Calcul formel
et
Mathématiques
avec
`Xcas`

Renée De Graeve
Maître de Conférence à Grenoble I

Remerciements

Je remercie :
– Bernard Parisse pour ses précieux conseils et ses remarques sur ce texte,

Ce document est disponible à l'adresse Internet suivante :
http://www-fourier.ujf-grenoble.fr/~parisse/cascmd_fr.pdf

Préface

Bernard Parisse
Maître de Conférences à l'Université de Grenoble I
Développeur du logiciel de calcul formel `giac` et de son interface `Xcas`. La version à jour se récupère sur ;
`http://www-fourier.ujf-grenoble.fr/~parisse/giac.html`

Table des matières

Index

1. **Dans l’index, selon le style on a une** commande **ou une** *option* **ou une** `valeur`

Pour commencer

0.1 Style de l'index et notations

0.1.1 Notes concernant l'index de ce manuel

Dans l'index de ce document, les commandes de `Xcas` seront écrites avec des caractères normaux, les options de ces commandes seront écrites en italique et les valeurs de commandes ou d'options seront écrites en mode typewriter.
Par exemple :
- affichage est une commande,
- *affichage* est une option de commande géométrique,
- `bleu` est une valeur.

0.1.2 Remarques concernant les notations

Au cours du document ce qu'il faut taper dans `Xcas` sera écrit en mode typewriter. On tape par exemple :
`affichage(droite(0,1+i),bleu)`
`droite(0,1+i, affichage=bleu)`
alors que dans l'index, les commandes, les *options des commandes* et `les valeurs de ces options` seront écrites différemment.

Lorsque l'on doit appuyer sur 2 touches en même temps on reliera ces deux touches avec `+`. Par exemple, si on doit appuyer en même temps sur `Alt` et sur `t` on écrira `Alt+t`.

Lorsque l'on veut indiquer le choix à faire dans un menu on reliera les différents sous-menus avec ►. Par exemple, pour indiquer comment ouvrir l'écran des sorties graphiques intermédiaires d'un programme on écrira soit :
menu `Cfg` puis `Montrer` puis `DispG` ou encore,
`Cfg`►`Montrer`►`DispG`

0.2 La librairie `giac` et ses interfaces sous Unix

`giac` est la bibliothèque C++ de fonctions de calcul formel.
Pour connaitre le numéro de la version de `giac` que vous utilisez, on tape :

```
version()
```

On obtient par exemple :

```
"giac 1.1.0, (c) B. Parisse and R. De Graeve, Institut
        Fourier, Universite de Grenoble I"
```

Sous Unix, on peut utiliser cette bibliothèque de calcul formel avec plusieurs interfaces :

- Xcas interface qui est une feuille de calcul permettant d'avoir des niveaux de différentes natures (calcul formel, géométrie dynamique et formelle, tableur formel, dessin tortue, langage de programmation etc...),
- icas ou giac interface en ligne de commande,
- texmacs,
- emacs,
- un programme C++,
- un module C++ pour créer de nouvelles fonctions.

0.2.1 Interface Xcas

On tape `xcas &` on a alors une feuille de calcul avec différents menus `Fich Edit Cfg Aide...` et différents boutons.
Cette interface est détaillée dans :
`L'interface Xcas de giac`
que l'on ouvre en cliquant sur le menu `Aide -> Interface`.
On va aussi détailler l'utilisation de cette interface ci-après (cf 1)

0.2.2 Interface en ligne de commande

Depuis une fenêtre de commandes, on peut taper directement une commande avec des quotes et précédée de `icas` par exemple :
`icas 'factor(1-x^2)'`
ou encore taper `icas` , on obtient :
`0>>` il suffit alors de taper des commandes par exemple :
`factor(1-x^2)`
on obtient :
`(1-x)*(1+x)`
`1>>` etc...
`ctrl D` termine alors la session.
Remarque
On peut aussi taper :
`icas nom_de_fichier`
pour executer les commandes contenues dans `nom_de_fichier`. **Attention**
Sous windows, le programme `icas` (interface texte de `giac`) existe mais il est compliqué à mettre en oeuvre en mode interactif car il faut disposer d'un programme terminal supportant les entrées et sorties console/écran. On ne l'utilise donc qu'en mode "batch" (par exemple pour faire des tests automatiques). Sous Mac OS et sous linux, `icas` fonctionne par contre très bien.

0.2.3 Interface texmacs

On peut utiliser `giac` dans une session de `texmacs` en tapant :
`texmacs &` puis en cliquant sur l'icône qui représente un écran de moniteur, on choisit `giac`. On a alors le symbole d'invite > pour utiliser les fonctions de calcul formel qui se trouvent dans le menu `Giac`.
On peut taper différentes commandes, même graphiques, et obtenir la réponse ou

le graphique, en dessous de la question. Il faut savoir que le pavé numérique ne fonctionne pas toujours bien dans `texmacs`, et que `^` se tape en se servant de la touche `tréma/^`.
De plus on peut aussi avoir de l'aide sur les différentes fonctions de `giac` en tapant `?` suivi du nom de la fonction.
Dans `texmacs` on peut alterner facilement des calculs faits avec `giac` et des explications : il suffit d'écrire les explications en dehors du cadre destiné à la question suivante puis, de cliquer à nouveau sur l'icône représentant un écran de moniteur et de choisir `giac` pour pouvoir utiliser à nouveau `giac`.
Remarque
Il est facile de recopier avec la souris une réponse écrite dans `texmacs`.

0.2.4 Interface `emacs`

Pour utiliser `giac` dans `emacs`, installer `mupacs` disponible sur :
`http://mupacs.sourceforge.net`.
Pour utiliser `giac` à l'intérieur d'`emacs` on tape :
`emacs nomdufichier.mu`
Il faut obligatoirement mettre `.mu`, quitte à renommer ensuite le fichier.
Cela ouvre une fenêtre `emacs` ayant dans ces menus `MuPAD`.
Dans ce menu `MuPAD` on sélectionne `Start MuPAD`.
Dans la dernière ligne de `emacs` (celle tout en bas) vous devez avoir :
`Command to start mupad : giac -emacs`
sinon, vous faites la modification et vous validez en appuyant sur `enter`.
Vous pouvez alors faire des calculs en vous servant des fonctions de `giac`.
Il apparait :

```
» TEXTWIDTH:=79:
// :2: parse error at end of input
79:undef}
```

Mais il ne faut pas s'en préoccuper....ca marche !
Les commandes graphiques seront exécutées sur un écran `Xdvi` et le graphique est traduit en Latex dans `casgraph.tex`.

0.2.5 Utilisation dans un programme ou un module `C++`

Pour l'utilisation de `giac` à l'intérieur d'un programme `C++` ou pour définir de nouvelles fonctions de `giac`, vous trouverez des explications au chapitre "Utilisation de `giac` à l'intérieur d'un programme".

0.2.6 Savoir avec quelle version on travaille : `version giac`

Pour savoir avec quelle version on travaille, on utilise la commande `version` ou `giac`.
On tape :

```
version()
```

Ou on tape :

```
giac()
```

On obtient par exemple :

```
"giac 0.4.0"
```

Chapitre 1

L'interface Xcas

1.1 Mise en route de l'interface Xcas

1.1.1 Sous Unix

On tape simplement :
`xcas &`
Remarque utile !
Si sous Linux, Xcas ne répond plus, tapez dans la fenêtre xterm :
`killall xcas` et relancez en tapant `xcas & enter`. Dans ce cas, il y a eu un fichier de sauvegarde automatique et lors de la reprise de Xcas on vous demande si vous voulez l'exécuter.

1.1.2 Sous Windows

Utilisez l'explorateur pour aller dans le répertoire où est installé Xcas.
Cliquez sur le fichier `xcasnfr.bat`.
Tout se passe ensuite presque comme sous Unix lorsqu'on tape `xcas &`.

1.1.3 Sous MacOS

Depuis le Finder, double-cliquez sur le fichier `xcas_image.dmg`.
À présent, double-cliquez sur l'icône disque Xcas.
Pour lancer Xcas, double-cliquez sur le programme Xcas du disque Xcas.

1.2 Les différents niveaux d'entrée

Cette interface va vous permettre d'ouvrir plusieurs sessions de calculs : ces sessions ont plusieurs niveaux d'entrée, sont indépendantes les unes des autres et peuvent être pliée ou dépliée.
Chaque session peut contenir des niveaux d'entrée numérotés qui contiendront :
- soit une ligne de commandes pour exécuter des commandes de Xcas ou pour exécuter des programmes avec un emplacement pour les sorties intermédiaires qui seront écrites en bleu et un emplacement pour la réponse (que l'on obtient si on a validé la commande avec `Enter`). L'emplacement pour la réponse est un éditeur d'expressions, ou un écran graphique selon la nature

de la commande (commande renvoyant une expression ou un graphique). Lorsqu'on a, sur un même niveau, plusieurs commandes separées par un point virgule ou par une virgule, ces commandes sont faites es unes après les autres et c'est la nature de la dernière commande qui détermine la nature de la sortie. Par exemple, la réponse à la ligne : `a:=1;cercle(0,a)` sera un cercle de centre 0 et de rayon 1.
Les instructions `print` d'un programme ou l'évaluation du temps d'un long calcul seront écrites en bleu. L'évaluation du temps est approximatif, si vous voulez plus de précision sur votre temps de calcul, il faut utiliser la commande `time` qui renvoie le temps mis pour l'évaluation en secondes.
`time` a comme argument une commande et renvoie deux temps (le temps CPU et le temps chronomètè en secondes) sous linux et seulement le temps CPU sur mac et win.
On tape par exemple :
`time(factor(x^10-1))`
On obtient en mode réel et sous Linux :
`[0.0022,0.0053107029]`
On tape par exemple :
`time(factor(x^100-1))`
On obtient en mode complexe et sous Linux :
`[1.93,1.371075654]`
Vous remarquerez alors : `Temps mis pour l'évaluation : 1.93` en vert. Le premier nombre est le temps CPU (temps mis par le processeur pour faire uniquement ces calculs, en seconde), le deuxième nombre est le temps mis pour faire le calcul comme si on chronomètrait. Pour les commandes qui donnent un résultat rapidement, elles sont exécutées plusieurs fois et la moyenne de temps de calcul est renvoyée.
On a aussi `time()` sans argument, qui renvoie le temps de calcul depuis le début de la session, on tape :
`time()`
On obtient par exemple :
`2.37`

– soit un éditeur d'expressions,
– soit un niveau de géométrie 2-d, son écran graphique, ses menus, ses boutons et ses lignes de commandes,
– soit un niveau de géométrie 3-d, son écran graphique, ses menus, ses boutons et ses lignes de commandes,
– soit un niveau de dessin tortue, son écran graphique, son éditeur de programmes et ses lignes de commandes,
– soit un tableur et son écran graphique,
– soit un éditeur de programmes,
– soit un commentaire,
– soit un regroupement de ces niveaux en un groupe.

Au sein d'une même session, les différents niveaux d'entrée ne sont pas indépendants, par exemple, une variable définie dans une ligne de commandes pourra être utilisée en géométrie ou dans le tableur.
L'ensemble de toutes ces sessions constitue votre espace de travail.
Le niveau actif est celui où se trouve le curseur et le niveau selectionné est obtenu

quand on clique sur son numéro, numéro qui sécrit alors sur fond noir.
On peut déplacer un niveau ou un groupe de niveau dans une session, ou le recopier dans une autre session.
Vous pouvez , à tout moment insérer un nouveau niveau ou encore changer l'entrée d'un niveau : `Enter` valide le changement de ce niveau et sélectionne l'entrée suivante, mais attention les niveaux suivants ne seront pas recalculés. Il est toutefois possible après une modification de réexécuter, soit tous les niveaux, soit les niveaux situés après la modification (menu `Edit` puis `Executer session` ou `Executer en-dessous`).
Il faut savoir qu'il suffit de faire :

– `Alt+c` pour ouvrir un niveau de type ligne de commentaires,
– `Alt+e` pour ouvrir un niveau de type éditeur d'éxpressions,
– `Alt+n` pour ouvrir un niveau de type ligne de commandes,
– `Alt+t` pour ouvrir un niveau de type tableur,
– `Alt+p` pour ouvrir un niveau de type éditeur de programmes.
– `Alt+g` pour ouvrir un niveau de type écran de géométrie plane,
– `Alt+h` pour ouvrir un niveau de type écran de géométrie 3,
– `Alt+d` pour ouvrir un niveau de type écran de dessin tortue,

1.3 Que voit-on au démarrage ?

Vous obtenez au démarrage l'ouverture d'une session avec de haut en bas :

1. La barre du menu général `Fich Edit Cfg...` contenant les fonctions de `Xcas` et ce qu'il faut pour les configurer et pour sauver ou charger une session de travail.
2. Une ligne des noms de sessions qui contiendra les noms (ou `Unnamed` si elles n'ont pas de noms) de vos différentes sessions. Au démarrage il n'y a qu'une session qui n'a pas de nom donc sur cette ligne il y a `Unnamed`,
3. Un bandeau général avec de gauche à droite :
   – Un bouton `?` permettant d'ouvrir le sous menu `Index` du menu `Aide`. Il sert aussi de touche de tabulation, car il ouvre le menu `Aide`à l'endroit indiqué par la ligne de commandes où figure le curseur,
   – Un bouton `Save`, sur fond rose, servant à sauver dans un fichier la session de calcul. Si on clique sur `Save`, on vous demande la première fois un nom de fichier d'extension `.xws` Quand la session a été sauvée le fond du bouton `Save` devient vert et `Unnamed` est remplacé par le nom de fichier, par exemple par `session1.xws`. Notez que les noms des sessions ont comme extension `.xws` et cette extension est rajoutée automatiquement si vous l'avez oubliée.
   – Un bouton `Config : exact real RAD 12 xcas 12.65M` ou `Config essai.xws : exact real RAD 12 xcas 12.65M` si on a sauvé la session sous le nom `essai.xws`.
   Ce bouton "ligne d'état" ouvre la fenêtre permettant de configurer le calcul formel et rappelle la configuration choisie : dans l'exemple, on est en mode exact et réel, les angles sont exprimés en radian, les calculs numériques sont faits avec 12 chiffres significatifs, le style de programmation est `xcas` et enfin les recources en mémoire demandées par le calcul sont

de 12.65M (c'est la taille mémoire en Mégabits utilisée par `Xcas`).
Sur cette "ligne d'état", le mode numérique sera noté `approx` et le mode complexe sera noté :
  - `cplx` si les variables ne sont pas considérées comme complexes (on a alors `re(z)=z` et `conj(z)=z`),
  - `CPLX` si les variables sont considérées comme complexes (on a alors `re(z)=re(z)` et `im(z)=im(z)` qui seront notés respectivement $\mathcal{R}(z)$ et $\mathcal{I}(z)$ dans la réponse).

  Notez que ce bouton "ligne d'état" ouvre la fenêtre de configuration qui est aussi obtenue à partir du menu : `Cfg`▶`Configuration du CAS`,
- Un bouton rouge `STOP` pour arrêter un calcul trop long,
- Un bouton `Kbd`, servant à faire apparaitre ou disparaitre un clavier scientifique. On remarquera sur ce clavier :
  - la touche `cmds` qui sert à faire apparaitre ou disparaitre une barre de boutons contenant les commandes du CAS, appelée bandeau du CAS. Dans le bandeau, on remarquera la touche `kbd` qui sert à faire apparaitre ou disparaitre le clavier.
  - la touche `msg`, servant à faire apparaitre ou disparaitre une fenêtre de messages facilement lisible grâce à sa barre de scroll. Cette fenêtre vous donne des messages comme **Success** pour dire que tout s'est bien passé, ou affiche une aide succinte sur la commande choisie à partir du menu général, ou ce qu'il faut insérer dans votre fichier LaTeXpour insérer la figure sauvée, par exemple, sous le nom `session1.eps` :
    Use `\includegraphics[width=\textwidth]{session1}`
    inside your latex document, with header
    `\usepackage{graphicx}`
  - la touche `abc`, servant à faire apparaitre ou disparaitre un clavier contenant les lettres en minuscule, par ordre alphabétique, ainsi que des signes de ponctuation. On notera qu'en appuyant sur la touche α on obtient un clavier contenant les lettres grecques et qu'en appuyant sur la touche `Maj` transforme le clavier avec des lettres en majuscule.
  - la touche ◀ qui permet de supprimer ce que l'on a sélectionné,
  - la touche `coller` qui permet de recopier ce que l'on a sélectionné,
  - la touche verte ↩ qui sert de touche `Enter`.
  - la touche `X` situé tout en haut à droite, permettant de faire disparaitre le clavier `Kbd`.
- Un bouton $times$ situé tout à droite, permettant de fermer la session en cours. Cela provoquera éventuellement un avertissement si les dernières modifications n'ont pas été sauvées.

4. Un premier niveau ou les deux premiers niveaux, selon le démarrage choisi.

1.4 Les menus

1.4.1 Le menu `Fich`

Il sert à la gestion des fichiers : on prendra l'habitude de nommer les fichiers contenant des fonctions écrites en langage `Xcas` avec des noms se terminant par

.cxx et les fichiers contenant des scripts (c'est à dire une suite d'instructions) avec des noms se terminant par .cas.
Fich a différents sous-menus :

- Nouvelle session pour ouvrir et créer une nouvelle session. Unnamed se mettra en surbrillance après les noms des sessions déjà ouvertes (la surbrillance indique le nom de la session ouverte) et il ne vous reste plus qu'à lui donner un nom se terminant par .xws (avec Sauver du menu Fich) pour pouvoir vous repérer,
- Ouvrir ou Alt+o pour ouvrir une nouvelle session et charger une session sauvée précédemment,
- Importer pour ouvrir une session que l'on a réalisée et sauvée soit avec le logiciel Maple dans un fichier en .mws (veillez à utiliser l'"ancien" format de sauvegarde avec Maple 9, 10,...), soit avec l'une des calculatrices ti89 ou Voyage200. Xcas récupère les commentaires et les entrées.
  On peut alors faire exécuter ces entrées avec Executer session du menu Edit de la session, mais il est préférable de faire une exécution pas à pas en validant chaque niveau, afin de voir où il y a des modifications à faire,
- Inserer pour insérer une session sauvée auparavant dans votre session,
- Sauver ou Alt+s ou le bouton Save pour sauver cette session (c'est à dire tous ses niveaux d'entrée et de sortie) dans le fichier de nom indiqué à coté du bouton Save. La première fois on vous demandera le nom du fichier de sauvegarde qui doit se terminer par .xws. Ce nom remplacera Unnamed dans la ligne des noms des sessions et sera en surbrillance. Il s'inscrira aussi sur le bouton de configuration à coté de Save.
- Sauver comme pour sauver en donnant un autre nom au fichier de sauvegarde de la session, ce fichier doit être un .xws,
- Sauver tout pour sauver tout votre espace de travail c'est à dire toutes les sessions, ce fichier doit être un .xws,
- Exporter comme pour sauver la session au format texte choisi (soit xcas, soit maple...)
- Fermer sans sauver pour fermer la session en cours sans la sauver,
- A propos ouvre la fenêtre des messages avec l'adresse http où vous pouvez vous procurer la dernière version de Xcas ainsi que l'adresse mail du développeur ! (voir aussi le menu Aide▶Internet)
- Imprimer pour imprimer la session en cours, il faut tout d'abord choisir son format d'impression et cocher ou ne pa cocher Paysage en utilisant le menu Cfg▶ Configuration generale, puis, choisir :
  - Pre-visualisation pour voir votre session en postscript (on vous demandera un nom de fichier .ps) ou,
  - vers imprimante pour imprimer votre session en postscript (on vous demandera le nom de l'imprimante) ou,
  - Previsualiser les niveaux selectionnes pour prévisualiser les niveaux selectionnés (on vous demandera un nom de fichier .eps pour chaque niveau, par exemple niveau3.eps). On pourra ensuite inclure ce fichier dans un texte LaTeX en mettant :
    - dans l'en-tête :
      \usepackage{graphicx}
    - et dans le texte à l'endroit désiré :

`\includegraphics{niveau3}`

- LaTeX convertit toute la session en un fichier LaTeX, qui sera compilé et affiché,
- `Capture ecran` permet de faire une capture d'écran qui sera sauvée dans un fichier de suffixe `.eps` (par exemple `window.eps`). On pourra ensuite inclure ce fichier dans un texte LaTeX en mettant :
  - dans l'en-tête :
    `\usepackage{graphicx}`
  - et dans le texte à l'endroit désiré :
    `\includegraphics{window}`
- `Quitter` ou `Alt+q` permet de quitter `Xcas` quand on a terminé.

1.4.2 Le menu `Edit`

- `Executer session` pour recalculer cette session entièrement,
- `Executer en-dessous` pour recalculer cette session à partir d'une entrée sélectionnée,
- `Enlever les reponses` pour supprimer les réponses à partir d'une entrée sélectionnée afin de pouvoir, devant un auditoire, refaire pas à pas l'exécution des différentes entrées.
- `Inserer saut de ligne` pour obtenir une nouvelle ligne sur le niveau d'entrée, là où se trouve le curseur et permet ainsi de passer à la ligne par exemple pour écrire dans un même niveau plusieurs lignes d'entrées séparées par `;` (on peut aussi taper `Shift+Enter`).
- `Annuler` ou raccourci `Ctrl+z` pour annuler la dernière exécution : par exemple vous avez effacé malencontreusement un niveau et vous voulez annuler cet effacement il faut faire `Annuler`, ou bien vous avez modifié et exécuter une ligne de commande pour annuler cette modification il faut faire `Annuler`, mais si il n'y a pas eu d'exécution (par exemple on a effacer une expression dans la ligne de commandes) il ne faut pas faire `Annuler`,
- `Redo` ou raccourci `Ctrl+y` pour revenir à ce qu'il y avait avant l'annulation : il faut faire 2 fois `Redo` si vous avez fait 2 fois `Annuler`,
- `Coller` permet de recopier, à l'endroit du curseur, ce que l'on a sélectionné avec la souris (analogue à la touche `coller` du clavier `Kbd`). On peut aussi sélectionner ce qui se trouve dans une ligne de commandes avec `Shift+les flèches de déplacement`.
- `Effacer niveaux selectionnes` permet de supprimer les niveaux sélectionnés,
- `selection->LaTeX` ou raccourci `Ctrl+t` pour traduire en `LaTeX` la question ou la réponse ou le contenu de l'éditeur d'expressions : on sélectionne une question ou une réponse ou le contenu d'un éditeur d'équations [1]. Remarquez le petit bouton `M` situé à l'intersection des 2 barres de scroll de l'éditeur d'expressions qui permet soit de recopier ce qui se trouve dans l'éditeur d'un seul clic, (avec `Selectionner tout`), soit d'évaluer la sélection (avec `Evaluer selection`), soit de retrouver la réponse initiale (avec `Annuler`), dans le cas où, on a fait une modification malen-

1. Si vous voulez traduire une équation en `LaTeX`, tapez votre équation entre deux `'` ce qui l'affichera dans un éditeur d'expressions.

contreuse car comme les réponses non graphiques sont affichées dans un éditeur d'expressions, elles peuvent donc être modifiées..., Lorsqu'on utilise `Edit->selection->LaTeX` ou `Ctrl+t`, la traduction en `LaTeX` apparait alors dans l'écran des messages (que l'on peut voir en cliquant sur le bouton `msg` de `Kbd`).
On peut la recopier dans un éditeur (`emacs, nedit, vi, ...`) par un clic avec le bouton du milieu.
Par exemple :
- on saisit dans un niveau de calcul formel `concat([1,2],3)` on met en surbrillance la réponse, puis on fait `Ctrl+t`, ce qui provoque l'inscription dans la partie reservée aux messages de :
  `\mbox{concat}([1,2],3)`
- on saisit `sqrt(1+3)`, on met en surbrillance la réponse, puis on fait `Ctrl+t`, ce qui provoque l'inscription dans la partie reservée aux messages de :
  `\sqrt{1+3}`
- on saisit `f(x):=sin(x)/x`, on met en surbrillance la réponse, puis on fait `Ctrl+t`, ce qui provoque l'inscription dans la partie reservée aux messages de :
  `\parbox{12cm}{\tt  (x)-{\tt\symbol{62}}(sin(2*x))/x }`

Il ne vous reste plus qu'à recopier le texte ainsi traduit, d'un coup de souris, dans une portion en mode mathématique de votre document LaTeX.
- `Fusionner niveaux` permet de mettre dans un même niveau, les niveaux sélectionnés,
- `Nouveau groupe` permet de créer un groupe,
- `Grouper niveaux` permet de créer un regroupement des niveaux sélectionnés, que l'on peut plier ou déplier en cliquant sur $\boxed{-}$ ou $\boxed{+}$ du menu propre à ce regroupement. On peut aussi donner un nom au regroupement ainsi créé.
- `Degrouper niveaux` effectue l'opération inverse de la précédente (on aplatit le groupe dont le numéro a été sélectionné (noirci) pour revenir à l'état précédent le regroupement).

1.4.3 Le menu `Cfg`

- `Configuration du CAS` permet de configurer le calcul formel (idem que le bouton "ligne d'état" situé entre `Save` et `Stop`),
- `Configuration graphique` permet de configurer le graphique par défaut : vous pouvez aussi avoir une configuration spécifique pour chaque graphique avec le bouton `cfg` du graphique, mais cela ne changera pas la configuration par défaut,
- `Configuration generale` permet de déterminer la taille des caractères, le navigateur, si on veut une aide automatique ou pas , le format de l'impression et de cocher ou pas `Paysage` (si cette case est cochée l'impression se fera selon la largeur de la feuille, si elle n'est pas cochée l'impression se fera selon la hauteur de la feuille), le nombre de lignes et de colonnes du tableur, et aussi le nombre d'appels récursifs autorisés,

- `Mode (syntax)` permet de travailler avec une syntaxe `Xcas` (qui est de type `C`), soit avec une syntaxe Maple ou TI ou MuPad ou Tortue,
- `Montrer` puis
  - `DispG` pour voir l'écran `DispG` sur lequel est enregistré toutes les commandes graphiques depuis le début de la session, sans distinction de niveau. Il permet en particulier de visualiser les affichages graphiques intermédiaires d'un programme (en effet seuls les objets graphiques renvoyés par `return` ou `retourne` peuvent être affichés en réponse dans un niveau où on exécute un programme). (voir aussi la commande `DispG()` 8.1.3 et 8.6.4).
  - `Clavier` pour avoir un clavier scientifique. Ce clavier se met en bas de la fenêtre.
  - `Bandeau` pour avoir les commandes de `Xcas` dans un bandeau. Ce bandeau se met juste après le clavier. Il permet d'afficher un sous-menu de manière persistante. Le bandeau est formé d'une ligne qui permet d'avoir facilement à sa disposition les commandes qui se trouvent dans les différents menus de la barre de menu. On appuie par exemple, sur `Geo` (écrit en rouge) puis, sur `Triangles` (écrit en rouge) pour avoir dans le bandeau les commandes géométriques de `Xcas` (écrites en noir) dessinant des triangles.
    La touche `home` permet de revenir au bandeau initial contenant :
    `Expression, Geo, ,Prg, Graphic.. BACK`.
    Les touches » et « permettent de circuler dans la ligne du bandeau.
    La touche `BACK` permet de revenir au bandeau précédent.
    La touche `cust` pour custom permet de mettre les noms de ses propres commandes dans un menu défini dans la variable `CST`, `CST` doit contenir une liste qui sera le menu affiché (voir aussi 2.5.1).
    Par exemple `CST:=[1,2,3]` va afficher comme menu `1, 2, 3`. On pourra donc faire un bandeau personnalisé en mettant dans `CST` les noms des fonctions que l'on veut utiliser ou les noms des fonctions que l'on a créées.
    Par exemple, on définit la fonction `f(x):=x^2+2*x+3` et on tape :
    `CST:=[evalc,["f",f],["euro",6.55957]]`
    On obtient quand on appuie sur `cust`, un bandeau qui contient comme menu :
    `evalc f euro`.
    puis, on définit la fonction `f(x):=x^2+2*x+3` et on peut ainsi appeler `f` depuis `eqw` :
    on met par exemple 3 dans `eqw` puis on appuie sur `f` et on obtient 18.
    On peut ainsi appeler `evalc()` dans une ligne d'entrée en appuyant sur `evalc` ou encore on utilise la touche `euro` dans un calcul.
  - `Msg` pour avoir la fenêtre des messages. La fenêtre des messages se met juste après le bandeau.
- `Cacher` puis
  - `DispG` pour ne plus voir l'écran `DispG` sur lequel est enregistré toutes les commandes graphiques effectuées depuis le début de la session,
  - `Clavier` pour ne plus voir le clavier scientifique,
  - `Bandeau` pour ne plus voir les commandes de `Xcas` dans le bandeau,

  - `Msg` pour ne plus voir la fenêtre des messages,
- `Langue de l'aide` pour choisir d'avoir l'aide en français, en anglais ou en espagnol,
- `Couleurs` pour choisir la couleur de l'affichage selon son type,
- `Police session` permet de changer la police et la taille des caractères de la session en choisissant la police et la taille de la fonte (par défaut police helvetica de taille 20),
- `Polices (toutes)` permet de changer la police et la taille des caractères de la session, du menu principal et du clavier en choisissant la police et la taille de la fonte (par défaut police helvetica de taille 20)
- `Navigateur` permet de donner le nom de votre navigateur,
- `Sauver préférences` permet de sauver les différentes configurations choisies avec le menu `Cfg` ou avec le bouton donnant la ligne d'état et ainsi de les retrouver pour des utilisations ultérieures.

1.4.4 Le menu `Aide`

Ce menu contient les différentes formes d'aide possible.

- `Index` Lorsque l'on sélectionne `Index`, cela ouvre un écran avec plusieurs plages donnant pour toutes les commandes, une aide succincte, les commandes proches, des exemples .
  Les différentes plages :
  Vous voyez sur la plage en haut et à gauche toutes les commandes utilisables classées par ordre alphabétique avec en dessous une ligne d'entrée qui permet de consulter cette liste facilement : il suffit de taper le début d'un nom dans cette ligne pour avoir le curseur à cet endroit dans la liste, vous pouvez ainsi aller directement à une lettre ou à une commande.
  En cliquant sur l'une de ces commandes, vous avez :
  - à droite les commandes proches de la commande mise en surbrillance,
  - en dessous la ligne d'entrée
  - en dessous de la ligne d'entrée, une aide succincte qui s'affiche aussi dans le bandeau général à l'endroit des messages et une aide plus complète s'affichera, soit dans Mozilla sous Linux, soit dans un écran à part. Sous Linux, il est commode d'ouvrir Mozilla et de l'icônifier pour pouvoir ouvrir cette aide si cela est necessaire.
  - des exemples que l'on peut copier pour les exécuter : il suffit d'avoir le curseur dans une ligne ce commande, puis de cliquer sur l'un des exemples pour que cet exemple soit recopié dans la ligne où se trouve le curseur.

  Il faut savoir aussi qu'en tapant le début d'une commande dans une ligne de commandes, puis sur la touche de tabulation, on obtient toutes les commandes de l'`Index` commençant par ce début : vous pouvez alors cliquer sur l'un des exemples de la commande mise en surbrillance pour que ce début de commande soit complété par cet exemple. Vous pouvez aussi taper `?nom_de_commande` pour avoir comme réponse l'aide succincte sur cette commande et éventuellement taper `?nom_de_commande` suivi de `,` (virgule) `1` ou `2` ou `3` pour indiquer le langage utilisé (`1` français, `2` anglais, `3` espagnol) dans la réponse : par exemple `?factor,2` pour avoir comme réponse l'aide de `factor` en anglais.

Remarque Quand on choisit les commandes à partir des menus, une aide succincte sur cette commande s'affiche dans la ligne des messages (cliquez sur `msg` du bandeau) et le manuel de Calcul formel s'ouvre à la bonne page ! Quand on choisit les commandes à partir du bandeau, seule une aide succincte sur cette commande s'affiche dans la ligne des messages (cliquez sur `msg` du bandeau).

- `Trouve` recherche le mot demandé dans toutes les pages du manuel Calcul formel.
- `Interface` contient de l'aide concernant l'interface de `Xcas`.
- `Manuels`
  1. `Calcul formel` contient l'aide générale qui concerne toutes les fonctions de calcul formel, de géométrie, de statistiques mais qui ne concerne pas les instructions de programmation, ni les instructions déplaçant la tortue.
  2. `Geometrie` contient une aide plus détaillée pour certaines commandes car cette aide est illustrée par des exercices (mais on n'a pas toutes les fonctions de géométrie !).
  3. `Programmation` contient une aide détaillée des instructions de programmation. Dans cette partie vous trouverez l'écriture de plusieurs algorithmes avec la traduction de ces algorithmes en langage `xcas MapleV MuPAD TI89/92`.
  4. `Tableur,statistiques` contient une aide détaillée concernant le tableur et les fonctions de statistiques ainsi que leurs utilisations dans le tableur.
  5. `Tortue` contient l'aide concernant les instructions qui sont utilisées dans l'écran de `dessin Tortue`. Dans cette partie vous trouverez plusieurs activités que l'on peut faire avec des enfants (du CP au CM2) dans le but de leur faire faire des mathématiques.
- `Internet`
  1. `Forum` permet d'accéder à un forum de discussion.
  2. `Ressources pedagogiques` contient des exercices sur différents sujets et de différents niveaux.
  3. `Mettre a jour l'aide` permet de mettre l'aide à jour.
- `Débuter en calcul formel`
  1. `Tutoriel` est un tutoriel. Il permet de faciliter la prise en main de `Xcas` pour faire du calcul formel.
  2. `solutions` contient les solutions des exercices du tutoriel.
- `Recreer les fichiers index de l'aide`
  Le bouton `Details` de la fenêtre d'index de l'aide ouvre le navigateur sur la page principale d'aide de la commande en cours si elle existe. Pour cela il utilise un fichier cache contenant les correspondances, mais il arrive que ce fichier cache ne soit pas à jour, ce menu permet de mettre à jour ce fichier cache (il faut bien sur avoir les droits d'écriture sur le fichier cache).

– `Exemples` Si vous sélectionnez par exemple l'exemple de nom `glace.xws` du sous menu `climat`, alors, ce fichier sera recopié du répertoire :
`/usr/local/share/giac/examples/Exemples/climat/`
dans votre répertoire courant et `Xcas` vous ouvrira une nouvelle session `glace.xws` qui contiendra ce fichier. Toutefois si un fichier dumême nom existe, on vous demandera auparavant si vous voulez le remplacer puis l'ouvrir ou bien ouvrir celui du répertoire courant sans l'écraser.

Remarques

– Quand on choisit les commandes à partir des menus, une aide succincte sur cette commande s'affiche dans la fenêtre des messages (cliquez sur le bouton `Msg`) et par défaut, le manuel de Calcul formel s'ouvre à la bonne page
– On peut activer ou désactiver ce mécanisme d'aide automatique dans le menu `Cfg,Configuration generale,Aide HTML automatique`.
– Quand on choisit les commandes à partir du bandeau, seule une aide succincte sur cette commande s'affiche dans la fenêtre des messages (cliquez sur la touche `msg` du bouton `Kbd` ou avec le menu `Edit`▶`Montrer`▶`Msg`).

1.4.5 Les menus des commandes de calcul

Ces menus permettent d'ouvrir un niveau adéquat par exemple :
– `CAS`▶`Nouvelle entree` ou `Alt+n` ouvrira une nouvelle ligne de commandes
– `Tableur`▶`Nouveau tableur` ou `Alt+t` ouvrira un tableur,
– `Geo`▶`Nouvelle figure`▶`graph,geo2d` ou `Alt+g` ouvrira un niveau de géométrie où les points ont des coordonnées décimales,
– `Geo`▶`Nouvelle figure`▶`geo2d exact` ouvrira un niveau de géométrie où les points ont des coordonnèes exactes,

Certains menus sont des menus dit "Assistant" car les commandes sont classées par thème et sont explicitées. Ces commandes sont facilement utilisables soit parce que l'aide s'ouvre sur la commande choisie (menu `CAS`), soit parce qu'une boite de dialogue vous demande de préciser les paramètres de la commande choisie (menus `Tableur`▶`Maths` ou menu `Graphic`).
Les autres menus contiennent les noms des commandes (le menu `Cmds` contient toutes les commandes de calcul formel, le menu `Geo` contient toutes les commandes de géométrie...)

Déscription des menus des commandes de calcul

Ces menus sont :
– `CAS` contient les fonctions de calcul formel classées par thème. Ce menu vous permet de connaitre le nom de la commande `Xcas` que vous cherchez car ce nom est suivi d'un bref déscriptif et l'`Index` de l'`Aide` s'ouvre automatiquement sur la commande choisie.
– `Tableur` ce menu est un menu "Assistant". On retrouve les trois sous-menus `Table Edit Maths` qui sont identiques aux menus d'un niveau de type tableur. Avec le menu `Tableur`▶`Maths` le tableur se remplit automatiquement grâce à une boite de dialogue qui vous demande de préciser les paramètres de la commande choisie à condition que le curseur soit dans

un niveau de type tableur (tableur que l'on ouvre avec `Alt+t`).

- `Graphic` contient les fonctions permettant de tracer des graphes de fonctions, des courbes en paramétrique ou en polaire, des solutions d'expressions différentielles, pour visualiser "l'escargot" des suites récurrentes...
On notera que le niveau utilisé pour tracer un graphe est un niveau d'entrée normale car une commande graphique ouvre automatiquement en réponse un écran graphique et une commande non graphique ouvre automatiquement en réponse un éditeur d'expressions.
- `Geo` contient les fonctions permettant de faire de la géométrie interactive. On notera le menu `Affichage` qui contient la commande `affichage` et ses paramètres concernant la couleur, les différentes sortes de lignes, les différentes sortes de points et l'emplacement des légendes.
- `Prg` contient les instructions permettant d'écrire des programmes.
- `Expression` contient les fonctions de calcul formel permettant de transformer une expression. Ces fonctions peuvent par exemple être appliquées à la sélection d'un éditeur d'expressions (la réponse d'une commande est mise dans un éditeur d'expressions, mais on peut aussi ouvrir un niveau éditeur d'expressions avec `Alt+e`).
- `Cmds` contient les fonctions mathématiques classées par thème.
- `Phys` contient toutes les unités physiques, les constantes physiques et des fonctions de conversion.
- `Scolaire` contient des commandes de calcul formel classées par niveau. Les sous-menus `Seconde Premiere Terminale` contient les différentes fonctions de calcul formel utilisables dans les classes de Lycée.
Vous pouvez rajouter ou supprimer des commandes dans ce menu selon vos besoins en modifiant le fichier `xcasmenu` (cf.la section 1.4.5).
- `Tortue` contient toutes les commandes qui sont valides dans l'écran de dessin Tortue (écran de dessin Tortue que l'on ouvre avec `Alt+t`). Ces commandes sont proches du langage LOGO et permettent de faire des dessins en donnant des ordres à un robot (une tortue).

Rajouter un menu

On peut redéfinir les menus que l'on voit au-delà du menu `Aide`. Par exemple, rajoutons un menu `Exo1` qui contiendra les commandes `equal2diff factor subst` qui seraient utiles pour faire un exercice numéroté 1.

Pour cela il faut ouvrir dans l'éditeur texte de votre choix (par exemple `emacs, vi, nedit, notepad, bloc.notes...` mais PAS `word, abiword, kword, openoffice...`) le fichier `xcasmenu` (sous Linux ou Mac, il faut modifier ce fichier dans le répertoire `/usr/share/giac/doc/fr` ou `/usr/local/share/giac/doc/fr`).

Vous devez voir :
```
Math/Constants/pi
Math/Constants/i
Math/Constants/e...
```
Vous tapez alors les trois lignes suivantes (une ligne par commande) :
```
Exo1/equal2diff
Exo1/factor
```

`Exo1/subst`
en laissant le reste
`Math/Constants/pi`
`Math/Constants/i`
`Math/Constants/e...`
inchangé.

Lorsque vous relancez `Xcas` vous avez maintenant après le menu `Aide` un menu `Exo1` qui contient les commandes `equal2diff factor subst`.

Remarque : le menu `Exemples` suit le même principe avec un fichier `xcasex`, mais il faut aussi avoir créé les fichiers de session exemple dans le répertoire de `xcasex`.

Supprimer un menu

Pour cela il faut ouvrir dans l'éditeur de votre choix le fichier `xcasmenu`. Il suffit alors d'effacer les lignes que vous ne voulez pas voir apparaitre en ayant soin de les sauver pour pouvoir vous en reservir ultérieurement !

1.5 Comment bien gérer son espace de travail

1.5.1 Pour sélectionner ou désélectionner un niveau

Pour sélectionner un niveau, il faut cliquer sur le numéro du niveau que l'on veut sélectionner : ce numéro apparait alors sur fond noir.

Pour sélectionner plusieurs niveaux qui se suivent et se trouvent sur l'écran, il faut cliquer sans relacher sur le premier numéro du groupe et se deplacer jusqu'au dernier numéro du groupe puis relacher le bouton de la souris : ces numéros apparaissent alors sur fond noir. Pour sélectionner plusieurs niveaux qui se suivent on peut aussi cliquer sur le premier numéro du groupe et `Shift+cliquer` sur le dernier numéro du groupe : cela sélectionne les niveaux intermédiaires.

Pour désélectionner, on clique ailleurs dans la session (on peut aussi cliquer sur un seul des niveaux parmi les niveaux sélectionnés ce qui sélectionnera uniquement ce niveau).

1.5.2 Pour remplir les niveaux

On peut retrouver l'historique de ce que l'on a tapé précédemment avec `Shift+`↑ et `Shift+`↓.
On peut recopier une suite d'instructions situés dans un niveau dans différents niveaux, mais il faut que chaque instruction soit écrite sur une ligne différente (faire `shift+enter` pour changer de ligne dans un niveau). Par exemple pour recopier :

```
C:=cercle(0,1,affichage=hidden_name);
Q:=projection(C,P)
```

dans 2 niveaux on sélectionne les 2 instructions avec la souris, puis on clique directement avec le bouton du milieu dans le numero du niveau sans le noircir.

1.6 Les différentes configurations

Les différentes configurations se font :
- pour la configuration du calcul formel avec le menu `Cfg`►`Configuration du CAS` ou avec le bouton donnant la ligne d'état (c'est la ligne écrite à coté de `Save` et qui rappelle une partie de cette configuration).
- pour la configuration graphique avec le menu `Cfg`►`Configuration graphique` .
- pour la configuration générale avec le menu `Cfg`►`Configuration generale`.

1.6.1 Configuration du Cas

- `Prog style` si on veut programmer en un autre langage : on peut choisir `maple`, `mupad` ou `ti89/92`,
- `eval` : nombre maximum d'évaluations récursives en mode interactif. Par exemple, si on exécute dans l'ordre `a:=b+1` et `b:=5` puis on évalue `a`, la valeur renvoyée sera `b+1` si `eval` vaut 1 et `6` si `eval` est plus grand que 1.
- `prog` : comme ci-dessus mais lorsqu'un programme est exécuté. On utilise en principe le niveau 1 d'évaluation à l'intérieur d'un programme.
- `recurs` pour déterminer le nombre maximum d'appels récursifs : par défaut c'est 50. Il n'y a pas de limite imposée par l'interface de `Xcas`, par contre si on met une borne trop élevée, c'est la pile du programme qui risque de déborder en provoquant un segmentation fault. Le moment où cela se produit dépend de la fonction, car la pile est aussi utilisée pour les appels récursifs des fonctions C++ de giac,
- `debug` : affiche des informations intermédiaires (en bleu) sur les algorithmes utilisés par giac en fonction du niveau (0 = pas d'info).
- `maxiter` : nombre maximal d'itérations pour la méthode de Newton
- `Flottants` on peut choisr `Standard Scientific Engineer` pour que l'affichage des nombres décimaux se fasse selon :
  - la notation standard : `150.12` sera noté `150.12`,
  - la notation scientifique : `150.12` sera noté `1.501200000000e+02`,
  - la notation des ingénieurs : `150.12` sera noté `150.12e0`,
- `Chiffres` pour détérminer le nombre de chiffres significatifs. Si on choisit moins de 14 chiffres, les calculs sont quand même faits avec 14 chiffres mais l'affichage n'en montre que ce que l'on demande.
- `epsilon` détermine la valeur de `epsilon` utilisée dans `epsilon2zero`,
- `proba` : si cette valeur est non nulle, Giac peut utiliser des algorithmes non déterministes et renvoyer une réponse qui a alors une probabilité d'être fausse inférieure à la valeur donnée. C'est par exemple le cas pour le calcul du déterminant d'une grande matrice à coefficients entiers.
- `approx` on coche cette case si on veut travailler en mode numérique : si cette case n'est pas cochée on travaille en mode exact ou formel. Lorsqu'on travaille en mode exact on peut avoir une évaluation numérique en tapant sur la touche num ou en utilisant la commande `evalf` ou encore en mettant des nombres decimaux (par exemple 1.0/2=0.5 alors que 1/2 est inchangé),
- `taches` pour faire du parallelisme, dans le futur...,
- `Base (Entiers)` pour travailler en base 10, 16 ou 8,

– `radian` est l'unité d'angle si cette case est cochée, sinon l'unité d'angle est le degré,
– `puissance croissante` pour que l'affichage des polynômes se fasse selon les puissances croissantes après l'utilisation de commandes comme `normal` ou `simplify`,
– `Complex` on coche cette case pour travailler en mode complexe : si cette case n'est pas cochée on travaille en mode réel. Certaines commandes cependant sont indépendante du mode choisi comme `cpartfrac`, `csolve` ou `cFactor`,
– `Variables_complexes` on coche cette case pour travailler avec des variables formelles complexes : si cette case n'est pas cochée les variables formelles sont réelles (si la variable `a` n'est pas affectée `re(a)=conj(a)=a` et `im(a)=0`). Mais attention dans `csolve` l'inconnue est toujours considérée comme une variable formelle complexe et dans `solve` l'inconnue est considérée comme une variable formelle complexe seulement si l'equation à résoudre contient `i` ou bien si on est en mode complexe,
– `puissance croissante` : affiche les développements de polynômes selon les puissances croissantes ou décroissantes
– `All_trig_sol` : renvoie toutes les solutions d'une équation trigonométrique (par exemple `solve(cos(x)=0)`) à l'aide de variables entières ou seulement les solutions principales.
– `Sqrt` : si `Sqrt` est coché, `factor` factorise les polynômes du second degré, même si les facteurs ne sont pas dans le corps de base des coefficients.
– `Appliquer` pour avoir cette configuration,
– `Sauver` pour avoir cette configuration par défaut,
– `Annuler` pour revenir à l'ancienne configuration.

1.6.2 Configuration du graphique avec le menu : `Cfg▶Configuration graphique`

La configuration du graphique se fait avec le menu `Cfg▶Configuration graphique`.
Mais chaque graphique peut avoir sa configuration propre en utilisant le bouton `cfg` du bloc situé en haut à droite de l'écran graphique. `Montrer les axes` on coche cette case pour voir les axes (en 3d on peut aussi choisir de ne voir que le trièdre en décochant `Montrer les axes` et en cochant `Triedre`),
`X-` et `X+` pour déterminer l'axe des x de la fenêtre de calcul,
`Y-` et `Y+` pour déterminer l'axe des y de la fenêtre de calcul,
`Z-` et `Z+` pour déterminer l'axe des z de la fenêtre de calcul,
`t-` et `t+` pour déterminer la variation du paramètre t utilisé dans les courbes en paramétriques ou en polaires,
`WX-` et `WX+` pour déterminer l'axe des x de la fenêtre de visualisation,
`WY-` et `WY+` pour déterminer l'axe des y de la fenêtre de visualisation,
`x_rot` pour faire tourner une figure en `3-d` selon l'axe des x : une partie de l'écran de configuration graphique s'ouvre automatiquement lorsqu'on fait un graphe en `3-d` selon une petite fenêtre qui remplace momentanḿent `Xcas`. On utilise les touches + ou − pour faire tourner la figure et on ferme cette petite fenêtre avec `OK`,
`z_rot` pour faire tourner une figure en `3-d` selon l'axe des z, à l'aide les touches

+ ou −,
x_scale pour faire changer l'échelle de l'axe des x,
z_scale pour faire changer l'échelle de l'axe des z,
class_min pour définir en statistiques le minimum des classes,
class_size pour définir en statistiques la taille des classes,
autoscale on coche cette case pour avoir un réglage automatique de l'échelle en mode 2-d ou 3-d, ce qui provoque un changement du réglage des x, des y et des z,
ortho pour avoir un repère orthonormé contenant la partie visible demandée,
-> W et W->XY si on veut recopier les plages XY dans W et vice-versa,
TX et TY permet de marquer, lorsqu'on a les axes, les points de coordonnées les multiples de TX et TY, on met TX et TY à zéro lorsque l'on ne veut pas ces points,
OK pour confirmer, et on obtient par exemple :
xyztrange(-6.0,6,-7,4,-10,10,-1,6.0,-6,6,-2,4,1) dans l'historique, commande qui définit les nouveaux paramètres de la fenêtre graphique,
Annuler pour revenir à l'ancienne configuration.

1.6.3 Configuration générale

Format d'impression on a le choix entre différents formats : A4, A5, A3, Lettre Enveloppe,
Landscape si cette case est cochée l'impression se fera selon la largeur de la feuille, si elle n'est pas cochée l'impression se fera selon la hauteur de la feuille,
Fonte historique pour déterminer la taille des fontes de l'historique,
Level si on choisit Tortue, cela ouvrira Xcas avec 2 niveaux : un écran de dessin tortue en niveau 1 et un éditeur de programmes en niveau 2,
Aide HTML automatique on coche cette case pour avoir l'aide detaillée à chaque appel de fonction se trouvant dans les menus. Si cette case n'est pas cochée on a toujours une aide succincte qui s'affiche dans le bandeau général à l'endroit des messages et on a peut aussi avoir de l'aide en tapant ? suivi du nom de la commande et éventuellement de , (virgule) 1 ou 2 ou 3 pour indiquer le langage utilisé (1 français, 2 anglais, 3 espagnol), (par exemple ?iquo ou ?iquo,2) ou, en cliquant sur le menu Aide pour ouvrir toutes les aides possibles,
Lignes pour déterminer le nombre de lignes de l'éditeur de matrices (ou du tableur),
Cols pour déterminer le nombre de colonnes de l'éditeur de matrices (ou du tableur),
Prévisualisation pour déterminer le logiciel utilisé pour voir les fichiers .ps (on met par exemple gv...),
Recurs pour déterminer le nombre de récursivité aurorisée,

1.7 Les différentes configurations avec les commandes

1.7.1 Le fichier .xcasrc

Le fichier .xcasrc du répertoire home (xcas.rc sous Windows) est un fichier qui est exécuté au lancement de Xcas et qui contient vos "préférences" (menu

`Cfg` sous-menu `Sauver préférences`) Dans ce fichier on a par exemple :

```
widget_size(20,267,56,735,557,0,1,0,7,"mozilla",0,"gv");
cas_setup(0,0,0,1,0,1e-10,12,1,0,0,0),maple_mode(0);
xyztrange(-1.0,5.0,-3.5,3.5,-10.0,10.0,-10.0,10.0,-10.0,10.0,
-5.72671232877,5.42671232877,1,0.0,1.0,1);
```

1.7.2 La configuration générale et la fonction : `widget_size`

`widget_size` a entre 1 et 12 arguments.
Les arguments dans l'ordre sont :

- argument numéro 0 : la taille des caractères, par exemple 20,
- arguments numéro 1 et 2 : l'abscisse et l'ordonnée du point dans la fenêtre `xterm`, où doit se produire l'affichage du coin supérieur gauche de `Xcas`, par exemple (58,49) (le repère a pour origine le point en haut et à gauche de la fenêtre `xterm` et l'axe des `Y` est dirigé vers le bas),
- arguments numéro 3 et 4 la largeur et la hauteur de `Xcas`, par exemple (697,563) (le repère est toujours celui de `xterm`),
- argument numéro 5 : 1 (resp 0) pour avoir (resp ne pas avoir) la présence du clavier,

- argument numéro 0 : 1 (resp 0) pour avoir (resp ne pas avoir) pour lancer le navigateur automatiquement afin d'afficher l'aide sur la commande sélectionnée dans un menu ou dans l'index,
- argument numéro 7 : 1 (resp 0) pour avoir (resp ne pas avoir) la présence du bandeau,

- argument numéro 8 : pas utilisé pour l'instant,
- argument numéro 9 : une chaîne de caractères indiquant le nom de votre navigateur internet (si vous travaillez avec Linux ou avec MacOS), celui qui s'ouvrira pour lire l'aide `html` ("" ou "builtin" designe le petit navigateur integré)
- argument numéro 10 : pour avoir le niveau de l'utilisateur 0=université, 1=terminale, 2=première, 3=seconde, 8=tortue,
- argument numéro 11 : nom de la commande utilisée permettant la previsualisation postscript : par exemple '"gv"

1.7.3 La configuration du cas avec la fonction : `cas_setup`

`cas_setup` a 9 arguments.
Les arguments dans l'ordre sont :
`approx, var_complexe ,complexe, radian, format_affichage, epsilon, chiffres, taches, increasing power`
et valent 1 ou 0 (sauf `taches`) pour dire si on veut ou on ne veut pas travailler avec cette option.
On tape :

```
cas_setup(1,0,0,1,0,1e-10,12,2,0)
```

`approx` vaut `1` : on travaille en mode approximatif (on met 0 pour numérique),
`var_complexe` vaut `0` : on travaille avec des variables réelles (on met 1 pour travailler avec des variables complexes),
`complex` vaut `0` (on travaille en mode réel),
`radian` vaut `1` (on travaille en radian). On met `radian` à `0` pour travailler en degré,
`format_affichage` vaut `0` : l'affichage sera selon le format standard (on met 1 pour le format scientifique, 2 pour le format ingenieur et 3 pour le format flottant hexadecimal (normalisé avec un 1er digit non nul ou dénormalisé)),
`epsilon` vaut `1e-10`,
`Chiffres` vaut `12`,
`taches` vaut `2`,
`puissance croissante` vaut `0` (on travaille en puissance décroissante).

1.7.4 Nombres de chiffres significatifs : `Digits DIGITS`

`Digits` ou `DIGITS` est le nom d'une variable qui contient le nombre de chiffres de l'affichage. On tape :

```
Digits
```

Ou on tape :

```
DIGITS
```

On obtient :

```
12
```

si dans la configuration du CAS il y a 12 dans la case `Chiffres`.

On peut changer la valeur de la case `Chiffres` de la configuration du CAS en affectant une valeur à `Digits` ou `DIGITS`. `Digits` ou `DIGITS` permet donc de spécifier le nombre de chiffres significatifs de l'affichage, pour cela on doit taper `Digits` puis `:=` et un nombre entier entre 1 et 1000.
On tape :

```
Digits:=3
```

Ou on tape :

```
DIGITS:=3
```

pour avoir 3 chiffres à l'affichage : Puis, on tape :

```
1.0/3
```

On obtient :

```
0.333
```

On tape :

```
4.0/3
```

On obtient :

`1.33`

IL y a deux cas à distinguer :

– La valeur de `Digits` ou `DIGITS` est inférieure à 14, `Xcas` travaille alors avec des flottants machines c'est à dire que les décimaux ont environ 14 chiffres significatifs car ils sont representés en binaire sur 52 bits et donc les calculs seront les mêmes quelque soit `n` inférieur à 14, seul l'affichage sera différent. En effet si `Digits:=n` ou `DIGITS:=n` avec `n` inférieur à 14 l'affichage sera fait avec `n` chiffres significatifs.
  On tape :
  `Digits:=5`
  Ou on tape :
  `DIGITS:=5`
  puis, on tape :
  `a:=4.0/3`
  On obtient :
  `1.3333`
  Mais, si on tape :
  `b:=a-1.333333333`
  On obtient :
  `3.3333e-10`
  ce qui prouve que `Xcas` travaille avec `a=1.33333333333333`.
– La valeur `n` de `Digits` ou `DIGITS` est supérieure à 14, `Xcas` travaille avec des flottants en multi-précision c'est à dire que les décimaux auront environ `n` chiffres significatifs. Donc si on tape `Digits:=n` ou `DIGITS:=n` avec `n` supérieur à 14 l'affichage et les calculs se feront avec `n` chiffres significatifs.
  On tape :
  `Digits:=20`
  Ou on tape :
  `DIGITS:=20`
  puis, on tape :
  `c:=4.0/3`
  On obtient 21 chiffres significatifs :
  `1.33333333333333333333`
  0n tape :
  `c-1.33333333333333333333`
  On obtient :
  `0.00000000000000000000`
  Dans ce cas les calculs correspondent à l'affichage.
  Exemples On tape :
  `Digits:=5`
  Ou on tape :
  `DIGITS:=5`
  puis
  `evalf((pi)^4)`
  On obtient :

`97.409`

puis

`evalf((pi)^4-97.409)`

On obtient :

`9.1034e-05`

puis

`evalf((pi)^4-97.40909103)`

On obtient :

`4.0018e-09`

On tape :

`Digits:=20`

Ou on tape :

`DIGITS:=20`

puis On tape

`evalf((pi)^4)`

On obtient :

`0.97409091034002437236e2`

puis

`evalf((pi)^4-97.40909103)`

On obtient :

`4.001776687801e-09`

Remarque
Si pour un calcul on veut par exemple avoir un affichage avec seulement n chiffres significatifs il faut utiliser `evalf` avec 2 arguments : cela ne change pas la valeur de `Digits`.
On tape :
`a:=862/7`
`evalf(a,5)`
On obtient :
`123.14`
`evalf(a)`
On obtient si dans la configuration utilisée `Digits` vaut 12 :
`123.142857143`

1.7.5 Choix du mode de langage `Xcas` ou Maple ou MuPad ou TI89 : `maple_mode`

`maple_mode` permet de spécifier le langage avec lequel on veut travailler.
Si on veut travailler en mode `Xcas`, on tape :

`maple_mode(0)`

Si on veut travailler en mode `Maple`, on tape :

`maple_mode(1)`

Si on veut travailler en mode `MuPad`, on tape :

`maple_mode(2)`

Si on veut travailler en mode `TI89`, on tape :

`maple_mode(3)`

1.7.6 Choix de l'unité d'angle : angle_radian

angle_radian permet de spécifier l'unité choisie pour les angles.
Si on veut travailler en radian, on tape :

```
angle_radian:=1
```

Si on veut travailler en degré, on tape :

```
angle_radian:=0
```

On peut aussi cocher ou décocher radian dans l'écran de la Configuration du CAS du menu Cfg.

1.7.7 Choix du mode approximatif ou exact : approx_mode

approx_mode permet de spécifier si on veut travailler avec des valeurs approchées ou avec des valeurs exactes.
Si on veut travailler en approximatif, on tape :

```
approx_mode:=1
```

Si on veut travailler en exact, on tape :

```
approx_mode:=0
```

On peut aussi cocher ou décocher approx dans l'écran de la Configuration du CAS du menu Cfg.

1.7.8 Choix du mode réel ou complexe : complex_mode

complex_mode permet de spécifier si on veut travailler avec des nombres complexes ou avec des nombres réels.
Si on veut travailler en complexe, on tape :

```
complex_mode:=1
```

Si on veut travailler en réel, on tape :

```
complex_mode:=0
```

On peut aussi cocher ou décocher Complex dans l'écran de la Configuration du CAS du menu Cfg.

1.7.9 Variables réelles ou complexes : complex_variables

complex_variables permet de spécifier si on veut travailler avec des variables formelles réelles (dans ce cas par exemple re(A)=A) ou complexes.
Si on veut travailler avec des variables formelles complexes, on tape :

```
complex_variables:=1
```

Si on veut travailler avec des variables formelles réelles, on tape :

```
complex_variables:=0
```

On peut aussi cocher ou décocher `Variables_ complexes` dans l'écran de la `Configuration du CAS` du menu `Cfg`.
Attention !!!
Dans `csolve` l'inconnue est toujours considérée comme une variable formelle complexe et dans `solve` l'inconnue est considérée comme une variable formelle complexe seulement si l'equation à résoudre contient `i` et si on est en mode complexe.
On tape en mode complexe (`Complexe` est coché et `Variables_complexes` est décoché) :

```
solve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)]
```

cela veut dire que la solution `r` est un nombre complexe qui vaut `` ` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)`` où `'x'` est arbitraire.
On tape en mode complexe (`Complexe` est coché et `Variables_complexes` est décoché) :

```
solve(expand(re(r*exp(-(i)*t))-1),r)
```

On obtient :

```
[1/(cos(t))]
```

car `(expand(re(r*exp(-(i)*t))-1)` renvoie une expression qui ne contient pas de `i` donc `solve` considère `r` comme une variable réelle.
Règle Pour ne pas se tromper, lorqu'on veut des solutions complexes, il faut utiliser `csolve`.
On tape en mode complexe ou réel :

```
csolve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)]
```

cela veut dire que la solution `r` est un nombre complexe qui vaut `` ` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)`` où `'x'` est arbitraire.

1.8 L'aide

On peut avoir de l'aide sur les différentes fonctions de calcul formel de plusieurs façons. On peut cocher la case `Aide HTML automatique` de la configuration générale pour avoir l'ouverture de l'aide detailléeà chaque appel d'une fonction se trouvant dans les menus ou ne pas cocher cette case pour avoir de l'aide seulement lorqu'on le désire, toutefois une aide succincte apparait dans la ligne des messages à chaque appel d'une fonction se trouvant dans les menus. On peut avoir accès à l'aide générale ou à l'aide par fonction tout le temps (voir ci-dessous).

1.8.1 Aide générale

Voir aussi : 1.4.4 pour le menu de `Aide`.
On ouvre l'aide en utilisant le menu `Aide`.
Il sufit aussi, de taper le début du nom d'une commande, puis d'appuyer sur la touche de tabulation (⭾) pour avoir l'aide sur les commandes commencant par ce début de nom ou,
de mettre en surbrillance dans un des menus `Cmds...Geo`, le nom d'une fonction pour avoir :
- cette fonction dans la ligne où se trouve le curseur,
- une explication sur ce que fait cette fonction dans l'écran des messages,
- une aide plus détaillée mais seulement en Français dans votre navigateur.

1.8.2 Aide sur une fonction : `findhelp` ou `?`

Chaque fois que l'on appelle une fonction à partir d'un menu une aide succincte sur cette fonction s'affiche dans l'écran des messages (cliquez sur `msg` du bandeau).
Pour avoir de l'aide dans l'historique dans la langue de son choix, on utilise `findhelp` ou `?` avec deux paramètres : le premier paramètre est le nom de la fonction et le deuxième est `1` pour avoir l'aide en français (ou `2` pour avoir l'aide en anglais ou `3` pour avoir l'aide en espagnol) sur cette fonction.
On tape par exemple :

```
findhelp(factor)
```

ou

```
?factor
```

On obtient alors l'aide sur la fonction `factor` en réponse. On peut aussi choisir la langue On tape par exemple :

```
findhelp(factor,2)
```

ou

```
?factor,2
```

On obtient alors l'aide sur la fonction `factor` en anglais en réponse.

1.9 Sauver et imprimer

Il est préférable de mettre les suffixe suivants :
`.cxx` (ou `.map` ou `.mu` ou `.ti` selon le mode) pour un script et par exemple,
`.xws` pour la session de travail,
`.cxx` pour une fonction,
`.tab` pour le tableur,
`.tex` pour le graphique en latex,
`.eps` pour le graphique et pouvoir ensuite inclure le fichier dans un texte en latex ou en postscipt,
`.png` pour le graphique et pouvoir ensuite inclure le fichier dans un texte en html.

1.9.1 Pour sauver une session

Il suffit d'utiliser le bouton `sauver` qui se trouve dans la barre de boutons pour sauver votre session ou encore le sous-menu `sauver` du menu `Fich`.
Lorsque vous sauvez votre session, le bouton `sauver` qui etait sur fond rouge est alors sur fond vert. La première fois on vous demande le nom du fichier de sauvegarde (en `.xws`), ce nom s'inscrit à la place de `<no filename>` et c'est dans ce fichier que se feront les sauvegardes ultérieures. Si vous voulez qu'une sauvegarde se fasse sous un autre nom Il suffit d'utiliser le sous-menu `Sauver comme` du menu `Fich`.
Vous avez dans un fichier `.xws` la suite des questions et des réponses, ainsi votre session de travail est transformée en un script qui pourra alors être réexécuté avec `Charger` du menu `Fich`.

1.9.2 Pour sauver un tableur

À l'ouverture d'un tableur (avec `Alt+t`), on vous demande un nom de variable : si vous donnez comme nom `mat`, il s'inscrit `mat.tab` à coté du bouton `Save` du tableur. Lorsque vous sauvez votre tableur avec ce bouton `Save` du tableur, vous sauvez à la fois les formules et les valeurs du tableur : votre tableur est transformé en un fichier qui pourra être chargé grâce à `Insérer` du menu `Fich` du tableur.

1.9.3 Pour sauver un programme

Vous sauvez les programme ou les scripts (suite d'instructions séparées par des points virgules) écrit dans un éditeur de programmes, avec le bouton `save` de cet éditeur de programmes, ou avec le sous-menu `sauver` du menu `Fich` de cet éditeur de programmes.
Le fichier contenant cette sauvegarde pourra être remis dans un éditeur de programmes grâce au menu `Fich` sous-menu :
`Charger` pour charger un fichier contenant des scripts ou des programmes dans l'éditeur de programmes.
ou
`Inserer` pour inserer un fichier contenant des scripts ou des programmes dans l'éditeur de programmes.
Appuyer sur `Save` a pour effet, la première fois, de vous demander le nom du fichier de sauvegarde. Ce nom sera le même, pour toutes les sauvegardes ultérieures. Si vous voulez, au cours de votre travail, sauver sous un autre nom, il faut utiliser le menu `Fich` de l'éditeur de programme sous-menu `Sauver comme`.

1.9.4 Pour imprimer

Il faut utiliser le menu `Fich` sous-menu `Imprimer session` puis :
`Imprimer` pour imprimer l'historique : vous n'avez pas besoin pour cela d'avoir Latex.... On vous demandera simplement le nom de l'imprimante ou bien,
`Fichier postscript` pour traduire votre session en un fichier postscript (`.ps`)

qui sera prêt à être imprimer (sous Unix ou Linux vous pourrez le voir avec la commande `gv`) ou bien,
`Pré-visualisation avec Latex` pour voir votre fichier avant impression, votre session est sauvée en un fichier Latex (`.tex`) qui sera traduit après compilation en un fichier (`.dvi`) (si vous n'avez pas sauvé auparavant ces fichiers auront pour nom `session.tex` et `session.dvi`) ou bien directement,
`Imprimer avec Latex` vous aurez les fichiers `.tex`) et `.dvi`.

1.10 Traduction Latex

1.10.1 Traduction Latex d'une entrée : `latex TeX`

`latex` (ou `TeX`) a comme argument une expression.
`latex` (ou `TeX`) renvoie l'écriture en latex de l'expression évaluée.
On tape :

```
latex(1/2)
```

On obtient :

```
"\frac{1}{2}"
```

On tape :

```
latex(1+1/2)
```

On obtient :

```
"\frac{3}{2}"
```

1.10.2 Imprimer la session ou/et la convertir en un fichier Latex

Pour sauver toute la session il faut cliquer sur `Save` de la barre des boutons de la session (on vous demande le nom par exemple `session.xws` et toute la session est sauvée dans le fichier `session.xws`). Si vous voulez l'imprimer en un fichier postscript, il faut choisir dans le menu `Fich` de la session, le sous-menu `Imprimer` puis `Pre-visualisation` et ensuite `Vers Imprimante` : cela crée un fichier postscript `session.ps`. Si vous voulez convertir votre session en un fichier Latex, il faut choisir dans le menu `Fich`, le sous-menu `Imprimer` puis, `Pre-visualisation (Latex)`, et ensuite `Vers Imprimante (Latex)` cela crée les fichiers `session.tex`, `session.dvi`, `session.ps` et `session.png`.

1.10.3 Traduction Latex d'un écran de géométrie

Voir aussi : 3.5 et 1.10.5 On veut traduire en Latex toutes les sorties graphiques réalisées depuis le dernier `erase()` c'est à dire toutes celles faites à partir des lignes de commandes et celles faites dans les écrans de géométrie.
On utilise pour cela la fonction `graph2tex`.
On tape par exemple dans une ligne de commandes :

```
graph2tex("truc.tex")
```

On obtient alors les graphiques sauvés en un fichier Latex `truc.tex` qui pourra être compilé et visualisé seul, ou encore être inséré dans un fichier Latex.
Ou encore, si on veut la traduction Latex d'un seul écran de géométrie, on appuie sur le bouton `Save` de cet écran de géométrie, on donne un nom de fichier par exemple `truc.cas`. Puis, on choisit dans le menu `Fich` de cet écran de géométrie, le sous-menu `Imprimer` puis, `Pre-visualisation (Latex)`, cela crée les fichiers `truc.tex`, `truc.dvi`, `truc.ps` et `truc.png` ou éventuellement si on veut imprimer, on choisit `Vers Imprimante (Latex)`.
On obtient alors l'écran de géométrie sauvé en un fichier Latex `truc.tex` qui pourra être compilé et visualisé seul, ou encore être inséré dans un fichier Latex, à condition d'enlever l'en-tête `\documentclass{article}...\begin{document}`, d'enlever à la fin `\end{document}` et de rajouter `\usepackage{pstricks}` dans l'en-tête du fichier dans lequel on l'insère.
Ou encore, on veut traduire en Latex une seule sortie graphique ou un seul dessin ou un seul graphique fait dans un des écrans de géométrie et on voudrait l'intégrer dans un texte Latex ou l'imprimer seul. Mais, on a déjà fait d'autres graphiques, soit à l'aide de commandes qui renvoient des sorties graphiques, soit dans différents écrans de géométrie, alors, pour avoir la traduction Latex du dessin que vous voulez, il faut utiliser la commande `erase()` cela n'effacera pas vos graphiques, mais cela aura pour effet de ne traduire en Latex que les tracés futurs. Donc on tape `erase()`, puis on revalide ce que l'on veut traduire en Latex, puis on tape par exemple :

```
graph2tex("truc1.tex")
```

Pour traduire en Latex et imprimer un seul graphique on peut aussi se servir du menu `Exporter/Imprimer` du bouton `M` de cet écran graphique :
`M▶ Exporter/Imprimer▶Imprimer(en Latex)`,
Ou encore, on peut sélectionner le niveau de l'écran graphique que l'on veut imprimer et utiliser le menu de la session :
`Fich▶Imprimer▶Imprime selection (latex)`.
Dans ces deux cas l'écran est sauvé selon 4 formats, sous les noms `session0.tex`, `session0.dvi`, `session0.ps` et `session0.png` (ou encore sous les noms `session<numero>.tex/dvi/ps/png`) (sauf si vous avez donné un autre nom ce qui est conseillé !).

1.10.4 Traduction Latex de l'écran `DispG`

On tape `DispG` pour voir cet écran, et on se sert du bouton `M▶ Exporter/Imprimer▶Imprime` `Latex)` de cet écran.
L'écran `DispG` s'efface avec la commande `ClrGraph()` ou `erase`.

1.10.5 Traduction Latex de l'écran `3-d` : `graph3d2tex`

Voir aussi : 3.5 et 1.10.3 On fait un graphique en `3-d` dans l'écran et on voudrait l'imprimer. On se sert de `M▶ Exporter/Imprimer▶Imprimer(en Latex)` ou de la fonction `graph3d2tex`.
On tape par exemple :

```
graph3d2tex("truc.tex")
```

On obtient alors l'écran `3-d` sauvé en un fichier Latex `truc.tex` qui pourra être inséré dans un fichier Latex.
On peut aussi sélectionner le niveau (en cliquant sur son numéro puis, utiliser le menu `Fich` sous menu `Imprimer`▶`Imprime sélection (latex)`).
Dans ce cas l'écran `3-d` est sauvé selon 4 formats, sous les noms `session0.tex`, `session0.dvi`, `session0.ps` et `session0.png` (ou encore sous les noms `session<numero>.tex/dvi/ps/png`) (sauf si vous avez donné un autre nom ce qui est conseillé !).

1.11 Traduction Mathml

1.11.1 Traduction Mathml d'une expression : `mathml`

`mathml` a comme argument une expression.
`mathml` renvoie l'écriture en mathml de l'expression évaluée.
On tape :

```
mathml(1/2)
```

On obtient :

```
"<?xml version=""1.0"" encoding=""iso-8859-1""?>
<!DOCTYPE html PUBLIC ""-//W3C//DTD XHTML 1.1 plus
MathML 2.0//EN""
""http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd""
[ <!ENTITY mathml
""http://www.w3.org/1998/Math/MathML""> ]> <html
xmlns=""http://www.w3.org/1999/xhtml""> <body> <math
mode=""display""
xmlns=""http://www.w3.org/1998/Math/MathML"">
<mfrac><mrow><mn>1</mn></mrow><mrow><mn>2</mn></mrow></mfrac>
</math><br/> </body> </html> "
```

1.11.2 Traduction Mathml du tableur

On utilise le menu `Fich->Exporter->Mathml`.

1.12 Traduction de fichiers Maple en fichier `Xcas` ou Mupad

1.12.1 Fichier Maple traduit en fichier `Xcas` : `maple2xcas`

`maple2xcas` a comme argument un nom de fichier qui contient un programme Maple en mode texte.
On tape :

```
maple2xcas("fichier1","fichier2")
```

On obtient :

```
la traduction de ce fichier en Xcas
```

1.12.2 Fichier Maple traduit en fichier Mupad : `maple2mupad`

`maple2mupad` a comme argument un nom de fichier qui contient un programme Maple en mode texte.
On tape :

```
maple2mupad("fichier1","fichier2")
```

On obtient :

```
la traduction de ce fichier en Mupad
```

1.13 Traduction d'un fichier Mupad en un fichier `Xcas` ou Maple

1.13.1 Fichier Mupad traduit en fichier `Xcas` : `mupad2xcas`

`mupad2xcas` a comme argument un un nom de fichier qui contient un programme Mupad en mode texte.
On tape :

```
mupad2xcas("fichier1","fichier2")
```

On obtient :

```
la traduction de ce fichier en Xcas
```

1.13.2 Fichier Mupad traduit en fichier Maple : `mupad2maple`

`mupad2maple` a comme argument un un nom de fichier qui contient un programme Mupad en mode texte.
On tape :

```
mupad2maple("fichier1","fichier2")
```

On obtient :

```
la traduction de ce fichier en Maple
```

Chapitre 2

Saisie

2.1 Pour écrire un commentaire : `Alt+c`

On peut à tout moment faire apparaitre une ligne pour écrire un commentaire avec `Alt+c`. Le commentaire s'écrit sans utiliser de guillemets et apparait en vert.
Le commentaire ne génère pas de réponse.
Le commentaire sert à commenter votre session.
Depuis un commentaire on peut ouvrir le navigateur à une adresse donnée On tape dans une ligne de commentaire :

```
Exercice 1
```

On obtient :

```
aucune réponse
```

Dans une ligne de commentaire, on peut ouvrir le navigateur à une adresse donnée :
On tape dans une ligne de commentaire :

```
Pour plus d'info cf :
@www-fourier.ujf-grenoble.fr/ parisse/giac/doc/fr/casrouge/index.html
```

On obtient :

```
l'ouverture dans le navigateur de l'index du document
sur l'algorithmique
```

Attention
Pour faire un commentaire dans un programme il faut utiliser la commande `comment` qui a comme argument une chaîne de caractères ou bien utiliser `//` qui doit etre suivi du commentaire et d'un retour à la ligne. Quand il y a un commentaire dans un programme, tout se qui se trouve entre `//` et le retour à la ligne n'est pas pris en compte par le programme.
On tape :

```
bs():={comment("bonjour"); return "Salut";}
```

On tape :

```
bs():={//"bonjour"
```

```
return "Salut";}
```

On obtient :

```
un programme ayant comme commentaire "bonjour"
```

2.2 L'éditeur d'expressions

Dans l'éditeur d'expressions, l'affichage ne se fait pas linéairement mais se fait en dimension `2d`.
Quand on a mis une expression dans l'éditeur d'expressions, on peut facilement sélectionner des sous-expressions et appeler les fonctions des menus sur ces sous-expressions puis appuyer sur `enter` pour avoir la réponse en dessous de l'éditeur d'expressions ou encore évaluer la sélection dans l'éditeur d'expressions avec le bouton `eval`.
Dans l'éditeur d'expressions, on peut utiliser les raccourcis suivants sur la sélection de sous-expressions :
`Ctrl+s` pour la commande `simplify`
`Ctrl+r` pour la commande `integrate`

2.2.1 Comment éditer une équation

Supposons que l'on veuille entrer l'expression $\frac{x+2}{x^2-4}$, on peut le faire de plusieurs façons :

- dans une ligne de commande à condition de mettre des parenthèses et on tape : `(x+2)/(x^2-4)`
- on se sert d'un éditeur d'expressions, éditeur obtenu avec `Alt+e`.
  On clique dans l'écran de l'éditeur d'expressions et on tape `x+2`, puis on sélectionne `x+2` (avec la souris ou avec la flèche vers le haut du bloc des flèches de direction) puis on tape sur `/` et `x`, on sélectionne `x` (avec la souris ou avec la flèche vers le haut) puis on tape `^2` puis on sélectionne $\mathtt{x}^2$ et on tape `-4` et on valide avec `enter` et on obtient $\frac{x+2}{x^2-4}$ comme réponse.

Supposons que l'on veuille simplifier l'expression $\frac{x+2}{x^2-4}$ on peut le faire de plusieurs façons :

- dans une ligne de commande à condition de mettre des parenthèses et on tape : `normal((x+2)/(x^2-4+1))`,
- on se sert d'un éditeur d'expressions, éditeur obtenu avec `Alt+e`.
  On tape la fraction $\frac{x+2}{x^2-4}$ (voir ci-dessus) puis, on sélectionne toute la fraction (avec la flèche vers le haut) et on sélectionne la fonction `normal` du menu `Math`▶`Réecriture` ce qui a pour effet d'avoir :
  `normal(`$\frac{x+2}{x^2-4}$`)` dans l'éditeur d'expressions, puis soit on valide avec `enter` et on obtient $\frac{1}{x-2}$ comme réponse, soit on appuie sur le bouton `eval` du clavier (si le clavier est présent), soit on sélectionne `Evaluer selection` dans le petit `M` (à l'intersection des deux barres de scroll), pour avoir la réponse dans l'éditeur d'expressions.

2.2.2 Comment sélectionner

On peut sélectionner directement avec la souris ou avec les flèches du bloc des flèches de direction.
On peut aussi considérer une expression comme un arbre et parcourir cet arbre à l'aide des flèches de direction.
Quand une expression est sélectionnée on peut :

– monter dans l'arbre en selectionnant le père avec la flèche vers le haut,
– descendre dans l'arbre en selectionnant le fils gauche avec la flèche vers le bas,
– selectionner le frère droit ou gauche avec la flèche vers la droite ou vers la gauche,
– échanger l'expression avec son frère avec `Ctrl` et la flèche vers la droite ou vers la gauche,

– supprimer un opérateur ou le nom d'une commande en sélectionnant l'expression et en tapant : `backspace`. Il faut noter qu'un deuxième `backspace` effacera les arguments

Exemple On clique dans l'écran de l'éditeur d'expressions et on tape `x+1` puis flèche vers le haut pour sélectionner `x+1`, puis `*` et on tape `x+2` puis flèche vers le haut pour sélectionner `x+2` puis `*` et on tape `x-1` puis flèche vers le haut pour sélectionner `x-1`.
On a alors dans l'écran de l'éditeur d'expressions l'expression `(x+1)*((x+2)*(x-1))`.
Puis on sélectionne le tout et on appuie sur `eval` (menu `Expression`►`Autres` ou bouton `eval` du clavier si on a choisi `Montrer-> Clavier` dans le menu `Cfg`) pour avoir les 3 facteurs sur le même plan (l'arbre n'est plus binaire).
Donc maintenant on obtient :
`(x+1)*(x+2)*(x-1).`
On peut ensuite soit

– sélectionner `(x+1)*(x+2)` avec la souris puis on sélectionne la commande `normal` du menu `Calc` sous-menu `Réecriture`.
  On obtient alors `(x^2+3· x+2)· (x+2).`
– sélectionner `x+1`, puis `Ctrl` et la flèche vers la droite, pour échanger `x+1` et `x+2`.
  Donc maintenant on obtient :
  `(x+2)*(x+1)*(x-1).`
  On peut ensuite sélectionner `(x+1)*(x-1)` avec la souris puis on sélectionne la commande `normal` du menu `Calc` sous-menu `Réecriture`.
  On obtient alors `(x+2)*(x^2-1).`

2.2.3 Comment éditer une chaîne de caractères

Lorsque le curseur est actif dans un éditeur d'expressions, il suffit d'appuyer sur la touche `"` pour transformer une expression en une chaîne de caractères et vice versa.

2.2.4 Utilité de l'éditeur d'expressions

Avec la souris on peut mettre en surbrillance toute l'expression ou seulement une partie de cette expression : c'est alors sur cette partie que l'on peut agir : par exemple la modifier avec une commande de réecriture (par exemple avec `normal`) ou la factoriser (par exemple avec `factor`),...
On tape dans un éditeur d'éxpressions :

`1+4`

On sélectionne `4` et on appelle `ifactor` du menu `Cmds`▶`Entier`, puis `enter`.
On obtient :

$$1+2^2$$

2.3 Les éditeurs de matrices et les tableurs

Il faut tout d'abord ouvrir un tableur avec `Alt+t`.
À chaque tableur est attaché un écran de géométrie, une barre de menu (`Fich Edit Statistiques`), des boutons (`reeval`,`val`,`Save`), deux cases (l'une donne le nom du fichier de sauvegarde et l'autre le nom de la cellule sélectionnée) et deux lignes (l'une contient une case de sélection et une ligne dite ligne de commandes qui sert soit à remplir la cellule sélectionnée, soit à afficher ce que l'on a mis dans la cellule sélectionnée, et l'autre est la ligne d'état qui rappelle la configuration du tableur et sert de bouton pour ouvrir un écran de configuration du tableur).

2.3.1 Les sauvegardes d'un tableur

Lorsque vous ouvrez un tableur, on vous demande son nom, par exemple `sim`, ainsi toutes les sauvegardes ultérieures du tableur se feront dans le fichier `sim.tab` et la matrice du tableur sera stockée dans la variable `sim`.
Vous avez la possibilté de changer ce nom au cours de votre travail et ce nom peut alors ne pas être le même pour le fichier et la variable : vous pouvez avoir `toto.tab` comme nom de fichier (nom qui est rappelé à coté du bouton `Save`) et `A` comme nom de variable (nom qui est rappelé dans la ligne d'état du tableur).
Les différentes sauvegardes se font avec le bouton `Save` ou à l'aide du menu `Fich`.

2.3.2 Les menus d'un tableur

Un tableur a trois menus qui sont :
- `Fich` Ce menu permet de sauver le tableur dans le fichier dont le nom figure à coté du bouton `save` , ou de le sauver dans un fichier sous un autre nom, ou de sauver une sous-matrice du tableur dans une variable, ou d'insérer un fichier contenant un tableur, ou de changer le nom de la variable contenant le tableur et aussi d'imprimer le tableur,
- `Edit` Ce menu permet de comfigurer le tableur (voir ci-après), de remplir, à partir de la cellule sélectionnée, toutes les cellules situées vers le bas ou vers la droite, avec la formule de la cellule sélectionnée. `Edit` permet aussi

d'ajouter ou d'effacer des lignes ou des colonnes, de trier les lignes ou les colonnes et de changer la taille des lignes et des colonnes.
– `Statistiques` Ce menu contient les fonctions de statistiques `1-d` et `2-d`.

2.3.3 La configuration d'un tableur

La configuration du tableur se fait avec le menu `Edit`▶`Configuration` de la barre de menus lié au tableur ou en ouvrant un écran de configuration du tableur en cliquant sur la ligne d'état du tableur.
On a :

– `Format` Pour accéder à un tableur il faut sélectionner `Tableur` .
  Pour accéder à un éditeur de matrices, il faut sélectionner `Matrice`.
  Pour vous faciliter l'entrée d'une matrice vous avez la possibilité de choisir son type, par exemple symétrique avec `Matrice symétrique`, (resp quelconque, hermitienne..)
– `Deplacer ->` pour deplacer le curseur automatiquement vers la droite après le remplissage d'une cellule,
– `Deplacer vers le bas` pour deplacer le curseur automatiquement vers le bas après le remplissage d'une cellule,
– `Changer le nombre de lignes` permet de changer le nombre de lignes,
– `Changer le nombre de colonnes` permet de changer le nombre de colonnes,
– `Recalculer automatiquement` pour avoir un recalcul automatique du tableur un après chaque modification,
– `Ne pas recalculer automatiquement` pour ne pas avoir un recalcul automatique du tableur un après chaque modification, mais seulement quand on appuie sur le bouton `reeval`,
– `Distribuer une matrice sur plusieurs cellules` permet de remplir une matrice ou un tableur avec une sous-matrice. En effet, pour remplir une matrice ou un tableur, on peut le faire case par case (en entrant un nombre dans chaque case), on peut aussi entrer une sous matrice (en entrant par exemple une expression qui s'évalue en une matrice) à condition d'avoir choisi de distribuer la matrice. Par exemple, si dans la première case on tape `idn(3)`, si on a cliqué sur `Distribuer une matrice sur plusieurs cellules`, cela a pour effet de remplir 3 lignes et 3 colonnes avec la matrice identité d'ordre 3,
– `Conserver une matrice dans une seule cellule` permet de mettre une matrice dans une case du tableur. Par exemple, si dans la première case on tape `idn(3)`,si on a cliqué sur `Conserver une matrice dans une seule cellule`, cela a pour effet de remplir la première case par la matrice identité d'ordre 3.
– `Portrait` pour avoir l'écran de représentation graphique du tableur à droite du tableur,
– `Paysage` pour avoir l'écran de représentation graphique du tableur en dessous du tableur,

– `Cacher graph` pour ne pas avoir d'écran de représentation graphique associé au tableur.

2.3.4 Les boutons d'un tableur

On a trois boutons :

– `reeval` permet d'évaluer le tableur ce bouton est utile quand on n'est pas en mode automatique mais en mode manuel,
– `val` permet d'avoir la valeur et non la formule dans la ligne d'entrée lorsqu'on clique sur une cellule,
– `Save` pour sauver le tableur sous le nom donné au début (identique au sous-menu `sauver` du menu `Fich`).

2.4 Les commandes d'effacement

2.4.1 Effacer dans le tableur

Pour effacer, on peut se servir du menu `Edit` sous-menu `Ajouter/Effacer` pour pouvoir supprimer la ligne ou la colonne courante ou, pour effacer les lignes ou les colonnes sélectionnées.

2.4.2 Effacer l'écran `DispG` de géométrie : `ClrGraph ClrDraw`

Dans un programme, toutes les sorties graphiques d'un programme seront effectuées dans l'écran `DispG` visible avec la commande `DispG;`. Pour effacer cet écran `DispG`, on utilise la commande `ClrGraph` ou `ClrDraw`.
On tape :

```
ClrGraph()
```

On obtient :

```
L'effacement de l'écran DispG de géométrie
```

2.4.3 Effacer les écrans de géométrie : `erase`

Voir aussi : 1.10.3 et 3.5 `erase` n'a pas d'argument.
La commande `erase()` efface l'écran `DispG`, mais n'efface ni les sorties graphiques, ni les écrans de géométrie mais influe sur la commande `graph2tex`. En effet, `erase` permet d'effacer l'historique interne des graphiques et ainsi de pouvoir traduire en Latex à l'aide de la commande `graph2tex` seulement les sorties graphiques faites postérieurement à la commande `erase`. On tape :

```
erase()
```

On obtient :

```
L'effacement de l'historique interne des graphiques
```

2.4.4 Effacer une ligne de commande : touche `esc`

L'effacement de la ligne de commande où se trouve le curseur se fait en appuyant sur la touche `esc` de `Xcas` ou sur la touche `Echap` de votre ordinateur.
On tape :

`sur la touche esc`

On obtient :

`L'effacement de la ligne de commande où se trouve le curseur`

2.4.5 Effacer les noms des variables d'une seule lettre minuscule : `rm_a_z`

`rm_a_z` n'a pas d'argument.
`rm_a_z` efface tous les noms des variables d'une seule lettre minuscule.
On connait le nom des variables affectées en utilisant la commande `VARS` ou en utilisant la touche noire `var` du bandeau.
On tape :

`VARS()`

On obtient :

`[A,B,a,b,eps]`

On tape :

`rm_a_z()`

On obtient :

`[a,b]`

Si on tape maintenant :

`VARS()`

On obtient :

`[A,B,eps]`

2.4.6 Effacer toutes les variables : `rm_all_vars`

`rm_all_vars` n'a pas d'argument.
`rm_all_vars` efface tous les noms des variables.
On connait le nom des variables affectées en utilisant la commande `VARS` ou en utilisant la touche noire `var` du bandeau .
On tape :

`VARS()`

On obtient :

`[A,B,a,b,eps]`

On tape :

`rm_all_vars()`

On obtient :

`[A,B,a,b,eps]`

On tape :

`VARS()`

On obtient :

`[]`

2.5 Les variables

2.5.1 Le nom des variables et la variable `CST`

Un nom de variables est une suite de lettres ou chiffres commençant par une lettre.
Attention ! ! ! il y a des noms qui sont déjà employés pas le système.
La variable `CST` permet de définir son propre menu, menu qui s'affichera dans le bandeau lorsqu'on appuie sur le bouton `cust` du bandeau.
On suppose que l'on a écrit deux fonctions `tor` et `pgcd`, et on tape par exemple :
`CST:=[diff,["tor",tor],["pgcd",pgcd],["euro",6.55957]]` qui affichera `diff tor pgcd euro` dans le bandeau du bouton `cust`.
Pour rajouter une fonction à `CST` on tape par exemple :
`CST:=concat(CST,evalc)`.
Attention
Dans l'exemple ci-dessus, `euro` n'est pas une variable qui contient `6.55957`, mais le nom d'un bouton du bandeau. Il y a donc une différence entre :
`CST:=[evalc,["f",f],["a",6.55957]]` et
`CST:=[evalc,["f",f],["a",a]]`, dans la deuxième formulation le bouton `a` va renvoyer la valeur stockée dans la variable `a` et cette valeur changera si on modifie `a`, à condition que lorsque l'on définit `CST`, la variable `a` ne soit pas encore affectée (sinon `a` est evaluée et `a` represente la valeur stockée du début), alors que dans la première formulation le bouton `a` va renvoyer `6.55957` quelque soit la valeur stockée dans la variable `a`.
On peut aussi quoter `a` pour que `a` ne soit pas évaluée et renvoie `a` lorsqu'on appuie sur le bouton `a` du menu `cust` du bandeau, par exemple :
`CST:=[evalc,["f",'f'],["a",'a']]`.
Pour revenir au bandeau initial il faut utiliser le bouton `home` (voir aussi page 66).

2.5.2 L'affectation : `:=` `=>` `sto` `Store`

On peut utiliser les fonctions infixées `:=` comme en Pascal ou `=>` comme le "sto" des calculatrices pour réaliser une affectation ou encore,

les fonctions préfixées, `sto` ou `Store`, d'arguments la valeur à affecter et le nom d'une variable.
: = (ou => ou `sto` ou `Store`) permet d'affecter une variable.
On tape (attention à l'ordre des arguments !) :

```
a:=4
```

ou

```
4=>a
```

ou

```
sto(4,a)
```

ou

```
Store(4,a)
```

On obtient :

```
4
```

: = ou => permettent aussi de définir des fonctions.
On tape :

```
f(x):=sin(x)/x
```

ou

```
sin(x)/x=>f(x)
```

ou encore

```
f:=x->sin(x)/x
```

ou

```
x->sin(x)/x=>f
```

On peut aussi définir une fonction par morceaux :

– une fonction définie par 2 valeurs,
  par exemple, pour définir la fonction g qui vaut 1 si $x > 0$ et -1 si $x \leq 0$ on tape : On tape :
  ```
  g(x):=ifte(x>0,1,-1)
  ```
  ce qui est équivalent à :
  ```
  g(x):=if x>0 then 1; else -1; end_if
  ```
  On tape (voir plus loin la différence avec `ifte`) :
  ```
  g(x):=when(x>0,1,-1)
  ```
  ou
  ```
  g(x):=quand(x>0,1,-1)
  ```
  ou
  ```
  g(x):=IFTE(x>0,1,-1)
  ```
  ou
  ```
  g(x):=x>0?1:-1
  ```

En effet `ifte` (ou `when` ou `quand` ou `IFTE`) a trois arguments : une condition et deux expressions et `?` est infixé avec la condition à gauche et à droite on met les deux expressions séparées par `:`.
Si la condition est vraie, `ifte` (ou `when` ou `quand` ou `IFTE` ou `?`) renvoie la première expression et sinon `ifte` (ou `when` ou `quand` ou `IFTE` ou `?`) renvoie la deuxième expression.
Remarque
La condition $x! = a$ peut être remplacée par le rèel $r = x - a$:
si $r == 0$ la condition est fausse et sinon elle est vraie.
On tape pour définir la fonction qui vaut partout 0 sauf en 1 où elle vaut 1 :

`h(x):=when(x-1,0,1)`

est équivalent à :

`h(x):=when(x!=1,0,1)`

Remarque : Différence entre `ifte` et les autres `when....`
On tape : `f(x):=ifte(x>0,1,0);`
`g(x):=when(x>0,x,-x)` ou `g(x):=quand(x>0,1,0)`
puis on tape `f(x)`
on obtient :
Ifte : Unable to check test Error : Bad Argument Value
ici `x` n'a pas de valeur : avec `ifte` ou `if then else end_if` il faut que la variable `x` soit affectée pour pouvoir tester la condition (quand on définit une fonction ce qui suit le := n'est pas évalué donc la définition de `f(x)` ne pose pas de problème).
Pour la définition de g avec `when....`, la variable `x` n'a pas besoin dêtre affectée.
On tape `g(x)`
on obtient :
`((x>0)? 1 : -1)`
car `?` est la version infixée de `when`.

– une fonction définie par n valeurs,
par exemple, pour définir la fonction g qui vaut -1 si $x < -1$, 0 si $-1 \leq x \leq 1$ et 1 si $x > 1$, on tape :

`g(x):=piecewise(x<-1,-1,x<=1,0,1)`

`piecewise` utilise des paires condition/valeur ou valeur est renvoyée si sa condition est vraie ce qui implique que les conditions précédentes sont fausses. Si le nombre d'arguments est impair, la dernière valeur est la valeur par défaut (comme dans un `switch`).
`piecewise` est la généralisation de `when`.
Pour définir la fonction f qui vaut -2 si $x < -2$, $3x + 4$ si $-2 \leq x < -1$, 1 si $-1 \leq x < 0$ et $x + 1$ si $x \geq 0$, on tape :

`f(x):=piecewise(x<-2,-2,x<-1,3x+4,x<0,1,x+1)`

On peut alors faire le graphe de f en tapant :

`plotfunc(f(x))`

2.5.3 L'affectation par référence dans une variable désignant un élément d'une liste ou d'une matrice : =<

On peut utiliser l'opérateur infixé =< pour stocker par référence le deuxième argument dans une variable (désignant un élément d'une liste ou d'une matrice) donnée comme premier argument.
Voir aussi 8.4.15 et 8.4.14. On tape :

```
a:=[1,2,3,4,5]
```

Pour changer la valeur de `a[1]` il est préférable de le faire par référence c'est à dire sans faire de recopie, on tape :

```
a[1]=<5
```

Dans un programme, il est préférable d'utiliser l'opérateur infixé =< pour changer un élément d'une liste ou d'une matrice contenue dans une variable . **Exemple**
On cherche pour tout $n \in \mathbb{N}$, la liste des nombres entiers k vérifiant $0 < k < 2^n$ et dont la somme des chiffres, dans l'écriture en base 2, est égale à p ou qui s'écrivent en base 2 avec des 0 et p 1.
On sait que `convert(k,base,2)` renvoie la liste de 0 et de 1 de l'écriture en base 2 de k en commençant par le chiffre des unités. On a, par exemple :
`convert(2,base,2)=[0,1]`.
On connait la longueur de la liste résultat qui est `comb(n,p)` puisque il peut y avoir n chiffres et que parmi ces n chiffres il doit y avoir p 1. On peut donc initialiser la liste par :
`L:=makelist(0,1,comb(n,p));`.
On peut aussi vouloir initialiser la liste `L` par la liste vide et dans ce cas il faut mettre `L:=[0$0]` et ne pas mettre `L:=[]`. La différence est subtile : `[0$0]` est une liste qui est crée lors de chaque exécution du programme alors que après la compilation du programme, `L:=[]` fait pointer `L` sur la liste `[]` et cette liste sera modifiée par les différents `L[k]=<j` en `LR` et restera modifiée en fin d'exécution ce qui fait que si on effectue une autre exécution du programme`L` est initialisée par `LR` car elle pointe sur la liste `LR`.
On tape :

```
truc(p,n):={
local j,k,L;
L:=makelist(0,1,comb(n,p));
k:=0;
for (j:=2^p-1;j<=2^n-2^(n-1-p);j++){
if (sum(convert(j,base,2))==p){
L[k]=<j;
k:=k+1;
};
}
return L;
}
:;
```

Puis : `J:=truc(10,17):;J[0];J[10]` renvoie : `Done,1023,2046`
`convert(1023,base,2)` renvoie `[1,1,1,1,1,1,1,1,1,1]`
`convert(2046,base,2)` renvoie `[0,1,1,1,1,1,1,1,1,1,1]`
Comme la liste J a pour longueur `comb(17,10)=19448` si on met dans le programme `L[k]:=j;` au lieu de `L[k]=<j;`, `Xcas` effectue 19448 recopie de cette liste ce qui allonge l'execution du programme....

2.5.4 L'incrémentation d'une variable : `+=` `-=` `*=` `/=`

`+=` est un raccourci pour ajouter une expression à la valeur contenue dans une variable.
On tape :

```
a:=a+4
```

ou

```
a+=4
```

`-=` est un raccourci pour soustraire une expression à la valeur contenue dans une variable.
On tape :

```
a:=a-4
```

ou

```
a-=4
```

`*=` est un raccourci pour multipliser la valeur contenue dans une variable par une expression.
On tape :

```
a:=a*4
```

ou

```
a*=4
```

`/=` est un raccourci pour diviser la valeur contenue dans une variable par une expression.
On tape :

```
a:=a/4
```

ou

```
a/=4
```

2.5.5 Archiver et désarchiver des variables et leur contenu : `archive` `unarchive`

`archive` a comme argument une chaine de caractère (le nom du fichier d'archive) et une liste de variables.
`archive` sauve le contenu de ces variables dans le fichier ayant pour nom la chaine donnée en argument.
Le format de sauvegarde est un format interne à `Xcas`, ce qui a comme intérêt de pouvoir relire ces valeurs plus rapidement. On tape :

```
archive("toto",[a,b,f])
```

On obtient :

```
crée le fichier "toto" contenant les valeurs des
variables a,b,f
```

`unarchive` a comme argument une chaine de caractère (le nom d'un fichier d'archive créé avec la commande `archive`).
`unarchive` lit les valeurs se trouvant dans le fichier ayant pour nom la chaine donnée en argument, si ce fichier a été créé avec la commande `archive`.
On tape :

```
unarchive("toto")
```

On obtient :

```
les contenus des variables sauvées avec archive
```

2.5.6 Copier sans l'évaluer le contenu d'une variable : `CopyVar`

`CopyVar` a comme arguments le nom de deux variables.
`CopyVar` copie sans l'évaluer le contenu de la première variable dans la deuxième variable.
On tape (attention à l'ordre) :

```
a:=c
c:=5
CopyVar(a,b)
b
```

On obtient :

```
c
```

puis on tape :

```
c:=10
b
```

On obtient :

`10`

Une modification du contenu de `c` va modifier le contenu de `b` car `b` contient `c`.
On tape :

`a:=d`

`b`

On obtient :

`10`

On tape :

`purge(c)`

`b`

On obtient :

`c`

puisque `b` contient `c`.

2.5.7 Faire une hypothèse sur une variable : `assume supposons`

`assume` ou `supposons` permet de faire des hypothèses sur une variable.
`assume` ou `supposons` a comme argument un nom de variable suivi d'une égalité ou d'une inégalité représentant l'hypothèse faite ou bien un nom de variable suivi d'une virgule et de son type. On peut mettre plusieurs hypothèses à condition de les relier par `and` ou `or` selon ce que l'on veut faire ! Toutefois, il faut utiliser `additionally` comme deuxième argument de `assume` pour spécifier le type de la variable et une plage de valeurs pour cette variable.
`assume` renvoie le nom de la variable sur laquelle on a fait les hypothèses ou le type de cette variable.
Attention Si on fait une autre hypothèse avec `assume`, l'ancienne hypothèse est effacée : si vous voulez rajoutez une nouvelle hypothèse il faut utiliser la commande `additionally` ou mettre `additionally` comme deuxième argument de `assume`.
Remarques Cela permet de faire de la géométrie interactive tout en faisant du calcul formel. Par exemple, si on met en géométrie :
`assume(a=2);assume(b=3); A:=point(a+i*b)`, la figure sera construite avec les valeurs données aux variables mais les calculs seront faits avec les variables symboliques `a` et `b` car pour toutes les sorties graphiques et seulement sur celles-ci la variable est évaluée.
On tape

`assume(a=2)`

Ou on tape

`supposons(a=2)`

Ou on tape

`assume(a=2)`

Ou on tape

`supposons(a:=2)`

Ou on tape directement :

`supposons(a=[2,-5,5,0.1])`

On obtient :

`un curseur permettant de faire varier a`

Lorsque l'on fait varier `a` la commande `assume(a=2)` se transforme en `supposons(a=[2.1,-5.0,5.0,0.`
et les niveaux qui suivent sont évalués. Si il n'y a rien sur le niveau suivant on aura `undef` en réponse.
Cela signifie que `a` peut varier entre `-5` et `5` avec un pas de `0.1` et que `a` vaut `2.1`.
Si sur les deux niveaux suivants on a
`evalf(a+2)` et `evalf(a+3)`
les réponses évolueront selon la position du curseur (curseur en `2.1`, on aura `4.1` et `5.1` puis curseur en `2.2`, on aura `4.2` et `5.2`). Mais si sur les deux niveaux suivants on a `a+2` et `a+3`, les réponses seront toujours `a+2` et `a+3`.
On tape pour supposer que la variable formelle `a` est positive :

`assume(a>0)`

On obtient :

`a`

On tape :

`assume(a)`

On obtient :

`assume[DOM_FLOAT,[line[0,+(infinity)]],[0]]`

cela signifie que `a` est une variable réelle appartenant à $[0;+\infty$ et que 0 est exclu (on a le domaine, l'intervalle et les valeurs exclues).
On tape pour supposer que la variable formelle `a` est dans $[2;4[\cup]6;\infty[$:

`assume((a>=2 and a<4) or a>6)`

On obtient :

`a`

On tape :

`assume(a)`

On obtient :

`assume[DOM_FLOAT,[[2,4],[6,+(infinity)]],[4,6]]`

cela signifie que a est une variable réelle appartenant à $[2;4] \cup [6;\infty[$ et que 4 et 6 sont exclus (on a le domaine, l'intervalle et les valeurs exclues).
On tape :

`abs(1-a)`

On obtient :

`-1+a`

On tape pour dire que b est un entier :

`assume(b,integer)`

On obtient :

`DOM_INT`

On tape :

`assume(b)`

On obtient :

`[DOM_INT]`

On tape pour dire que b est un entier supérieur strictement à 5 :

`assume(b,integer);`

`assume(b>5,additionally)`

On obtient :

`DOM_INT`

puis

`b`

On tape :

`assume(b)`

On obtient :

`[DOM_INT]`

Remarque
Lorsque `assume` a comme argument une seule égalité et que la commande est tapée dans une ligne d'entrée d'un écran de géométrie, cela met un petit curseur en haut et à droite de cet écran. Le nom du paramètre est noté à droite du curseur. Ce curseur permet de changer la valeur du paramètre et cette valeur sera notée à gauche du curseur. On tape par exemple :
`assume(a=[2,-10,10,0.1])`
Cela signifie que tous les calculs seront faits avec `a` quelconque, à condition que les points aient des coordonnées exactes, mais que la figure sera tracée avec `a=2` et que l'on pourra faire varier cette figure avec le petit curseur en fonction de `a` de `-10` à `+10`, avec un pas de `0.1`. Si on met `assume(a=[2,-5,5)`, `a` varie de `-5` à `+5` avec un pas de `(5-(-5))/100`), et si on met `assume(a=2)`, `a` varie de `WX-` à `WX+` et le pas est `((WX+)-(WX-))/100`.

Attention En géométrie il faut donc travailler avec des coordonnées exactes par exemple :
`A:=point(i);assume(b:=2); B:=point(b);` puis on tape :
`longueur(A,B);`
On obtient :
`sqrt((-b)^2+1)`
Mais :
`A:=point(0.0+i);assume(b:=2); B:=point(b);` puis on tape :
`longueur(A,B);`
On obtient la valeur approchée de $\sqrt{(1+4)}$:
`2.2360679775`
Attention Un paramètre défini par `assume` n'est évalué que pour les sorties graphiques, sinon il faut utiliser `evalf`.
Exemple : On tape :
`dr(m):=ifte(m==2,droite(x=1),droite(x+(m-2)*y-1))`
puis dans un niveau de géométrie, on tape :
`supposons(a=[2.0,-5,5,0.1])`
`dr(evalf(a))`
qui renvoie `droite(x=2)` lorsque `a:=2` et `droite(y=(-5*x+5))` lorsque `a:=2.2` alors que
`dr(a)` renvoie `droite(y=(-1/(a-2)*x+1/(a-2)))` quelque soit `a` et il y aura donc une erreur pour `a=2`....

Attention à la différence entre `assume` et `element`
Si `b:=element(0..3,1,0.1)` est tapé dans une ligne d'entrée d'un écran de géométrie, cela met aussi un petit curseur en haut et à droite de cet écran avec `b=1` et on pourra faire varier `b` avec le petit curseur de `0` à `3` avec un pas de `0.1`. Mais la variable `b` n'est pas formelle !
On tape

```
a;b
```

On obtient :

```
(a,1)
```

2.5.8 Faire une hypothèse suppl'ementaire sur une variable : `additionally`

`additionally` permet de faire des hypothèses supplémentaires sur une variable. En effet, si on fait une autre hypothèse avec `assume`, l'ancienne hypothèse est effacée. Donc, si vous voulez rajoutez une nouvelle hypothèse il faut utiliser la commande `additionally` ou mettre `additionally` comme deuxième argument de `assume`.
`additionally` a les mêmes arguments que `assume` ou `supposons` : un nom de variable suivi d'uneégalité ou d'une inégalité représentant l'hypothèse faite ou bien un nom de variable suivi d'une virgule et de son type. On peut mettre plusieurs hypothèses à condition de les relier par `and` ou `or` selon ce que l'on veut faire !
On est obligé d'utiliser `additionally` pour spécifier le type de la variable et une plage de valeurs pour cette variable.
On tape pour dire que `b` est un entier supérieur strictement à 5 :

```
assume(b,integer);
additionally(b>5)
```

ou bien

```
assume(b,integer);
assume(b>5,additionally)
```

On obtient :

```
DOM_INT,b
```

puis

```
b
```

On tape :

```
assume(b)
```

On obtient :

```
[DOM_INT]
```

2.5.9 Connaitre les hypothèses faites sur une variable : `about`

`about` a comme argument un nom de variable.
`about` permet de connaitre les hypothèses faites sur cette variable.
On tape :

```
assume(a,real);additionally(a>0)
```

ou

```
assume(a,real);assume(a>0,additionally)
```

puis,

```
about(a)
```

On obtient :

```
assume[DOM_FLOAT,[0,+(infinity)],[0]]
```

`assume[ ]` signifie que l'on a une liste d'un type particulier.
Le dernier `0` veut dire que `0` est exclus de l'intervalle `[0,+(infinity)]`.
On tape :

```
assume(b,real);additionally(b>=0 and b<2)
```

ou

```
assume(b,real);assume(b>=0 and b<2,additionally)
```

puis,

```
about(b)
```

On obtient :

```
assume[DOM_FLOAT,[0,2],[2]]
```

Le dernier `2` veut dire que `2` est exclus de l'intervalle `[0,2]`.
On tape :

```
about(x)
```

On obtient :

```
x
```

ce qui veut dire que `x` est une variable formelle.

2.5.10 Effacer le contenu d'une variable : `purge DelVar`

`purge` ou `DelVar` permet d'effacer le contenu d'une variable ou d'annuler une hypothèse faite sur cette variable.
On tape :

```
purge(a)
```

si `a` n'est pas affecté on obtient en mode direct `"a not assigned"` et sinon l'ancienne valeur est renvoyée (ou les hypothèses faites sur cette variable sont renvoyées) et la variable `a` redevient formelle et sans hypothèse.
On peut aussi taper :

```
purge(a,b)
```

pour effacer le contenu des variables `a` et `b`.

2.5.11 Effacer le contenu de toutes les variables : `restart`

`restart` permet d'effacer le contenu de toutes les variables et d'annuler les hypothèses faites sur ces variables.
On tape :

```
A:=point(1+i);assume(n>0);
```

puis

```
restart
```

On obtient :

```
[A,n]
```

si les variables `[A,n]` avaient été les seules variables affectées.

2.5.12 Accès aux réponses : `ans(n)`

`ans` doit être utilisé si on travaille sans modifier les lignes déjà validées. En effet, les questions et les réponses sont numerotées en partant de 0 et ce numéro ne correspond pas aux numéros des lignes d'entrée, puisque l'on peut, par exemple, modifier la première ligne après avoir rempli 4 lignes et cette modification aura comme numéro 4.
Si $n \geq 0$, `ans(n)` permet de désigner la réponse de numéro $n+1$ et,
Si $n < 0$, `ans(n)` permet de désigner la $(-n)$-ième réponse précédente.
Ainsi :
`ans()` ou `ans(-1)` désigne la réponse précédente,
`ans(0)` désigne la première réponse (celle correspondant à la première commande demandée). **Attention** Si vous avez effacé des niveaux, les réponses de ces niveaux ne sont pas effacées et sont comptées dans les `ans(n)` .

2.5.13 Pour ne pas afficher la réponse : `nodisp :;`

Pour ne pas encombrer la feuille, on peut vouloir que la réponse ne s'affiche pas : pour cela il faut appliquer la fonction `nodisp` à la question ou bien terminé la question par `:;`.
On tape par exemple dans une ligne d'entrée :

```
A:=[1,2,3,4]
```

On obtient :

```
[1,2,3,4]
```

et `ans()` désigne `[1,2,3,4]` c'està dire `A`.
On tape :

```
nodisp(A:=[1,2,3,4])
```

Ou on tape :

```
A:=[1,2,3,4]:;
```

On obtient :

```
Done
```

ce qui veut dire que l'affectation a bien eu lieu et dans ce cas `ans()` désigne `Done`.

2.5.14 Accès aux questions : `quest(n)`

`quest` doit être utilisé si on travaille sans modifier les lignes déjà validées. En effet, les questions et les réponses sont numerotées en partant de 0 et ce numéro ne correspond pas aux numéros des lignes d'entrée, puisque l'on peut, par exemple, modifier la première ligne après avoir rempli 4 lignes et cette modification aura comme numéro 4.
Si $n \geq 0$, `quest(n)` permet de désigner la question de numéro n et, si $n < 0$ `quest(n)` permet de désigner la $(-n)$-ième question précédente.
Ainsi :
`quest()` ou `quest(-1)` désigne la question précédente,
`quest(0)` désigne la première question demandée.
Remarque
Pour ne pas avoir à retaper une commande, on peut se déplacer dans la liste des commandes tapées précédemment en mettant le curseur dans une ligne de commandes et en tapant une ou plusieurs fois `Ctrl`+↑ ou `Ctrl`+↓.

2.6 Les répertoires

2.6.1 Comment créer un répértoire sur vôtre disque dur

`Xcas` peut se déplacer dans le répertoire de vôtre disque dur. On rappelle que sous `Unix` la commande `mkdir` permet de créer de nouveaux répertoires.
On tape par exemple (sous `Unix`) : `mkdir toto`
et cela crée le sous répertoire `toto` à partir du répertoire courant.

2.6.2 Comment sauver un fichier dans un répértoire de vôtre disque dur

Vous avez crée un répertoire sur votre disque dur qui s'appelle `toto`.
Travailler dans ce répertoire depuis `Xcas` est facile : il suffit de taper dans la ligne de commande :
`cd` puis entre guillemets, le chemin pour accéder à ce répértoire.
On tape :

```
cd("toto")
```

ou en indiquant entre guillemets, un chemin depuis le répertoire racine par exemple :

```
cd("/home/texte/toto")
```

Ainsi les fichiers que vous sauverez depuis `Xcas`, dans vôtre disque dur seront sauvés dans ce répertoire.
Attention !!!
Quand on change de répértoire les variables qui sont conservées dans des fichiers d'extention `.cas` sont celles du nouveau répértoire. Donc les variables changent quand on change de répértoire.

2.6.3 Comment créer un répértoire de travail : `NewFold`

On peut aussi créer des répertoires de travail qui sont internes à `Xcas` avec la commande `NewFold`. Cela permet d'avoir des variables de même nom mais dans des répertoires différents.
Depuis `Xcas` on tape par exemple :

```
NewFold(toto)
```

On obtient :

```
la création du sous-répertoire "toto"
```

2.6.4 Comment aller dans un répértoire de travail : `SetFold`

On utilise la commande `SetFold` pour aller dans un répértoire de travail interne à `Xcas`.
On tape par exemple dans `Xcas` :

```
SetFold(toto)
```

On obtient :

```
on se trouve dans le sous-répertoire "toto"
```

On tape par exemple dans `Xcas` :

```
SetFold(home)
```

Ou

```
SetFold(main)
```

On obtient :

```
on se trouve dans le répertoire courant
```

Attention !!!
Quand on change de répértoire de travail les variables sont celles du nouveau répértoire : vous pouvez donc avoir des variables de même nom contenant des valeurs différentes dans différents répertoires. Donc les variables changent quand on change de répértoire de travail.

2.6.5 Nom du répértoire en cours : `GetFold`

`GetFold` renvoie le nom du répértoire dans lequel on se trouve. On tape par exemple dans `Xcas` :

```
SetFold(toto)
```

puis

```
GetFold()
```

On obtient :

```
toto([])
```

2.6.6 Effacer un répértoire vide : `DelFold`

`DelFold` efface le répértoire vide de nom donné dans l'argument. `Xcas` renvoie une erreur si ce répertoire n'est pas vide. On tape par exemple dans `Xcas` :

```
DelFold(toto)
```

On obtient :

```
([])
```

2.6.7 Comment connaitre les variables et les répértoires créés : `VARS`

`VARS` n'a pas d'argument.
`VARS()` renvoie la liste des variables du répértoire courant et des noms des sous-répértoires du répértoire courant.
On peur aussi utiliser le bouton `var` du bandeau si on a choisi `Montrer->Bandeau` dans le menu `Cfg`. Ce bouton `var` affiche dans le bandeau les noms des variables affectées et les noms des sous-répértoires du répértoire courant.
Attention ! ! !
Quand on change de répértoire les variables sont celles du nouveau répértoire. Donc les variables changent quand on change de répértoire.

2.6.8 Lire un fichier depuis `Xcas` : `read`

Pour valider une ou des fonctions se trouvant dans un fichier ou pour exécuter une suite d'instructions se trouvant dans un fichier, on utilise `read` en mettant le nom du fichier entre des guillemets (`"..."`).
On tape :
`read("pgcd.cxx")`
ou encore si `pgcd.cxx` se trouve dans le répértoire `toto` :
`read("/home/texte/toto/pgcd.cxx")` ou `read("toto/pgcd.cxx")`
On peut aussi faire :
`Charger session` du menu `Fich` puis mettre `pgcd.cxx` comme nom de fichier ou si `pgcd.cxx` se trouve dans le répértoire `toto` mettre `toto/pgcd.cxx` comme nom de fichier.
Remarque

On utilise plutôt `read` pour des fichiers contenant des fonctions (fichiers `.cxx`) et `Charger session` pour des fichiers contenant des scripts (fichiers `.cas`).
Par exemple un script contenant des instructions géométriques sera exécuté mais restera figé avec `read` (on ne peut pas faire bouger les points) par contre avec `Charger session` un script contenant des instructions géométriques sera exécuté et sera interactif (on peut faire bouger les points).

Chapitre 3

Le graphique

3.1 Généralités

Si le graphe dépend d'une fonction utilisateur, il faut que la fonction soit définie lorsque le(s) paramètre(s) a (ont) une valeur formelle, ce qui peut se faire en testant le type du paramètre, comme dans l'exemple suivant : Je définis f avec le test du type du paramètre et g sans le test par :

```
f(x):= {
  if (type(x)!=DOM_FLOAT) return 'f'(x);
  while(x>0){ x--; }
  return x;
}:;
g(x):= {
  while(x>0){ x--; }
  return x;
}:;
```

Si je tape :
`F:=plotfunc(f(x))` ou `G:=plotfunc(g(x))` j'obtiens le même graphe.
Le problème apparait lorsque x n'a pas de valeur et que l'on réutilise `G`.
Mais si on fait :
`G:=plotfunc(g(x))` puis `symetrie(droite(y=x),G)` ou même simplement `G` on a l'erreur :
`"Unable to eval test in loop : x>0.0 Error: Bad Argument Value"`
parce que l'évaluation de $g(x)$ ne peut pas se faire si x est formel.
Par contre, `F:=plotfunc(f(x))` puis `symetrie(droite(y=x),F)` renvoie bien le symétrique du graphe par rapport à la première bissectrice grâce au test de la ligne :
`if (type(x)!=DOM_FLOAT) return 'f'(x);`. D'ou l'intérêt de rajouter le test.
Par contre on peut taper directement sans provoquer d'erreurs :
`symetrie(droite(y=x),plotfunc(g(x)))`.
Explications
Il faut savoir que dans les réponses de certaines commandes (par exemple `G:=plotfunc(g(x))`)
il va figurer l'expression formelle de `g(x)` (par exemple `G` contient `expr("curve(group[pnt[x+(i)*g(x)`

Lors de l'évaluation de `G` il y aura une erreur car `x+(i)*g(x)` ne pourra pas être évalué puisque l'évaluation de `g(x)` provoque l'évaluation du test `x>0` qui ne peut pas être évalué car `x` n'a pas de valeur ....d'où une erreur mais si dans la fonction figure le test : `if (type(x)!=DOM_FLOAT) return 'g'(x);` cela supprime l'évaluation de `g(x)` et donc l'erreur due au test `x>0`.

En effet, `F:=plotfunc(f(x))` puis `symetrie(droite(y=x),F)` renvoie bien le symétrique du graphe par rapport à la première bissectrice grâce au test de la ligne :
`if (type(x)!=DOM_FLOAT) return 'f'(x);`.
Par contre on peut taper directement sans provoquer d'erreurs :
`symetrie(droite(y=x),plotfunc(g(x)))`

3.2 L'écran graphique et ses boutons

Un écran graphique 2-d ou 3-d s'ouvre automatiquement en réponse d'une commande graphique 2-d ou 3-d. À un écran graphique 2-d ou 3-d est attaché des boutons situés en haut et à droite de cet écran.
Un écran de géométrie plane s'ouvre avec les touches `Alt+g` : c'est un écran graphique 2-d interactif muni de lignes d'entrée, d'une barre de menus contenant les menus `Fich Edit` et d'un bouton `Save`. Cet écran graphique est interactif : on peut définir des points et des segments en cliquant avec la souris.
Un écran de géométrie 3-d s'ouvre avec les touches `Alt+h` : c'est un écran graphique 3-d muni de lignes d'entrée, d'une barre de menus contenant les menus `Fich Edit` et d'un bouton `Save`. Les boutons d'un écran graphique 2-d et 3-d sont les mêmes en apparence mais leurs contenus sont quelquefois différents :

- les flèches de couleur rouge servent à se promener sur l'axe des x,
- les flèches de couleur verte servent à se promener sur l'axe des y,
- les flèches de couleur bleue servent en 2-d à augmenter ou à diminuer l'échelle des y (pour faire un zoom selon les y), et en 3-d à se promener sur l'axe des z,
- les boutons `in` (resp `out`) permettent de faire des zooms c'est à dire augmente (resp diminue) l'échelle des x et des y,
- le bouton $\perp$ permet d'avoir un repère orthonormé,
- le bouton `cfg` permet de changer de configuration (cf 3.4)
- le bouton ▶| permet de démarrer ou d'arrêter une animation,
- le bouton `M` permet avec ses menus
  - `Voir` d'avoir des menus simulant les boutons ci-dessus,
  - `Trace` d'intervenir sur une trace,
  - `Animation` d'intervenir sur une animation,
  - `3-d` d'avoir différentes vues d'un graphique 3-d,
  - `Export/Imprimer` d'imprimer le graphique avec un fichier `.eps` ou de traduire le graphique en Latex.

3.3 La configuration de l'écran graphique

Avant de faire un tracé, il faut régler les différents paramètres de la configuration de l'écran graphique :
le menu `Cfg▶Configuration graphique` (cf section 1.6.2) règle les paramètres de tous les graphiques qui se feront lors de la session. On peut changer ensuite ses paramètres au coup par coup avec le bouton `cfg` attaché à chaque écran graphique (cf3.4).
Les commandes du `cas` qui ont comme réponses un graphique 2-d ou 3-d seront tapées dans une ligne d'entrée. Toutefois les commandes du `cas` qui ont comme réponses un graphique 2-d peuvent aussi être tapées soit dans une ligne d'entrée d'un écran de géométrie. **Attention !** Un écran de géométrie est un écran graphique interactif.
Les commandes graphiques se trouvent dans le sous-menu `Graphic` du menu `Cmds`.
Les commandes de géométrie se trouvent dans le menu `Geo`.

3.4 Configuration graphique avec `cfg`

Le bouton `cfg` permet de régler la fenêtre graphique.

- `WX-,...,WZ` désignent les coordonnées de ce qui est visible. Pour effacer facilement les cases, on peut mettre les plages en surbrillance en utilisant soit `Shift+Tab` qui met en surbrillance la plage qui précède celle où se trouve le curseur, soit `Ctrl+Tab` qui met en surbrillance la plage qui est suit celle où se trouve le curseur.
- `Pixels` en 2-d, permet de définir les valeurs de `X-tick` et de `Y-tick` en prenant comme unité le pixel,
- `X-tick, Y-tick` pour modifier l'écartement des graduations mises sur les axes x et y, l'unité est le pixel si `Pixels` a été coché ou sinon l'unité depent des unités définies par `WX-,WX+,WY-,WY+`,
- `ry,rz,rx` en 3-d, sont les valeurs en degrés des angles d'Euler : ces valeurs permettent de retrouver facilement la position que l'on a choisie pour le repère (cf 10.2),
- `Montrer les noms` pour voir les noms des points ou des objets géométrique ou ne pas les voir,
- `Autoname` il faut mettre une lettre. Ainsi, au fur et à mesure que l'on clique dans l'écran de géométrie, les noms des points seront définis automatiquement en commençant par cette lettre en sautant les points déjà définis,
- `Montrer les axes` pour voir les axes ou ne pas les voir,
- `animate` pour donner le temps d'affichage des objets lors d'une animations (en faisant bouger la souris à droite ou à gauche on augmente ou on diminue ce temps ou on tape un nombre),
- `Round` pour avoir les coordonnèes visibles arrondies automatiquement au dizaines,
- `Default` remet les paramètre choisis par défaut,
- `Autoscale` choisit les échelles de façon automatique et appropriée,

– `Apply` a le même effet que la touche `enter` (pour valider une valeur sans fermer la fenêtre `cfg`),
– `OK` pour valider et fermer la fenêtre `cfg`,
– `Annuler` pour annuler (annule cequi n'a pas été validé par `enter` ou `Apply`) et fermer la fenêtre `cfg`.

3.5 Pour transformer un graphique en un fichier Latex

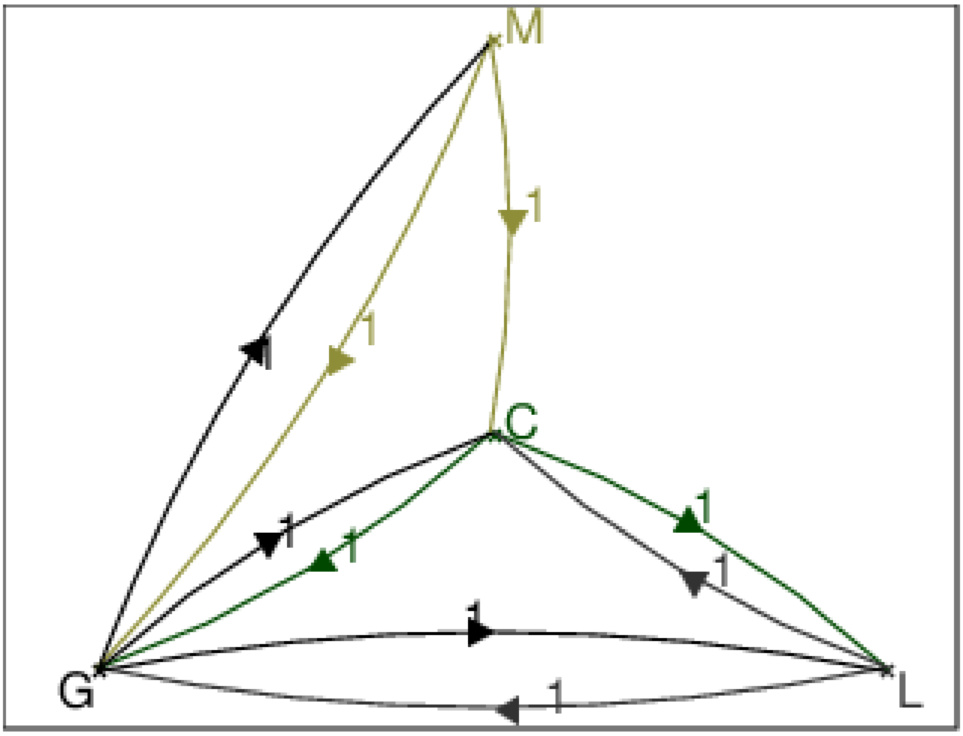
Voir aussi : 1.10.3 et 1.10.5 Il faut employer la commande `graph2tex("nom.tex")` (ou pour un graphique 3-d `graph3d2tex("nom.tex")`) pour transformer tous les graphiques réalisés en le fichier Latex `nom.tex`.
Ce fichier pourra être visualisé seul ou bien inséré dans un autre fichier Latex en otant l'en tête `\documentclass{article}...\begin{document}`, et le `\end{document}` de la fin et de rajouter `\usepackage{pstricks}` dans l'en-tête du fichier dans lequel on l'insère.
Attention Dans ce fichier tous les graphiques seront superposés : pour n'avoir qu'un seul graphique, il faut supprimer les niveaux contenant les autres graphiques avant de faire `graph2tex("nom.tex")`.

3.6 Graphe d'une matrice de transition probabiliste : `graphe_probabiliste`

`graphe_probabiliste` a comme argument une matrice de transition probabiliste A ayant au plus 7x7 entrées. On peut rajouter en option la liste des positions des sommets du graph associé à la matrice A.
On tape :

```
graphe_probabiliste([[1/2,1/3,1/12,1/12],[1/3,1/2,1/6,0],
          [0,0,1/2,1/2],[1/4,1/4,1/4,1/4]])
```

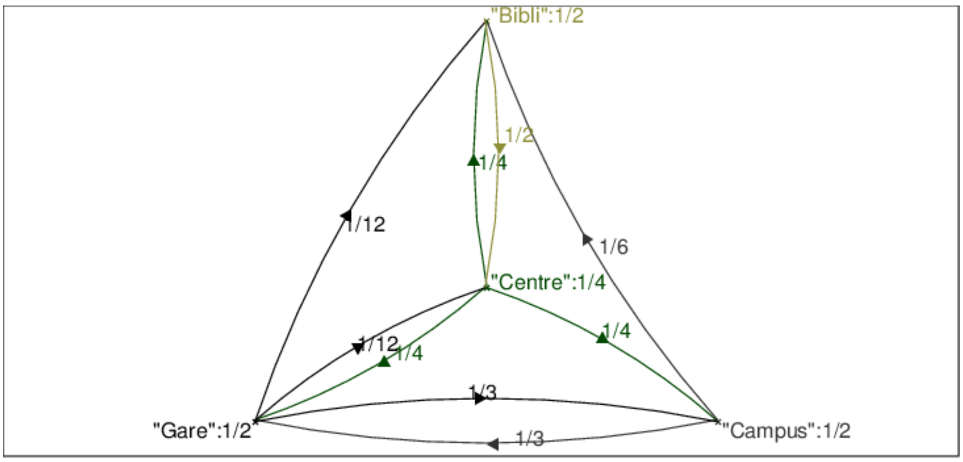
On obtient :



On tape :

```
graphe_probabiliste([[1/2,1/3,1/12,1/12],[1/3,1/2,1/6,0],
              [0,0,1/2,1/2],[1/4,1/4,1/4,1/4]],
            ["Gare","Campus","Bibli","Centre"]])
```

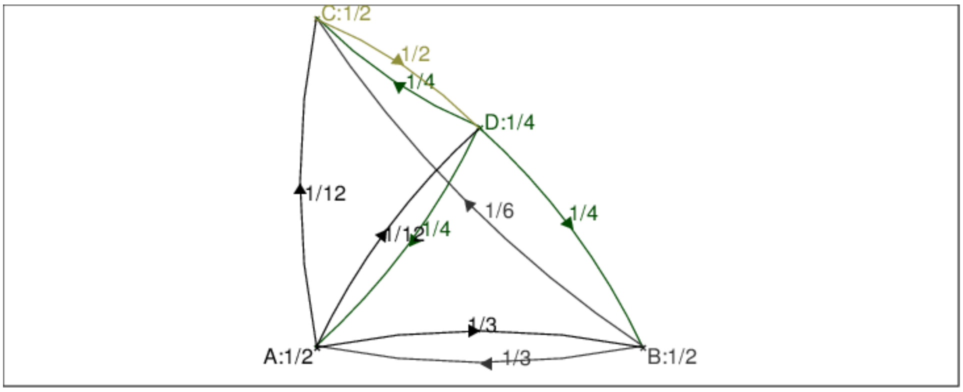
On obtient :



On tape :

```
graphe_probabiliste([[1/2,1/3,1/12,1/12],[1/3,1/2,1/6,0],
  [0,0,1/2,1/2],[1/4,1/4,1/4,1/4]],[0,1,i,1/2+2/3*i])
```

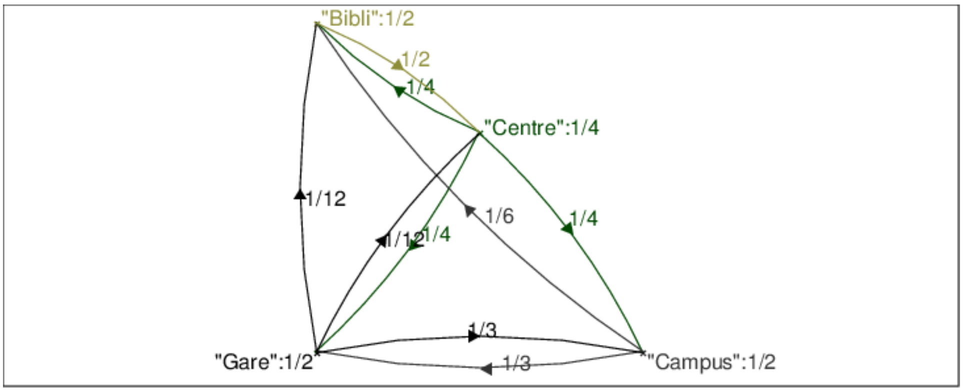
On obtient :



On tape :

```
graphe_probabiliste([[1/2,1/3,1/12,1/12],[1/3,1/2,1/6,0],
             [0,0,1/2,1/2],[1/4,1/4,1/4,1/4],
           ["Gare","Campus","Bibli","Centre"]],
                   [0,1,i,1/2+2/3*i])
```

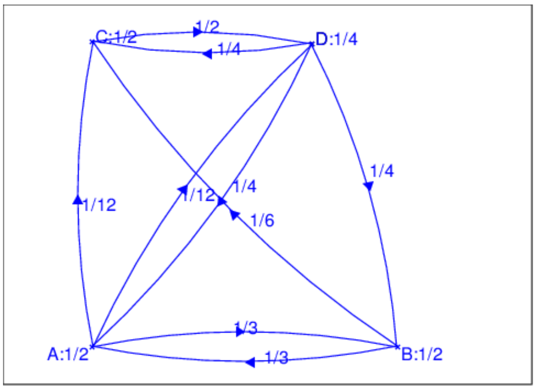
On obtient :



Si on ne met pas les "" autour d'un nom, si ce nom est le nom d'une variable qui contient une valeur c'est cette valeur que sera affichée sinon les noms s'afficheront sans "".
On peut aussi définir les points pour définir le graphe, par exemple, on tape dans un nieau de géométrie 2d :
:

```
A:=point(0);
B:=point(1);
C:=point(i);
D:=point(2/2+2i/3);
graphe_probabiliste([[1/2,1/3,1/12,1/12],[1/3,1/2,1/6,0],
[0,0,1/2,1/2],[1/4,1/4,1/4,1/4]],[A,B,C,D])
```

On peut alors bouger les points `A,B,C,D` en mode pointeur.
On obtient après avoir bougé `D` :



3.7 Graphe d'une fonction : `plotfunc funcplot DrawFunc Graph`

3.7.1 Graphe en 2-d

`plotfunc(f(x),x)` trace la représentation graphique de $y = f(x)$ et `plotfunc(f(x),x=a..b)` trace la représentation graphique de $y = f(x)$ lorsque $a \leq x \leq b$.
On tape :

```
plotfunc(x^2-2)
```

ou

```
plotfunc(a^2-2,a=-1..2)
```

On obtient :

```
la représentation graphique de y=x^2-2
```

On peut rajouter un paramètre pour indiquer le saut d'échantillonnage en x c'est à dire le pas en x que l'on veut utiliser pour faire le graphe en utilisant `xstep`.
On tape :

```
plotfunc(x^2-2,x,xstep=1)
```

On obtient :

```
une ligne polygonale qui est la représentation
grossière de y=x^2-2
```

On peut aussi spécifier le nombre de points d'échantillonage de la fonction à représenter en utilisant `nstep` à la place de `xstep`. Par exemple, on tape :

```
plotfunc(x^2-2,x=-2..3,nstep=30)
```

3.7.2 Graphe en 3-d

`plotfunc` a deux arguments principaux et éventuellement le saut d'échantillonnage des variables (`xstep=` et `ystep=`) c'est à dire le pas en x et en y que l'on choisi pour le graphe. On peut aussi spécifier le nombre de points d'échantillonnage de la fonction à représenter en utilisant `nstep`.
Les deux arguments principaux de `plotfunc` sont : une expression de deux variables ou une liste de plusieurs expressions de deux variables et la liste des deux variables.
`plotfunc` trace la (ou les) surfaces définie par le premier argument.
On peut faire tourner ce graphique selon l'axe des `x`, l'axe des `y` ou l'axe des `z`. Pour cela, il faut cliquer avec la souris dans la fenêtre graphique en dehors du parallélépipéde servant à la représentation, puis faire bouger la souris (sans relacher son bouton) ou utiliser les touches `x`, `X`, `y`, `Y`, `z` et `Z`.
On tape :

```
plotfunc( x^2+y^2,[x,y])
```

On obtient :

```
Un graphique en 3-d représentant z=x^2+y^2
```

On tape :

```
plotfunc(x*y,[x,y])
```

On obtient :

La surface z=x*y

On tape :

```
plotfunc([x*y-10,x*y,x*y+10],[x,y])
```

On obtient :

Les surfaces z=x*y-10, z=x*y et z=x*y+10

Pour n'avoir qu'une portion de surface on peut indiquer l'intervalle de variation dans le deuxième et le troisème argument.
On tape :

```
plotfunc(x*sin(y),[x=0..2,y=-pi..pi])
```

On obtient :

`Une portion de surface` $z = x * y$

On peut rajouter un paramètre pour indiquer le saut d'échantillonnage en x et en y c'est à dire le pas en x et en y que l'on veut utiliser pour faire le graphe, en utilisant`xstep` et `ystep`.
On tape :

```
plotfunc(x*sin(y),[x=0..2,y=-pi..pi],xstep=1,ystep=0.5)
```

On obtient :

`Une portion de surface` $z = x * y$

On peut aussi spécifier le nombre de points d'échantillonage de la fonction à représenter en utilisant `nstep` à la place de `xstep` et `ystep`. Par exemple, on tape :

```
plotfunc(x*sin(y),[x=0..2,y=-pi..pi],nstep=300)
```

On obtient :

`Une portion de surface` $z = x * y$

Remarque
Si vous voulez l'impression ou la traduction en Latex, il faut utiliser :
`M▶Exporter/Imprimer▶Print(with Latex).`

3.7.3 Graphe "3-d" avec les couleurs de l'arc en ciel

`plotfunc` permet aussi de représenter une expression `Xpr` dépendant de deux variables à valeur dans $\mathbb{R}$, en représentant `z=Xpr` par une couleur. Cela permet de visualiser les points ayant même cote.
Les deux arguments principaux de `plotfunc` sont alors `i*Xpr` et la liste des noms de deux variables.
Pour n'avoir qu'une portion de surface on peut indiquer l'intervalle de variation dans le deuxième et le troisème argument.
On peut rajouter un paramètre pour indiquer le pas en x et en y que l'on veut utiliser pour faire le graphe, en utilisant`xstep` et `ystep`.
On peut aussi spécifier le nombre de points d'échantillonage de la fonction à représenter en utilisant `nstep` à la place de `xstep` et `ystep`.
On tape :

```
plotfunc(i*x*sin(y),[x=0..2,y=-pi..pi])
```

On obtient :

`Une portion de surface` $z = x * y$ `avec les couleurs de l'arc en ciel`

3.7.4 Graphe en "4D"

`plotfunc` permet aussi de représenter une expression `Xpr` à valeur dans $\mathbb{C}$ mais non imaginaire pur : on représente `abs(Xpr)` selon Oz et `arg(Xpr)` par une couleur. Cela permet de visualiser les points ayant même argument.
Si l'expression `Xpr` est imaginaire pur c'est `Xpr/i` qui est representé en dégradé (cf 3.7.3) Les deux arguments principaux de `plotfunc` sont alors une expression de deux variables à valeur dans $\mathbb{C}$ et la liste des noms des deux variables.
On peut aussi spécifier le nombre de points d'échantillonnage de la fonction à représenter en utilisant `nstep` et demander un affichage en une forme pleine (`affichage=rempli`).
`plotfunc` trace la surface aux couleurs de l'arc en ciel définie par le module du premier argument soit `z=abs(Xpr)`, chaque couleur est une valeur de `arg(Xpr)`.

On peut faire tourner ce graphique selon l'axe des `x`, l'axe des `y` ou l'axe des `z`. Pour cela, il faut cliquer avec la souris dans la fenêtre graphique en dehors du parallélépipéde servant à la représentation, puis faire bouger la souris (sans relacher son bouton) ou utiliser les touches `x`, `X`, `y`, `Y`, `z` et `Z`.
On tape :

```
plotfunc((x+i*y)^2,[x,y])
```

On obtient :

```
Un graphique en 3-d coloré représentant
z=abs((x+i*y)^2 et permettant de visualiser les points
ayant même argument
```

On tape :

```
plotfunc((x+i*y)^2,[x,y],affichage=rempli)
```

On obtient :

La surface précedente selon une forme pleine aux couleurs de l'arc en ciel

Pour n'avoir qu'une portion de surface on peut indiquer l'intervalle de variation dans le deuxième et le troisème argument.
On tape :

```
plotfunc((x+i*y)^2,[x=-1..1,y=-2..2], nstep=900,
affichage=rempli)
```

On obtient :

La portion de la surface précedente selon une forme pleine aux couleurs de l'arc en ciel avec x entre -1 et 1, y entre -2 et 2 et avec 900 points d'échantillonnage

Remarque
Si vous voulez l'impression ou la traduction en Latex, il faut utiliser :
`M▶Exporter/Imprimer▶Print(with Latex)`.

3.8 Graphe 2-d pour compatibilité Maple : `plot graphe`

`plot(f(x),x)` trace la représentation graphique de $y = f(x)$.
On tape :

```
plot(x^2-2,x)
```

On obtient :

```
la représentation graphique de y=x^2-2
```

On peut rajouter un paramètre pour indiquer le saut d'échantillonnage en x c'est à dire le pas en x que l'on veut utiliser pour faire le graphe. On tape :

```
plot(x^2-2,xstep=1)
```

ou encore

```
plot(x^2-2,x,xstep=1)
```

On obtient :

```
une ligne polygonale qui est la représentation
grossière de y=x^2-2
```

On peut aussi spécifier le nombre de points d'échantillonage de la fonction à représenter en utilisant `nstep` à la place de `xstep`. Par exemple, on tape :

```
plot(x^2-2,x=-2..3,nstep=30)
```

3.9 Surface 3-d pour compatibilité Maple `plot3d graphe3d`

`plot3d` a trois arguments une fonction de deux variables (ou une expression de deux variables ou une liste de trois fonctions de deux variables ou encore une liste de trois expressions de deux variables) et les noms de ces deux variables.
`plot3d` trace la surface définie par le premier argument (soit $z = f(x, y)$, soit $x = f(u, v), y = g(u, v), z = h(u, v)$).
On peut faire tourner ce graphique selon l'axe des `x`, l'axe des `y` ou l'axe des `z`. Pour cela, il faut cliquer avec la souris dans la fenêtre graphique en dehors du parallélépipéde servant à la représentation, puis faire bouger la souris (sans relacher son bouton) ou utiliser les touches `x`, `X`, `y`, `Y`, `z` et `Z`.
On tape :

```
plot3d(x*y,x,y)
```

On obtient :

La surface $z = x * y$

On tape :

```
plot3d([v*cos(u),v*sin(u),v],u,v)
```

On obtient :

Le cône $x = v * \cos(u), y = v * \sin(u), z = v$

Pour n'avoir qu'une portion de surface on peut indiquer l'intervalle de variation dans le deuxième et le troisème argument.
On tape :

```
plot3d([v*cos(u),v*sin(u),v],u=0..pi,v=0..3)
```

On obtient :

Une portion du cône $x = v * \cos(u), y = v * \sin(u), z = v$

3.10 Graphe d'une droite et les tangentes à un graphe

3.10.1 Tracé d'une droite : `line droite`

Voir aussi : 9.10.1 et 10.5.1 pour la droite en géométrie et 9.10.1 et 10.5.2 pour la droite orientée.
`droite` a comme argument son équation cartésienne :
- en 2-d
  une équation de droite,
- en 3-d
  deux équations de plan.

`droite` définit et trace la droite d'équation donnée en argument.
On tape :

```
droite(2*y+x-1=0)
```

On obtient :

```
le tracé de la droite 2*y+x-1=0
```

On tape :

```
droite(y=1)
```

On obtient :

```
le tracé de la droite horizontale y=1
```

On tape :

```
droite(x=1)
```

On obtient :

```
le tracé de la droite verticale x=1
```

On tape :

```
droite(x+2*y+z-1=0,z=2)
```

On obtient :

```
le tracé de la droite x+2*y+1=0 dans le plan z=2
```

On tape :

```
droite(y=1,x=1)
```

On obtient :

```
le tracé de la droite verticale passant par (1,1,0)
```

Remarque
`droite` définit une droite orientée :

- Lorsque la droite 2-d est donnée par son équation, on met cette équation sous la forme "membre_gauche-membre_droite=`ax+by+c=0`", cela détermine son vecteur normal `[a,b]` et l'orientation est donnée par le vecteur `[b,-a]`) (ou encore son orientation est définie par le produit vectoriel 3-d de son vecteur normal (de cote 0) et du vecteur de coordonnées [0,0,1]).
  Par exemple `droite(y=2*x)` définit une droite orientée par le vecteur de coordonnées `[1,2]`.
- Lorsque la droite 3-d est donnée par deux équations de plans, son orientation est définie par le produit vectoriel des normales aux plans (en mettant les équations des plans sous la forme "membre_gauche-membre_droite=0" on détermine les normales orientées de ces plans).
  Par exemple, `droite(x=y,y=z)` est orientée par :
  `cross([1,-1,0],[0,1,-1])=[1,1,1]`.

3.10.2 Tracé d'une droite horizontale en 2-d : `LineHorz`

`LineHorz` a comme argument une expression Xpr.
`LineHorz` trace la droite horizontale $y = Xpr$.
On tape :

```
LineHorz(1)
```

On obtient :

```
le tracé de la droite y=1
```

3.10.3 Tracé d'une droite verticale en 2-d : `LineVert`

`LineVert` a comme arguments une expression Xpr.
`LineVert` trace la la droite verticale $x = Xpr$.
On tape :

```
LineVert(1)
```

On obtient :

```
le tracé de la droite x=1
```

3.10.4 Tangente à un graphe en 2-d : `LineTan droite_tangente`

`LineTan` (resp `droite_tangente`) a deux arguments : une expression Xpr de la variable x et une valeur $x0$ de x.
Attention pour `LineTan` il ne faut pas mettre de parenthèses ou alors il faut les mettre à l'extérieur.
`LineTan` (resp `droite_tangente`) trace la tangente en $x = x0$ à la représentation graphique de $y = Xpr$.
On tape :

```
(LineTan ln(x),1)
```

Ou on tape :

`droite_tangente(ln(x),1)`

On obtient :

`le tracé de la droite y=x-1`

On tape :

`equation(LineTan ln(x),1)`

Ou on tape :

`equation(droite_tangente(ln(x),1))`

On obtient :

`l'équation y=(x-1)`

)

3.10.5 Tangente en un point d'un graphe en 2-d : `tangent tangente`

Voir aussi : 9.10.8 pour la géométrie plane et 10.6.3 pour la géométrie 3-d.
`tangent` a deux arguments : un objet géométrique et un point `A`.
Mais quand l'objet géométrique est le graphe `G` d'une fonction 2-d, le deuxième argument doit être soit, un nombre réel x_0, soit un point `A` situé sur `G`.

Par exemple on tape :
```
G:=plotfunc(g(x),x)
tangent(G, 1.2)
```
trace la tangente au graphe `G` de la fonction `g` au point d'abscisse `x=1.2`,
ou on tape :
```
A:=point(1.2+i*g(1.2))
tangent(G, A)
```
trace la tangente au point `A` du graphe `G` de la fonction `g`.
Par exemple, pour avoir le tracé de la tangente au graphe de $g(x) = x^2$ au point d'abscisse $x_0 = 1$, on tape :

`g(x):=x^2; G:=plotfunc(g(x),x)`

`T:=tangent(G,1)`

ou on tape :

`T:=tangent(G,point(1+i))`

On obtient

La tangente au graphe de $g(x) = x^2$ au point 1+i

L'équation de la tangente est alors obtenue en tapant :

`equation(T)`

3.10.6 Tracé d'une droite donnée par un point et sa pente : `DrawSlp`

`DrawSlp` a comme argument trois réels `a,b,m`.
`DrawSlp` renvoie et trace la droite de pente `m` et passant par le point de coordonnées `a,b`.
On tape :

```
DrawSlp(2,1,-1)
```

On obtient :

```
la droite d'équation y=(-x+3) qui a pour pente -1 et
          qui passe par le point(2,1)
```

3.10.7 Intersection d'un graphe en 2-d avec les axes

Pour avoir l'ordonnée de l'intersection du graphe de f avec l'axe des y on tape :

```
f(0)
```

le point de coordonnées : $(0, f(0))$ est donc le point d'intersection du graphe de f avec l'axe des y.
Pour avoir l'intersection du graphe de f avec l'axe des x il faut résoudre $f(x) = 0$.
On peut essayer d'avoir les valeurs exactes des abscisses de ces points en tapant :

```
solve(f(x),x)
```

ou avoir les valeurs approchées de ces abscisses en utilisant la représentation graphique puis, en utilisant `fsolve` pour avoir une meilleure précision.

3.11 Représentation graphique d'inéquations à 2 variables : `plotinequation inequationplot`

`plotinequation([f1(x,y)<a1,..,fk(x,y)<ak],[x=x1..x2,y=y1..y2])`
trace la surface du plan définie par les inéquations à 2 variables :

$$f1(x, y) < a1$$

$$...$$

$$fk(x, y) < ak$$

$$x1 < x < x2$$

$$y1 < y < y2$$

On tape :

```
plotinequation(x^2-y^2<3,
[x=-2..2,y=-2..2],xstep=0.1,ystep=0.1)
```

On obtient :

```
la partie contenant l'origine et délimitée par
     l'hyperbole x^2-y^2=3 est remplie
```

On tape :

```
plotinequation([x+y>3,x^2<y],
[x-2..2,y=-1..10],xstep=0.2,ystep=0.2)
```

On obtient :

```
le morceau du plan définit par -2<x<2,y<10,x+y>3,y>x^2
est rempli
```

Attention
Si on ne met pas les bornes pour x et y ce sont les valeurs de `X-,X+,Y-,Y+` mises dans la configuration générale du graphique (`Cfg▶Configuration graphique`) qui seront prises en compte.

3.11.1 Aire sous une courbe : `area aire`

`aire` ou `area` calcule de façon approchée l'aire sous la courbe $y = f(x)$ comprise entre $x = a$ et $x = b$.
`aire` ou `area` a quatre arguments : l'expression $f(x)$, $x = a..b$, un entier n et le nom de la méthode numérique choisie pour faire calcul.
La méthode numérique est choisie parmi :
`trapezoid, left_rectangle, right_rectangle, middle_point`
ou `trapeze, rectangle_gauche, rectangle_droit, point_milieu`
et aussi `simpson` (méthode de Simpson), `rombergt` (accélération de convergence de Romberg avec la méthode des trapèzes), `rombergm` (accélération de convergence de Romberg avec la méthode du point milieu) et `gauss15` (avec une quadrature de Gauss adaptative à 15 points).
La valeur de l'entier n est le nombre de subdivisions que l'on a choisi pour les calculs faits avec les méthodes `trapezoid, left_rectangle, right_rectangle, middle_point` ou `trapeze, rectangle_gauche, rectangle_droit, point_milieu, simpson` alors que pour `gauss15`, `rombergt` et `rombergm` le nombre de subdivisions est 2^n.
Ainsi,`area(f(x),x=a..b,n,trapeze)` calcule l'aire de n trapèzes :
le troisième argument est un entier n, et le quatrième argument est le nom de la méthode numérique d'intégration lorsqu'on partage $[a, b]$ en n parties égales.
`area(f(x),x=a..b,n,rombergt)` revient à calculer l'aire de 2^n trapèzes qui sont accélérés.
On tape :

```
area(x^2,x=0..1,8,trapeze)
```

On obtient :

```
0.3359375
```

On tape :

```
area(x^2,x=0..1,8,point_milieu)
```

On obtient :

```
0.33203125
```

Donc comme $f(x) = x^2$ est convexe on a l'aire est dans l'intervalle :
$]0.33203125, 0.3359375[$:
On tape :

```
area(x^2,x=0..1,3,rombergt)
```

On obtient une meilleur approximation :

```
0.333333333333
```

On tape :

```
area(x^2,x=0..1,3,rombergm)
```

On obtient une meilleur approximation :

```
0.333333333333
```

On tape :

```
area(x^2,x=0..1,3,gauss15)
```

On obtient :

```
1/3
```

On tape :

```
area(x^2,x=0..1)
```

On obtient :

```
1/3
```

3.12 Représentation graphique de l'aire sous une courbe : `tracer_aire graphe_aire aire_graphe plotarea areaplot`

- Avec deux arguments, `aire_graphe` ou `plotarea` permet de représenter et d'afficher avec 3 digits l'aire sous une courbe.
  Ainsi, `plotarea(f(x),x=a..b)` trace l'aire sous la courbe $y = f(x)$ pour $a < x < b$, c'est à dire la portion du plan définie par les inéquations $a < x < b$ et selon le signe de $f(x)$ $0 < y < f(x)$ ou $0 > y > f(x)$.
  On tape :

  ```
  plotarea(sin(x),x=0..2*pi)
  ```

  On obtient :

  ```
  la portion de plan situé dans les deux arches de sin(x)
  ```

– Avec quatre arguments, `aire_graphe` ou `plotarea` permet de représenter l'aire qui est caculée avec la méthode numérique choisie parmi :
`trapezoid, left_rectangle, right_rectangle, middle_point`
ou `trapeze, rectangle_gauche, rectangle_droit, point_milieu`.
Ainsi, `plotarea(f(x),x=a..b,n,trapeze)` trace l'aire de n trapèzes : le troisième argument est un entier n, et le quatrième argument est le nom de la méthode numérique d'intégration lorsqu'on partage $[a, b]$ en n parties égales.
On tape :

```
plotarea(x^2,x=0..1,5,trapeze)
```

Ou on tape pour voir la courbe en rouge :

```
plotarea(x^2,x=0..1,5,trapeze);
plot(x^2,x=0..1,affichage=rouge)
```

On obtient :

```
les 5 trapèzes qui sont utilisés dans la méthode
dite des trapèzes, pour approcher une intégrale
```

On tape :

```
plotarea(x^2,x=0..1,5,point_milieu)
```

Ou on tape pour voir la courbe en rouge :

```
plotarea(x^2,x=0..1,5,point_milieu);
plot(x^2,x=0..1,affichage=rouge)
```

On obtient :

```
les 5 rectangles qui sont utilisés dans la méthode
dite du point milieu, pour approcher une intégrale
```

Remarque 1 On peut aussi taper, pour n'avoir que la valeur de l'aire :

```
plotarea(x^2,x=0..1,5,trapeze)[0,3];
```

On obtient :

```
0.34
```

Remarque 2 Si on utilise `plotarea` avec le menu `Graphic->Courbes->plotarea` une boite de dialogue s'ouvre : vous entrez, l'expression de la fonction, le nom de la variable, les bornes de l'intervalle `xmin,xmax`, le pas `xstep` (on a alors `n=(xmax-xmin)/xstep`), la méthode d'intégration et aussi la couleur du dessin (on retrouve en effet le bouton `Attribut` en haut et à gauche de la boite de dialogue).

3.13 Lignes de niveaux : plotcontour contourplot DrwCtour

`plotcontour(f(x,y),[x,y])` (ou `DrwCtour(f(x,y),[x,y])` ou encore `contourplot(f(x,y),[x,y])`) trace les 11 lignes de niveaux $z = -10$, $z = -8$,.., $z = 0$, $z = 2$,.., $z = 10$ de la surface définie par $z = f(x, y)$.
On tape :

```
plotcontour(x^2+y^2,[x=-3..3,y=-3..3],[1,2,3],
   affichage=[vert,rouge,noir]+[rempli$3])
```

On obtient :

```
le graphe des trois cercles x^2+y^2=n pour n=1,2,3;
les zones comprises entre ces cercles sont remplies
avec la couleur verte,rouge ou noire
```

On tape :

```
plotcontour(x^2-y^2,[x,y])
```

On obtient :

```
le graphe des 11 hyperboles x^2-y^2=n pour
n=-10,-8,..10
```

Pour visualiser la surface, on tape (`plotfunc(f(x,y),[x,y])` trace la représentation graphique de $z = f(x, y)$, voir 3.7.2) :

```
plotfunc( x^2-y^2,[x,y])
```

On obtient :

```
Un graphique en 3-d représentant z=x^2+y^2
```

On peut faire tourner ce graphique selon l'axe des `x`, l'axe des `y` ou l'axe des `z`. Pour cela, il faut cliquer avec la souris dans la fenêtre graphique en dehors du parallélépipéde servant à la représentation, puis faire bouger la souris (sans relacher son bouton) ou utiliser aux touches `x`, `X`, `y`, `Y`, `z` et `Z`.

3.14 Graphe d'une fonction par niveaux de couleurs : `plotdensity densityplot`

`plotdensity(f(x,y),[x,y])` ou encore `densityplot(f(x,y),[x,y])` trace le graphe de $z = f(x, y)$ dans le plan en représentant z par une des couleurs de l'arc en ciel.
On tape :

```
plotdensity(x^2-y^2,[x=-2..2,y=-2..2],xstep=0.1,ystep=0.1)
```

On obtient :

```
Un graphique en 2-d représentant pour chaque z,
l'hyperbole définie par x^2-y^2=z par une couleur de
l'arc en ciel
```

On remarquera que l'on a l'echelle des couleurs en dessous du graphe.

3.15 Courbe implicite : plotimplicit implicitplot

plotimplicit ou implicitplot permet de tracer des courbes ou des surfaces définies de façon implicite par une expression. Pour que Xcas ne cherche pas à factoriser l'expression, la commande plotimplicit ou implicitplot peut être utilisée avec l'option unfactored ou sans_factoriser mise comme dernier paramètre, :

– avec unfactored l'expression ne sera pas modifiée,
– sans unfactored Xcas réduit l'expression au même dénominateur puis cherche à factoriser le numérateur.

3.15.1 Courbe implicite en 2-d

– plotimplicit(f(x,y),x,y) ou plotimplicit(f(x,y),[x,y]) trace la représentation graphique de la courbe définie implicitement par $f(x, y) = 0$ lorsque x (resp y) varie selon WX-, WX+ (resp WY-, WY+) défini dans cfg,
– plotimplicit(f(x,y),x=0..1,y=-1..1) ou plotimplicit(f(x,y),[x=0..1,y=-1..1]) trace la représentation graphique de la courbe définie implicitement par $f(x, y) = 0$ lorsque $0 \leq x \leq 1$ et $-1 \leq y \leq 1$ (mettre des bornes un peu plus grandes pour ne pas avoir de manques !).

On peut éventuellement rajouter encore deux paramètres pour spécifier le saut d'échantillonnage des variables avec xstep= et ystep=, c'est à dire le pas en x et en y que l'on choisi pour le graphe.
On tape :

```
plotimplicit(x^2+y^2-1,[x,y])
```

Ou on tape :

```
plotimplicit(x^2+y^2-1,x,y,unfactored)
```

On obtient :

```
Le dessin du cercle unité
```

On tape :

```
plotimplicit(x^2+y^2-1,x,y,xstep=0.2,ystep=0.3)
```

Ou on tape :

```
plotimplicit(x^2+y^2-1,[x,y],xstep=0.2,ystep=0.3)
```

Ou on tape :

```
plotimplicit(x^2+y^2-1,[x,y],
xstep=0.2,ystep=0.3,unfactored)
```

On obtient :

```
Le dessin du cercle unité
```

On tape :

```
plotimplicit(x^4+y^4=x^2+y^2)
```

On obtient :

```
Le dessin du symbole infini
```

On tape :

```
plotimplicit(x^2+4*y^3-k)$(k=1..5)
```

On obtient :

```
Le dessin de 5 courbes de la forme du chapeau de
Napoléon
```

3.15.2 Surface implicite en 3-d

- `plotimplicit(f(x,y,z),x,y,z)` trace la représentation graphique de la surface définie implicitement par : $f(x, y, z) = 0$,
- `plotimplicit(f(x,y,z),x=0..1,y=-1..1,z=-1..1)` trace la représentation graphique de la surface définie implicitement par $f(x, y, z) = 0$ lorsque $0 \leq x \leq 1$, $-1 \leq y \leq 1$ et $-1 \leq z \leq 1$.

On peut éventuellement rajouter trois paramètres pour spécifier le saut d'échantillonnage des variables (`xstep=`, `ystep=` et `zstep=`) c'est à dire le pas en x, en y et en z que l'on choisi pour le graphe.
On tape :

```
plotimplicit(x^2+y^2+z^2-1,x,y,z,
xstep=0.2,ystep=0.1,zstep=0.3)
```

On tape :

```
plotimplicit(x^2+y^2+z^2-1,x,y,z,
xstep=0.2,ystep=0.1,zstep=0.3,unfactored)
```

On obtient :

```
Le dessin de la sphère unité
```

On tape :

```
plotimplicit(x^2+y^2+z^2-1,x=-1..1,y=-1..1,z=-1..1)
```

On obtient :

```
Le dessin de la sphère unité
```

3.16 Courbe et surface en paramétrique : `plotparam` `paramplot` `DrawParm` `courbe_parametrique`

3.16.1 Courbe 2-d en paramétrique

`plotparam(f(t)+i*g(t),t)` (resp `plotparam(f(t)+i*g(t),t=t1..t2))` trace la représentation paramétrique de la courbe définie par $x = f(t), y = g(t)$ (resp par $x = f(t), y = g(t)$ et $t1 \geq t \geq t2$).
Si on ne précise pas les bornes de l'intervalle de variation du paramètre ce sont les valeurs de `t-` et `t+` (cf 1.6.2) qui seront ces bornes.
On tape :

```
plotparam(cos(x)+i*sin(x),x)
```

ou

```
plotparam([cos(x),sin(x)],x)
```

On obtient :

```
Le dessin du cercle unité
```

On peut péciser les bornes de l'intervalle de variation du paramètre.
On tape si dans la configuration du graphique `t` va de -4 à 1 :

```
plotparam(sin(t)+i*cos(t))
```

ou encore :

```
plotparam(sin(t)+i*cos(t),t=-4..1)
```

ou encore :

```
plotparam(sin(x)+i*cos(x),x=-4..1)
```

On obtient :

```
Le dessin de l'arc du cercle unité allant de -4 à 1
```

On peut rajouter un paramètre pour indiquer le saut d'échantillonnage du paramètre t avec `tstep=` c'est à dire le pas en t que l'on veut utiliser pour faire le graphe.
On tape si dans la configuration du graphique `t` va de -4 à 1 :

```
plotparam(sin(t)+i*cos(t),t,tstep=0.5)
```

Ou on tape :

```
plotparam(sin(t)+i*cos(t),t=-4..1,tstep=0.5)
```

On obtient :

Le dessin grossier de l'arc du cercle unité allant de -4 à 1

3.16.2 Surface 3-d en paramétrique : plotparam paramplot DrawParm courbe_parametrique

plotparam a deux arguments principaux et éventuellement les sauts d'échantillonnage des variables avec ustep= et vstep=, c'est à dire le pas en u et en v que l'on choisi pour le graphe.
Les deux arguments principaux de plotparam sont : une liste de trois expressions de deux variables et la liste des deux variables.
plotparam([f(u,v),g(u,v),h(u,v)],[u,v]) trace la surface définie par le premier argument soit $x = f(u, v), y = g(u, v), z = h(u, v)$).
On peut faire tourner ce graphique selon l'axe des x, l'axe des y ou l'axe des z. Pour cela, il faut cliquer avec la souris dans la fenêtre graphique en dehors du parallélépipéde servant à la représentation, puis faire bouger la souris (sans relacher son bouton) ou utiliser les touches x, X, y, Y, z et Z.
On tape :

```
plotparam([v*cos(u),v*sin(u),v],[u,v])
```

On obtient :

Le cône $x = v * \cos(u), y = v * \sin(u), z = v$

Pour n'avoir qu'une portion de surface on peut indiquer l'intervalle de variation dans le deuxième et le troisème argument.
On tape :

```
plotparam([v*cos(u),v*sin(u),v],[u=0..pi,v=0..3])
```

On obtient :

Une portion du cône $x = v * \cos(u), y = v * \sin(u), z = v$

On tape :

```
plotparam([v*cos(u),v*sin(u),v],[u=0..pi,v=0..3],
             ustep=0.5,vstep=0.5)
```

On obtient :

Une portion du cône $x = v * \cos(u), y = v * \sin(u), z = v$

Remarque
Si vous voulez l'impression ou la traduction en Latex, il faut utiliser :
M▶Exporter/Imprimer▶Print(with Latex).

3.17 Courbes de Bézier : bezier

Soient $n + 1$ points P_j de contrôle ($j = 0..n$) et L la séquence de ces points.
La courbe de Bézier ayant les points de la séquence L comme points de contrôle, a comme équation paramétrique :
$\sum_{j=0}^{n} comb(n, j)t^j(1 - t)^{n-j} * L[j]$.
bezier(L,plot) renvoie le tracé de la courbe d'équation paramétrique : $\sum_{j=0}^{n} comb(n, j)t^j(1-$

$t)^{n-j} * L[j]$.
`parameq(bezier(L))` renvoie l'équation paramétrique de la courbe de Bézier ayant comme points de contrôle les points de la séquence `L`.
On tape :

```
bezier(1,1+i,2+i,3-i,plot)
```

On obtient :

```
Le tracé de la courbe de Bézier ayant comme points de
contrôle les points d'affixe 1,1+i,2+i,3-i
```

On tape :

```
parameq(bezier(1,1+i,2+i,3-i))
```

On obtient :

```
L'équation paramétrique de la courbe précédente
```

On tape :

```
bezier(point([0,0,0]),point([1,1,0]),point([0,1,1]),plot)
```

On obtient :

```
Le tracé de la courbe de Bézier ayant comme points de
contrôle les points
point([0,0,0]),point([1,1,0]),point([0,1,1])
```

On tape :

```
parameq(bezier(point([0,0,0]),point([1,1,0]),point([0,1,1])))
```

On obtient :

```
L'équation paramétrique de la courbe précédente
```

3.18 Courbe en polaire : `plotpolar polarplot DrawPol courbe_polaire`

`plotpolar(f(t),t)` trace la représentation polaire de la courbe définie par : $\rho = f(t)$.
On tape si dans la configuration du graphique `t` va de 0 à 10 :

```
plotpolar(t,t)
```

On obtient :

```
La spirale ρ=t est dessinée
```

On peut rajouter un paramètre (`tstep=`) pour indiquer le saut d'échantillonnage en t c'est à dire le pas en t que l'on veut utiliser pour faire le graphe. On tape si dans la configuration du graphique `t` va de 0 à 10 :

```
plotpolar(t,t,tstep=1)
```

ou :

```
plotpolar(t,t=0..10,tstep=1)
```

On obtient :

La spirale ρ=t est dessinée grossièrement

3.19 Tracé d'une suite récurrente : plotseq seqplot graphe_suite

`plotseq(f(x),a,n)` ou `plotseq(f(t),t=a,n)` permet de visualiser les n premiers termes d'une suite récurrente définie par :
$u_0 = a,\ \ u_n = f(u_{n-1})$
On tape :

```
plotseq(sqrt(1+x),3,5)
```

On obtient :

Le dessin de y=sqrt(1+x), de y=x et des 5 premiers termes de la suite u_0=3 et u_n=sqrt(1+u_(n-1))

3.20 Le champ des tangentes : plotfield fieldplot

On peut tracer le champ des tangentes de léquation différentielle $y' = f(t, y)$ ou du système déquations différentielles $x' = u(x, y), y' = v(x, y)$ et on peut spécifier les plages de valeurs des paramètres.

- Soit $f(t, y)$ une expression dependant de deux variables t et y, alors `plotfield(f(t,y),[t,y])` trace le champ des tangentes de l'équation différentielle $y' = f(t, y)$ où y représente une variable réelle et t est représenté en abscisse,
- Soit $V = [u(x, y), v(x, y)]$ est un vecteur 2-d de coordonnées deux expressions dépendant de 2 variables x, y mais indépendant du temps, alors `plotfield(V,[x,y])` trace le champ des tangentes du système $[x'(t) = u(x, y), y'(t) = v(x, y)]$,
- Les plages de valeurs de t, y ou de x, y peuvent être spécifiées par `t=tmin..tmax`, `x=xmin..xmax`, `y=ymin..ymax` à la place du nom de variable seul.
- On peut spécifier le cadrage en mettant par exemple :
`plotfield(f(t,y),[t=tmin..tmax,y=ymin..ymax])`
- On peut spécifier que le champ des tangentes soit, dans un repère orthonormé, de norme 1 avec l'option `normalize`. Sans l'option `normalize` le point de contact est l'origine du vecteur tangent et avec l'option `normalize` le point de contact se trouve au milieu des tangentes.
- On peut aussi spécifier la valeur des pas en t et en y avec `xstep=...` et `ystep=...`.

On tape :

```
plotfield(4*sin(t*y),[t=0..2,y=-3..7])
```

On obtient :

```
Des segments de pente 4*sin(t*y) sont tracés en
différents points. Ces segments représentent les
vecteurs tangents dirigés selon les t croissants et
dont l'origine est le point de contact
```

On tape :

```
plotfield(4*sin(t*y),[t=0..2,y=-3..7],normalize,
xstep=0.7,ystep=0.7))
```

On obtient :

```
Des segments de longueur 1 et de pente 4*sin(t*y) qui
représentent les tangentes au point situé en leur
milieu. Ces points espacés de 0.7
```

On tape :

```
plotfield(5*[-y,x],[x=-1..1,y=-1..1])
```

On obtient :

Des vecteurs $[-y, x]$ sont tracés aux points (x, y). Ces vecteurs représentent des vecteurs tangents en leur origine aux courbes solutions du système $x(t)' = -y, y(t)' = x$. Ils sont dirigés selon les t croissants.

On tape :

```
plotfield(5*[-y,x],[x=-1..1,y=-1..1],normalize)
```

On obtient :

Des segments de longueur 1 et de pente $-y/x$ qui représentent les tangentes au point situé en leur milieu aux courbes solutions du système $x(t)' = -y, y(t)' = x$.

3.21 Tracé de solutions d'équation différentielle : `plotode` `odeplot`

On peut tracer les solutions de léquation différentielle $y' = f(t, y)$ ou du système déquations différentielles $x' = u(t, x, y), y' = v(t, x, y)$ et on peut spécifier les plages de valeurs des paramètres.

- `plotode(f(t,y),[t,y],[t0,y0])` trace en fonction du temps la solution $y(t)$ de l'équation différentielle $y' = f(t, y)$ passant par le point `(t0,y0)`, où $f(t, y)$ désigne une expression dépendant de la variable de temps t et de la variable y.
- Par défaut, t varie dans les 2 directions. On peut spécifier la plage du temps par le paramètre optionnel `t=tmin..tmax`.

– Lorsque $y = (X, Y)$ est un vecteur de longueur 2 et f à valeurs dans $\mathbb{R}^2$, on peut également représenter dans l'espace (t, X, Y) ou dans le plan (X, Y) la solution d'une équation différentielle $y' = f(t, y)$ c'est à dire $[X', Y'] = [f(t, X, Y)]$. Pour cela, il suffit de remplacer y par le noms des variables X, Y et la valeur initiale par les deux valeurs initiales des variables au temps t_0.

On tape :

```
plotode(sin(t*y),[t,y],[0,1])
```

On obtient :

Le graphe de la solution de y'=sin(t,y) passant par le point (0,1) est tracé

On tape :

```
S:=odeplot([h-0.3*h*p, 0.3*h*p-p],
         [t,h,p],[0,0.3,0.7])
```

On obtient le graphe dans l'espace de la solution de

$$[h, p]' = [h - 0.3hp, 0.3hp - p] \quad [h, p](0) = [0.3, 0.7]$$

Pour avoir le graphe dans le plan, on ajoute l'option `plan` ou `plane`

```
S:=odeplot([h-0.3*h*p, 0.3*h*p-p],
      [t,h,p],[0,0.3,0.7],plan)
```

Pour visualiser les valeurs de la solution, se reporter à la section 4.3.6 On tape :

```
plotfield(5*[-y,x],[x=-1..1,y=-1..1],normalize)
```

```
plotode(5*[-y,x],[t=0..1,x,y],[0,0.3,0.7],tstep=0.05,plan)
```

On obtient :

Le graphe de la solution de x'=-y,y'=x pour t=0 passant par le point (0.3,0.7) est tracé

Exemple On trace 4 solutions du système d'équations différentielles dépendant de 2 paramètre a et b :
$x' = -y + b$
$y' - 1 + (x - a)^2 + (y - b)^2$
Les conditions initiales sont :
pour $t = 0$ $x0 = a + 1, y0 = b + 0.5$
pour $t = 0$ $x0 = a + 1, y0 = b + 0.1$
pour $t = 0$ $x0 = a + 0.827, y0 = b + 0.827$
pour $t = 0$ $x0 = a - 1.1, y0 = b+$
On tape :

```
avril(a,b):={
local L;
L:=NULL;
L:=L,affichage(plotode([-y+b,-1+(x-a)^2+(y-b)^2],[t=-3..3,x,y],[0,a+1,
plan),94+epaisseur_ligne_8);
L:=L,affichage(plotode([-y+b,-1+(x-a)^2+(y-b)^2], [t=-3..3,x,y],[0,a+1
plan),4+epaisseur_ligne_8);
L:=L,affichage(plotode([-y+b,-1+(x-a)^2+(y-b)^2],[t=-6..3.65,x,y],
[0,a+0.827,b+0.827],plan),1+epaisseur_ligne_4);
L:=L,affichage(plotode([-y+b,-1+(x-a)^2+(y-b)^2], [t=-1.3..1.3,x,y],[0
plan),1+epaisseur_ligne_4);
return L;
}:;
```

Puis on tape par exemple :

```
affichage(cercle(0,5,3*pi/4,4*pi/3),4+epaisseur_ligne_4);
affichage(cercle(0,5,5*pi/3,2*pi+pi/4),4+epaisseur_ligne_4);
affichage(segment(5*exp(-i*pi/3),5*exp(-2*i*pi/3)),4+epaisseur_ligne_4
avril(-1.4,-1);
```

3.22 Tracé interactif des solutions d'équation différentielle : interactive_plotode interactive_odeplot

`interactive_plotode(f(t,y),[t,y])` trace le champ des tangentes de l'équation différentielle $y' = f(t, y)$ dans l'écran `DispG` et `interactive_plotode(f(t,y),[t=a...b,y])` trace le champ des tangentes pour `t` allant de `a` à `b` de l'équation différentielle $y' = f(t, y)$ dans l'écran `DispG`.
Lorsqu'on clique sur un point, on obtient le tracé de la solution de $y' = f(t, y)$ passant par ce point.
On peut faire autant de tracés que l'on veut (un tracé se fait chaque fois que l'on clique sur un point avec la souris). On termine les tracés en tapant sur la touche `Esc` ou `Echap`.
On peut aussi spécifier, comme dans `plotfield`, que le champ des tangentes soit de norme 1 avec l'option `normalize`. **Attention** Si on ne veut pas de superposition avec les dessins faits auparavant, il ne faut pas oublier de taper `ClrGraph`, avant d'utiliser `interactive_plotode`, pour effacer l'écran `DispG`. On tape :

```
interactive_plotode(-y+x+1,[x=-4..4,y])
```

On obtient :

```
Le champ des tangentes est tracé ainsi que la solution
de y'=sin(t,y) passant par le point qui a été cliqué
avec la souris
```

IL se trouve que l'on sait résoudre cette équation : les solutions sont `y(x)=C*exp(-x)+x` et on peut donc vérifier...
On tape :

```
interactive_plotode(sin(t*y),[t=-4..4,y])
```

On obtient :

```
Le champ des tangentes est tracé ainsi que la solution
de y'=sin(t,y) passant par le point qui a été cliqué
avec la souris
```

On tape :

```
interactive_plotode(sin(t*y),[t=-4..4,y],normalize)
```

On obtient :

```
Le tracé du champ des tangentes avec une norme égale à
1 et le graphe de la solution de y'=sin(t,y) passant
par le point qui a été cliqué avec la souris
```

3.23 Tracé interactif des solutions d'équation différentielle dans un niveau de géométrie : `plotfield fieldplot` et `plotode odeplot`

Dans un niveau de géométrie, le menu `Graphe->Slopefield/Ode(2d)` ouvre une boite de dialogues qui demande :

- si on veut que soit tracé le champ des tangentes,
- si on veut que ces tangentes soient normalisées dans un repère orthonormé,
- la valeur de y',
- le nom des variables,
- les différentes valeurs de cadrage et de pas.

Lorsqu'on appuie sur `OK`, l'écran de géométrie est en mode `plotode` et si l'on a coché `Field`, le champ des tangentes apparait et la commande correspondante s'inscrit au niveau suivant de l'écran de géométrie, par exemple :

```
plotfield(sin(t*y),[t=-5.7..5.7,y=-5.7..5.7],normalize,
                    xstep=0.7,ystep=0.7)
```

Si on a coché `Field` et `||=1`, et que $y' = \sin(t * y)$.
Ensuite, il suffit de cliquer en différents points de lécran de géométrie pour avoir les tracés des solutions passant par ces points et les commandes correspondantes stockées dans une variable, par exemple :

```
A:=plotode(sin(t*y),[t,y],point(-2.863,1.327),plan)
```

Pour terminer, il suffit de changer de mode, par exemple passer en mode `Repere`.
Il faut noter que le mode `plotode` n'est pas accessible directement : on doit réouvrir la boite de dialogue avec le menu `Graphe->Slopefield/Ode(2d)`.
Si on trouve que le champ des tangentes est génant, on peut le supprimer facilement en supprimant le niveau correspondant à sa commande.

3.24 Faire une animation en 2-d, 3-d ou "4D"

`Xcas` permet d'animer des graphes en 2-d, 3-d ou "4D" en calculant une fois pour toute une suite d'objets graphiques et en affichant chaque objet de la sequence en boucle.

- Le temps d'affichage d'un objet peut se régler avec `animate` dans `cfg` (plus le nombre est petit et plus le temps d'affichage est petit i.e la vitesse d'animation est grande).
- Si on met `animate` à `0`, à chaque clic de la souris dans l'écran graphique, on a un affichage.
- Le nombre d'images peut se régler avec un argument de la forme `frames=` ou `trames=`.
- On peut interrompre ou relancer l'affichage en boucle en cliquant sur le bouton ▶| (à droite de `M`).

3.24.1 Animation d'un graphe 2-d :`animate`

`animate` permet de créer une animation en boucle d'un graphe de fonctions dépendant d'un paramètre. Le paramètre doit être indiqué en 3ème argument de `animate`, le nombre d'images en 4ème argument sous la forme `frames=` ou `trames=`, les autres arguments sont identiques à ceux de la commande `plot`, section 3.8, p. 120.
On tape :

```
animate(sin(a*x),x=-pi..pi,a=-2..2,trames=10,couleur=rouge)
```

On obtient :

une à une la représentation graphique de `y=sin(`a`x)`
pour 11 valeurs de a entre -2 et 2

3.24.2 Animation d'un graphe 3-d :`animate3d`

`animate3d` permet de créer une animation en boucle d'un graphe 3-d de fonctions dépendant d'un paramètre. Le paramètre doit être indiqué en 3ème argument de `animate3d`, le nombre d'images en 4ème argument sous la forme `frames=` ou `trames=`, les autres arguments sont identiques à ceux de la commande `plotfunc`, voir section 3.7.2, p. 118.
On tape :

```
animate3d(x^2+a*y^2,[x=-2..2,y=-2..2],a=-2..2,
        frames=10,affichage=rouge+rempli)
```

On obtient :

une à une la représentation graphique de `z=x^2+`a`*y^2`
pour 11 valeurs de a entre -2 et 2

3.24.3 Animation d'une séquence d'objets graphiques :`animation`

`animation` permet de dessiner chaque objet d'une suite d'objets graphiques avec un temps d'affichage donné. En général les objets de la suite dépendent d'un paramètre, il faut alors créer une suite en faisant varier ce paramètre.
`animation` a comme paramètre une séquence d'objets graphiques.
Remarque
Si on veut que dans l'animation plusieurs objets graphiques soient affichès en même temps, il faut mettre ces objets dans une liste, par exemple :
On tape :

```
plotfunc(x^2);animation([point(1),segment(1,1+i),
        point(1+i)],droite(y=2*x-1))
```

On obtient :

le graphe de $y = x^2$ puis une animation de 2 objets (le premier objet est 2 points et un segment et le deuxième une droite)

Attention
Pour définir la séquence d'objets graphiques avec `seq` on peut quoter ou ne pas quoter la commande dessinant l'objet graphique.
On peut aussi spécifier le pas de la séquence si on utilise 5 arguments pour `seq` : l'objet graphique, le nom du paramètre, sa valeur minimum, sa valeur maximum et le pas.
On tape :

```
animation(seq(plotfunc(cos(a*x),x),a,0,10))
```

On obtient :

La suite des différentes représentations de la courbe définies par $y = \cos(ax)$, pour $a = 0, 1, 2..10$

On tape :

```
animation(seq(plotfunc(cos(a*x),x),a,0,10,0.5))
```

ou

```
animation(seq(plotfunc(cos(a*x),x),a=0..10,0.5))
```

On obtient :

La suite des différentes représentations de la courbe définies par $y = \cos(ax)$, pour $a = 0, 0.5, 1, 1.5..10$

On tape :

```
animation(seq(plotfunc([cos(a*x),sin(a*x)],x=0..2*pi/a),
                    a,1,10))
```

On obtient :

La suite des différentes représentations des 2 courbes définies par $y = \cos(ax)$ et $y = \sin(ax)$, pour $a = 1..10$ et pour $x = 0..2\pi/a$

On tape :

```
animation(seq(plotparam([cos(a*t),sin(a*t)],
t=0..2*pi),a,1,10))
```

On obtient :

La suite des différentes représentations des courbes définies paramétriquement par $x = \cos(at)$ et $y = \sin(at)$, pour $a = 1..10$ et pour $t = 0..2\pi$

On tape :

```
animation(seq(plotparam([sin(t),sin(a*t)],
t,0,2*pi,tstep=0.01),a,1,10))
```

On obtient :

La suite des différentes représentations des courbes paramétrées définies par $x = \sin(t), y = \sin(at)$, pour $a = 0..10$ et $t = 0..2\pi$

On tape :

```
animation(seq(plotpolar(1-a*0.01*t^2,
t,0,5*pi,tstep=0.01),a,1,10))
```

On obtient :

La suite des différentes représentations des courbes polaires définies par $\rho = 1 - a * 0.01 * t^2$, pour $a = 0..10$ et $t = 0..5\pi$

On tape :

```
plotfield(sin(x*y),[x,y]);
animation(seq(plotode(sin(x*y),[x,y],[0,a]),a,-4,4,0.5))
```

On obtient :

Le champ des tangentes de y'=sin(xy) et la suite des différentes courbes intégrales passant par le point $(0;a)$ pour a=-4,-3.5...3.5,4

On tape :

```
animation(seq(affichage(carre(0,1+i*a),rempli),a,-5,5))
```

On obtient :

La suite des différents carrés définis par les points 0 et 1+i*a pour $a = -5..5$

On tape :

```
animation(seq(droite([0,0,0],[1,1,a]),a,-5,5))
```

On obtient :

La suite des différentes droites définies par les points [0,0,0] et [1,1,a] pour $a = -5..5$

On tape :

```
animation(seq(plotfunc(x^2-y^a,[x,y]), a=1..3))
```

On obtient :

La suite des différentes représentations 3-d des surfaces définies par $x^2 - y^a$, pour $a = 1..3$ avec les couleurs de l'arc en ciel

On tape :

```
animation(seq(plotfunc((x+i*y)^a,[x,y],
      affichage=rempli),a=1..10))
```

On obtient :

La suite des différentes représentations "4D" des surfaces définies par $(x + i * y)^a$, pour $a = 0..10$ avec les couleurs de l'arc en ciel

Remarque 0n peut construire la séquence avec un programme, par exemple on veut dessiner les segments de longueur 1, $\sqrt{2}...\sqrt{20}$ construit avec un triangle rectangle de côtés 1 et le segment précédent.
Voici ce programme (bien mettre `c:=evalf(..)` pour que les calculs soient approchés sinon le temps de calcul est trop long) :

```
essai(n):={
local a,b,c,j,aa,bb,L;
a:=1;
b:=1;
L:=[point(1)];
for(j:=1;j<=n;j++){
L:=append(L,point(a+i*b));
c:=evalf(sqrt(a^2+b^2));
aa:=a;
bb:=b;
a:=aa-bb/c;
b:=bb+aa/c;
}
L;
}
```

Puis on tape :

```
animation(essai(20))
```

On voit, en boucle, chaque point, l'un après l'autre, avec un temps d'affichage plus ou moins grand selon la valeur de `animate` de `cfg`.
Ou on tape :

```
L:=essai(20); s:=segment(0,L[k])$(k=0..20)
```

On voit les 21 segments.
Puis on tape :

```
animation(s)
```

On voit, en boucle, chaque segment, l'un apres l'autre avec un temps d'affichage plus ou moins grand selon la valeur de `animate` de `cfg`.

Chapitre 4

Calcul numérique

4.1 Codage des réels et des décimaux

Voici comment sont codées les nombres réels lorsque le nombre de chiffres significatifs demandés est inférieur ou égal à 16 (par exemple `Digits:=15`).
On écrit d, un nombre réel ou décimal, sous la forme :
$d = 2^{\alpha}(1+m)$ avec $0 < m < 1$ et $-2^{10} < \alpha \geq 2^{10}$.
On utilse 64 bits pour représenter ce nombre :

– le premier bit pour le signe de d (0 pour '+' et 1 pour '-'),
– les 11 bits suivant sont pour codés l'exposant (on code $\alpha + 2^{10} - 1$),
– les 52 derniers sont pour codés la mantisse m.

Codage de 2^{α} :
$\alpha = 0$ est codé 011 1111 1111
$\alpha = 1$ est codé 100 0000 0000
$\alpha = 4$ est codé 100 0000 0011
$\alpha = 5$ est codé 100 0000 0100
$\alpha = -1$ est codé 011 1111 1110
$\alpha = -4$ est codé 011 1111 1011
$\alpha = -5$ est codé 011 1111 1010
$\alpha = 2^{10}$ est codé 111 1111 1111
$\alpha = 2^{-10} - 1$ est codé 000 0000 0000.
Remarque
$2^{-52} = 0.2220446049250313e - 15$

4.1.1 Un exemple : codage de 3.1 et de 3

– codage de 3.1 :
On a :
$3.1 = 2*(1+1/2+1/2^5+1/2^6+1/2^9+1/2^{10}+....) = 2*(1+1/2+\sum_{k=1}^{\infty} 1/2^{4*k+1}+1/2^{4*k+2})$
donc $\alpha = 1$ et $m = 1/2+\sum_{k=1}^{\infty} 1/2^{4*k+1}+1/2^{4*k+2}$
On obtient le codage de 3.1 :
40 (01000000), 8 (00001000), cc (11001100), cc (11001100),
cc (11001100), cc (11001100), cc (11001100), cd (11001101),
le dernier octet est 1101 car il y a eu un arrondi du dernier bit a 1, car le

chiffre suivant etait 1.

- codage de 3 :
  On a :
  $3 = 2 * (1 + 1/2)$
  On obtient le codage de 3 :
  40 (01000000), 8 (00001000), 0 (00000000), 0 (00000000), 0 (00000000), 0 (00000000), 0 (00000000), 0 (00000000).

4.1.2 Différence de codage entre (3.1-3) et 0.1

- codage de 0.1 :
  On a :
  $0.1 = 2^{-4}*(1+1/2+1/2^4+1/2^5+1/2^8+1/2^9+...) = 2^{-4}*\sum_{k=0}^{\infty} 1/2^{4*k}+1/2^{4*k+1}$
  donc $\alpha = 1$ et $m = 1/2 + \sum_{k=1}^{\infty} 1/2^{4*k} + 1/2^{4*k+1}$
  On obtient le codage de 0.1 :
  the code of3f (00111111), b9 (10111001), 99 (10011001), 99 (10011001), 99 (10011001), 99 (10011001), 99 (10011001), 9a (10011010),
  le dernier octet est 1010 car il y a eu un arrondi les 2 derniers bits 01 sont devenus 10 car le chiffre suivant etait 1.
- codage de a :=3.1-3 :
  L'exposant sera donc $\alpha = -4$ (qui correspond à $2 * 2^{-5}$) et les bits qui correspondent à la mantisse vont débuter à $1/2 = 2 * 2^{-6}$: ainsi les nombres de la mantisse subissent un décalage vers la gauche de 5 places et on obtient :
  3f (00111111), b9 (10111001), 99 (10011001), 99 (10011001),
  99 (10011001), 99 (10011001), 99 (10011001), 9a (10100000),
  On voit alors que :
  $a > 0.1$ et que $a - 0.1 = 1/2^{50} + 1/2^{51}$ (car 100000-11010=110)

`Remarque`
Ce qui précéde permet d'expliquer pourquoi lorsque `Digits:=15` :
`floor(1/(3.1-3))` renvoie `9` et non `10`.

4.2 Évaluation des réels : `evalf approx` et `Digits`

On peut évaluer une expression numérique grâce à la commande `evalf` ou `approx`. En mettant un deuxième argument `n` à `evalf` (ou `approx`), on peut spécifier le nombre de chiffres significatifs de l'approximation.
Mettre ce deuxième argument a l'avantage de ne pas modifier la valeur de `Digits` (i.e. la case `Chiffres` de la configuration du CAS).
Attention l'affichage tiendra compte de la valeur `p` de `Digits` si `p<15` et si le deuxième argument `n` de `evalf` est supérieur à `p` mais les calculs seront faits avec `n` chiffres significatifs.
Exemple 1 Avec `Digits:=12`.
On tape : `a:=1234+1/7`
On obtient : `8639/7`
On tape : `b:=evalf(a,9)`
On obtient : `1234.14286` (9 chiffres significatifs avec un arrondi)
On tape : `c:=evalf(a,16)`

On obtient : `0.1234142857142857e4` (16 chiffres significatifs)
On tape : `d:=evalf(a,21)`
On obtient : `0.123414285714285714286e4` (21 chiffres significatifs)
On tape : `7*a,7*b,7*c,7*d`
On obtient :
`8639,8639.00002,0.8639000000000001e4,0.863900000000000000000e4`
Mais avec `Digits:=7`.
On tape : `a:=1234+1/7`
On obtient : `8639/7`
On tape : `b:=evalf(a,9)`
On obtient : `1234.143` (à l'affichage juste 7 chiffres significatifs)
On tape : `b-1234`
On obtient : `0.14286` (ce qui prouve que `b` vaut 1234.14286)
On tape : `b-1234.14286`
On obtient : `0.0` (ce qui prouve encore que `b` vaut 1234.14286)
On tape : `c:=evalf(a,16)`
On obtient : `0.1234142857142857e4`
On tape : `d:=evalf(a,21)`
On obtient : `0.123414285714285714286e4`
Exemple 2 Avec `Digits:=7` ou si dans la configuration du `cas` (menu `Cfg`) on a choisit `Chiffres=7`.
On tape :

```
evalf(sqrt(2))
```

On obtient :

```
1.414214
```

On tape :

```
evalf(sqrt(2),3)
```

On obtient :

```
1.41
```

On tape :

```
evalf(sqrt(2),3)-1.414214
```

On obtient :

```
-0.000214
```

ce qui montre que `evalf(sqrt(2),3)` est le nombre `1.414`
On tape :

```
evalf(sqrt(2),10)
```

On obtient :

```
1.414214
```

On obtient toujours un affichage avec 7 chiffres significatifs si `n` est supérieur ou égal à 7 :

`1.414214`

On tape :

`evalf(sqrt(2))-1.414214`

On obtient toujours lorsque `Digits:=7`) :

`-4.376269e-07`

ce qui montre que `Xcas` fait les calculs avec 14 chiffres significatifs.
Par contre, dés que le 2-ième argument `n` de `evalf` est strictement supérieur à 14 l'affichage se fait avec `n` chiffres significatifs.
On tape :

`evalf(sqrt(2),15),evalf(sqrt(2),16)`

On obtient :

`1.41421356237310,1.414213562373095`

On tape :

`evalf(sqrt(2),20)`

On obtient :

`1.4142135623730950488`

et cela n'a pas modifié la configuration du CAS.
On peut changer le nombre de chiffres significatifs avec la variable `DIGITS` ou `Digits`.
On tape :

`DIGITS:=20`

Cela a pour effet de changer `Configuration du CAS` et de mettre 20 dans la case `Chiffres`.

`evalf(sqrt(2))`

On obtient 20 chiffres significatifs :

`1.4142135623730950488`

Notation : Le nombre réel 10^{-4} est un nombre exact alors que $1e{-}4$ est un nombre approché.
On tape :

`evalf(10^-5)`

On obtient :

`1e-05`

On tape :

`evalf(10^15)`

On obtient :

`1e+15`

On tape :

`evalf(sqrt(2))*10^-5`

On obtient si `Digits:=20` :

`0.14142135623730950488e-4`

Remarques On tape :

`DIGITS:=20`

`a:=evalf(sqrt(2))`

On obtient :

`1.4142135623730950488`

On tape :

`evalf(a,10)`

On obtient :

`1.4142135624`

On tape :

`evalf(sqrt(2),10)`

On obtient :

`1.414213562373`

On tape :

`DIGITS:=10`

`b:=evalf(sqrt(2))`

On obtient :

`1.414213562`

On tape :

`evalf(b,10)`

On obtient :

`1.414213562`

On tape :

```
evalf(sqrt(2),10)
```

On obtient :

```
1.414213562
```

Attention
Si vous définissez une fonction $F(a)$ qui renvoie une séquence formée par un nombre fractionnaire p/q et un entier n alors `evalf(F(a))` renvéra une approximation de p/q avec n chiffres significatifs ! Il faut donc écrire `evalf([F(a)])` pour avoir une liste constituée d'une approximation de p/q et de n.

4.3 Quelques fonctions

4.3.1 Solution approchée d'une équation : `newton`

`newton` a comme arguments : une expression `Xpr`, le nom de la variable de cette expression (par défaut `x`), et trois valeurs `a` (par défaut `a=0`), `eps` (par défaut `eps=1e-8`) and `nbiter` (par défaut `nbiter=12`).
`newton(Xpr,x,a,eps,nbiter)` calcule de façon approchée par la méthode de Newton, une solution `x` proche de `a` de l'équation `Xpr=0`. Le nombre maximum d'itérations est `nbiter` et la précision demandée est `eps`.
On tape :

```
newton(x^2-2,x,1)
```

On obtient :

```
1.41421356237
```

On tape :

```
newton(x^2-2,x,-1)
```

On obtient :

```
-1.41421356237
```

On tape :

```
newton(cos(x)-x,x,0)
```

On obtient :

```
0.739085133215
```

4.3.2 Calcul approché du nombre dérivé : `nDeriv`

`nDeriv` a comme arguments : une expression `Xpr`, le nom de la variable de cette expression (par défaut `x`), and `h` (par défaut `h=0.001`).
`nDeriv(f(x),x,h)` calcule de façon approchée la valeur de la dérivée de l'expression `f(x)` au point `x` et renvoie :
`(f(x+h)-f(x+h))/2*h`.
On tape :

```
nDeriv(x^ 2,x)
```

On obtient :

```
((x+0.001)^2-(x+-0.001)^2)*500.0
```

On tape :

```
subst(nDeriv(x^ 2,x),x=1)
```

On obtient :

```
2
```

On tape :

```
nDeriv(exp(x^ 2),x,0.00001)
```

On obtient :

```
(exp((x+1e-05)^2)-exp((x+-1e-05)^2))*50000
```

On tape :

```
subst(exp(nDeriv(x^ 2),x,0.00001),x=1)
```

On obtient :

```
5.43656365783
```

On a `2.0*e=5.43656365692`

4.3.3 Calcul approché d'intègrales avec la méthode de Romberg : `romberg` `nInt`

`romberg` ou `nInt` a comme arguments : une expression `Xpr`, le nom de la variable de cette expression (par défaut `x`), et deux valeurs `a,b`.
`romberg(Xpr,x,a,b)` ou `nInt(Xpr,x,a,b)` calcule de façon approchée l'intégrale $\int_a^b Xpr\ dx$ par la méthode de Romberg.
On tape :

```
romberg(exp(x^2),x,0,1)
```

On obtient :

```
1.46265174591
```

4.3.4 Calcul approché d'intègrales par une quadrature de Gauss adaptative à 15 points : `gaussquad`

`gaussquad` a comme arguments : une expression `Xpr`, le nom de la variable de cette expression (par défaut `x`), et deux valeurs `a,b`.
`gaussquad(Xpr,x,a,b)` calcule de façon approchée l'intégrale $\int_a^b Xpr\ dx$ par une méthode adaptative (Ernst Hairer) utilisant des quadratures de Gauss à 15 points (méthode d'ordre 30). On commence par faire la quadrature à 15 points sur l'intervalle $[a, b]$ tout entier et on estime l'erreur par une méthode emboitée (à 14 et 6 points choisis parmi les 15 points), si l'erreur relative [1] est inférieure à la tolérance, l'algorithme s'arrête. Sinon on divise $[a, b]$ en deux, et on calcule la quadrature sur chaque morceau, on estime l'erreur, si on dépasse la tolérance on divise en deux l'intervalle amenant l'erreur la plus grande, et ainsi de suite, on divise toujours en deux l'intervalle amenant l'erreur la plus grande, jusqu'à ce que l'erreur relative soit plus petite que la tolérance ou que le nombre de subdivisions dépasse le nombre maximal de subdivisions autorisées.
On tape :

```
gaussquad(exp(x^2),x,0,1)
```

On obtient :

```
1.46265174591
```

On tape :

```
gaussquad(exp(-x^2),x,-1,1)
```

On obtient :

```
1.49364826562
```

4.3.5 Solution approchée de y'=f(t,y) : `odesolve`

Soit f une fonction de $\mathbb{R}^2$ dans $\mathbb{R}$.
`odesolve` renvoie la valeur approchée $y(t1)$ de la solution de léquation differentielle $y' = f(t, y)$ lorsque $y(t_0) = y_0$.
`odesolve` a comme paramètres :

- `odesolve(f(t,y),[t,y],[t0,y0],t1)` ou
  `odesolve(f(t,y),t=t0..t1,y,y0)` ou
  `odesolve(t0..t1,f,y0)` ou
  `odesolve(t0..t1,(t,y)->f(t,y),y0)`
  renvoie la valeur approchée de $y(t1)$ lorsque $y(t)$ est la solution de $y'(t) = f(t, y(t))$ qui vérifie $y(t0) = y0$.
- On peut ajouter un paramètre optionnel pour indiquer la discrétisation en temps souhaitée (`tstep=valeur`). Cette valeur n'est pas forcément respectée par le solver. Par défaut `tstep=0.3`)
- On peut indiquer en paramètre optionnel `curve` pour obtenir la liste des $[t, [y(t)]]$ calculés au lieu de la seule valeur de $y(t1)$.

1. on calcule l'erreur relative en divisant l'erreur estimée par l'intégrale approchée de la valeur absolue de `Xpr`

On tape :

```
odesolve(sin(t*y),[t,y],[0,1],2)
```

ou :

```
odesolve(sin(t*y),t=0..2,y,1)
```

ou :

```
odesolve(0..2,(t,y)->sin(t*y),1)
```

ou encore on définit la fonction :

```
f(t,y):=sin(t*y)
```

et on tape :

```
odesolve(0..2,f,1)
```

On obtient :

```
[1.82241255675]
```

puis on tape :

```
odesolve(0..2,f,1,tstep=0.3)
```

On obtient :

```
[1.82241255675]
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.5)
```

On obtient :

```
[1.82241255675]
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.5,curve)
```

On obtient :

```
[[0.0,[1.0]],[0.3906,[1.07811817892]],[0.760963058921,[1.30972370161]],[1.07
```

On tape :

```
odesolve(sin(t*y),t=0..2,y,1,curve)
```

Ou on tape :

```
odesolve(sin(t*y),t=0..2,y,1,tstep=0.3,curve)
```

On obtient :

```
[[0.0,[1.0]],[0.3781,[1.07309655677]],[0.6781,[1.24392692452]],[0.9781,[1.512
```

4.3.6 Solution approchée du système v'=f(t,v) : `odesolve`

On cherche une valeur approchée du vecteur $v(t)$ de $\mathbb{R}^n$ qui est la solution du système : $v'(t) = f(t, v(t))$.
L'expression $f(t, v(t))$ peut être donnée :
soit par un vecteur ayant comme coordonnées des expressions de t et des coordonnées x_i de v,
soit par une fonction f de $\mathbb{R} \times \mathbb{R}^n$ dans $\mathbb{R}^n$.

- L'expression $f(t, v(t))$ est donnée par un vecteur d'expressions Si v est un vecteur de coordonnées $[x_1, .., x_n]$ dépendant de t et si $f(t, v(t))$ est donné par un vecteur ayant comme coordonnées les expressions dépendant de t et des x_i : $[Xpr_1, ..., Xpr_n]$, si la valeur initiale de v en $t = t_0$ est le vecteur de coordonnées $[x0_1, ..., x0_n]$ alors l'instruction
  `odesolve(`$[Xpr_1, ..., Xpr_n], t = t_0..t_1, [x_1, ..., x_n], [x0_1, ..., x0_n]$`)`
  renverra une valeur approchée de v au temps $t = t1$. Le paramètre optionnel `curve` permet d'avoir les valeurs intermédiaires sous forme d'une liste de couples $[t, v(t)]$ calculés.
  Pour résoudre le système :
  $x'(t) = -y(t)$
  $y'(t) = x(t)$
  On tape :
  `odesolve([-y,x],t=0..pi,[x,y],[0,1])`
  On obtient :
  `[-1.79045146764e-15,-1]`
- L'expression $f(t, v(t))$ est donnée par f une fonction de $\mathbb{R} \times \mathbb{R}^n$ dans $\mathbb{R}^n$.
  `odesolve(t0..t1,(t,v)->f(t,v),v0)` ou
  `odesolve(t0..t1,f,v0)`
  calcule de façon approchée la valeur de $v(t1)$ lorsque le vecteur de coordonnées $v(t)$ de $\mathbb{R}^n$ est la solution de $v'(t) = f(t, v(t))$ qui vérifie $v(t0) = v0$.
  Le paramètre optionnel `curve` permet d'avoir les valeurs intermédiaires sous forme d'une liste de couples $[t, v(t)]$ calculés.
  Pour résoudre le système :
  $x'(t) = -y(t)$
  $y'(t) = x(t)$
  On tape :
  `odesolve(0..pi,(t,v)->[-v[1],v[0]],[0,1])`
  Ou on définit la fonction :
  `f(t,v):=[-v[1],v[0]]`
  puis on tape :
  `odesolve(0..pi,f,[0,1])`
  On obtient :
  `[-1.79045146764e-15,-1]`
  On définit la fonction :
  `f(t,v):=[-v[1],v[0]]`
  puis on tape :

  `odesolve(0..pi/4,f,[0,1],curve)`
  On obtient :

```
[[0.1781,[-0.177159948386,0.984182072936]],
 [0.3781,[-0.369155338156,0.929367707805]],
 [0.5781,[-0.54643366953,0.837502384954]],
 [0.7781,[-0.701927414872,0.712248484906]]]
```

4.4 Résolution numérique d'équations avec nSolve

nSolve permet de résoudre numériquement des équations non polynomiales : $f(x) = 0$ pour $x \in]a, b[$ (nSolve est une commande compatible ti).
Les paramètres de nSsolve sont f(x)=0, x, ou x=x0 où x0 est un point de $]a, b[$.
On tape :

```
nSolve((cos(x))=x,x)
```

On obtient soit :

```
0.739085133215
```

soit une solution complexe :

```
-9.10998745394-2.95017086176*i
```

En effet, si on ne précise pas la valeur qui démarre l'itération, Xcas démarre l'itération avec une valeur aléatoire réelle ou complexe.
On vérifie :
cos(-9.10998745394-2.95017086176*i)=-9.10998745394-2.95017086176*i
On tape :

```
nSolve((cos(x))=x,x=0)
```

On obtient :

```
0.739085133215
```

On tape :

```
nSolve((cos(x))=x,x=-9-3*i))
```

On obtient :

```
-9.10998745394-2.9501708617*i
```

4.5 Résolution d'équations avec fsolve

fsolve permet de résoudre numériquement des équations non polynomiales : $f(x) = 0$ pour $x \in]a, b[$.
fsolve a comme arguments $f(x) = 0$ et $x = a..b$ ou $f(x) = 0$, x et $a..b$.
fsolve donnera aussi les racines numériques complexes si dans la configuration du CAS on a coché Complexe. Si Complexe est décoché il faut utiliser cfsolve pour avoir les racines numériques complexes.
On tape :

```
fsolve(sin(x)=0,x=0..10)
```

Ou on tape :

```
fsolve(sin(x)=0,x,0..10)
```

On obtient :

```
[0.0,3.14159265359,6.28318530718,9.42477796077]
```

On peut rajouter en dernier argument la valeur de l'échantillonage en spécifiant la valeur de `xstep` ou la valeur de `nstep` (nombre de découpages de l'intervalle $]a, b[$).
On tape :

```
fsolve(sin(x)=0,x=0..10,xstep=1 )
```

On obtient :

```
[0.0,3.14159265359,6.28318530718,9.42477796077]
```

On tape :

```
fsolve(sin(x)=0,x=0..10,nstep=10)
```

On obtient :

```
[0.0,3.14159265359,6.28318530718,9.42477796077]
```

On peut utiliser différents algorithmes pour résoudre numériquement $f(x) = 0$ pour $x \in]a, b[$.
Si on veut indiquer la mèthode, les paramètres de `fsolve` sont `f(x)=0`, `x`, `a..b` ou selon les méthodes un point `x0` de $]a, b[$ et le nom de la méthode utilisée.
Les différentes méthodes sont détaillées ci dessous.

4.5.1 `fsolve` avec l'option `bisection_solver`

Cet algorithme de dichotomie est le plus simple mais aussi le plus lent. Il permet d'encadrer le zéro d'une fonction sur un intervalle. À chaque itération, on coupe l'intervalle en deux, on calcule la valeur au point milieu et, le signe de la fonction en ce point nous dit sur quel morceau de l'intervalle on doit recommencer l'iteration.
On tape :

```
fsolve((cos(x))=x,x,-1..1,bisection_solver)
```

Ou on tape :

```
fsolve((cos(x))=x,x=-1..1,bisection_solver)
```

On obtient :

```
[0.739085078239,0.739085137844]
```

On tape :

```
fsolve((cos(x))=x,x,0,bisection_solver)
```

On obtient :

```
Bad Argument Type
```

4.5.2 `fsolve` avec l'option `brent_solver`

La méthode de Brent combine l'interpolation de f et la dichotomie. Cette méthode est rapide.
On tape :

```
fsolve((cos(x))=x,x,-1..1,brent_solver)
```

On obtient :

```
[0.73908513321 5,0.739085133215]
```

On tape :

```
fsolve((cos(x))=x,x,0,brent_solver)
```

On obtient :

```
Bad Argument Type
```

4.5.3 `fsolve` avec l'option `falsepos_solver`

L'algorithme de "fausse position" est itératif et est basé sur l'interpolation linéaire : on calcule la valeur de f au point d'intersection de la droite passant par les points d'affixe $a + i * f(a)$ et $b + i * f(b)$ avec l'axe des x. Cette valeur permet de savoir sur quelle partie de l'intervalle se trouve la racine, on peut ainsi recommencer l'iteration.
La convergence est linéaire mais est plus rapide que la dichotomie (bisection).
On tape :

```
fsolve((cos(x))=x,x,-1..1,falsepos_solver)
```

On obtient :

```
[0.739085133215,0.739085133215]
```

4.5.4 `fsolve` avec l'option `newton_solver`

La méthode de Newton est la méthode standard. L'algorithme démarre par une valeur initiale x_0, on cherche l'intersection x_1 de la tangente en x_0 au graphe de f, avec l'axe des x, puis à chaque itération on recommence en prenant x_1 comme valeur x_0 :
La suite des x_i est donc définie par :
$x_0 = x_0$ et $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
La méthode de Newton quand elle converge, converge de façon quadratique pour les racines simples.
On tape :

```
fsolve(cos(x)=x,x,0,newton_solver)
```

On obtient :

```
0.739085133215
```

4.5.5 `fsolve` avec l'option `secant_solver`

La méthode de la sécante est une méthode simplifiée de la méthode de Newton. Le calcul de $f'(x_n)$ se fait de façon approchée : cela peut être utile quand le calcul de la derivée est couteux.
On a :
$x_{i+1} = x_i - \frac{f(x_i)}{f'_{est}}$ et $f'_{est} = \frac{f(x_i)-f(x_{i-1})}{(x_i-x_{i-1})}$
Pour le calcul de x_1 on utilise la méthode de Newton.
La convergence pour les racines simples est d'ordre $(1+\sqrt{5})/2 = 1.62$.
On tape :

```
fsolve((cos(x))=x,x,-1..1,secant_solver)
```

On obtient :

```
[0.739085078239,0.739085137844]
```

On tape :

```
fsolve((cos(x))=x,x,0,secant_solver)
```

On obtient :

```
0.739085133215
```

4.5.6 `fsolve` avec l'option `steffenson_solver`

La méthode de Steffenson est la plus rapide de toutes les méthodes.
Elle combine la méthode de Newton avec l'accélération du "delta-deux" d'Aitken : avec la méthode de Newton on obtient la suite x_i et l'accélération de convergence produit la suite :
$R_i = x_i - \frac{(x_{i+1}-x_i)^2}{(x_{i+2}-2x_{i+1}+x_i)}$
On tape :

```
fsolve(cos(x)=x,x,0,steffenson_solver)
```

On obtient :

```
0.739085133215
```

On tape :

```
fsolve(cos(x)=x,x,-1..1,steffenson_solver)
```

On obtient :

```
0.739085133215
```

4.6 Résolution des systèmes d'équations avec `fsolve`

On propose six méthodes pour résoudre numériquement des systèmes d'équations de la forme $f(x) = 0$.

Remarque

`fsolve` donnera aussi les racines numériques complexes si dans la configuration du CAS on a coché `Complexe`. Si `Complexe` est décoché il faut utiliser `cfsolve` pour avoir les racines numériques complexes.
Trois méthodes utilisent la matrice jacobienne $f'(x)$ et leurs noms se terminent par `j_solver`.
Les trois autres méthodes utilisent des méthodes d'approximation de $f'(x)$ et utilisent uniquement f.
Les six méthodes utilisent une itération de type Newton :
$x_{n+1} = x_n - f'(x_n)^{-1} * f(x_n)$.
Les quatre méthodes `hybrid*_solver` utilisent aussi une méthode de descente de gradient lorsque l'itération Newtonienne donne un pas trop grand.
La longueur du pas est calculé sans facteur d'échelle pour `hybrid_solver` et `hybridj_solver` ou avec facteur d'echelle (calculé à partir de $f'(x_n)$) pour `hybrids_solver` et `hybridsj_solver`

4.6.1 `fsolve` avec l'option `dnewton_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[2,2],dnewton_solver)
```

On obtient :

```
[1.0,1.0]
```

4.6.2 `fsolve` avec l'option `hybrid_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[2,2],
cos(x)=x,x,0,hybrid_solver)
```

On obtient :

```
[1.0,1.0]
```

4.6.3 `fsolve` avec l'option `hybrids_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[2,2],hybrids_solver)
```

On obtient :

```
[1.0,1.0]
```

4.6.4 `fsolve` avec l'option `newtonj_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[0,0],newtonj_solver)
```

On obtient :

```
[1.0,1.0]
```

4.6.5 `fsolve` avec l'option `hybridj_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[2,2],hybridj_solver)
```

On obtient :

```
[1.0,1.0]
```

4.6.6 `fsolve` avec l'option `hybridsj_solver`

On tape :

```
fsolve([x^2+y-2,x+y^2-2],[x,y],[2,2],hybridsj_solver)
```

On obtient :

```
[1.0,1.0]
```

4.7 Résolution sur $\mathbb{C}$ d'équations ou de systèmes `cfsolve`

`cfsolve` effectue la résolution numérique sur $\mathbb{C}$ d'une équation ou d'un système, même si `Complexe` est décoché dans la configuration du CAS.
`fsolve` donne aussi les racines numériques complexes d'une équation ou d'un système si dans la configuration du CAS on a coché `Complexe`.
On tape :

```
cfsolve(sin(x)=2)
```

On obtient :

```
[1.57079632679-1.31695789692*i,1.57079632679+1.31695789692*i]
```

On tape :

```
cfsolve(cos(x)=2)
```

On obtient :

```
[1.31695789692*i,-1.31695789692*i]
```

On tape :

```
cfsolve([x^2+y+2,x+y^2+2],[x,y])
```

On obtient :

```
[[0.50000000000014794+1.65831239517770439*i,
0.500000000000000000-1.65831239517771331*i],
[0.50000000000014794-1.65831239517770439*i,
0.500000000000000000+1.65831239517771331*i],
[-0.4999999999999994291-1.32287565553229745*i,-0.49999999999999999999-1.32287565
```

4.8 Racines numériques d'un polynôme : `proot`

`proot` a comme argument un polynôme ou le vecteur de composantes les coefficients d'un polynôme (par ordre décroissant).
`proot` renvoie un vecteur dont les composantes sont les racines numériques non multiples du polynôme.
On peut mettre en option un entier n pour préciser le nombre n de chiffres significatifs de la réponse. Pour chercher les racines numériques de $P(x) = x^3 + 1$, on tape :

```
proot([1,0,0,1])
```

ou on tape :

```
proot(x^3+1)
```

On obtient :

```
[0.5+0.866025403784*i,0.5-0.866025403784*i,-1.0]
```

On tape pour avoir 20 chiffres significatifs :

```
proot([1,0,0,1],20)
```

ou on tape :

```
proot(x^3+1)
```

On obtient :

```
[-1.000000000000000000,
0.500000000000000000-0.866025403784438647*i,
0.500000000000000000+0.866025403784438647*i]
```

On tape pour avoir les racines numériques de $x^2 - 3$:

```
proot([1,0,-3])
```

ou :

```
proot(x^2-3)
```

On obtient :

```
[1.73205080757,-1.73205080757]
```

On tape pour avoir les racines numériques de $P(x) = x^2 - 3$ avec 20 chiffres significatifs :

```
proot([1,0,-3],20)
```

ou on tape :

```
proot(x^2-3,20)
```

On obtient :

```
[-1.732050807568877294,1.732050807568877294]
```

On tape pour avoir les racines numériques de $P(x) = x^10 - 15 * x^8 + 90 * x^6 - 270 * x^4 + 405 * x^2 - 243$:

```
proot([1,0,-15,0,90,0,-270,0,405,0,-243])
```

ou on tape :

```
proot(x^10-15*x^8+90*x^6-270*x^4+405*x^2-243)
```

On obtient :

```
[1.73205080757,-1.73205080757,1.73205080757,-1.73205080757,
1.73205080757,-1.73205080757,1.73205080757,-1.73205080757,
1.73205080757,-1.73205080757]
```

4.9 Factorisation numérique d'une matrice : `cholesky` `qr` `lu` `svd`

Pour avoir les factorisations numériques de :
– Cholesky,
– QR,
– LU,
– svd,
d'une matrice, on se repotera à la section 6.51.

Chapitre 5

Les unités et les constantes physiques

Les constantes physiques (sous-menu `Constante`), les fonctions de conversion (sous-menu `Unit_convert`), les préfixes (sous-menu `Unit_prefix`) et les unités classées par thème, se trouvent dans le menu `Phys`.

5.1 Les unités

5.1.1 La notation des unités

Les noms des unités sont précédés du symbole _ ("underscore"). Par exemple `2_m` for 2 meters.
Vous pouvez mettre un préfixe devant le nom d'une unité qui indique une multiplication par une puissance de 10. Par exemple `k` ou `K` pour kilo (indique une multiplication par 10^3), `D` pour déca (indique une multiplication par 10), `d` pour déci (indique une multiplication par 10^{-1}) etc...
Lorsqu'on combine un nombre réel avec des unités on crée un objet-unité.
On tape :

```
10.5_m
```

On obtient :

```
un objet-unité valant 10.5 mètres
```

On tape :

```
10.5_km
```

On obtient :

```
un objet-unité valant 10.5 kilomètres
```

5.1.2 Les calculs avec des unités

On peut faire les opérations de base (+, -, *, /) avec des objets-unités.
Dans les opérations, on peut utiliser des unités différentes (mais compatibles pour

+ et -) et le résultat sera exprimé selon l'unité correspondante. Pour la multiplication et la division de deux unités différentes `_u1` et `_u2` l'unité résultat s'écrit `_(u1*u2)` ou `_(u1/u2)` (ne pas oublier les parenthèses ! ! !)
On peut aussi élever un objet-unité à une puissance entière : on obtient l'objet-unité correspondant.
Il faut noter que lors d"une addition ou d'une soustraction, le résultat sera exprimé selon l'unité du premier terme de l'opération.
On tape :

```
1_m+100_cm
```

On obtient :

```
2_m
```

On tape :

```
100_cm+1_m
```

On obtient :

```
200_cm
```

On tape :

```
1_m*100_cm
```

On obtient :

```
100_(cm*m)
```

On tape :

```
3_h +10_mn-(1_h+45_mn)
```

On obtient :

```
1.41666666667_h
```

On tape :

```
10_mn+3_h-(1_h+45_mn)
```

On obtient :

```
85.0_mn
```

5.1.3 La conversion d'un objet-unité dans une autre unité : `convert convertir =>`

`convert` permet d'obtenir la conversion d'un objet-unité dans une autre unité qui est le deuxième paramètre.
`=>` est la version infixée de `convert` ou `convertir`.
On tape :

```
convert(2_h+30_mn,_mn)
```

ou bien

```
2_h+30_mn=>_mn
```

On obtient :

```
150_mn
```

On tape :

```
convert(1_m*100_cm,_m^2)
```

ou bien

```
convert(100_(cm*m),_m^2)
```

ou bien

```
100_(cm*m)=>_m^2
```

On obtient :

```
1_m^2
```

On tape :

```
convert(1_h,_s)
```

Ou on tape :

```
1_h=>_s
```

On obtient :

```
3600_s
```

On tape :

```
convert(60_mn,_h)
```

Ou on tape :

```
60_mn=>_h
```

On obtient :

```
1.0_h
```

Remarque
Il faut mettre un espace avant l'unité si le nombre d'unité se trouve dans une variable ou si c'est une constante :
On tape :

```
convert(pi _rad,_deg)
```

Ou on tape :

```
pi _rad=>_deg
```

On obtient :

`180.0_deg`

On tape :

`a:=180`

`convert(a _deg,_rad)`

Ou on tape :

`a _deg=>_rad`

On obtient :

`3.14159265358_rad`

5.1.4 Les changements d'unités en unités MKSA : `mksa`

`mksa` permet d'obtenir la conversion d'un objet-unité en un objet-unité exprimé en unités `MKSA`.
On tape :

`mksa(15_C)`

On obtient :

`15_A*s`

On tape :

`mksa(1_Hz)`

On obtient :

`1_s^(-1)`

5.1.5 Les conversions entre degré Célsius et degré Fahrenheit : `Celsius2Fahrenheit` et `Fahrenheit2Celsius`

`Celsius2Fahrenheit` permet de convertir les degrés Celsius en degrés Fahrenheit.
On tape :

`Celsius2Fahrenheit(a)`

On obtient :

`(a*9)/5+32`

On tape :

`Celsius2Fahrenheit(0)`

On obtient :

```
32
```

`Fahrenheit2Celsius` permet de convertir les degrés Fahrenheit en degrés Celsius.
On tape :

```
Fahrenheit2Celsius(a)
```

On obtient :

```
((a-32)*5)/9
```

On tape :

```
Fahrenheit2Celsius(212))
```

On obtient :

```
100
```

5.1.6 Mise en facteur d'une unité : `ufactor`

`ufactor` permet de factoriser une unité dans un objet-unité : on obtient un objet-unité multiplié par les unités `MKSA` restantes .
On tape :

```
ufactor(3_J,_W)
```

On obtient :

```
3_(W*s)
```

On tape :

```
ufactor(3_W,_J)
```

On obtient :

```
3_(J/s)
```

5.1.7 Simplifier une unité : `usimplify`

`usimplify` permet de simplifier une unité dans un objet-unité.
On tape :

```
usimplify(3_(W*s))
```

On obtient :

```
3_J
```

5.1.8 Les préfixes disponibles pour les noms d'unités

Vous pouvez mettre des préfixes devant les noms d'unités : chaque préfixe correspond au nom de l'unité muliplié par une puissance de 10.
Voici les différents préfixes disponibles :

Préfixe	Nom	(*10^) n	Préfixe	Nom	(*10^) n
Y	yota	24	d	déci	-1
Z	zêta	21	c	cent	-2
E	exa	18	m	mili	-3
P	péta	15	mu	micro	-6
T	téra	12	n	nano	-9
G	giga	9	p	pico	-12
M	méga	6	f	femto	-15
k ou K	kilo	3	a	atto	-18
h ou H	hecto	2	z	zepto	-21
D	déca	1	y	yocto	-24

Remarque
Bien sûr vous ne pouvez pas utiliser le préfixe avec une unité intégrée si la combinaison donne une autre unité intégrée.
Par exemple, `1_a` est un are et `1_Pa` est un pascal et non `10^15_a`.

5.2 Les constantes physiques

5.2.1 La notation des constantes physiques

Les noms des constantes physiques commencent et se terminent par le caractère _ ("underscore"). Il ne faut pas confondre les constantes physiques avec les constantes symboliques, par exemple, e, π sont des constantes symboliques alors que _c_, _NA_ sont des constantes physiques.
On tape :

```
_c_
```

On obtient la vitesse de la lumière dans le vide :

```
299792458_m*s^-1
```

On tape :

```
_NA_
```

On obtient le nombre d'Avogadro :

```
6.0221367e23_gmol^-1
```

5.2.2 Bibliothèque des constantes physiques

Vous trouverez certaines constantes physiques dans le menu `Phys` sous-menu `Constante` ou encore dans l'aide.
Pour avoir la valeur d'une constante il suffit de taper le nom de la constante dans la

ligne de commande de Xcas et de valider (sans oublier de mettre un _ au début et un _ à la fin du nom).
Voici la bibliothèque des constantes :

Nom	Déscription
_NA_	Nombre d'Avogadro
_k_	Constante de Boltzmann
_Vm_	Volume molaire
_R_	Constante universelle des gaz
StdT	Température standard
StdP	Pression standard
sigma	Constante de Stefan-Boltzmann
_c_	Vitesse de la lumière
epsilon0	Permitivité du vide
_mu0_	Perméabilité du vide
_g_	Accélération de la gravité
_G_	Constante gravitationnelle
_h_	Constante de Planck
hbar	Constante de Dirac
_q_	Charge de l'électron
_me_	Masse élémentaire de l'électron
qme	Rapport q/me (charge/masse de l'électron)
_mp_	Masse élémentaire du proton
mpme	Rapport mp/me (masse du proton/masse de l'électron)
alpha	Constante de structure fine
phi	Quantum de flux magnétique
_F_	Constante de Faraday
Rinfinity	Constante de Rydberg
_a0_	Rayon de Bohr
muB	Magnéton de Bohr
muN	Magnéton nucléaire
lambda0	Longueur d'onde du photon (ch/e)
_f0_	Fréquence du photon (e/h)
lambdac	Longueur d'onde du Compton
rad	1 radian
twopi	2*pi radians
angl	Angle de 180 degrés
_c3_	Constante de la loi de répartition de Wien
_kq_	k/q (Boltzmann/charge de l'électron)
epsilon0q	epsilon0/q (permitivité /charge de l'électron)
qepsilon0	q*epsilon0 (charge de l'électron*permitivité)
epsilonsi	Constante diélectrique du silicium
epsilonox	Constante diélectrique du bioxyde de silicium
_I0_	Intensité de référence

Chapitre 6

Les fonctions de calcul formel

6.1 Les constantes symboliques : `e pi infinity inf i euler_gamma`

`e` ou `%e` désigne le nombre $\exp(1)$;
`pi` ou `%pi` désigne le nombre π.
`infinity` désigne ∞.
`+infinity` ou `inf` désigne $+\infty$.
`-infinity` ou `-inf` désigne $-\infty$.
`i` ou `%i` désigne le nombre complexe i.
`euler_gamma` désigne la constante d'Euler. On a :
`euler_gamma=limit(sum(1/k,k,1,n)-ln(n),n,+infinity)=`γ.

6.2 Les booléens

6.2.1 Les valeurs d'un booléen : `true false`

Un booléen a comme valeur `true` ou `false`.
On a les synonymes suivant :
`true` ou `TRUE` ou `1` et,
`false` ou `FALSE` ou `0`.
Les tests ou les conditions sont des fonctions booléennes.

6.2.2 Les tests : `==, !=, >, >=, <, <=`

`==, !=, >, >=, <, <=` sont des opérateurs infixés.
`a==b` teste l'égalité entre `a` et `b` et renvoie `1` si `a` est égal à `b` et `0` sinon.
`a!=b` renvoie `1` si `a` est différent de `b` et `0` sinon.
`a>=b` renvoie `1` si `a` est supérieur ou égal à `b` et `0` sinon.
`a>b` renvoie `1` si `a` est strictement supérieur à `b` et `0` sinon.
`a<=b` renvoie `1` si `a` est inférieur ou égal à `b` et `0` sinon.
`a<b` renvoie `1` si `a` est strictement inférieur à `b` et `0` sinon.
On tape pour définir la fonction booléenne qui vaut `true` sur $]0;+\infty[$ et qui vaut `false` sur $]-\infty;0]$:

```
f(x):=ifte(x>0,true,false)
```

On tape :

```
f(0)==0
```

On obtient :

```
1
```

Attention
`a=b` n'est pas un booléen ! ! ! !
Pour tester l'égalité entre `a` et `b` il faut mettre `a==b`.

6.2.3 Les opérateurs booléens : `or xor and not`

`or` (ou `||`), `xor`, `and` (ou `&&`) sont des opérateurs infixés.
`not` est un opérateur préfixé.
Soient `a` et `b` deux booléens :
`(a or b)` ou `(a || b)` renvoie `0` (ou `false`) si `a` et `b` valent 0 et renvoie `1` (ou `true`) sinon.
`(a xor b)` renvoie `1` si `a` vaut 1 et `b` vaut 0 ou si `a` vaut 0 et `b` vaut 1 et renvoie 0 si `a` et `b` valent 0 ou si `a` et `b` valent 1 (c'est le "ou exclusif").
`(a and b)` ou `(a && b)` renvoie `1` (ou `true`) si `a` et `b` valent 1 et `0` (ou `false`) sinon.
sinon.
`not(a)` renvoie `1` (ou `true`) si `a` vaut 0 (ou `false`), et `0` (ou `false`) si `a` vaut 1 (ou `true`).
On tape :

```
1>=0 or 1<0
```

On obtient :

```
1
```

On tape :

```
1>=0 xor 1>0
```

On obtient :

```
0
```

On tape :

```
1>=0 and 1>0
```

On obtient :

```
1
```

On tape :

```
not(0==0)
```

On obtient :

```
0
```

6.2.4 Transformer une expression en liste : `exp2list`

`exp2list` renvoie la liste `[Xpr0,Xpr1]` lorsque l'argument est `(var=Xpr0) or (var=Xpr1)`.
`exp2list` est utile en mode TI pour utiliser la réponse renvoyée par `solve`.
On tape :

```
exp2list((x=2) or (x=0))
```

On obtient :

```
[2,0]
```

On tape :

```
exp2list((x>0) or (x<2))
```

On obtient :

```
[0,2]
```

En mode TI on tape :

```
exp2list(solve((x-1)*(x-2)))
```

On obtient :

```
[1,2]
```

6.3 Évaluation des booléens : `evalb`

On peut évaluer une expression booléenne grâce à la commande `evalb` cette commande sert surtout pour la compatibilité Maple car en `Xcas`, les booléens sont toujours évalués.
On tape :

```
evalb(sqrt(2)>1.41)
```

Ou on tape :

```
sqrt(2)>1.41
```

On obtient :

```
1
```

On tape :

```
evalb(sqrt(2)>1.42)
```

Ou on tape :

```
sqrt(2)>1.42
```

On obtient :

```
0
```

6.4 Les opérateurs bit à bit

6.4.1 Les opérateurs `bitor, bitxor, bitand`

Les entiers peuvent etre entrés avec la notation 0x... en hexadécimal par exemple 0x1f représente 16+15=31 en décimal. On peut faire afficher les entiers en hexadécimal (bouton de la ligne d'état du cas avec le bouton `Base (Entiers)`).
`bitor` est le `ou` logique inclusif bit à bit.
On tape :

`bitor(0x12,0x38)`

ou on tape :

`bitor(18,56)`

On obtient :

`58`

en effet :
`18` s'écrit `0x12` en base 16 et `0b010010` en base 2,
`56` s'écrit `0x38` en base 16 et `0b111000` en base 2,
`bitor(18,56)` s'écrit `0b111010` en base 2 et donc vaut `58`.

`bitxor` est le `ou` logique exclusif bit à bit.
On tape :

`bitxor(0x12,0x38)`

ou on tape :

`bitxor(18,56)`

On obtient :

`42`

en effet :
`18` s'écrit `0x12` en base 16 et `0b010010` en base 2,
`56` s'écrit `0x38` en base 16 et `0b111000` en base 2,
`bitxor(18,56)` s'écrit `0b101010` en base 2 et donc vaut `42`.

`bitand` est le `et` logique bit à bit.
On tape :

`bitand(0x12,0x38)`

ou on tape :

`bitand(18,56)`

On obtient :

`16`

en effet :
`18` s'écrit `0x12` en base 16 et `0b010010` en base 2,
`56` s'écrit `0x38` en base 16 et `0b111000` en base 2,
`bitand(18,56)` s'écrit `0b010000` en base 2 et donc vaut `16`.

6.4.2 Distance de Hamming bit à bit : `hamdist`

La distance de Hamming bit à bit est la somme des valeurs absolues des différences bit à bit des 2 nombres c'est-à-dire le nombre de bits différents.
On tape :

```
hamdist(0x12,0x38)
```

ou on tape

```
hamdist(18,56)
```

On obtient :

```
3
```

en effet :
`18` s'écrit `0x12` en base 16 et `0b010010` en base 2,
`56` s'écrit `0x38` en base 16 et `0b111000` en base 2,
`hamdist(18,56)` vaut `1+0+1+0+1+0` et donc vaut `3`.

6.5 Les chaînes de caractères

6.5.1 Écriture d'une chaîne ou d'un caractère : `"`

Les chaînes de caractères s'écrivent en utilisant `"` (guillemets) comme délimiteurs.
Un caractère est une chaîne ayant un caractère ; en effet les délimiteurs `'` ou (`quote`) servent à préciser que l'on ne doit pas évaluer la variable mise entre les quotes.
Exemple :
`"a"` est un caractère mais `'a'` ou `quote(a)` désigne la variable `a` non évaluée.
Les caractères d'une chaîne sont repérés par un indice (comme pour les listes).
Pour accéder à un élément d'une chaîne, on tape l'indice de cet élément entre des crochets pour des indices qui commencent à 0
ou bien
on tape son indice entre des doubles crochets pour des indices qui commencent à 1.
Attention ! Pour toutes les autres fonctions de `Xcas` (autres que l'accés à un élément), l'indice du premier élément est 0.
Exemple :
On tape :

```
"bonjour"[2]
```

On obtient :

```
"n"
```

On tape :

```
"bonjour"[[2]]
```

On obtient :

"o"

Remarque
Lorsque l'on tape une chaîne de caractères dans la ligne de commande, cela génère un écho en réponse.
Exemple :
On tape :

"bonjour"

On a `"bonjour"` s'inscrit comme question et on a `bonjour` comme réponse.
On tape :

"bonjour"+", ca va?"

On obtient :

"bonjour, ca va?"

6.5.2 Écriture du caractère "retour à la ligne" : `"\n"`

`"\n"` désigne le caractère retour à la ligne.
On tape :

"bonjour, \n ca va?"

On obtient :

"bonjour,
ca va?"

6.5.3 Longueur d'une chaîne : `size length`

`size` ou `length` renvoie la longueur de la chaîne (ou d'ne liste).
On tape :

size("bonjour")

Ou on tape :

length("bonjour")

On obtient :

6.5.4 Début, milieu et fin d'une chaîne : `head mid tail`

- `head(s)` renvoie le premier caractère de la chaîne `s`.
  On tape :
  `head("bonjour")`
  On obtient :
  `"b"`
- `mid(s,p,q)` renvoie un morceau de longueur `q` de la chaîne `s` commençant au caractère d'indice `p` (attention ! les indices commencent à 0).
  On tape :
  `mid("bonjour",2,4)`
  On obtient :
  `"njou"`
- `tail(s)` renvoie la chaîne `s` privée de son premier caractère.
  On tape :
  `tail("bonjour")`
  On obtient :
  `"onjour"`

6.5.5 Partie droite et gauche d'une chaîne : `droit ou right, gauche ou left`

- `droit(s,n)` ou `right(s,n)` renvoie les n derniers caractères de la chaîne `s`.
  On tape :
  `droit("bonjour",4)`
  Ou on tape :
  `right("bonjour",4)`
  On obtient :
  `"jour"`
- `gauche(s,n)` ou `left(s,n)` renvoie les n premiers caractères de la chaîne `s`.
  On tape :
  `gauche("bonjour",3)`
  Ou on tape :
  `left("bonjour",3)`
  On obtient :
  `"bon"`

6.5.6 Concaténation d'une suite de mots : `cumSum`

`cumSum` permet de faire la concaténation d'une liste de chaînes.
`cumSum` a comme argument une liste de chaînes.
`cumSum` renvoie une liste de chaînes, l'élément d'indice k étant obtenu en concaténant les chaînes la précédant (i.e celles d'indice $0...k-1$) avec la chaîne d'indice k.
Si `l` est une liste de chaînes `cumSum` renvoie la liste `lr` égale à `sum(l[j],j=0..k)$(k=0..size(l)-1)`.
On tape :

`cumSum("Madame ","bon","jour")`

On obtient :

```
"Madame ","Madame bon","Madame bonjour"
```

6.5.7 Le code ASCII d'un caractère : `ord`

`ord` a pour argument une chaîne `s` (resp une liste `l` de chaînes).
`ord` renvoie le code ASCII du premier caractère de `s` chaîne ou la liste des codes ASCII des premiers caractères des éléments de `l`.
On tape :

```
ord("a")
```

On obtient :

```
97
```

On tape :

```
ord("abcd")
```

On obtient :

```
97
```

On tape :

```
ord(["abcd","cde"])
```

On obtient :

```
[97,99]
```

On tape :

```
ord(["a","b","c","d"])
```

On obtient :

```
[97,98,99,100]
```

6.5.8 Le code ASCII d'une chaîne : `asc`

`asc` a pour argument une chaîne `s`.
`asc` renvoie la liste des codes ASCII des caractères de `s`.
On tape :

```
asc("abcd")
```

On obtient :

```
[97,98,99,100]
```

On tape :

```
asc("a")
```

On obtient :

```
[97]
```

6.5.9 La chaîne associée à une suite d'ASCII : `char`

`char` a pour argument une liste `l` des code ASCII.
`char` renvoie la chaîne dont les caractères ont pour codes ASCII les éléments de la liste `l`.
On tape :

```
char([97,98,99,100])
```

On obtient :

```
"abcd"
```

On tape :

```
char(97)
```

On obtient :

```
"a"
```

On tape :

```
char(353)
```

On obtient :

```
"a"
```

En effet 353-256=97.

6.5.10 Repérer un caractère dans une chaîne : `inString`

`inString` a deux paramètres : une chaîne de caractères `S` et un caractère `c`.
`inString` est une fonction qui teste si le caractère `c` est dans la chaîne de caractères `S`.
`inString` renvoie `-1` si `c` n'est pas dans `S` et,
sinon renvoie `"l'indice de sa première apparition"`.
On tape :

```
inString("abcded","d")
```

On obtient :

```
3
```

On tape :

```
inString("abcd","e")
```

On obtient :

```
-1
```

6.5.11 Concaténer des objets en une chaîne : `cat`

`cat` a comme paramètre une séquence d'objets.
`cat` concaténe ces objets en une chaîne de caractères.
On tape :

```
cat("abcd",3,"d")
```

On obtient :

```
"abcd3d"
```

On tape :

```
c:=5
cat("abcd",c,"e")
```

On obtient :

```
"abcd5e"
```

On tape :

```
purge(c)
cat(15,c,3)
```

On obtient :

```
"15c3"
```

6.5.12 Concaténer des objets en une chaîne : +

+ est un opérateur préfixé ou infixé qui a comme paramètre une séquence d'objets comprenant une chaîne de caractères.
+ concaténe ces objets en une chaîne de caractères.
Attention
Quand l' opérateur + est préfixé il faut le quoté c'est à dire l'écrire `'+'`
On tape :

```
'+'("abcd",3,"d")
```

Ou on tape :

```
"abcd"+3+"d"
```

On obtient :

```
"abcd3d"
```

On tape :

```
c:=5
```

On tape ensuite :

```
"abcd"+c+"e"
```

Ou on tape ensuite :

```
'+'("abcd",c,"e")
```

On obtient :

```
"abcd5e"
```

6.5.13 Pour concaténer des nombres et des chaînes : `cat +`

`+` ou `cat` évalue les arguments et les concaténe en une chaîne. cela permet donc de transformer un nombre réel en une chaîne de caractères.
On tape :

```
"="+123
```

Ou on tape :

```
cat("=",123)
```

On obtient :

```
"=123"
```

On tape :

```
a:=123
```

puis,

```
"On a obtenu : "+a)
```

ou

```
cat("On a obtenu : ",a)
```

On obtient :

```
"On a obtenu : 123"
```

6.5.14 Pour transformer un nombre réel ou entier en une chaîne : `string`

`string` évalue son argument et le transforme en une chaîne de caractères.
On tape :

```
string(1.32*10^20)
```

On obtient :

```
"1.23e+20"
```

On tape :

```
a:=1.32*10^4)
```

```
string(a+a)
```

On obtient :

```
"26400"
```

6.5.15 Pour transformer un nombre réel en une chaîne : `format`

`format` évalue son argument et le transforme en une chaîne de caractères du format indiqué.
Le format est indiqué par une lettre et un nombre :
- `f` (pour format flottant) suivi du le nombre de chiffres après la virgule.
  On tape :
  `format(sqrt(2)*10^10,f13)`
  On obtient :
  `"14142135623.7308959960938"`
- `s` (pour format scientifique) suivi du nombre de chiffres significatifs.
  On tape :
  `format(sqrt(2)*10^10,s13)`
  On obtient :
  `"14142135623.73"`
- `e` (pour format ingénieur) suivi du nombre de chiffres significatifs augmenté de 1.
  On tape :
  `format(sqrt(2)*10^10,e13)`
  On obtient :
  `"1.4142135623731e+10"`

6.5.16 Pour transformer une chaîne en un nombre ou en une commande : `expr`

Voici , selon la chaîne, les transformations que fait `expr` :
- `expr` permet de transformer une chaîne de chiffres ou lettres en un nombre entier ayant comme écriture en base 8,ou 10 ou 16 cette chaîne de chiffres. Il faut bien sûr que l'écriture soit valide et représente un nombre entier dans la base donnée : toutefois `expr("019")` renvoie le nombre décimale `19.0` (voir aussi 8.5.5).
  Attention
  Si la chaîne commence par `0x`, `expr` transforme cette chaîne en le nombre entier écrit en base 16, sinon,
  si la chaîne commence par 0, `expr` transforme cette chaîne en le nombre entier écrit en base 8 et sinon
  `expr` transforme cette chaîne en le nombre entier écrit en base 10.
  On tape :
  `expr("123")`
  On obtient :
  `123`
  On tape :
  `expr("0123")`
  On obtient :
  `83`
  En effet : $1*8^2+2*8+3=83$ On tape :
  `expr("0x12f")`
  On obtient :
  `303`

En effet $1 * 16^2 + 2 * 16 + 15 = 303$
Remarque On tape :

`expr(char([48,50,56,56]))`

On obtient un nombre flottant :

`288.0`

En effet `char([48,50,56,56])` renvoie `"0288"`.
Or un nombre entier commençant par 0 est un nombre écrit en octal, du coup il ne doit pas contenir de chiffre supérieur ou égal à 8. Ici le nombre a des chiffres égaux à 8, il est donc réinterprété (après détection d'erreur de saisie en octal) comme un nombre flottant.
On tape :

`expr(char([48,50,55,55]))`

On obtient un nombre exact car l'écriture `"0277"` est valide :

`191`

En effet `char([48,50,55,55])` renvoie `"0277"` et `0277` est l'écriture en base 8 de `191` (7+8*7+64*2=191).
On tape :

`expr(char([48,48,50,51]))`

On obtient un nombre exact car l'écriture `"0023"` est valide :

`19`

En effet `char([48,50,55,55])` renvoie `"0023"` et `0023` est l'écriture en base 8 de `19` (3+2*8=19).

– `expr` permet aussi de transformer une chaîne de chiffres représentant un nombre décimal en ce nombre.
On tape :

`expr("123.4567")`

On obtient :

`123.4567`

On tape :

`expr("123e-5")`

On obtient :

`0.00123`

– `expr` permet aussi de transformer une chaîne de caractères pouvant être interprétée comme une commande.
On tape :

`expr("a:=1")`

On obtient :

`1 et, la variable a contient 1`

6.6 Écriture en base b d'un entier

6.6.1 Transformer un entier en la liste des coefficients de son écriture en base b : `convert convertir`

`convert` ou `convertir` permet de faire différente conversions selon l'option choisie par le deuxième argument. Pour convertir un entier `n` en son écriture en base `b`, cette option est `base`. Les arguments de `convert` ou `convertir` sont alors un entier `n`, `base` et la valeur de la base `b`.

`convert` ou `convertir` renvoie la liste des coefficients de l'écriture en base `b` de l'entier `n`.
On tape :

```
convert(123,base,8)
```

On obtient :

```
[3,7,1]
```

On vérifie en tapant `expr("0173")` ou `horner(revlist([3,7,1]),8)` ou `convert([3,7,1],base,8)` et on obtient bien `123`
On tape :

```
convert(142,base,12)
```

On obtient :

```
[10,11]
```

6.6.2 Transformer la liste des coefficients d' une écriture en base b en un entier : `convert convertir`

`convert` ou `convertir` permet de faire différentes conversions selon l'option choisie par le deuxième argument.
Pour convertir la liste des coefficients d' une écriture en base `b` d'un entier `n` en l'entier `n`, cette option est `base`. Les arguments de `convert` ou `convertir` sont alors une list `l`, `base` et la valeur de la base `b`.
`convert` ou `convertir` renvoie l'entier `n`.
On tape :

```
convert([3,7,1],base,8)
```

Ou

```
horner(revlist([3,7,1]),8)
```

On obtient :

```
123
```

On tape :

```
convert([10,11],base,12)
```

Ou

```
horner(revlist([10,11]),12)
```

On obtient :

```
142
```

6.7 Les entiers (et les entiers de Gauss)

Dans tout ce paragraphe, on peut utiliser des entiers de Gauss (nombres complexes de la forme $a + i * b$ avec a et b dans $\mathbb{Z}$), à la place des entiers dans les différentes fonctions.

6.7.1 La factorielle : `factorial`

`Xcas` peut gérer des nombres entiers en précision infinie.
On tape :

```
factorial(100)
```

On obtient :

```
9332621544394415268169923885626670049071596826438162
1468592963895217599993229915608941463976156518286253
697920827223758251185210916864000000000000000000000000
```

6.7.2 Le PGCD : `gcd igcd`

`gcd` ou `igcd` désigne le PGCD de deux (ou de plusieurs) entiers ou rationnels ou entiers de Gauss (pour les polynômes voir alors 6.27.7).
On tape :

```
gcd(18,15)
```

ou

```
igcd(18,15)
```

On obtient :

```
3
```

On tape :

```
gcd(5,2+i)
```

ou

```
igcd(5,2+i)
```

On obtient :

```
2+i
```

en effet $(2 + i) * (2 - i) = 5$
On tape :

```
gcd(1+3i,7-4i)
```

ou

```
igcd(1+3i,7-4i)
```

On obtient :

```
2+i
```

en effet $(1+i)*(2+i) = 1+3*i$ et $-i*(2+i)*(3+2i) = (7-4i)$
On tape :

```
igcd(15/7,50/9)
```

ou

```
gcd(15/7,50/9)
```

On obtient :

```
5/63
```

en effet $\frac{15}{7} = 27\frac{5}{63}$ et $\frac{50}{9} = 70\frac{5}{63}$.
On tape :

```
gcd(18,15,21,36)
```

ou

```
igcd(18,15,21,36)
```

On obtient :

```
3
```

On tape :

```
gcd([18,15,21,36])
```

On obtient :

```
3
```

On peut aussi mettre comme paramètres deux listes de même longueur (ou une matrice ayant 2 lignes), dans ce cas `gcd` renvoie le PGCD des éléments de même indice (ou d'une même colonne). On tape :

```
gcd([6,10,12],[21,5,8])
```

Ou on tape :

```
gcd([[6,10,12],[21,5,8]])
```

On obtient :

```
[3,5,4]
```

On peut aussi utiliser la librairie Pari qui a une fonction `gcd` plus générale car `pari("gcd",x,y)` fonctionne aussi lorsque x et y sont rationnels et aussi lorsque x et y sont des listes ou des matrices qui n'ont pas forcément la même dimension (c'est alors le type de y qui donne le type du résultat.
On tape :

```
pari("gcd",5/7,50/9)
```

On obtient :

```
5/63
```

car $\frac{5}{7} = 9 * \frac{5}{63}$ et $\frac{50}{9} = 70 * \frac{5}{63}$ On tape :

```
pari("gcd", [4,3],[20,30,50,75]))
```

On obtient une matrice A de dimension 4×2, c'est aussi une liste de même longueur que y i.e. de longueur 4 :

```
[[4,1],[2,3],[2,1],[1,3]]
```

car gcd$(4, 20) = 4$, gcd$(3, 20) = 1$, gcd$(4, 30) = 2$, gcd$(3, 30) = 3$...
Pour obtenir ce résultat avec `Xcas`, on doit taper 2 instructions :

```
gcd( [4,4,4,4],[20,30,50,75])
```

On obtient la première colonne de A :

```
[4,2,2,1]
```

et on tape

```
gcd( [3,3,3,3],[20,30,50,75])
```

On obtient la deuxième colonne de A :

```
[1,3,1,3]
```

: On tape :

```
diag(pari("gcd", [5,4,3,2],[20,30,50,75])))
```

ou on tape :

```
gcd([5,4,3,2],[20,30,50,75])
```

On obtient :

```
[5,2,1,1]
```

Un exemple
Déterminer le pgcd de $4n + 1$ et de $5n + 3$ quand $n \in \mathbb{N}$.
On définit :

```
f(n):=gcd(4*n+1,5*n+3)
```

Puis on tape le programme `essai(n)` qui renvoie pour $j = -n$ à n la liste des valeurs de j, a lorsque le pgcd de $4j + 1$ et $5j + 3$ est égal à $a \neq 1$:

```
essai(n):={
  local j,a,L;
  L:=NULL;
  for (j:=-n;j<n;j++) {
    a:=f(j);
    if (a!=1) {
      L:=L,[j,a];
    }
  }
  return L;
}
```

Puis on tape :

```
essai(20)
```

On obtient :

```
[-16,7],[-9,7],[-2,7],[5,7],[12,7],[19,7]
```

On voit donc que $4n + 1$ et $5n + 3$ sont soit premiers entre eux soit leur pgcd vaut 7 lorsque $n \in [-16, -9, -2, 5, 12, 19]$ c'est à dire lorsque $n = 5 + k * 7$.
On doit donc montrer que :
si $n! = 5 + k * 7$ pour $k \in \mathbb{Z}$, $4n + 1$ et $5n + 3$ sont premiers entre eux, et
si $n = 5 + k * 7$ pour $k \in \mathbb{Z}$, $4n + 1$ et $5n + 3$ ont 7 comme pgcd.

6.7.3 Le PGCD : `Gcd`

`Gcd` est la forme inerte de `gcd`. Voir la section 6.27.8 sur les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ pour utiliser cette instruction.
On tape :

```
Gcd(18,15)
```

On obtient :

```
gcd(18,15)
```

6.7.4 Le PGCD d'une liste d'entiers : `lgcd`

`lgcd` désigne le PGCD des éléments d'une liste d'entiers (ou d'une liste de polynômes).
On tape :

```
lgcd([18,15,21,36])
```

On obtient :

```
3
```

On peut aussi utiliser la librairie Pari et taper :

```
pari("content",([18,15,21,36])
```

On obtient :

`3`

Attention
On ne peut pas mettre comme paramètres deux listes de même longueur.

6.7.5 Le PPCM : `lcm`

`lcm` désigne le PPCM de deux entiers ou de deux rationnels (ou deux polynômes voir alors 6.27.10).
On tape :

`lcm(18,15)`

On obtient :

`90`

On tape :

`lcm(5,2+i))`

On obtient :

`5`

On tape :

`lcm(1+3i,7-4i)`

On obtient :

`11+3*i`

en effet $(1+i)*(1-2*i)*(3+2i) = 11+3*i = (3-i)*(3+2i) = (1+i)*(7-4i)$
On tape :

`lcm(15/7,50/9)`

On obtient :

`150`

en effet `gcd(15/7,50/9)=5/63` et $\frac{15}{7} * \frac{50}{9} = 150 * \frac{5}{63}$

6.7.6 Factoriser un entier : `ifactor factoriser_entier`

`ifactor` a comme paramètre un entier.
`ifactor` décompose cet entier (ou entier de Gauss) en produit de facteurs premiers.
On tape :

`ifactor(90)`

On obtient :

```
2*3^2*5
```

On tape :

```
ifactor(-90)
```

On obtient :

```
(-1)*2*3^2*5
```

On tape :

```
ifactor(90i)
```

On obtient :

```
(1+i)^2*3^2*(2+i)*(2-i)
```

On tape :

```
ifactor(90+5i)
```

On obtient :

```
(i)*(2+i)*(2-i)^3*(3-2*i)
```

6.7.7 Liste des facteurs d'un entier : `facteurs_premiers ifactors`

`ifactors` a comme paramètre un entier (ou une liste d'entiers).
`ifactors` effectue aussi la décomposition de cet entier (ou des entiers de la liste) en produit de facteurs premiers, mais le résultat est donné sous la forme d'une liste (ou d'une liste de listes), formée par les diviseurs premiers et par leur multiplicité.
On tape :

```
ifactors(90)
```

On obtient :

```
[2,1,3,2,5,1]
```

On tape :

```
ifactors(-90)
```

On obtient :

```
[-1,1,2,1,3,2,5,1]
```

On tape :

```
ifactors(90i)
```

On obtient :

```
[1+i,2,3,2,2+i,1,2-i,1]
```

On tape :

```
ifactors(90+5i)
```

On obtient :

```
[i,1,2+i,1,2-i,3,3-2*i,1]
```

On tape :

```
ifactors([36,52])
```

On obtient :

```
[[2,2,3,2],[2,2,13,1]]
```

6.7.8 Matrice des facteurs d'un entier : maple_ifactors

maple_ifactors a comme paramètre un entier n (ou une liste d'entiers).
maple_ifactors effectue aussi la décomposition de cet entier (ou des entiers de la liste) en produit de facteurs premiers, mais selon la syntaxe Maple : le résultat est donné sous la forme d'une liste, formée de +1 ou -1 (pour le signe) et d'une matrice ayant 2 colonnes et dont les lignes sont formées des diviseurs premiers de n et de leur multiplicités (ou d'une liste de listes...).
On tape :

```
maple_ifactors(90)
```

On obtient :

```
[1,[[2,1],[3,2],[5,1]]]
```

On tape :

```
maple_ifactor([36,52])
```

On obtient :

```
[[1,[[2,2],[3,2]]],[1,[[2,2],[13,1]]]]
```

6.7.9 Liste des diviseurs d'un entier : idivis divisors

idivis ou divisors donne la liste des diviseurs d'un entier (ou d'une liste d'entiers).
On tape :

```
idivis(36)
```

On obtient :

```
[1,2,4,3,6,12,9,18,36]
```

On tape :

```
idivis([36,22])
```

On obtient :

```
[[1,2,4,3,6,12,9,18,36],[1,2,11,22]]
```

6.7.10 Quotient entier infixé de la division euclidienne : `div`

`div` est un opérateur infixé qui désigne le quotient entier q de la division euclidienne des deux entiers ou de deux entiers de Gauss a et b donnés en argument ($a = b * q + r$ avec $0 \leq r < b$).
On tape :

```
148 div 5
```

On obtient :

```
29
```

On tape :

```
factorial(148) div (factorial(145)+2)
```

On obtient :

```
3176375
```

```
factorial(148) div factorial(145)+2
```

On obtient :

```
3176378
```

On tape :

```
(25+12*i) div (5+7*i)
```

On obtient :

```
3-2*i
```

6.7.11 Quotient entier de la division euclidienne : `iquo intDiv`

`iquo` (ou `intDiv`) désigne le quotient entier q de la division euclidienne des deux entiers a et b donnés en argument ($a = b * q + r$ avec $0 \leq r < b$).
`iquo` travaille avec des entiers ou des entiers de Gauss.
Pour les entiers de Gauss, on choisit q pour que $b * q$ soit le plus proche possible de a et on peut montrer que l'on peut choisir r tel que $|r|^2 \leq |b|^2/2$.
On tape :

```
iquo(148,5)
```

On obtient :

```
29
```

On tape :

```
iquo(factorial(148),factorial(145)+2)
```

On obtient :

```
3176375
```

ou encore

```
iquo(25+12*i,5+7*i)
```

On obtient :

```
3-2*i
```

On a :
$a - b * q = -4 + i$ et on a $|-4+i|^2 = 17 < |5+7*i|^2/2 = 74/2 = 37$

6.7.12 Reste entier de la division euclidienne : `irem remain smod mods mod %`

– `irem` (ou `remain`) désigne le reste entier r de la division euclidienne des deux entiers a et b donnés en argument ($a = b * q + r$ avec $0 \leq r < b$).
Pour les entiers de Gauss, on choisit q pour $b * q$ soit le plus proche possible de a et on peut montrer que l'on peut choisir r tel que $|r|^2 \leq |b|^2/2$.
On tape :
```
irem(148,5)
```
On obtient :
```
3
```
`irem` travaille avec des entiers longs ou des entiers de Gauss.
On tape :
```
irem(factorial(148),factorial(45)+2 )
```
On obtient :
```
111615339728229933018338917803008301992120942047239639312
```
ou encore
```
irem(25+12*i,5+7*i)
```
On obtient :
```
-4+i
```
On a :
$a - b * q = -4 + i$ et on a $|-4+i|^2 = 17 < |5+7*i|^2/2 = 74/2 = 37$

– `smod` ou `mods` est préfixé et a 2 entiers comme arguments.
`smod` ou `mods` désigne le reste entier symétrique s de la division euclidienne des deux entiers a et b donnés en argument ($a = b * q + s$ avec $-b/2 < s \leq b/2$).
On tape :
```
smod(148,5)
```
On obtient :
```
-2
```

– `mod` ou `%` sert à désigner un nombre modulaire.
`mod` ou `%` est infixé et renvoie un nombre modulaire. **Attention** `%` doit être suivi d'un espace.

On tape :
```
148 mod 5
```
ou
```
148 % 5
```

On obtient :

`-2 % 5`

ce qui veut dire que $148 \bmod 5 = -2 \bmod 5$ et que $148 \bmod 5$ et $-2\ \%\ 5$ sont des éléments de $Z/5Z$ (voir 6.33 pour avoir les opérations possibles dans $Z/5Z$).

Remarque

Si on veut passer d'un élément de $Z/5Z$ à un élément de Z, on tape `% 0` : On tape :

`(148% 5)% 0`

On obtient :

`-2`

6.7.13 Le quotient et le reste de la division euclidienne : `iquorem`

`iquorem` donne la liste du quotient q et du reste entier r de la division euclidienne des deux entiers a et b donnés en argument ($a = b * q + r$ avec $0 \leq r < b$).

.

On tape :

`iquorem(148,5)`

On obtient :

`[29,3]`

6.7.14 Test de parité : `even est_pair`

`even` ou `est_pair` a comme argument un entier `n`.

`even` ou `est_pair` renvoie `1` si `n` est pair et renvoie `0` si `n` est impair.

On tape :

`even(148)`

On tape :

`est_pair(148)`

On obtient :

`1`

On tape :

`even(149)`

On tape :

`est_pair(149)`

On obtient :

`0`

6.7.15 Test de non parité : `odd est_impair`

`odd` ou `est_impair` a comme argument un entier `n`.
`odd` ou `est_impair` renvoie `1` si `n` est impair et renvoie `0` si `n` est pair.
On tape :

```
odd(148)
```

On tape :

```
est_impair(148)
```

On obtient :

```
0
```

On tape :

```
odd(149)
```

On tape :

```
est_impair(149)
```

On obtient :

```
1
```

6.7.16 Test de pseudo-primalité : `is_pseudoprime`

- Si `n` est premier, `is_pseudoprime(n)` renvoie `2` (vrai),
- Si `n` est pseudo-premier, `is_pseudoprime(n)` renvoie `1` (vrai),
- Si `n` n'est pas premier, `is_pseudoprime(n)` renvoie `0` (faux).

Définition : Pour les nombres inférieurs à 10^{14} être pseudo-premier et premier c'est la même chose ! ...mais au delà de 10^{14} un nombre pseudo-premier est premier avec une probabilité très forte (cf l'algorithme de Rabin et de Miller-Rabin dans la partie Algorithmique et traduction `Xcas` avec le menu `Aide->Manuels->Programmation`).
On tape :

```
is_pseudoprime(100003)
```

On obtient :

```
2
```

On tape :

```
is_pseudoprime(9856989898997)
```

On obtient :

```
2
```

On tape :

```
is_pseudoprime(14)
```

On obtient :

`0`

On tape :

`is_pseudoprime(98569898989977789789)`

On obtient :

`1`

6.7.17 Test de primalité : `is_prime isprime isPrime`

`is_prime(n)` renvoie `1` (vrai) ou `0` (faux) selon que son argument est premier ou non. `isprime` ou `isPrime` renvoie `true` ou `false`.
Utiliser la commande `is_prime(n,1)` (ou `pari("isprime",n,1)`) pour obtenir un certificat de primalité par le test "p-1" de Selfridge-Pocklington-Lehmer (voir la documentation de PARI/GP, depuis le menu `Aide->Manuels->PARI-GP`)) et `pari("isprime",n,2)` ou `is_prime(n,2)` pour utiliser le test APRCL ou `is_prime(n)` pour utiliser un test mixte.
En interne isprime appelle `is_prime`, et si la reponse est `0` il la transforme en `false`, sinon il renvoie `true`. On ne peut donc pas obtenir de certificat de primalité avec `isprime`, par contre les arguments de `isprime` et `is_prime` sont identiques.
On tape :

`is_prime(100003)`

On obtient :

`1`

On tape :

`isprime(100003)`

On obtient :

`true`

On tape :

`is_prime(98569898989987)`

On obtient :

`1`

On tape :

`is_prime(14)`

On obtient :

`0`

On tape :

```
isprime(14)
```

On obtient :

```
false
```

Pour obtenir un certificat de primalité pour $n = 9856989898997789789$, on tape :

```
is_prime(9856989898997789789,1)
```

ou :

```
pari("isprime",9856989898997789789,1)
```

On obtient les coefficients prouvant la primalité par le test "p-1" de Selfridge-Pocklington-Lehmer :

```
[[2,2,1],[19,2,1],[941,2,1],[1873,2,1],[94907,2,1]]
```

sinon, on tape :

```
pari("isprime",9856989898997789789,2)
```

Ou on tape :

```
is_prime(9856989898997789789,2)
```

On obtient :

```
1
```

6.7.18 Nombre pseudo-premier : `nprimes`

`nprimes(n)` renvoie le nombre de nombres pseudo-premier (ou premier) qui sont inférieurs ou égaux à n.
On tape :

```
nprimes(5)
```

On obtient :

```
3
```

On tape :

```
nprimes(10)
```

On obtient :

```
4
```

6.7.19 N-ième nombre pseudo-premier : `ithprime`

`ithprime(n)` désigne le n-ième nombre pseudo-premier (ou premier).
On tape :

```
ithprime(75)
```

On obtient :

```
379
```

6.7.20 Nombre pseudo-premier après `n` : `nextprime`

`nextprime(n)` désigne le plus petit nombre pseudo-premier (ou premier) plus grand que `n`.
On tape :

```
nextprime(75)
```

On obtient :

```
79
```

6.7.21 Nombre pseudo-premier avant `n` : `prevprime`

`prevprime(n)` désigne le premier nombre pseudo-premier (ou premier) trouvé avant `n`.
On tape :

```
prevprime(75)
```

On obtient :

```
73
```

6.7.22 Le `n`-ième nombre premier : `ithprime`

`ithprime(n)` désigne le `n`-ième nombre premier inférieur à 10000 (pour l'instant !).
On tape :

```
ithprime(75)
```

On obtient :

```
379
```

On tape :

```
ithprime(1229)
```

On obtient :

```
9973
```

On tape :

```
ithprime(1230)
```

On obtient :

```
ithprime(1230)
```

car cela désigne un nombre premier > 10000.

6.7.23 Identité de Bézout : `iegcd igcdex bezout_entiers`

`iegcd(a,b)` ou `igcdex(a,b)` désigne le PGCD étendu (identité de Bézout) de deux entiers.
`iegcd(a,b)` ou `igcdex(a,b)` renvoie `[u,v,d]` vérifiant `au+bv=d` et tel que `d=gcd(a,b)`.
On tape :

```
iegcd(48,30)
```

On obtient :

```
[2,-3,6]
```

En effet :

$$2 \cdot 48 + (-3) \cdot 30 = 6$$

6.7.24 Résolution de au+bv=c dans $\mathbb{Z}$: `iabcuv`

`iabcuv(a,b,c)` donne `[u,v]` vérifiant `au+bv=c`.
Il faut bien sûr que `c` soit un multiple de `gcd(a,b)` pour obtenir une solution.
On tape :

```
iabcuv(48,30,18)
```

On obtient :

```
[6,-9]
```

6.7.25 Restes chinois : `ichinrem, ichrem`

`ichinrem([a,p],[b,q])` ou `ichrem([a,p],[b,q])` désigne une liste `[c,lcm(p,q)]` formée de deux entiers.
Le premier nombre `c` est tel que

$$\forall k \in \mathbb{Z}, \quad d = c + k \times \text{lcm}(p, q)$$

vérifie

$$d = a \pmod p, \quad d = b \pmod q$$

Si `p` et `q` sont premiers entre eux, il existe toujours une solution `d` et toutes les solutions sont alors congrues modulo `p*q`.
Exemples :

1. Trouver les solutions de :

$$\begin{cases} x = & 3 \pmod 5 \\ x = & 9 \pmod{13} \end{cases}$$

On tape :

```
ichinrem([3,5],[9,13])
```

ou on tape :

```
ichrem([3,5],[9,13])
```

On obtient :

`[-17,65]`

ce qui veut dire que `x=-17 (mod 65)`
On peut aussi taper :

`ichrem(3% 5,9% 13)`

On obtient :

`-17% 65`

2. Trouver les solutions de :

$$\begin{cases} x = & 3 \pmod 5 \\ x = & 4 \pmod 7 \\ x = & 1 \pmod 9 \end{cases}$$

On tape tout d'abord :

`tmp:=ichinrem([3,5],[4,7])`

ou on tape :

`tmp:=ichrem([3,5],[4,7])`

On obtient :

`[-17,35]`

puis on tape

`ichinrem([1,9],tmp)`

ou on tape :

`ichrem([1,9],tmp)`

On obtient :

`[-17,315]`

ce qui veut dire que `x=-17 (mod 315)`
On peut aussi taper directement :

`ichinrem([3% 5,4% 7,1% 9])`

On obtient :

`-17% 315`

6.7.26 Reste chinois pour des polynômes connus modulo plusieurs entiers : `ichinrem, ichrem`

`ichrem` (ou `ichinrem`) peut aussi être utiliser pour trouver les coefficients de polynômes qui sont connus modulo plusieurs entiers, par exemple trouver :
$ax + b$ modulo $315 = 5 \times 7 \times 9$ tel que :

$$\begin{cases} a = & 3 \pmod 5 \\ a = & 4 \pmod 7 \\ a = & 1 \pmod 9 \end{cases}, \quad \begin{cases} b = & 1 \pmod 5 \\ b = & 2 \pmod 7 \\ b = & 3 \pmod 9 \end{cases}$$

On tape :

```
ichrem((3x+1)% 5,(4x+2)% 7,(x+3)% 9)
```

On obtient :

```
(-17% 315)× x+156% 315
```

ce qui veut dire que `a=-17 (mod 315)` et `b=156 (mod 315)`.

6.7.27 Reste chinois pour des listes d'entiers : `chrem`

`chrem` a comme argument deux listes de même longueur.
`chrem` renvoie une liste de deux entiers.
Par exemple, `chrem([a,b,c],lcm(p,q,r))` désigne une liste formée de deux entiers :
le premier entier est un nombre `x` vérifiant :
`x=a mod p` et `x=b mod q` et `x=c mod r`.
Il existe toujours une solution `x` si `p` et `q` sont premiers entre eux, et toutes les solutions sont congrues modulo `p*q*r`
ATTENTION à l'ordre des paramètres, en effet on a :
`chrem([a,b],[p,q])=ichrem([a,p],[b,q])=`
`ichinrem([a,p],[b,q])`
Exemples :
Trouver les solutions de :

$$\begin{cases} x = & 3 \pmod 5 \\ x = & 9 \pmod{13} \end{cases}$$

On tape :

```
chrem([3,9],[5,13])
```

On obtient :

```
[-17,65]
```

ce qui veut dire que `x=-17 (mod 65)` Trouver les solutions de :

$$\begin{cases} x = & 3 \pmod 5 \\ x = & 4 \pmod 6 \\ x = & 1 \pmod 9 \end{cases}$$

On tape :

```
chrem([3,4,1],[5,6,9])
```

On obtient :

```
[28,90]
```

ce qui veut dire que `x=28 (mod 90)`
Remarque
`chrem` peut aussi être utiliser pour trouver les coefficients de polynômes qui sont connus modulo plusieurs entiers, par exemple trouver $ax + b$ modulo $315 = 5 \times 7 \times 9$ tel que :

$$\begin{cases} a = & 3 \pmod 5 \\ a = & 4 \pmod 7 \\ a = & 1 \pmod 9 \end{cases}, \quad \begin{cases} b = & 1 \pmod 5 \\ b = & 2 \pmod 7 \\ b = & 3 \pmod 9 \end{cases}$$

On tape :

```
chrem([3x+1,4x+2,x+3],[5,7,9])
```

On obtient :

```
[-17x+156),315]
```

ce qui veut dire que `a=-17 (mod 315)` et que `b=156 (mod 315)`.

6.7.28 Résolution de $a^2 + b^2 = p$ dans $\mathbb{Z}$: `pa2b2`

`pa2b2` décompose un entier p premier, congru à 1 modulo 4, en la somme de deux carrés : $p = a^2 + b^2$.
Le résultat est donné sous la forme d'une liste.
On tape :

```
pa2b2(17)
```

On obtient :

```
[4,1]
```

en effet $17 = 4^2 + 1^2$

6.7.29 Indicatrice d'Euler : `euler Phi`

`euler` (ou `Phi`) désigne l'indicatrice d'Euler d'un entier.
`euler(n)` (ou `Phi(n)`) est égale au cardinal de l'ensemble des nombres inférieurs à `n` qui sont premiers avec `n`.
On tape :

```
euler(21)
```

On obtient :

```
12
```

En effet l'ensemble :
E={2,4,5,7,8,10,11,13,15,16,17,19} correspond aux nombres inférieurs à 21 qui sont premiers avec 21, et E a comme cardinal 12.
Euler a introduit cette fonction pour formuler la généralisation du petit théorème de Fermat qui dit "si n est premier et si a est premier avec n alors $a^{n-1} = 1 \mod n$".
La généralisation est (puisque si n est premier, $euler(n) = n - 1$) :
$a^{euler(n)} = 1 \mod n$ si a et n sont premiers entre eux.
On tape :

```
powmod(5,12,21)
```

On obtient :

```
1
```

6.7.30 Symbole de Legendre : `legendre_symbol`

Lorsque n est premier, on définit le symbole de Legendre de a noté $\left(\frac{a}{n}\right)$ par :

$$\left(\frac{a}{n}\right) = \begin{cases} 0 & \text{si } a = 0 \mod n \\ 1 & \text{si } a \neq 0 \mod n \text{ et si } a = b^2 \mod n \\ -1 & \text{si } a \neq 0 \mod n \text{ et si } a \neq b^2 \mod n \end{cases}$$

Quelques propriétés

– Si n est premier :

$$a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right) \mod n$$

–

$$\begin{aligned} \left(\frac{p}{q}\right).\left(\frac{q}{p}\right) &= (-1)^{\frac{p-1}{2}}.(-1)^{\frac{q-1}{2}} \text{ si } p \text{ et } q \text{ sont impairs et positifs} \\ \left(\frac{2}{p}\right) &= (-1)^{\frac{p^2-1}{8}} \\ \left(\frac{-1}{p}\right) &= (-1)^{\frac{p-1}{2}} \end{aligned}$$

`legendre_symbol` a deux paramètres a et n et renvoie le symbole de Legendre $\left(\frac{a}{n}\right)$.

On tape :

```
legendre_symbol(26,17)
```

On obtient :

```
1
```

On tape :

```
legendre_symbol(27,17)
```

On obtient :

```
-1
```

On tape :

```
legendre_symbol(34,17)
```

On obtient :

```
0
```

6.7.31 Symbole de Jacobi : `jacobi_symbol`

Lorsque n n'est pas premier on définit le symbole de Jacobi de a, noté encore $\left(\frac{a}{n}\right)$, à partir du symbole de Legendre et de la décomposition de n en facteur premier.
Soit

$$n = p_1^{\alpha_1}..p_k^{\alpha_k}$$

où p_j est premier and α_j est un entier pour $j = 1..k$. Le symbole de Jacobi de a est définit par :

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} ... \left(\frac{a}{p_k}\right)^{\alpha_k}$$

`jacobi_symbol` a deux paramètres a et n et renvoie le symbole de Jacobi $\left(\frac{a}{n}\right)$.
On tape :

```
jacobi_symbol(25,12)
```

On obtient :

```
1
```

On tape :

```
jacobi_symbol(35,12)
```

On obtient :

```
-1
```

On tape :

```
jacobi_symbol(33,12)
```

On obtient :

```
0
```

6.8 Analyse combinatoire

6.8.1 La factorielle : `factorial !`

`factorial` a comme paramètre un entier n.
`factorial(n)` calcule $n!$ (on peut aussi écrire directement `n!`).
On tape :

```
factorial(10)
```

ou

```
10!
```

On obtient :

```
3628800
```

6.8.2 Les coefficients binomiaux : `comb nCr`

`comb` (ou `nCr`) a comme paramètre deux entiers positifs `n` et `p`.
`comb(n,p)` ou `nCr(n,p)` calcule C_n^p.
On a :$C_n^p = \frac{n!}{p!(n-p)!} = \frac{n(n-1)...(n-p+1)}{p!}$.
On tape :

```
comb(5,2)
```

On obtient :

```
10
```

Remarque 1
On peut définir `comb` (ou `nCr`) avec comme paramètre `n` entier négatif et `p` entier positif par :
`comb(n,p)`=$\frac{n(n-1)...(n-p+1)}{p!}$ On tape :

```
comb(-5,2)
```

On obtient :

```
15
```

En effet $-5 * -6/2 = 15$
On tape :

```
comb(-5,3)
```

On obtient :

```
-35
```

En effet $-5 * -6 * -7/6 = -35$
Remarque 2
On a aussi `binomial(n,p)=comb(n,p)` lorsque `n` et `p` sont deux entiers positifs mais `binomial` peut admettre un paramètre supplémentaire et dans ce cas `binomial(n,p,a)` calcule $C_n^p * a^p * (1-a)^{n-p}$.

6.8.3 Les arrangements : `perm nPr`

`perm` ou `nPr` a comme paramètre deux entiers n et p.
`perm(n,p)` ou `nPr(n,p)` calcule A_n^p.
On tape :

```
perm(5,2)
```

On obtient :

```
20
```

6.8.4 Les nombres entiers aléatoires : `rand alea hasard`

`rand` a comme paramètre un entier n ou aucun paramètre.

- `rand(n)` renvoie un entier p aléatoire vérifiant $0 \leq p < n$.
  On tape :
  `rand(10)`
  On obtient par exemple :
  `8`
- `rand()` renvoie un réel r aléatoire vérifiant $0 \leq p < 1$.
  On tape :
  `rand()`
  On obtient par exemple :
  `0.72392661823`

Remarque Il faut utiliser `srand(n);` (avec `n` un entier) ou `srand;` pour initialiser la suite des nombres aléatoires.

6.9 Les rationnels

6.9.1 Transformer un nombre décimal en rationnel : `exact float2rational`

`float2rational` ou `exact` a comme paramètre un nombre décimal `d` et renvoie un nombre rationnel `q` qui approche `d` à moins de `epsilon`. On définit `epsilon` dans la configuration du `cas` (menu `Cfg`) ou avec la commande `cas_setup`.
On tape :

`float2rational(0.3670520231)`

On obtient pour `epsilon=1e-10` :

`127/346`

Essayez d'entrer :

`123/12+57/21`

On obtient :

`363/28`

Puis :

`evalf(363/28)`

On obtient :

`12.9642857143`

On tape :

`float2rational(12.9642857143)`

On obtient :

```
363/28
```

Si on mélange les deux représentations par exemple :

```
1/2+0.7
```

On obtient

```
1.2
```

On tape :

```
1/2+float2rational(0.7)
```

On obtient :

```
6/5
```

6.9.2 Partie entière et fractionnaire : `propfrac propFrac`

`propfrac(A/B)` ou `propFrac(A/B)` écrit la fraction $\frac{A}{B}$ après simplification en $\frac{a}{b}$ (avec $\gcd(a, b) = 1$) sous la forme :

$$q + \frac{r}{b} \text{ avec } 0 \leq r < b$$

Pour les fractions rationnelles on se reportera à 6.30.8.
On tape :

```
propfrac(42/15)
```

On obtient :

```
2+4/5
```

On tape :

```
propfrac(43/12)
```

On obtient :

```
3+7/12
```

6.9.3 Numérateur d'une fraction après simplification : `numer`, `getNum`

`numer` ou `getNum` a comme argument une fraction et renvoie le numérateur de cette fraction simplifiée (pour les fractions rationnelles on se reportera à 6.30.2).
On tape :

```
numer(42/12)
```

Ou :

```
getNum(42/12)
```

On obtient :

7

Si on veut le numérateur de cette fraction non simplifiée il faut quoter l'argument (pour les fractions rationnelles on se reportera à 6.30.1).
On tape :

`numer('42/12')`

Ou :

`getNum('42/12')`

On obtient :

`42`

6.9.4 Dénominateur d'une fraction après simplification : `denom getDenom`

`denom` ou `getDenom` a comme argument une fraction et renvoie le dénominateur de cette fraction simplifiée (pour les fractions rationnelles on se reportera à 6.30.4)
On tape :

`denom(42/12)`

Ou :

`getDenom(42/12)`

On obtient :

`2`

Si on veut le dénominateur de cette fraction non simplifiée il faut quoter l'argument (pour les fractions rationnelles on se reportera à 6.30.3).
On tape :

`denom('42/12')`

Ou :

`getDenom('42/12')`

On obtient :

`12`

6.9.5 Numérateur et dénominateur d'une fraction : `f2nd fxnd`

`f2nd` (ou `fxnd`) a comme argument une fraction et renvoie, la liste formée par le numérateur et le dénominateur de cette fraction simplifiée (pour les fractions rationnelles on se reportera à 6.30.5).
On tape :

`f2nd(42/12)`

On obtient :

`[7,2]`

6.9.6 Simplification d'un couple : simp2

simp2 a pour argument deux entiers ou une liste de deux entiers représentant une fraction (pour deux polynômes voir alors 6.30.6).
simp2 renvoie une liste formée par le numérateur et le dénominateur de cette fraction simplifiée.
On tape :

```
simp2(18,15)
```

On obtient :

```
[6,5]
```

On tape :

```
simp2([42,12])
```

On obtient :

```
[7,2]
```

6.9.7 Développement en fraction continue d'un réel : dfc

dfc a comme argument un nombre réel ou fractionnaire ou décimal a et un entier n (ou un réel epsilon).
dfc renvoie une liste représentant le développement en fractions continues de a d'ordre n (ou de précision epsilon c'est à dire le développement en fractions continues qui approche a ou evalf(a) à moins de epsilon, par défaut epsilon est égal à la valeur du epsilon définit dans la configuration du cas à l'aide du menu Cfg▶Configuration du CAS).
On peut aussi utiliser convert avec l'option confrac : dans ce cas la valeur de epsilon est égal à la valeur du epsilon définit dans la configuration du CAS à l'aide du menu Cfg▶Configuration du CAS (voir 6.23.26).
Remarques
Si le dernier élément de la liste résultat est une liste il représente la période et si le dernier élément de la liste résultat n'est pas entier, il représente le reste r ($a = a0 + 1/.... + 1/an + 1/r$).
Si on a dfc(a)=[a0,a1,a2,[b0,b1]] cela veut dire que :
$$a = a0 + \cfrac{1}{a1 + \frac{1}{a2+\frac{1}{b0+\frac{1}{b1+\frac{1}{b0+...}}}}}$$
Si on a dfc(a)=[a0,a1,a2,r] cela veut dire que :
$$a = a0 + \cfrac{1}{a1 + \frac{1}{a2+\frac{1}{r}}}$$
On tape :

```
dfc(sqrt(2),5)
```

On obtient :

```
[1,2,[2]]
```

On tape :

```
dfc(evalf(sqrt(2)),1e-9)
```

Ou :

```
dfc(sqrt(2),1e-9)
```

On obtient :

```
[1,2,2,2,2,2,2,2,2,2,2,2,2]
```

On tape :

```
convert(sqrt(2),confrac,'dev')
```

On obtient si dans la configuration du `cas` `epsilon=1e-9` :

```
[1,2,2,2,2,2,2,2,2,2,2,2,2]
```

et `dev` contient `[1,2,2,2,2,2,2,2,2,2,2,2,2]`
On tape :

```
dfc(9976/6961,5)
```

On obtient :

```
[1,2,3,4,5,43/7]
```

En effet on tape :
`1+1/(2+1/(3+1/(4+1/(5+7/43))))`
et on obtient :
`9976/6961`
On tape :

```
convert(9976/6961,confrac,'l')
```

On obtient si dans la configuration du `cas` `epsilon=1e-9` :

```
[1,2,3,4,5,6,7]
```

et `l` contient `[1,2,3,4,5,6,7]`
On tape :

```
dfc(pi,5)
```

On obtient :

```
[3,7,15,1,292,(-113*pi+355)/(33102*pi-103993)]
```

On tape :

```
dfc(evalf(pi),5)
```

On obtient (si on travaille avec 12 chiffres significatifs) :

```
[3,7,15,1,292,1.57581843574]
```

On tape :

```
dfc(evalf(pi),1e-9)
```

Ou :

```
dfc(pi,1e-9)
```

Ou (si `epsilon=1e-9` dans la configuration du `cas`) :

```
convert(pi,confrac,'ll')
```

On obtient :

```
[3,7,15,1,292]
```

6.9.8 Transformer une fraction continue en un réel : `dfc2f`

`dfc2f` a comme argument une liste représentant le développement en fraction continue d'un nombre rationnel ou d'un nombre quadratique (nombre qui est racine d'une équation du second degré) : quand le développement est périodique, le dernier élément de la liste est une liste qui représente la période du développement en fraction continue et si le dernier élément de la liste n'est ni une liste ni un entier cet élément représente le reste r ($a = a0 + 1/.... + 1/an + 1/r$).
`dfc2f` renvoie le nombre rationnel ou le nombre quadratique ayant l'argument comme développement en fraction continue.
On tape :

```
dfc2f([1,2,[2]])
```

On obtient :

```
1/(1/(1+sqrt(2))+2)+1
```

Après simplification avec `normal` :

```
sqrt(2)
```

On tape :

```
dfc2f([1,2,3])
```

On obtient :

```
10/7
```

On tape :

```
normal(dfc2f([3,3,6,[3,6]]))
```

On obtient :

```
sqrt(11)
```

On tape :

```
dfc2f([1,2,3,4,5,6,7])
```

On obtient :

```
9976/6961
```

En effet on tape :
`1+1/(2+1/(3+1/(4+1/(5+1/(6+1/7)))))`
et on obtient :
`9976/6961`
On tape :

```
dfc2f([1,2,3,4,5,43/7])
```

On obtient :

```
9976/6961
```

En effet on tape :
`1+1/(2+1/(3+1/(4+1/(5+7/43))))`
et on obtient :
`9976/6961`

6.9.9 Le n^{ime} nombre de Bernoulli : `bernoulli`

`bernoulli` a comme argument un entier n.
`bernoulli` renvoie le n^{ime} nombre de Bernoulli $B(n)$.
On a :

$$\frac{t}{e^t-1}=\sum_{n=0}^{+\infty}\frac{B(n)}{n!}t^n$$

On rappelle que les polynômes de Bernoulli B_k sont définis par :

$$B_0=1$$

$$B_k{}'(x)=kB_{k-1}(x)$$

$$\int_0^1 B_k(x)dx=0$$

On a alors :
$B(n)=B_n(0)$
On tape :

```
bernoulli(6)
```

On obtient :

6.9.10 Accès aux fonctions de PARI/GP : commande `pari`

La commande `pari` sans argument exporte les fonctions de pari qui n'ont pas d'homonymes sous `Xcas` en leur nom habituel sous PARI/GP.
Toutes les commandes sont aussi exportées sous leur nom d'origine avec le préfixe `pari_`.

La commande `pari` avec en premier argument un chaine de caractère - le nom d'une commande PARI - et d'éventuels autres arguments exécute la commande PARI avec les autres arguments.
Par exemple :

- On tape :
  `pari()` puis `weber(1+i)` ou directement
  `pari("weber",1+i)`, cela exécute la commande `weber` de PARI avec comme argument `1+i`. La commande `weber` n'existant pas sous `Xcas`.
- On tape :
  `pari("content",[25,15,50,75])` ou
  `pari_content([25,15,50,75])`, cela exécute la commande `content` de PARI avec comme argument `[25,15,50,75]` et renvoie `5` qui est le pgcdde la liste d'entiers donnée en argument alors que `pari()` puis `content([25,15,50,75])` exécute la commande `content` de `Xcas` et renvoie aussi `5` car `content` est une commande `Xcas` qui a comme argument un polynôme donné sous sa forme symbolique ou par la liste de ses coefficients et qui renvoie le même résultat que la commande `content` de PARI car on a :
  `content([25,15,50,75])=content(25x^3+15x^2+50x+75)=5`
- On tape :
  `pari("gcd",[4,3,2,15],[20,30,50,75])` ou
  `pari_gcd([4,3,2,15],[20,30,50,75])`, cela exécute la commande `gcd` de PARI d'arguments `L1=[4,3,2,15]` et `L2=[20,30,50,75]` qui renvoie la matrice :
  `[[4,1,2,5],[2,3,2,15],[2,1,2,5],[1,3,1,15]]` :c'est la matrice `M[j,k]` égale au pgcd des entiers `L1[j]` et `L2[k]` alors que, :
  `pari()` puis
  `gcd([4,3,2,15],[20,30,50,75])` renvoie `[4,3,2,15]` car `gcd` est une commande `Xcas` qui avec comme argument 2 listes `L1` et `L2` renvoie la liste `L3[j]` des pgcd de `L1[j]` et `L2[j]` c'est à dire la diagonale de la matrice précédente car on a :
  `gcd([4,3,2,15],[20,30,50,75])=`
  `diag(pari("gcd", [4,3,2,15],[20,30,50,75]))`

La documentation de PARI/GP est disponible depuis le menu Aide->Manuels.

6.10 Les réels

6.10.1 Évaluer un réel et nombre de digits : `evalf` et `Digits`, `DIGITS`)

Il faut distinguer le nombre réel de sa valeur numérique qui est un nombre flottant.
La précision des nombres flottants, en nombre de chiffres est contrôlée par la va-

riable `Digits` qui vaut 12 par défaut.
Une expression peut être évaluée en flottant grâce à la commande `evalf`.
Attention ! ! ! !
Les nombres flottants sont contagieux (!) car une expression qui comporte un nombre flottant est calculée en flottant.
On tape :

```
1+1/2
```

On obtient :

```
3/2
```

On tape :

```
1.0+1/2
```

On obtient :

```
1.5
```

On tape :

```
exp(pi*sqrt(20))
```

On obtient :

```
exp(pi*2*sqrt(5))
```

Avec la commande `evalf`, on tape :

```
evalf(exp(pi*2*sqrt(5)))
```

On obtient :

```
1263794.75367
```

On tape :

```
1.1^20
```

On obtient :

```
6.72749994933
```

On tape :

```
sqrt(2)^21
```

On obtient :

```
sqrt(2)*2^10
```

On tape pour avoir 30 chiffres significatifs :

```
Digits:=30
```

On tape pour avoir la valeur numérique de $e^{\pi\sqrt{163}}$:

```
evalf(exp(pi*sqrt(163)))
```

On obtient :

```
0.262537412640768743999999999985e18
```

6.10.2 Les fonctions infixées de base sur les réels : `+`, `-`, `*`, `/`, `^`

`+`, `-`, `*`, `/`, `^` sont les opérateurs habituels pour faire des additions, des soustractions, des multiplications, des divisions et des élévations à une puissance entière ou fractionnaire.
On tape :

```
3+2
```

On obtient :

```
5
```

On tape :

```
3-2
```

On obtient :

```
1
```

On tape :

```
3*2
```

On obtient :

```
6
```

On tape :

```
3/2
```

On obtient :

```
3/2
```

On tape :

```
3.2/2.1
```

On obtient :

```
1.52380952381
```

On tape :

```
3^2
```

On obtient :

```
9
```

On tape :

```
3.2^2.1
```

On obtient :

```
11.5031015682
```

Remarque
Si vous avez une touche carrée ou une touche cube sur votre clavier vous pouvez l'utiliser, par exemple : 3^2 vaut 9.
Remarque sur les puissances fractionnaires
Par définition $a^fracpq = \exp(fracpq * \ln(a))$ et donc a^fracpq n'est défini que pour $a > 0$.
Il y a donc une différence entre :
$\sqrt[n]{a}$ et a^frac1n lorsque n est impair.
Si on veut, par exemple, tracer la courbe $y = \sqrt[3]{x^3 - x^2}$, il faut taper :
`plotfunc([(x^3-x^2)^(1/3),-(x^2-x^3)^(1/3)],x,xstep=0.01)`
on peut aussi taper :
`plotimplicit(y^3=x^3-x^2)`

6.10.3 Les fonctions prefixées de base sur les réels : `rdiv`

`rdiv` est l'opérateur prefixé pour faire des divisions.
On tape :

```
rdiv(3,2)
```

On obtient :

```
3/2
```

On tape :

```
rdiv(3.2,2.1)
```

On obtient :

```
1.52380952381
```

6.10.4 La fonction racine n-ième : `root`

`root` a deux arguments : un entier n et un nombre a.
`root` renvoie la racine n-ième de a (i.e. $a^{1/n}$).
On tape :

```
root(3,2)
```

ou on tape :

```
2^(1/3)
```

On obtient :

```
2^(1/3)
```

On tape :

```
root(3,2.0)
```

ou on tape :

```
2.^(1/3)
```

On obtient :

```
1.259921049892
```

On tape :

```
root(3,sqrt(2))
```

ou on tape :

```
sqrt(2)^(1/3)
```

On obtient :

```
2^(1/6)
```

6.10.5 La fonction exponentielle integrale Ei : `Ei`

`Ei` a comme argument un nombre complexe a.
`Ei` calcule les valeurs de la fonction Ei au point a.
On a par définition :

$$Ei(x) = \int_{t=-\infty}^{x} \frac{\exp(t)}{t} dt$$

Pour $x > 0$, on prolonge par la valeur principale de l'intégrale (les morceaux en 0^- et 0^+ se compensent). On a :

$$Ei(0) = -\infty, \quad Ei(-\infty) = 0$$

Lorsque l'on est proche de $x = 0$ on sait que :

$$\frac{\exp(x)}{x} = \frac{1}{x} + 1 + \frac{x}{2!} + \frac{x^2}{3!} + ... + \frac{x^n}{(n-1)!}....$$

on a donc pour $x \in \mathbb{C} - \mathbb{R}^+$, (la fonction est discontinue sur $\mathbb{R}^+$) :

$$Ei(x) = \ln(-x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + ...$$

où γ = la constante d'Euler = 0.57721566490..
sur l'axe $x > 0$ on prend : $Ei(x) = \ln(x) + \gamma + x + \frac{x^2}{2.2!} + \frac{x^3}{3.3!} + ...$
On tape :

```
Ei(1.)
```

On obtient :

```
1.89511781636
```

On tape :

```
Ei(-1.)
```

On obtient :

```
-0.219383934396
```

On tape :

```
Ei(1.)-Ei(-1.)
```

On obtient :

```
2.11450175075
```

On tape :

```
int((exp(x)-1)/x,x=-1..1.)
```

On obtient :

```
2.11450175075
```

On tape :

```
evalf(Ei(-1)-sum((-1)^n/n/n!,n=1..100))
```

On obtient la constante d'Euler γ :

```
0.577215664901532860606507
```

Lorsque `Ei` a comme argument 2 nombres : a réel et b entier positif, on a :
`Ei(a,1)=-Ei(-a)`
`Ei(a,2)=exp(-a)+a*Ei(-a) =exp(-a)-a*Ei(a,1)`
`Ei(a,3)=(exp(-a)-a*(exp(-a)+a*Ei(-a)))/2=(exp(-a)-a*Ei(a,2))/2`
`Ei(a,4)=(exp(-a)-a*(exp(-a)-a*(exp(-a)+a*Ei(-a)))/2)/3=(exp(-a)-a*Ei(a`
`Ei(a,n)= (exp(-a)-a*Ei(a,n-1))/n-1` pour n entier ≥ 2
Donc :
`Ei(a,1)=-Ei(-a) a*Ei(a,1)+Ei(a,2)=exp(-a)` donc $a * Ei(a,1) + Ei(a,2) = exp(-a)$
`a*Ei(a,2)+2*Ei(a,3)=exp(-a)` donc $a^2 * Ei(a,1) - 2 * Ei(a,3) = exp(-a) * (a-1)$
`a*Ei(a,3)+3*Ei(a,4)=exp(-a)` donc $a^3 * Ei(a,1) + 3! * Ei(a,4) = exp(-a) * (a^2 - a + 2)$ `a*Ei(a,4)+4*Ei(a,4)=exp(-a)` donc
$a^4 * Ei(a,1) - 4! * Ei(a,4) = exp(-a) * (a^3 - a^2 + 2a - 3!)$
$a^5 * Ei(a,1) + 5! * Ei(a,4) = exp(-a) * (a^4 - a^3 + 2a^2 - 3!a + 4!)$
etc...

6.10.6 La fonction cosinus integral Ci : `Ci`

`Ci` a comme argument un nombre complexe a.
`Ci` calcule les valeurs de la fonction Ci au point a.
On a par définition :

$$Ci(x) = \int_{t=+\infty}^{x} \frac{\cos(t)}{t} dt = \ln(x) + \gamma + \int_0^x \frac{cos(t) - 1}{t}\, dt$$

On a : $Ci(0) = -\infty$, $Ci(-\infty) = i\pi$, $Ci(+\infty) = 0$. Lorsque l'on est proche de $x = 0$ on sait que

$$\frac{\cos(x)}{x} = \frac{1}{x} - \frac{x}{2} + \frac{x^3}{4!} + ... + (-1)^n \frac{x^{2n-1}}{(2n)!}....$$

ce qui donne par intégration le développement en séries de Ci.
On tape :

```
Ci(1.)
```

On obtient :

```
0.337403922901
```

On tape :

```
Ci(-1.)
```

On obtient :

```
0.337403922901+3.14159265359*i
```

On tape :

```
Ci(1.)-Ci(-1.)
```

On obtient :

```
-3.14159265359*i
```

On tape :

```
int((cos(x)-1)/x,x=-1..1.)
```

On obtient :

```
-3.14159265359*i
```

6.10.7 La fonction sinus integral Si : `Si`

`Si` a comme argument un nombre complexe a.
`Si` calcule les valeurs de la fonction Si au point a.
On a par définition

$$Si(x) = \int_{t=0}^{x} \frac{\sin(t)}{t} dt$$

On a $Si(0) = 0$, $Si(-\infty) = -\frac{\pi}{2}$, $Si(+\infty) = \frac{\pi}{2}$. Lorsque l'on est proche de $x = 0$ on sait que :

$$\frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + ... + (-1)^n \frac{x^{2n}}{(2n+1)!}....$$

ce qui donne par intégration le développement en séries de Si en 0. On observe aussi que `Si` est une fonction impaire.
On tape :

```
Si(1.)
```

On obtient :

```
0.946083070367
```

On tape :

```
Si(-1.)
```

On obtient :

```
-0.946083070367
```

On tape :

```
Si(1.)+Si(-1.)
```

On obtient :

```
0
```

On tape :

```
Si(1.)-Si(-1.)
```

On obtient :

```
1.89216614073
```

On tape :

```
int(sin(x)/x,x=-1..1.)
```

On obtient :

```
1.89216614073
```

6.10.8 La fonction de $Heaviside$: `Heaviside`

`Heaviside` a comme argument un nombre a.
`Heaviside` calcule les valeurs de la fonction $Heaviside$ au point a.
On a par définition :

$$Heaviside(x) = 0 \text{ si } x < 0 \text{ et } 1 \text{ sinon}$$

On tape :

```
Heaviside(2)
```

On obtient :

```
1
```

On tape :

```
Heaviside(-4)
```

On obtient :

```
0
```

6.10.9 La distribution de $Dirac$: `Dirac`

`Dirac` a comme argument un nombre a.
`Dirac` est la distribution de $Dirac$, c'est la distribution associée à la fonction `Heaviside`.
On a par définition :

$$Dirac(x) = 0 \text{ si } x \neq 0 \text{ et } \infty \text{ sinon}$$

et si $a \geq 0$ et $b \neq 0$ on a :

$$\int_b^a Dirac(x)dx = 1$$

$$\int_b^a Dirac(x)f(x)dx = [Heaviside(x)f(x)]_b^a - \int_b^a Heaviside(x)f'(x)dx = f(0)$$

$$\int_{-\infty}^{+\infty} Dirac(x) * f(x)dx = f(0)$$

On tape :

```
int(Dirac(x)*sin(x),x,-1,2)
```

On obtient :

```
sin(0)
```

On tape :

```
int(Dirac(x-1)*sin(x),x,-1,2)
```

On obtient :

```
sin(1)
```

6.10.10 La fonction erf : `erf`

`erf` a comme argument un nombre a.
`erf` calcule les valeurs de la fonction erf au point a.
On a par définition :

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

On a :

$$erf(+\infty) = 1$$

$$erf(-\infty) = -1$$

En effet on sait que :

$$\int_0^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

On tape :

```
erf(1)
```

On obtient :

```
0.84270079295
```

On tape :

```
erf(1/(sqrt(2)))*1/2+0.5
```

On obtient :

```
0.841344746069
```

Remarque
Il y a une relation entre les fonctions `erf` et `normal_cdf` :
`normal_cdf`$(x) = \frac{1}{2} + \frac{1}{2}$`erf`$(\frac{x}{\sqrt{2}})$
En effet :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$
donc avec le changement de variables $t = u * \sqrt{2}$ on a :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = \frac{1}{2} + \frac{1}{2}erf(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

6.10.11 La fonction $erfc$: `erfc`

`erfc` a comme argument un nombre a.
`erfc` calcule les valeurs de la fonction $erfc$ au point a.
On a par définition :

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt = 1 - erf(x)$$

On a :

$$erfc(0) = 1$$

$$erfc() = -1$$

En effet on sait que :

$$\int_0^{+\infty} e^{-t^2} dt = \frac{\sqrt{\pi}}{2}$$

On tape :

```
erfc(1)
```

On obtient :

```
0.15729920705
```

On tape :

```
1- erfc(1/(sqrt(2)))*1/2
```

On obtient :

```
0.841344746069
```

Remarque
Il y a une relation entre les fonctions `erfc` et `normal_cdf` :
$\texttt{normal\_cdf}(x) = 1 - \frac{1}{2}\texttt{erfc}(\frac{x}{\sqrt{2}})$
En effet :
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$
donc avec le changement de variables $t = u * \sqrt{2}$
normal_cdf$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = 1 - \frac{1}{2}\text{erfc}(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

6.10.12 La fonction Γ : `Gamma`

`Gamma` a comme argument un nombre a.
`Gamma` calcule les valeurs de la fonction Γ au point a.
On a par définition :

$$\Gamma(a) = \int_0^{+\infty} e^{-t} t^{a-1} dt, \text{ si } a > 0$$

et on utilise la formule :

$$\Gamma(a+1) = a * \Gamma(a) \text{ si } a \text{ n'est pas un entier negatif}$$

Donc :

$$\Gamma(1) = 1$$

$$\Gamma(a+1) = a * \Gamma(a)$$

et ainsi :

$$\Gamma(n+1) = n!$$

On tape :

```
Gamma(5)
```

On obtient :

```
24
```

On tape :

```
Gamma(1/2)
```

On obtient :

```
sqrt(pi)
```

On tape :

```
Gamma(0.7)
```

On obtient :

`1.29805533265`

On tape :

`Gamma(-0.3)`

On obtient :

`-4.32685110883`

En effet : `Gamma(0.7)=-0.3*Gamma(-0.3)`
On tape :

`Gamma(-1.3)`

On obtient :

`3.32834700679`

En effet :
`Gamma(0.7)=-0.3*Gamma(-0.3)=(-0.3)*(-1.3)*Gamma(-1.3)`

6.10.13 La fonction γ incomplète : `igamma`

`igamma` a 2 ou 3 arguments, un nombre $a > 0$ et un nombre x et éventuellement 1 comme 3-ième argument si on veut que la fonction soit régularisée (i.e. divisée par `Gamma(a)`.
`igamma` est la fonction γ qui est la fonction Γ incomplète de paramètres a et x.
On a par définition :
`igamma(a,x)`=$\gamma(a, x) = \int_0^x e^{-t}t^{a-1}dt$, si $a > 0$ On tape :

`igamma(2.0,3.0)`

On obtient :

`0.800851726529`

On tape :

`igamma(4.0,3.0)`

On obtient :

`2.11660866731`

On tape :

`igamma(4.0,3.0,1)`

On obtient :

`0.352768111218`

car `Gamma(4)=6` et `2.11660866731/6=0.352768111218` On tape :

```
igamma(4.0,20.0)
```

On obtient :

```
5.99998077768
```

On tape :

```
igamma(4.0,20.0,1)
```

On obtient :

```
0.99999679628
```

car `Gamma(4)=6` et `5.99998077768/6=0.99999679628`

6.10.14 La fonction β : `Beta`

`Beta` a comme argument deux réels a, b ou trois réels a, b, p ou trois réels et 1 $a, b, p, 1$.

- avec 2 arguments a, b, `Beta` calcule les valeurs de la fonction β au point a, b de $\mathbb{R}^2$.
  On a par définition :
  $$\beta(x, y) = \frac{\Gamma(x) * \Gamma(y)}{\Gamma(x + y)}$$
  On a :
  $$\beta(1, 1) = 1$$
  $$\beta(n, 1) = \frac{1}{n}$$
  et :
  $$\beta(n, 2) = \frac{1}{n(n + 1)}$$
  On a :
  `Beta(a,b)`=$\int_0^1 t^(a - 1) * (1 - t)^(b - 1)dt$
  `Beta(a,b)` est défini pour a et b réels positifs (pour que l'intégrale $\int_0^1 t^(a - 1) * (1 - t)^(b - 1)dt$ soit convergente).
  On tape :
  ```
  Beta(5,2)
  ```
  On obtient :
  ```
  1/30
  ```
  On tape :
  ```
  simplify(Beta(5,-3/2))
  ```
  On obtient :
  ```
  256/15
  ```
  On tape :
  ```
  Beta(x,y)
  ```
  On obtient :
  ```
  Gamma(x)*Gamma(y)/Gamma(x+y)
  ```

On tape :

`Beta(5.1,2.2)`

On obtient :

`0.0242053671402`

- avec 3 arguments a, b, p c'est la fonction Beta incomplète pour p entre 0 et 1, c'est :
  `Beta(a,b,p)`=$\int_0^p t^(a-1)*(1-t)^(b-1)dt$, l'integrale va de 0 à p au lieu de 0 à 1 pour la fonction Beta. On tape :

  `Beta(5,2,0.5)`

  On obtient :

  `0.00364583333333`

- avec 4 arguments $a, b, p, 1$ si on met 1 en 4eme argument cela calcule la fonction Beta incomplète regularisée i.e. la fonction Beta incomplète qui est divisé par Beta(a,b). On tape :

  `Beta(5,2,0.5,1)`

  On obtient :

  `0.109375`

  en effet `Beta(5,2)=1/30` et `0.00364583333333*30=0.109375`

6.10.15 Les derivées de la fonction DiGamma : `Psi`

`Psi` a comme arguments un réel a et un entier n (par défault $n = 0$).
`Psi` est la valeur de la n-ième dérivée de la fonction DiGamma au point a.
La fonction DiGamma est la dérivée de $\ln(\Gamma(x))$.
On tape :

`Psi(3,1)`

On obtient :

`pi^2/6-5/4`

On peut ommettre le paramètre n lorsque $n = 0$.
Lorsque `Psi` a comme seul paramètre un nombre a, `Psi` renvoie la valeur de la fonction DiGamma au point a :
on a donc `Psi(a,0)=Psi(a)`.
On tape :

`Psi(3)`

On obtient :

`Psi(1)+3/2`

On tape :

`evalf(Psi(3))`

On obtient :

`.922784335098`

6.10.16 La fonction ζ : `Zeta`

`Zeta` a comme argument un réel x.
`Zeta` renvoie pour $x > 1$: $\displaystyle\sum_{n=1}^{+\infty} \frac{1}{n^x}$.
On tape :

```
Zeta(2)
```

On obtient :

```
pi^2/6
```

On tape :

```
Zeta(4)
```

On obtient :

```
pi^4/90
```

6.10.17 Les fonctions de Airy : `Airy_Ai` et `Airy_Bi`

`Airy_Ai` et `Airy_Bi` a comme argument un réel x.
`Airy_Ai` et `Airy_Bi` sont deux solutions indépendantes de l'équation :
$y'' - x * y = 0$.
On définit :
$Airy\_Ai(x) = (1/\pi) \int_0^\infty \cos(t^3/3 + x * t) dt$
$Airy\_Bi(x) = (1/\pi) \int_0^\infty (e^{-t^3/3} + \sin(t^3/3 + x * t)) dt$
On a aussi :
`Airy_Ai(x)` vérifie :
$\texttt{Airy\_Ai}(\texttt{x}) = \texttt{Airy\_Ai}(0) * \texttt{f}(\texttt{x}) + \texttt{Airy\_Ai}'(0) * \texttt{g}(\texttt{x})$ et
`Airy_Bi` vérifie :
$\texttt{Airy\_Bi}(\texttt{x}) = \sqrt{3}(\texttt{Airy\_Ai}(0) * \texttt{f}(\texttt{x}) - \texttt{Airy\_Ai}'(0) * \texttt{g}(\texttt{x}))$
où f et g sont deux séries entières solutions de $w'' - x * w = 0$:

$$f(x) = \sum_{k=0}^{\infty} 3^k \left(\frac{\Gamma(k + \frac{1}{3})}{\Gamma(\frac{1}{3})} \right) \frac{x^{3k}}{(3k)!}$$

$$g(x) = \sum_{k=0}^{\infty} 3^k \left(\frac{\Gamma(k + \frac{2}{3})}{\Gamma(\frac{2}{3})} \right) \frac{x^{3k+1}}{(3k+1)!}$$

On tape :

```
Airy_Ai(1)
```

On obtient :

```
0.135292416313
```

On tape :

```
Airy_Bi(1)
```

On obtient :

`1.20742359495`

On tape :

`Airy_Ai(0)`

On obtient :

`0.355028053888`

On tape :

`Airy_Bi(0)`

On obtient :

`0.614926627446`

6.11 Les permutations

Une permutation p de longueur n est une bijection de $[0..n-1]$ sur $[0..n-1]$ et est représentée par la liste : $[p(0), p(1), p(2)...p(n-1)]$.
Par exemple, la permutation p représentée par $[1, 3, 2, 0]$ est l'application de $[0, 1, 2, 3]$ sur $[0, 1, 2, 3]$ définie par : $p(0) = 1,\ p(1) = 3,\ p(2) = 2,\ p(3) = 0$.
Un cycle c d'ordre p est représenté par la liste $[(a_0, ..., a_{p-1})]$ $(0 \leq p \leq n-1)$ et c'est une permutation telle que :
$c(a_i) = a_{i+1}$ pour $(i = 0..p-2)$, $c(a_{p-1}) = a_0$ et $c(a_i) = a_i(i = p+1..n)$.
Un cycle c est représenté par la liste et une décomposition en cycles par une liste de listes.
Par exemple, le cycle c représenté par la liste $[3, 2, 1]$ est la permutation c définie par $c(3) = 2,\ c(2) = 1,\ c(1) = 3,\ c(0) = 0$ (qui est représenté en tant que permutation par la liste $[0, 3, 1, 2]$).

6.11.1 Permutation aléatoire : `randperm`

`randperm` a comme argument un entier n.
`randperm` renvoie une permutation aléatoire de $[0..n-1]$.
On tape :

`randperm(3)`

On obtient :

`[2,0,1]`

6.11.2 Permutation précédente : `prevperm`

`prevperm` a comme argument une permutation.
`prevperm` renvoie la permutation précédente dans l'ordre lexicographique, ou `undef` s'il n'y en a pas.
On tape :

`prevperm([0,3,1,2])`

On obtient :

`[0,2,3,1]`

6.11.3 Permutation suivante : `nextperm`

`nextperm` a comme argument une permutation.
`nextperm` renvoie la permutation suivante dans l'ordre lexicographique, ou `undef` s'il n'y en a pas.
On tape :

```
nextperm([0,2,3,1])
```

On obtient :

```
[0,3,1,2]
```

6.11.4 Décomposition en cycles : `permu2cycles`

`permu2cycles` a comme argument une permutation.
`permu2cycles` renvoie sa décomposition en cycles.
On tape :

```
permu2cycles([1,3,4,5,2,0])
```

On obtient :

```
[[0,1,3,5],[2,4]]
```

Dans la réponse les cycles d'ordre 1 sont omis sauf celui qui vaut $[n-1]$ (pour pouvoir déterminer la valeur de n).
On tape :

```
permu2cycles([0,1,2,4,3,5])
```

On obtient :

```
[[5],[3,4]]
```

On tape :

```
permu2cycles([0,1,2,3,5,4])
```

On obtient :

```
[[4,5]]
```

6.11.5 Produit de cycles : `cycles2permu`

`cycles2permu` a comme argument une liste de cycles.
`cycles2permu` renvoie la permutation (de longueur n la plus petite possible) égale au produit des cycles de la liste donnée en argument (voir `permu2cycles`).
On tape :

```
cycles2permu([[1,3,5],[2,4]])
```

On obtient :

```
[0,3,4,5,2,1]
```

On tape :

`cycles2permu([[2,4]])`

On obtient :

`[0,1,4,3,2]`

On tape :

`cycles2permu([[5],[2,4]])`

On obtient :

`[0,1,4,3,2,5]`

6.11.6 Transformer un cycle en permutation : `cycle2perm`

`cycle2perm` a comme argument un cycle.
`cycle2perm` renvoie la permutation de longueur la plus petite possible correspondant au cycle donné en argument (voir aussi `permu2cycles` et `cycles2permu`).
On tape :

`cycle2perm([1,3,5])`

On obtient :

`[0,3,2,5,4,1]`

6.11.7 Transformer une permutation en une matrice : `permu2mat`

`permu2mat` a comme argument une permutation p de longueur n.
`permu2mat` renvoie la matrice obtenue en permutant, selon la permutation p, les lignes de la matrice identité d'ordre n.
On tape :

`permu2mat([2,0,1])`

On obtient :

`[[0,0,1],[1,0,0],[0,1,0]]`

Remarques
Pour faire une permutation sur les composantes d'un vecteur, par exemple faire agir la permutation `[2,0,1]` sur `[10,20,30]`, on tape :
`P:=permu2mat([2,0,1]);V:=[10,20,30];P*V`
ou bien, on tape `p:=[2,0,1];V:=[10,20,30];V[p[j]]$(j=0..2) :`
et on obtient `[30,10,20]`.
Pour faire une permutation sur les lignes d'une matrice carrée, par exemple faire agir la permutation `[2,0,1]` sur les lignes de `[[1,2,3],[4,5,6],[7,8,9]]`, on tape :
`P:=permu2mat([2,0,1]);A:=[[1,2,3],[4,5,6],[7,8,9]];P*A` et on obtient `[[4,5,6],[7,8,9],[1,2,3]]`.
Pour faire une permutation sur les colonnes d'une matrice carrée, par exemple faire agir la permutation `[2,0,1]` sur les colonnes de `[[1,2,3],[4,5,6],[7,8,9]]`, on tape :
`P:=permu2mat([2,0,1]);A:=[[1,2,3],[4,5,6],[7,8,9]];A*tran(P))`
et on obtient `[[3,1,2],[6,4,5],[9,7,8]]`.

6.11.8 Reconnaitre une permutation : `is_permu`

`is_permu` a comme argument une liste de dimension s.
`is_permu` est une fonction booléenne qui renvoie 1 ou 0 selon que l'argument est ou n'est pas une permutation de $0, 1, s-1$.
On tape :

```
is_permu([2,1,3])
```

On obtient :

```
0
```

On tape :

```
is_permu([2,1,3,0])
```

On obtient :

```
1
```

6.11.9 Reconnaitre un cycle : `is_cycle`

`is_cycle` est une fonction booléenne qui renvoie 1 ou 0 selon que l'argument est ou n'est pas un cycle.
On tape :

```
is_cycle([2,1,3])
```

On obtient :

```
1
```

On tape :

```
is_cycle([2,1,3,2])
```

On obtient :

```
0
```

6.11.10 Composition de deux permutations : `p1op2`

`p1op2` a comme arguments deux permutations.
`p1op2` renvoie la permutation obtenue par la composition :

$$1^{\text{er}}\text{arg} \circ 2^{\text{ième}}\text{arg}$$

On tape :

```
p1op2([3,4,5,2,0,1],[2,0,1,4,3,5])
```

On obtient :

```
[5,3,4,0,2,1]
```

Attention
C'est la permutation donnée comme deuxième argument qui est effectuée la première.

6.11.11 Produit d'un cycle et d'une permutation : `c1op2`

`c1op2` a comme arguments un cycle et une permutation.
`c1op2` renvoie la permutation obtenue par la composition :

$$1^{\text{er}}\text{arg} \circ 2^{\text{ième}}\text{arg}$$

On tape :

`c1op2([3,4,5],[2,0,1,4,3,5])`

On obtient :

`[2,0,1,5,4,3]`

Attention
C'est la permutation donnée comme deuxième argument qui est effectuée la première.

6.11.12 Produit d'une permutation et d'un cycle : `p1oc2`

`p1oc2` a comme arguments une permutation et un cycle.
`p1oc2` renvoie la permutation obtenue par la composition :

$$1^{\text{er}}\text{arg} \circ 2^{\text{ième}}\text{arg}$$

On tape :

`p1oc2([3,4,5,2,0,1],[2,0,1])`

On obtient :

`[4,5,3,2,0,1]`

Attention
C'est le cycle donné comme deuxième argument qui est effectué en premier.

6.11.13 Produit de deux cycles : `c1oc2`

`c1oc2` a comme arguments deux cycles.
`c1oc2` renvoie la permutation obtenue par la composition :

$$1^{\text{er}}\text{arg} \circ 2^{\text{ième}}\text{arg}$$

On tape :

`c1oc2([3,4,5],[2,0,1])`

On obtient :

`[1,2,0,4,5,3]`

Attention
C'est le cycle donné comme deuxième argument qui est effectué en premier.

6.11.14 Signature d'une permutation : `signature`

`signature` a comme argument une permutation.
`signature` renvoie la signature de la permutation donnée en argument.
La signature d'une permutation vaut :

- 1 si elle peut se décomposer en un produit pair de transpositions,
- -1 si elle peut se décomposer en un produit impair de transpositions.

La signature d'un cycle d'ordre k est : $(-1)^{k+1}$.
On tape :

```
signature([3,4,5,2,0,1])
```

On obtient :

```
-1
```

En effet `permu2cycles([3,4,5,2,0,1])=[[0,3,2,5,1,4]]`.

6.11.15 Inverse d'une permutation : `perminv`

`perminv` a comme argument une permutation.
`perminv` renvoie la permutation inverse de la permutation donnée en argument.
On tape :

```
perminv([1,2,0])
```

On obtient

```
[2,0,1]
```

6.11.16 Inverse d'un cycle : `cycleinv`

`cycleinv` a comme argument un cycle.
`cycleinv` renvoie le cycle inverse du cycle donné en argument.
On tape :

```
cycleinv([2,0,1])
```

On obtient

```
[1,0,2]
```

6.11.17 Ordre d'une permutation : `permuorder`

`permuorder` a comme argument une permutation.
`permuorder` renvoie l'ordre k de la permutation p donnée en argument (c'est le plus petit entier m telle que p^m soit l'identité).
On tape :

```
permuorder([0,2,1])
```

On obtient

On tape :

```
permuorder([3,2,1,4,0])
```

On obtient

```
6
```

6.11.18 Groupe engendré par deux permutations : `groupermu`

`groupermu` a comme argument deux permutations `a` et `b`.
`groupermu` renvoie le groupe des permutations engendré par `a` et `b`.
On tape :

```
groupermu([0,2,1,3],[3,1,2,0])
```

On obtient

```
[[0,2,1,3],[3,1,2,0],[0,1,2,3],[3,2,1,0]]
```

6.12 Les complexes

Vous trouverez dans le menu `Math (Cmplx)` les fonctions ayant comme paramètre une expression à valeur complexe.
Remarque
Les nombres complexes sont utilisés pour représenter un point sur l'écran graphique : par exemple, le graphe de $y = f(x)$ est l'ensemble des points $x + i * f(x)$ pour x variant entre `WX-` et `WX+` (`WX-` et `WX+` sont initialisés avec le menu `Cfg`▶`Configuration graphique`).

6.12.1 Les fonctions de base sur les complexes : `+,-,*,/,^`

`+,-,*,/,^` sont les opérateurs habituels pour faire des additions, des soustractions, des multiplications, des divisions et des élévations à une puissance entière ou fractionnaire.
On tape :

```
(1+2*i)^2
```

On obtient :

```
-3+4*i
```

6.12.2 La partie réelle d'un nombre complexe : `re real`

`re` (ou `real`) a comme argument un nombre complexe (resp un point A).
`re` (ou `real`) renvoie la partie réelle de ce nombre complexe (resp le point d'affixe la partie réelle de l'affixe de A).
On tape :

```
re(3+4*i)
```

On obtient :

```
3
```

6.12.3 La partie imaginaire d'un nombre complexe : `im imag`

`im` (ou `imag`) a comme argument un nombre complexe (resp un point A).
`im` (ou `imag`) renvoie la partie imaginaire de ce nombre complexe (resp le point d'affixe la partie imaginaire de l'affixe de A).
On tape :

```
im(3+4*i)
```

On obtient :

```
4
```

6.12.4 Écriture des complexes sous la forme `re(z)+i*im(z)` : `evalc`

`evalc` a comme argument un nombre complexe `z`.
`evalc` renvoie ce nombre complexe, écrit sous la forme `re(z)+i*im(z)`.
On tape :

```
evalc(sqrt(2)*exp(i*pi/4))
```

On obtient :

```
1+i
```

6.12.5 Le module d'un nombre complexe : `abs`

`abs` a comme argument un nombre complexe.
`abs` renvoie le module de ce nombre complexe.
On tape :

```
abs(3+4*i)
```

On obtient :

```
5
```

6.12.6 L'argument d'un nombre complexe : `arg`

`arg` a comme argument un nombre complexe.
`arg` renvoie l'argument de ce nombre complexe.
On tape :

```
arg(3+4.i)
```

On obtient :

```
atan(4/3)
```

6.12.7 Le nombre complexe normalisé : `normalize unitV`

`normalize` ou `unitV` a comme argument un nombre complexe.
`normalize` ou `unitV` renvoie le nombre complexe divisé par le module de ce nombre complexe.
On tape :

```
normalize(3+4*i)
```

On obtient :

```
(3+4*i)/5
```

6.12.8 Le nombre complexe conjugué : `conj`

`conj` a comme argument un nombre complexe.
`conj` renvoie le complexe conjugué de ce nombre complexe.
On tape :

```
conj(3+4*i)
```

On obtient :

```
3-4*i
```

6.12.9 Multiplier par le complexe conjugué : `mult_c_conjugate multiplier_conjugue_complexe`

Si une expression a un dénominateur complexe, `mult_c_conjugate` multiplie le numérateur et le dénominateur de cette expression par le complexe conjugué du dénominateur.
Si une expression n'a pas de dénominateur complexe, `mult_c_conjugate` multiplie le numérateur et le dénominateur de cette expression par le complexe conjugué du numérateur.
On tape :

```
mult_c_conjugate((2+i)/(2+3*i))
```

On obtient :

```
(2+i)*(2+3*(-i))/((2+3*(i))*(2+3*(-i)))
```

On tape :

```
mult_c_conjugate((2+i)/2)
```

On obtient :

```
(2+i)*(2+-i)/(2*(2+-i))
```

6.12.10 Barycentre de complexes : `barycenter barycentre`

Voir aussi : 9.9.10 et 10.4.7.
`barycentre` a comme argument soit des listes de longueur 2 : chaque liste représente un point A_j ou l'affixe a_j d'un point suivi par un coefficient réel α_j, soit une matrice ayant deux colonnes : la première colonne contient des points A_j ou des nombres complexes a_j représentant l'affixe de ces points et la deuxième colonne contient des coefficients réels α_j.
`barycentre` renvoie le point ou l'affixe du point qui est le barycentre des points A_j d'affixes a_j affectés des coefficients réels α_j lorsque $\sum \alpha_j \neq 0$. Si $\sum \alpha_j = 0$, `barycentre` renvoie une erreur.
Attention pour avoir un nombre complexe il faut demander l'affixe du barycentre, sinon vous avez le tracé du point barycentre dans l'écran géométrique.
On tape :

```
affixe(barycentre([1+i,2],[1-i,1]))
```

Ou on tape :

```
affixe(barycentre([[1+i,2],[1-i,1]]))
```

On obtient :

```
(3+i)/3
```

6.13 Les expressions algébriques

6.13.1 Qu'est-ce qune expression

Il ne faut pas confondre expression et fonction.
Une expression est une combinaison de nombres et de variables reliés par des opérations alors qu'une fonction associe à une variable une expression.
Par exemple :
`a:=x^2+2*x+1` définit une expression et `b(x):=x^2+2*x+1` définit une fonction.
On obtient la valeur de l'expression `a` en 0, avec `subst(a,x=0)` et la valeur de la fonction `b` en 0, avec `b(0)`.
On peut aussi considérer une expression comme un arbre.
Par exemple, si `A:=3+2*x/y` ou si `B:=sin(2x+3)` on a :

- le sommet de l'arbre `A` est l'opérateur `'+'` et le sommet de l'arbre `B` est la fonction `'sin'`.
  On a :
  `sommet(A)` renvoie `'+'`
  `sommet(B)` renvoie `'sin'`
- les feuilles de l'arbre `A` sont `3,2*x/y` (car `'+'` est un opérateur binaire) et la feuille de l'arbre `B` est `2*x+3` (car `'sin'` est une fonction d'une variable).
  On a :
  `feuille(A)` renvoie `3,2*x/y`
  `feuille(B)` renvoie `2*x+3`

– la feuille `2*x/y` est elle-même un arbre de sommet `'*'` etc...
– Les variables `A` et `B` permettent de retrouver la structure d'arbre en effet :
`A[0],A[1],A[2],A[2,0],A[2,1],A[2,2],A[2,3],A[2,3,0],A[2,3,1]`
renvoie : `'+',3,2*x,'*',2,x,1/y,'inv',y`
`B[0],B[1,0],B[1,1],B[1,2],B[1,1,0],B[1,1,1],B[1,1,2]`
renvoie : `'sin','+',2*x,3,'*',2,x`

Attention
Si `C:=2*x-y` alors
`sommet(C)` renvoie `'+'` et
`feuille(C)` renvoie `2*x,-y`
car l'expression est `C` s'écrit `2*x+(-y)`
De même si `D:=x/3` alors
`sommet(D)` renvoie `'*'` et
`feuille(C)` renvoie `x,1/3`
car l'expression est `D` s'écrit `x*1/3`
On a alors :
`C[0],C[1],C[1,0],C[1,1],C[1,2],C[2],C[2,0],C[2,1]`
renvoie : `'+',2*x,'*',2,x,-y,'-',y` (`'-'` est le "moins" unaire).
Et :
`D[0],D[1],D[2],D[2,0],D[2,1],B[1,1,2]`
renvoie : `'*',x,1/3,'inv',3`
Remarque
Ce qui suit n'est valable que pour des programmeurs confirmés avec les manipulations qui suivent, les erreurs sont faciles !
On peut changer le sommet ou une feuille terminale en affectant l'une des variables
`A[0],A[1],A[2],A[2,0],A[2,1],A[2,2],A[2,3],A[2,3,0],A[2,3,1]`
ou `B[0],B[1,0],B[1,1],B[1,2],B[1,1,0],B[1,1,1],B[1,1,2]`
MAIS il faut faire cela avec prudence car le système ne fait pas de vérifications et n'enverra pas de message d'erreurs.
On tape :
`A:=3+2*x/y`
`A[0]:='*'` Et `A` renvoie `6*x/y` Maintenant :`A[1]` renvoie `6`
On peut taper :
`A[1]:=2+z` Maintenant `A` renvoie `(2+z)*x/y`
On tape :
`B:=sin(2*x+3)`
`B[0]:=cos` Et `B` renvoie `cos(2*x+3)` Puis, on tape :
`B[1,2]:=y` et `B` renvoie `cos(2*x+y)`

6.13.2 Pour évaluer une expression : `eval`

`eval` sert à évaluer une expression.
`eval` a un ou deux argument(s) : une expression et éventuellement le niveau souhaité de l'évaluation.
Il faut savoir que `Xcas` évalue toujours les expressions, sans avoir besoin de la commande `eval` : le niveau d'évaluation est indiqué dans la case `eval` de la `Configuration du CAS` du menu `Cfg` et vaut par défaut vaut 25.

La commande `eval` est surtout utile lorsqu'on veut évaluer une sous-expression dans l'éditeur d'expressions.
On tape :

```
a:=2
```

On obtient :

```
2
```

On tape :

```
eval(2+3*a)
```

ou

```
2+3*a
```

On obtient :

```
8
```

On tape :

```
purge(r);purge(p);a:=1+i*r
```

```
r:=p+1;p:=-4;
```

on peut alors avoir différentes évaluation de `a` selon le niveau dévaluation demandé :

– on tape :

  ```
  a
  ```

  On obtient :

  ```
  1-3*i
  ```

– on tape :

  ```
  eval(a,1)
  ```

  On obtient :

  ```
  1+(i)*r
  ```

– on tape :

  ```
  eval(a,2)
  ```

  On obtient :

  ```
  1+(i)*(p+1)
  ```

– on tape :

  ```
  eval(a,3)
  ```

  On obtient :

  ```
  1-3*i
  ```

Remarque
Pour les objets géométriques, en plus de l'évaluation exacte (au niveau 25 par défaut), `Xcas` rajoute une évaluation numérique (au niveau 1) au moment de l'affichage pour pouvoir représenter les objets géométriques dependant de paramètres définis par `assume` ou par une affectation numérique.

Voici différents exemples :

– On tape :

```
purge(r);
R:=point(1+i*r);
r:=-3;
```

Le niveau correspondant à `R:=point(1+i*r);` affichera dans tous les cas le point et sa légende car l'évaluation numérique pour l'affichage de ce niveau est faite au moment de l'affichage donc après que `r` ait été défini.

– On tape :

```
purge(r);
purge(p);
R:=point(1+i*r);
r:=p+1;
p:=-4;
```

Le point `R` n'apparait pas car l'évaluation numérique au moment de l'affichage n'est faite qu'au niveau 1. Ainsi `r` est remplacé par `p` mais `p` n'est pas remplacé donc la commande `R:=point(1+i*r);` n'affiche rien.

– On tape

```
purge(r);
R:=point(1+i*r):;
r:=-3;
eval(R,1);
```

La commande `eval(R,1)` renvoie `point(1+(i)*r)` et dessine le point `R` et sa légende. En effet la réponse est évaluée formellement au niveau 1 ce qui donne la réponse `point(1+i*r)` puis pour la représentation graphique, `point(1+i*r)` est évalué numériquement (sans toucher aux légendes) ce qui permet d'afficher le point. La légende n' apparait pas, lorsqu'on évalue un objet géométrique, mais ici, lorsqu'on fait `eval(R,1)`, `R` est évalué en un objet géométrique, mais l'objet géométrique lui-même n'est pas évalué. Donc `eval(R,1)` dessine le point `R` et sa légende.

– On tape

```
purge(r);
purge(p);
R:=point(1+i*r):;
r:=p+1;
p:=-4;
eval(R,1);
```

La commande `eval(R,1)` renvoie `point(1+(i)*r)`, mais ne dessine pas le point `R`. En effet la réponse est évaluée formellement au niveau 1 ce qui donne la réponse `point(1+i*r)` puis pour la représentation graphique, `point(1+i*r)` est évalué numériquement au niveau 1 ce qui ne permet pas d'afficher le point.

– On tape :

```
purge(r);
R:=point(1+i*r):;
r:=-3;
eval(R,2);
```

La commande `eval(R,2)` renvoie `point(1,-3)` et le dessin du point `R` sans sa légende. En effet, quand on fait `eval(R,2)`, alors `R` est évalué en un objet géométrique, et cet l'objet géométrique est lui-même évalué et donc la légende disparait.

– On tape :
```
purge(r);
purge(p);
R:=point(1+i*r):;
r:=p+1;
p:=-4;
eval(R,2);
```
La commande `eval(R,2)` renvoie `point(1,p+1)` et le dessin du point `R` sans sa légende. En effet, quand on fait `eval(R,2)`, alors `R` est évalué en un objet géométrique, et cet l'objet géométrique est lui-même évalué et donc la légende disparait.

6.13.3 Pour changer le niveau dévaluation : `eval_level`

On peut alors avoir différentes évaluation d'une variable selon le niveau n dévaluation demandé : ce nombre n est le nombre de la case `eval` de la configuration du CAS qui représente le nombre maximum d'évaluations récursives en mode interactif (cf 1.6.1). On peut changer ce nombre avec `eval_level` qui a comme argument l'entier $n > 0$.
`eval_level()` renvoie la valeur de `eval` de la configuration du CAS.
`eval_level(0)` met la valeur de `eval` de la configuration du CAS à 0 et du coup plus rien ne sera évalué. On peut revenir en situation normale en changeant la valeur de `eval` de la configuration du CAS.
On tape :

```
purge(a,b,c);a:=b+1; b:=c+1;c:=3;
```

On tape :

```
eval_level(1)
```

puis

```
a,b,c
```

On obtient :

```
b+1,c+1,3
```

et `eval` de la configuration du CAS vaut 1.
On tape :

```
purge(a,b,c);a:=b+1; b:=c+1;c:=3;
```

On tape :

```
eval_level(2)
```

puis

```
a,b,c
```

On obtient :

```
c+1+1,4,3
```

et `eval` de la configuration du CAS vaut 2.
On tape :

```
purge(a,b,c);a:=b+1; b:=c+1;c:=3;
```

On tape :

```
eval_level(3)
```

puis

```
a,b,c
```

On obtient :

```
5,4,3
```

et `eval` de la configuration du CAS vaut 3.
On tape ensuite :

```
eval_level()
```

On obtient :

```
3
```

ce qui veut dire que `eval` de la configuration du CAS vaut 3.
Remarque
On peut préciser le niveau dévaluation d'une variable sans changer la configuration du CAS en utilisant `eval` avec 2 arguments une expression et son niveau d'évaluation. On tape :

```
purge(a,b,c);a:=b+1; b:=c+1;c:=3;
```

puis

```
eval(a,0),eval(a,1),eval(a,2),eval(a,3)
```

On obtient :

```
a,b+1,c+1+1,5
```

Attention
`eval_level` est fait pour être exécuté seul, sinon il y aura forcément des effets de frontière entre la valeur du niveau d'évaluation utilisé pour les commandes de la ligne (en principe le prcédent) et la nouvelle valeur qui s'appliquera après. Si on tape :

```
purge(a,b,c);a:=b+1; b:=c+1;c:=3;
```

puis sur un même niveau :

```
eval_level(3);a,b,c
```

si `eval` de configuration du CAS vaut 1 alors `a,b,c` seront évalués au niveau 1, puis `eval` de configuration du CAS vaudra 3.

6.13.4 Pour évaluer une expression en mode Maple : `evala`

`evala` sert en mode Maple à évaluer une expression contenant des extentions algébriques, par contre `Xcas` évalue toujours les expressions, sans avoir besoin de la commande `evala`.

6.13.5 Pour ne pas évaluer une expression : `quote hold ou '`

Si on ne veut pas qu'une expression soit évaluée dans un calcul, il faut la quoter, soit avec `'`, soit à l'aide de la fonction `quote` (ou `hold`).
Remarque Lorsqu'on tape par exemple `a:=quote(a)` (ou `a:=hold(a)`) cela a pour effet de purger la variable `a` et cette instruction renvoie la valeur de cette variable (ou les hypothèses faites sur cette variable).
Donc `a:=quote(a)` est synonyme de `purge(a)` (c'est pour avoir la compatibilité Maple).
On tape :

```
a:=2;quote(2+3*a)
```

ou

```
a:=2;'2+3*a'
```

On obtient :

```
(2,2+3.a)
```

6.13.6 Pour forcer à évaluer une expression : `unquote`

Si on veut qu'une expression quotée soit évaluée dans un calcul, il faut utiliser la fonction `unquote`.
Par exemple dans une affectation, la variable est quotée c'est à dire non évaluée.
On peut forcer son évaluation pour cela on tape :

```
purge(b);a:=b;unquote(a):=3
```

On obtient :

```
a et b contiennent 3
```

6.13.7 Distributivité : `expand fdistrib developper`

Si on veut effectuer, sur une expression, la distributivité de la multiplication par rapport à l'addition on utilise la fonction `expand` ou `developper` ou `fdistib`.
Ou on tape :

```
developper((x+1)*(x-2))
```

Ou on tape :

```
expand((x+1)*(x-2))
```

Ou on tape :

```
fdistrib((x+1)*(x-2))
```

On obtient :

```
x^2-x-2
```

Remarque On peut aussi utiliser `convert` avec l'option `'+'` ou sa version infixée=> avec l'option + : On tape :

```
(x+1)*(x-2)=>+
```

Ou on tape :

```
convert((x+1)*(x-2),'+')
```

On obtient :

```
x^2-x-2
```

6.13.8 Forme canonique : `canonical_form`

`canonical_form` a comme paramètre un trinôme du second degré que l'on veut mettre sous la forme canonique.
Exemple :
Mettre sous la forme canonique :

$$x^2 - 6x + 1$$

On tape :

```
canonical_form(x^2-6*x+1)
```

On trouve :

```
(x-3)^2-8
```

6.13.9 Multiplier par la quantité conjuguée : `mult_conjugate multiplier_conju`

`mult_conjugate` a comme argument une expression avec un dénominateur ou un numérateur comportant des racines carrées :

- `mult_conjugate` a comme argument une expression avec un dénominateur comportant des racines carrées.
  `mult_conjugate` multiplie le numérateur et le dénominateur de cette expression par la quantité conjuguée du dénominateur.
- `mult_conjugate` a comme argument une expression avec un dénominateur ne comportant pas de racines carrées.
  `mult_conjugate` multiplie le numérateur et le dénominateur de cette expression par la quantité conjuguée du numérateur.

On tape :

```
mult_conjugate((2+sqrt(2))/(2+sqrt(3)))
```

On obtient :

```
(2+sqrt(2))*(2-sqrt(3))/((2+sqrt(3))*(2-sqrt(3)))
```

On tape :

```
mult_conjugate((2+sqrt(2))/(sqrt(2)+sqrt(3)))
```

On obtient :

```
(2+sqrt(2))*(-sqrt(2)+sqrt(3))/
((sqrt(2)+sqrt(3))*(-sqrt(2)+sqrt(3)))
```

On tape :

```
mult_conjugate((2+sqrt(2))/2)
```

On obtient :

```
(2+sqrt(2))*(2-sqrt(2))/(2*(2-sqrt(2)))
```

6.13.10 Séparation des variables : split

`split` a deux arguments : une expression dépendant de deux variables et la liste de ces deux variables.
Si l'expression est factorisable avec deux facteurs qui ne dependent chacun que d'une des 2 variables, `split` renvoie une liste formée par ces deux facteurs et sinon renvoie la liste `[0]`.
On tape :

```
split((x+1)*(y-2),[x,y])
```

Ou on tape :

```
split(x*y-2*x+y-2,[x,y])
```

On obtient :

```
[x+1,y-2]
```

On tape :

```
split(x^2*y^2-1,[x,y])
```

On obtient :

```
[0]
```

6.13.11 Factorisation : `factor factoriser`

`=>*` est la version postfixée de `factor`.
`factor` ou `factoriser` a comme paramètre une expression.
`factor` ou `factoriser` factorise cette expression sur le corps de ses coefficients si `Complex` et `Sqrt` sont décochés dans l'écran de configuration du `CAS`.
Remarque On peut aussi utiliser `convert` avec l'option `'*'` ou sa version infixée=> avec l'option `*` : On tape :

```
x^2-x-2=>*
```

Ou on tape :

```
convert(x^2-x-2,'*')
```

On obtient :

```
(x-2)*(x+1)
```

Exemples :

1. Factoriser dans $\mathbb{Z}$ (on doit décoche `Complex` dans l'écran de configuration du `CAS`) :
$$x^4 - 1$$
On tape :
```
factor(x^4-1)
```
Ou on tape :
```
x^4-1=>*
```
On obtient :
```
(x^2+1)*(x+1)*(x-1)
```
Les coefficients sont entiers donc la factorisation se fera avec des polynômes à coefficients entiers.

2. Factoriser dans $\mathbb{C}$:
$$x^4 - 1$$
Pour avoir une factorisation complexe, on coche `Complex` dans l'écran de configuration du `CAS` (bouton donnant la ligne d'état) on tape :
```
factor(x^4-1)
```
Ou on tape :
```
x^4-1=>*
```
On obtient :
```
(x-1)*(x+1)*(x+i)*(x-i)
```

3. Factoriser dans $\mathbb{Z}$ (on doit décoche `Complex` dans l'écran de configuration du `CAS`) :
$$x^4 + 1$$
On tape :

```
factor(x^4+1)
```

Ou on tape :

```
x^4+1=>*
```

On obtient :

```
x^4+1
```

car $x^4 + 1$ ne se fatorise pas sur les entiers.

4. Factoriser sur les entiers de Gauss :

$$x^4 + 1$$

Si l'on veut une factorisation sur les entiers de Gauss, on coche `Complex` dans l'écran de configuration du `CAS` (bouton donnant la ligne d'état), on tape :

```
factor(x^4+1)
```

Ou on tape :

```
x^4+1=>*
```

On obtient :

```
(x^2+i)*(x^2+-i)
```

5. Factoriser dans $\mathbb{R}$:

$$x^4 + 1$$

Si l'on veut une factorisation réelle, afin de connaitre le réel qui sert dans la factorisation, on coche `Complex` dans l'écran de configuration du `CAS` et on tape tout d'abord :

```
solve(x^4+1,x)
```

On obtient :

```
[sqrt(2)/2+(i)*sqrt(2)/2,sqrt(2)/2+(i)*(-(sqrt(2)/2)),

-sqrt(2)/2+(i)*sqrt(2)/2,-sqrt(2)/2+(i)*(-(sqrt(2)/2))]
```

On voit que les racines dépendent de $\sqrt{2}$ donc on tape :

```
factor(sqrt(2)*(x^4+1))
```

On obtient :

```
sqrt(2)*(x^2+sqrt(2)*x+1)*(x^2+(-(sqrt(2)))*x+1)
```

Ou bien on tape :

```
factor(x^4+1,sqrt(2)))
```

On obtient :

```
(x^2+sqrt(2)*x+1)*(x^2+(-(sqrt(2)))*x+1)
```

Remarques
Si on coche `Sqrt` dans la configuration du `CAS`, les trinômes du second degré (et seulement du second degré) seront factorisés même si les facteurs ne sont pas dans le corps de base des coefficients.
Pour factoriser dans $\mathbb{C}$ l'expression x^4+1, il faut cocher `Complex` et `Sqrt` dans l'écran de configuration du `CAS` et taper :
`factor(x^4+1)`
ou taper (avec juste `Sqrt` coché) :
`cfactor(x^4+1)` (cf `cfactor`).

6.13.12 Factorisation dans $\mathbb{C}$: `cFactor factoriser_sur_C cfactor`

`cFactor` ou `cfactor` ou `factoriser_sur_C` a comme paramètre une expression que l'on veut factoriser sur le corps des complexes sans avoir besoin d'être en mode complexe. Lorsqu'il y a plus de 2 variables la factorisation se fait sur les entiers de Gauss.
Exemples

1. Factoriser dans $\mathcal{C}$:
$$x^4-1$$
On tape :
```
cFactor(x^4-1)
```
On obtient :
```
-((x+-i)*((-i)*x+1)*((-i)*x+i)*(x+1))
```
2. Factoriser dans $\mathcal{C}$:
$$x^4+1$$
On tape :
```
cFactor(x^4+1)
```
On obtient :
```
(x^2+i)*(x^2+-i)
```
Puis, on tape :
```
cFactor(sqrt(2)*(x^2+i))*cFactor(sqrt(2)*(x^2+-i))
```
On obtient :
```
sqrt(2)*1/2*(sqrt(2)*x+1-i)*(sqrt(2)*x-1+i)*sqrt(2)*
        1/2*(sqrt(2)*x+1+i)*(sqrt(2)*x-1-i)
```
Mais si on tape, :
```
cFactor(sqrt(2)*(x^4+1))
```
On obtient :
```
sqrt(2)*(x^2+sqrt(2)*x+1)*(x^2+(-(sqrt(2)))*x+1)
```

6.13.13 Zéros d'une expression : `zeros`

`zeros` a comme paramètre une expression.
`zeros` renvoie la liste des éléments qui annulent l'expression.
Selon le mode choisi, si on est en mode réel (`complex_mode:=0`) les zéros seront réels et si on est en mode complexe (`complex_mode:=1`) les zéros seront complexes.
On tape en mode réel :

```
zeros(x^2+4)
```

On obtient :

```
[]
```

On tape en mode complexe :

```
zeros(x^2+4)
```

On obtient :

```
[-2*i,2*i]
```

On tape en mode réel :

```
zeros(ln(x)^2-2)
```

On obtient :

```
[exp(sqrt(2)),exp(-(sqrt(2)))]
```

On tape en mode réel :

```
zeros(ln(y)^2-2,y)
```

On obtient :

```
[exp(sqrt(2)),exp(-(sqrt(2)))]
```

On tape en mode réel :

```
zeros(x*(exp(x))^2-2*x-2*(exp(x))^2+4)
```

On obtient :

```
[ln(2)/2,2]
```

6.13.14 Zéros complexe d'une expression : `cZeros`

`cZeros` a comme paramètre une expression.
`cZeros` renvoie la liste des éléments complexes qui annulent l'expression.
Remarque
Différence entre `zeros` et `cZeros` : En mode complexe `zeros` renvoie le même résultat que `cZeros` (pour `cZeros` que l'on soit en mode complexe ou réel cela importe peu). Ainsi, si on ne veut pas que le résultat dépende du mode, pour avoir les solutions complexes il est préférable d'utiliser `cZeros`.
On tape en mode réel ou complexe :

```
cZeros(x^2+4)
```

On obtient :

```
[-2*i,2*i]
```

On tape :

```
cZeros(ln(x)^2-2)
```

On obtient :

```
[exp(sqrt(2)),exp(-(sqrt(2)))]
```

On tape :

```
cZeros(ln(y)^2-2,y)
```

On obtient :

```
[exp(sqrt(2)),exp(-(sqrt(2)))]
```

On tape :

```
cZeros(x*(exp(x))^2-2*x-2*(exp(x))^2+4)
```

On obtient :

```
[[log(sqrt(2)),log(-sqrt(2)),2]]
```

6.13.15 Regrouper et simplifier : `regrouper regroup`

`regrouper` a comme paramètre une expression.
`regrouper` effectue les simplifications évidentes sur une expression en regroupant des termes.
On tape :

```
regrouper(x+3*x+5*4/x)
```

On obtient :

```
20/x+4*x
```

6.13.16 Développer et simplifier : `normal`

`normal` a comme paramètre une expression.
`normal` renvoie l'expression développée et simplifiée.
On tape :

```
normal(x+3*x+5*4/x)
```

On obtient :

```
(4*x^2+20)/x
```

On tape :

```
normal((x-1)*(x+1))
```

On obtient :

```
x^2-1
```

Attention `normal` est moins efficace que `simplify` et on est quelquefois obligé de faire plusieurs fois la commande `normal`.
On tape :

```
normal(3-54*sqrt(1/162))
```

On obtient :

```
(-9*sqrt(2)+9)/3
```

On tape :

```
normal((-9*sqrt(2)+9)/3)
```

On obtient :

```
-(3*sqrt(2))+3
```

6.13.17 Simplifier : `simplify` `simplifier`

`simplify` simplifie l'expression de façon automatique.
On tape :

```
simplify((x-1)*(x+1))
```

On obtient :

```
x^2-1
```

On tape :

```
simplify(3-54*sqrt(1/162))
```

On obtient :

```
-3*sqrt(2)+3
```

Attention `simplify` est plus efficace lorsqu'on est en mode `radian` pour simplifier des expressions trigonométriques (pour cela on coche `radian` dans la configuration du `cas` ou bien on tape `angle_radian:=1`).
On tape :

```
simplify((sin(3*x)+sin(7*x))/sin(5*x))
```

On obtient :

```
4*(cos(x))^2-2
```

6.13.18 Pour réécrire les résultats selon son choix : `autosimplify`

`autosimplify` a pour argument une commande (comme `factor`, `simplify` ...) qui sera utilisée pour réécrire les résultats dans `Xcas` (la valeur de l'argument de `autosimplify` au lancement est `regroup`).
Si on ne veut pas de simplifications automatique il faudra taper au début de la session : `autosimplify(nop)`.
Si on veut changer le mode de simplification en cours de session il faudra que le nouveau `autosimplify(...)` se trouve seul sur une ligne. On tape en début de session :

```
autosimplify(1)
```

ou

```
autosimplify(regroup)
```

Puis on tape :

```
1+x^2-2
```

On obtient :

```
x^2-1
```

On tape sur une ligne :

```
autosimplify(factor)
```

Puis on tape :

```
1+x^2-2
```

On obtient :

```
(x-1)*(x+1)
```

On tape :

```
autosimplify(nop)
```

ou

```
autosimplify()
```

ou

```
autosimplify(0)
```

Puis on tape :

```
1+x^2-2
```

On obtient :

```
1+x^2-2
```

On tape :

```
autosimplify(regroup)
```

Puis on tape :

```
1+x^2-2
```

On obtient comme en début de session :

```
x^2-1
```

6.13.19 Simplifier à l'aide de fractions rationnelles : `ratnormal`

`ratnormal` simplifie l'expression sous forme de fraction irréductible.
On tape :

```
ratnormal((x^3-1)/(x^2-1))
```

On obtient :

```
(x^2+x+1)/(x+1)
```

On tape :

```
ratnormal((-2x^3+3x^2+5x-6)/(x^2-2x+1))
```

On obtient :

```
(-2*x^2+x+6)/(x-1)
```

6.13.20 Substituer une valeur à une variable : `|`

`|` est une fonction infixée qui substitue des valeurs à des variables : `|` a deux arguments : une expression dependant d'un paramètre et une égalité (paramètre=valeur de substitution,paramètre=valeur de substitution,...) . On tape :

```
a^2+1|a=2
```

On obtient même si la variable `a` est affectée :

```
5
```

On tape :

```
a^2+b|a=2,b=3
```

On obtient même si les variables `a` et `b` sont affectées :

```
7
```

6.13.21 Substituer une valeur à une variable : `subst substituer`

`subst` a deux ou trois arguments : une expression dépendant d'un paramètre et une égalité (paramètre=valeur de substitution) ou une expression dependant d'un paramètre , le paramètre et la valeur de substitution.

– `subst` effectue la substitution demandée dans l'expression à condition que le paramètre ne soit pas affecté car `subst` évalue tout d'abord l'expression et remplace donc le paramètre (si il a été affecté) par sa valeur sans tenir compte de la valeur de substitution donné par le deuxième paramètre.
  On tape :
  ```
  subst(a^2+1,a=2)
  ```
  ou :
  ```
  subst(a^2+1,a,2)
  ```
  On obtient si la variable `a` n'est pas affectée :
  ```
  5
  ```

Si la variable `a` est affectée, il faut taper auparavant `purge(a)` pour obtenir `5`.
Lorsque l'on veut substituer plusieurs variables, pour eviter de faire plusieurs substitutions à la suite, on met comme deuxième argument la liste de ces variables et comme troisième argument la liste de les valeurs de substitution (ou encore on met comme deuxième argument la liste formée des noms de variables = valeur de substitution).
On tape :

```
subst(a^2+b,[a,b],[2,1])
```

Ou on tape :

```
subst(a^2+b,[a=2,b=1])
```

On obtient si les variables `a` et `b` ne sont pas affectées :

```
2^2+1
```

– `subst` permet aussi d'effectuer des changements de variables dans une intégrale mais `subst` ne gère les changements de variable dans une integrale que si le changement de variable est de la forme `x=f(u)`. Dans ce cas il faut quoter l'intégrale pour que celle-ci ne soit pas calculée si on utilise `integrate` ou bien il faut utiliser la commande `Int`. Dans les deux cas il faut spécifier le nom de la variable d'intégration même si celle-ci est `x`.
On tape :

```
subst('integrate(sin(x^2)*x,x,0,pi/2)',x=sqrt(t))
```

Ou on tape :

```
subst(Int(sin(x^2)*x,x,0,pi/2),x=sqrt(t))
```

On obtient

```
integrate(sin(t)*sqrt(t)*1/2*1/t*sqrt(t),t,0,(pi/2)^2)
```

On tape :

```
subst('integrate(sin(x^2)*x,x)',x=sqrt(t))
```

Ou on tape :

```
subst(Int(sin(x^2)*x,x),x=sqrt(t))
```

On obtient

```
integrate(sin(t)*sqrt(t)*1/2*1/t*sqrt(t),t)
```

6.13.22 Substituer une valeur à une variable (compatibilité Maple et Mupad) : `subs`

En `Maple` et en `Mupad` la commande synonyme de `subst` est `subs`, mais l'ordre des paramètres de `subs` n'est pas le même en `Maple` et en `Mupad`.
Ainsi les arguments de `subs` sont :

– En mode `Maple`, la fonction `subs` a deux arguments : une égalité (paramètre=valeur de substitution) et une expression dépendant du paramètre.
Pour faire plusieurs substitutions, `subs` a deux arguments : une liste dégalité (paramètre=valeur de substitution) et une expression dépendant de ces paramètres.

– En mode `Mupad` ou `Xcas` ou `TI`, la fonction `subs` a deux ou trois arguments : une expression dépendant d'un paramètre et une égalité (paramètre=valeur de substitution) ou une expression dépendant d'un paramètre,

le paramètre et la valeur de substitution.
Pour faire plusieurs substitutions, la fonction `subs` a deux ou trois arguments : une expression dépendant de paramètres et une liste d'égalité (paramètre=valeur de substitution) ou une expression dépendant de paramètres, la liste des paramètres et la liste des valeurs de substitution.

`subs` effectue la substitution demandée dans l'expression à condition que le paramètre ne soit pas affecté car `subs` évalue tout d'abord l'expression et remplace donc le paramètre par sa valeur sans tenir compte de la valeur de substitution donnée par le deuxième paramètre.
On tape en mode Maple :

```
subs(a=2,a^2+1)
```

On obtient (si la variable `a` n'est pas affectée, sinon il faut taper auparavant `purge(a)`) :

```
5
```

Lorsque l'on veut substituer plusieurs variables :
On tape, en mode `Maple` :

```
subs([a=2,b=1],a^2+b)
```

On obtient (si les variables `a` et `b` ne sont pas affectées, sinon il faut taper auparavant `purge(a,b)`) :

```
2^2+1
```

On tape, en mode `Mupad` ou `Xcas` ou `TI` :

```
subs(a^2+1,a=2)
```

ou :

```
subs(a^2+1,a,2)
```

On obtient (si la variable `a` n'est pas affectée, sinon il faut taper auparavant `purge(a)`) :

```
5
```

Lorsque l'on veut substituer plusieurs variables :
On tape, dans les modes `Mupad Xcas TI` :

```
subs(a^2+b,[a=2,b=1])
```

ou on tape

```
subs(a^2+b,[a,b],[2,1])
```

On obtient (si les variables `a` et `b` ne sont pas affectées, sinon il faut taper auparavant `purge(a,b)`) :

```
2^2+1
```

6.13.23 Substituer dans une expression, une expression algébrique par une variable : `algsubs`

`algsubs` permet de substituer dans une expression, une expression algébrique par une autre expression algébrique.
`algsubs` a 2 arguments :
une équation entre 2 expressions algébriques `Xpr1=Xpr2` et
une expression dans laquelle `algsubs` remplacera l'expression algébrique `Xpr1` par `Xpr2`.
On tape :

```
algsubs(x^2=u,1+x^2+x^4)
```

On obtient :

```
u^2+u+1
```

On tape :

```
algsubs(a*b/c=d, 2*a*b^2/c)
```

On obtient :

```
2*b*d
```

On tape :

```
algsubs(2a=p^2-q^2,algsubs(2c=p^2+q^2,c^2-a^2))
```

On obtient :

```
p^2*q^2
```

6.13.24 Éliminer une (ou des) variable(s) dans une liste d'équations : `eliminate`

`eliminate` permet d'éliminer une ou plusieurs variables dans une liste d'équations algébriques, équations algébriques qui sont verifiées par plusieurs variables.
`eliminate` a 2 arguments :
une liste d'équations `L1` et
la variable ou la liste des variables à éliminer.
`eliminate` renvoie la liste `L2` des équations qui sont vérifiées par les variables non éliminées et pour lesquelles `=0` est sous entendu.
On tape :

```
eliminate([x=v0*t,y=y0-g*t^2],t)
```

On obtient si la variable `t` n'est pas affectée et si les autres variables ne sont pas affectées (si les autres variables sont affectées, elles sont remplacées par leur valeurs ce qui modifie le résultat) :

```
[x^2*g-v0^2*y0+v0^2*y]
```

On tape :

```
eliminate([x=2*t,y=1-10*t^2,z=x+y-t],t)
```

On obtient si la variable `t` n'est pas affectée et si les variables `x,y,z` ne sont pas affectées :

```
[x+2*y-2*z,-10*y^2+20*y*z-y-10*z^2+1]
```

On tape :

```
eliminate([x+y+z+t-2,x*y*t=1,x^2+t^2=z^2],[x,z])
```

On obtient si les variables `x` et `z` ne sont pas affectées et si les autres variables ne sont pas affectées (si les autres variables sont affectées, elles sont remplacées par leur valeurs ce qui modifie le résultat) :

```
[y^3*t+2*y^2*t^2-4*y^2*t-4*y*t^2+4*y*t+2*y+2*t-4]
```

Attention
Si la réponse est `[1]` ou `[-1]`, cela veut dire que la ou les variables ne peuvent pas être élinminées.
Si la réponse est `[]`, cela veut dire que certaines équations déterminent les valeurs des variables à éliminer et que ces valeurs vérifient toutes les équations.
On tape :

```
x:=2;y:=-5
eliminate([x=2*t,y=1-10*t^2],t)
```

On obtient si la variable `t` n'est pas affectée :

```
[1]
```

On ne peut donc pas éliminer `t` des deux équations :
$2 = 2t, -5 = 1 - 10t^2$.
On tape :

```
x:=2;y:=-9
eliminate([x=2*t,y=1-10*t^2],t)
```

On obtient si la variable `t` n'est pas affectée :

```
[]
```

En effet la première équation :
$2 = 2t$ donne $t = 1$ et $t = 1$ vérifie la deuxième équation $-9 = 1 - 10t^2$.
On tape :

```
x:=2;y:=-9
eliminate([x=2*t,y=1-10*t^2,z=x+y-t],t)
```

On obtient si la variable `t` n'est pas affectée :

```
[z+8]
```

En effet la première équation :
$2 = 2t$ donne $t = 1$, $t = 1$ vérifie la deuxième équation $-9 = 1 - 10t^2$ et il reste la troisième équation $z = 2 - 9 - 1 = -8$ soit $z + 8 = 0$.

6.13.25 Évaluer une primitive : `preval`

`preval` a trois paramètres : une expression `F(x)` dépendant de la variable `x`, et deux expressions `a` et `b`.
`preval` effectue `F(b)-F(a)`.
`preval` est utile pour calculer une intégrale définie à partir d'une primitive : on calcule une primitive, puis on évalue cette primitive entre les deux bornes de l'intégrale.
On tape :

```
preval(x^2+x,2,3)
```

On obtient :

```
6
```

6.13.26 Sous-expression d'une expression : `part`

`part` a deux arguments : une expression et un entier n.
`part` évalue l'expression puis renvoie la n-ième sous-expression de l'expression.
On tape :

```
part(x^2+x+1,2)
```

On obtient :

```
x
```

On tape :

```
part(x^2+(x+1)*(y-2)+2,2)
```

On obtient :

```
(x+1)*(y-2)
```

On tape :

```
part((x+1)*(y-2)/2,2)
```

On obtient :

```
y-2
```

6.14 Valeurs de u_n

6.14.1 Tableau de valeurs des termes d'une suite : `tablefunc table_fonction`

`tablefunc` ou `table_fonction` est une commande qui s'utilise à l'intérieur d'un tableur (que l'on ouvre avec `Alt+t`) et qui .
remplit deux colonnes donnant la table des valeurs d'une fonction.
`tablefunc(ex,n,n0,1)`, où `ex` est une expression dépendant de `n`, remplira le tableur avec les valeurs de la suite $u_n = ex$ pour $n = n0,\ n0 + 1,\ n0 + 2, ...$

Exemple : Affichage des valeurs de $u_n = \sin(n)$ On ouvre un tableur avec `Alt+t`.
Puis, on sélectionne une case du tableur (par exemple `C0`) et on tape dans la ligne de commande du tableur :

```
tablefunc(sin(n),n,0,1)
```

On obtient :

```
deux colonnes : n et sin(n)
```

- dans la colonne `n` il y a la valeur du pas (qui doit être égal à 1) et la valeur de `n0` (ici 0), puis une formule `C2+C$1` qui a été recopiée vers le bas.
- dans la colonne `sin(n)` il y a `"Tablefunc"`, puis une formule qui a été aussi recopiée vers le bas.
  Les valeurs de la suite $\mathtt{u_n} = \sin(\mathtt{n})$ s'affichent alors en face des `n` correspondants à partir de `n=n0` (ici 0).

6.14.2 Valeurs d'une suite récurrente ou d'un système de suites récurrentes : `seqsolve`

Voir aussi `rsolve` 6.14.3.
`seqsolve` a comme argument l'expression ou la liste des expressions qui défini(ssen)t une/des relation(s) de récurrence, par exemple $f(x, n)$ si la relation de récurrence est $u_{n+1} = f(u_n, n)$ (resp $g(x, y, n)$ si la relation de récurrence est $u_{n+2} = g(u_n, u_{n+1}, n) = g(x, y, n)$), le nom des variables utilisées (par exemple $[x, n]$ (resp $[x, y, n]$)) et les valeurs de départ de la suite : par exemple a si $u_0 = a$ (resp $[a, b]$ si $u_0 = a$ et $u_1 = b$).

La relation de récurrence doit comporter une partie homogène linéaire, la partie non homogène doit être une combinaison linéaire de produit de polynôme en n par une suite géométrique en n. `seqsolve` renvoie alors la valeur de la suite en fonctions de n.

Exemples :

- Valeurs de la suite $u_0 = 3,\ u_{n+1} = 2u_n + n$
  On tape :
  ```
  seqsolve(2x+n,[x,n],3)
  ```
  On obtient :
  ```
  -n-1+4*2^n
  ```
  On peut aussi taper rsolve(u(n+1)=2*u(n)+n,u(n),u(0)=3) (cf6.14.3)
- Valeurs de la suite $u_0 = 3,\ u_{n+1} = 2u_n + n3^n$
  On tape :
  ```
  seqsolve(2x+n*3^n,[x,n],3)
  ```
  On obtient :
  ```
  (n-3)*3^n+6*2^n
  ```
- Valeurs de la suite $u_0 = 0, u_1 = 1,\ u_{n+1} = u_n + u_{n-1}$ pour $n > 0$.
  On tape :
  ```
  seqsolve(x+y,[x,y,n],[0,1])
  ```
  On obtient :
  ```
  (5+sqrt(5))/10*((sqrt(5)+1)/2)^(n-1)+
  (5-(sqrt(5)))/10*((-sqrt(5)+1)/2)^(n-1)
  ```
- Valeurs de la suite $u_0 = 0, u_1 = 1,\ u_{n+2} = 2 * u_{n+1} + u_n + n + 1$ pour $n > 0$.
  À la main, on trouve $u_2 = 3$, $u_3 = 9$ $u_4 = 24$ etc...
  On tape :

```
seqsolve(x+2y+n+1,[x,y,n],[0,1])
```

On obtient :

```
(-4*n-3*(-(sqrt(2)-1))^n*sqrt(2)+2*(-(sqrt(2)-1))^n+3*(sqrt(2)+1)
```

On vérifie pour `n:=4` on obtient bien `24`

Ou on tape car on a $u_{n+1} = 2u_n + v_n + n$ et $v_{n+1} = u_n$ (donc $v_n = u_{n-1}$) avec $u_0 = 0$ et $u_1 = 2u_0 + v_0 + 0 = 1$ donc $v_0 = 1$:

```
seqsolve([2x+y+n,x],[x,y,n],[0,1])
```

On obtient :

```
[(-1)/2-(-2-3*sqrt(2))/8*(sqrt(2)+1)^n-
(-2+3*sqrt(2))/8*(-sqrt(2)+1)^n-1/2*n,
-(-4+sqrt(2))/8*(sqrt(2)+1)^n-
(-4-sqrt(2))/8*(-sqrt(2)+1)^n-1/2*n]
```

On vérifie pour `n:=4` on obtient bien `24`

– Valeurs de la suite $u_0 = 0, v_0 = 1,\ u_{n+1} = u_n + 2v_n, v_{n+1} = u_n + n + 1$ pour $n > 0$.
  On tape :

  ```
  seqsolve([x+2*y,n+1+x],[x,y,n],[0,1])
  ```

  On obtient :

  ```
  [(-2*n-(-1)^n+2^n*4-3)/2,((-1)^n+2*2^n-1)/2]
  ```

– Valeurs de la suite $u_0 = 0, v_0 = 1,\ u_{n+1} = u_n + 2v_n + n + 1, v_{n+1} = u_n$ pour $n > 0$.
  On tape :

  ```
  seqsolve([x+2*y+n+1,x],[x,y,n],[0,1])
  ```

  On obtient :

  ```
  [(-2*n-(-1)^n*3+2^n*8-5)/4,(-2*n+(-1)^n*3+2^n*4-3)/4]
  ```

– Valeurs de la suite $u_0 = 0, v_0 = 1,\ u_{n+1} = u_n + v_n, v_{n+1} = u_n - v_n$ pour $n > 0$.
  On tape :

  ```
  seqsolve([x+y,x-y],[x,y,n],[0,1])
  ```

  On obtient :

  ```
  [(-4*n-3*(-(sqrt(2)-1))^n*sqrt(2)+
  2*(-(sqrt(2)-1))^n+ 3*(sqrt(2)+1)^n*sqrt(2)+
  2*(sqrt(2)+1)^n-4)/8,
  (-4*n+(-(sqrt(2)-1))^n*sqrt(2)+
  4*(-(sqrt(2)-1))^n-(sqrt(2)+ 1)^n*sqrt(2)+
  4*(sqrt(2)+1)^n)/8]
  ```

– Valeurs de la suite $u_0 = 2, v_0 = 0,\ u_{n+1} = 4 * v_n + n + 1, v_{n+1} = u_n$, pour $n > 0$.
  On tape :

  ```
  seqsolve([4y+n+1,x],[x,y,n],[2,0])
  ```

  On obtient :

  ```
  [(-8)/9+2*2^n-(-8)/9*(-1)^n*2^n-1/3*n,
  (-5)/9+2^n-4/9*(-1)^n*2^n-1/3*n]
  ```

6.14.3 Valeurs d'une suite récurrente ou d'un système de suites récurrentes : `rsolve`

Voir aussi `seqsolve` 6.14.2.
`rsolve` a comme argument la ou les relation(s) de récurrence, le nom des variables utilisées et les valeurs de départ de la suite.

La relation de récurrence est :

- soit une partie homogène linéaire, la partie non homogène doit être une combinaison linéaire de produit de polynôme en n par une suite géométrique en n. Par exemple $u_{n+1} = 2u_n + n3^n$,
- soit une fonction homographique. Par exemple $u_{n+1} = \dfrac{u_n - 1}{u_n - 2}$

`rsolve` renvoie alors une matrice dont les lignes sont les valeurs de la suite en fonctions de n.

Remarques
Contrairement à `seqsolve`, `rsolve` est plus malléable car avec `rsolve` :

- la suite ne débute pas forcément par `u(0)`,
- on peut donner plusieurs valeurs de départ par exemple `u(0)^2=1`, c'est pourquoi `rsolve` renvoie une liste,
- on écrit la relation de récurrence comme en mathématiques.

Exemples :

- Valeurs de la suite $u_0 = 3,\ u_{n+1} = 2u_n + n$
  On tape :
  ```
  rsolve(u(n+1)=2u(n)+n,u(n),u(0)=3)
  ```
  On obtient :
  ```
  [-1+4*2^(n+1-1)-n]
  ```
- Valeurs de la suite $u_1^2 = 1,\ u_{n+1} = 2u_n + n$
  On tape :
  ```
  rsolve(u(n+1)=2u(n)+n,u(n),u(1)^2=1)
  ```
  On obtient :
  ```
  [[-1-(-3)/2*2^(n+1-1)-n,
  -1-(-1)/2*2^(n+1-1)-n]]
  ```
- Valeurs de la suite $u_0 = 3,\ u_{n+1} = 2u_n + n3^n$
  On tape :
  ```
  rsolve(u(n+1)=2u(n)+(n)*3^n,u(n),u(0)=3)
  ```
  On obtient :
  ```
  [-3*3^(n+1-1)+6*2^(n+1-1)+n*3^(n+1-1)]
  ```
- Valeurs de la suite $u_0 = 4, u_{n+1} = \dfrac{u_n - 1}{u_n - 2}$
  On tape :
  ```
  rsolve(u(n+1)=(u(n)-1)/(u(n)-2),u(n),u(0)=4)
  ```
  On obtient :
  ```
  [((10*sqrt(5)+30)*((sqrt(5)-3)/2)^n+30*sqrt(5)-70)/
      (20*((sqrt(5)-3)/2)^n+10*sqrt(5)-30)]
  ```
- Valeurs de la suite $u_0 = 0, u_1 = 1,\ u_{n+1} = u_n + u_{n-1}$ pour $n > 0$.
  On tape :
  ```
  rsolve(u(n+1)=u(n)+u(n-1),u(n),u(0)=0,u(1)=1)
  ```
  On obtient :

```
[(5+sqrt(5))/10*((sqrt(5)+1)/2)^(n+1-1-1)+
(5-sqrt(5))/10*((-sqrt(5)+1)/2)^(n+1-1-1)]
```

– Valeurs de la suite $u_0 = 0, u_1 = 1,\ u_{n+1} = 2 * u_n + u_{n-1} + n$ pour $n > 0$.
  On tape :
  `rsolve(u(n+1)=2*u(n)+u(n-1)+n,u(n),u(0)=0,u(1)=1)`
  On obtient :
```
[(-1)/2-(-2-3*sqrt(2))/8*(sqrt(2)+1)^(n+1-1)-
(-2+3*sqrt(2))/8*(-sqrt(2)+1)^(n+1-1)-1/2*n]
```
  Ou on tape :
  `rsolve([u(n+1)=2*u(n)+v(n)+n,v(n+1)=u(n)],`
  `[u(n),v(n)],u(0)=0,v(0)=1)` On obtient :
```
[[(-1)/2-(-2-3*sqrt(2))/8*(sqrt(2)+1)^(n+1-1)-
(-2+3*sqrt(2))/8*(-sqrt(2)+1)^(n+1-1)-1/2*n,
-(-4+sqrt(2))/8*(sqrt(2)+1)^(n+1-1)-
(-4-sqrt(2))/8*(-sqrt(2)+1)^(n+1-1)-1/2*n]]
```
– Valeurs de la suite $u_0 = 0, v_0 = 1,\ u_{n+1} = u_n + v_n, v_{n+1} = u_n - v_n,$.
  On tape :
  `rsolve([u(n+1)=u(n)+v(n),v(n+1)=u(n)-v(n)],`
  `[u(n),v(n)],[u(0)=0,v(0)=1])`
  On obtient :
```
[[1/2*2^((n-1)/2)+1/2*(-(sqrt(2)))^(n-1),
(-1+sqrt(2))/2*2^((n-1)/2)+
(-1-sqrt(2))/2*(-(sqrt(2)))^(n-1)]]
```
– Valeurs de la suite $u_0 = 2, v_0 = 0,\ u_{n+1} = 4 * v_n + n + 1, v_{n+1} = u_n,$.
  On tape :
  `rsolve([u(n+1)=4*v(n)+n+1,v(n+1)=u(n)],`
  `[u(n),v(n)],[u(0)=2,v(0)=0])`
  On obtient :
```
[[(-8)/9+2*2^(n+1-1)-(-8)/9*(-1)^(n+1-1)*2^(n+1-1)-
1/3*n,(-5)/9+2^(n+1-1)-4/9*(-1)^(n+1-1)*2^(n+1-1)-
1/3*n]]
```

6.14.4 Tableau de valeurs et graphe d'une suite récurrente : `tableseq` `table_suite` et `plotseq` `graphe_suite`

`tableseq` ou `table_suite` est une commande qui s'utilise à l'intérieur d'un tableur (que l'on ouvre avec `Alt+t`) et qui .
remplit une colonne avec $\mathtt{u_0},\ \mathtt{u_{n+1}} = \mathtt{f(u_n)}$ (récurrence sur un terme) ou plus générallement $u_0, ..., u_k,\ u_{n+k+1} = f(u_n, u_{n+1}, ..., u_{n+k})$. `tableseq` ou `table_suite` remplit une colonne à partir de la cellule sélectionnée ou à partir de `0` si c'est le nom de la colonne qui est sélectionnée.
Voir aussi `plotseq` (section 3.19) pour la représentation graphique des suites récurrentes.

Exemples :

– Affichage des valeurs de la suite $u_0 = 3.5,\ u_n = \sin(u_{n-1})$
  On ouvre un tableur avec `Alt+t`.
  Puis, on sélectionne une case du tableur (par exemple `B0`) et pour avoir les

valeurs de $u_0 = 3.5,\ u_n = \sin(u_{n-1})$, on tape, dans la ligne de commande du tableur :

`tableseq(sin(u),u,4)`

On obtient :

`une colonne contenant sin(n), n, 3.5 et une formule evalf(subst(B$0,B$1,B2))`

Les valeurs de la suite $\mathtt{u_0} = 4,\ \mathtt{u_n} = \mathtt{sin}(\mathtt{u_{n-1}})$ s'affichent dans la colonne `B`.

– Affichage des valeurs de la suite de Fibonacci $u_0 = 1, u_1 = 1\ u_{n+2} = u_n + u_{n+1}$
  Après avoir selectionné `B0`, on tape, dans la ligne de commande du tableur :
  `tableseq(x+y,[x,y],[1,1])`
  On obtient, les premiers termes de la suite de Fibonacci :

ligne	B
x+y	
1	x
2	y
3	1
4	1
5	2
..	..
7	5
..	..

6.15 Les fonctions infixées ou opérateur

Un opérateur est une fonction infixée.

6.15.1 Les opérateurs usuels :`+, -, *, /, ^`

`+, -, *, /, ^` sont les opérateurs habituels pour faire des additions, des soustractions, des multiplications, des divisions et des élévations à une puissance.

6.15.2 Les autres opérateurs de `Xcas`

– `$` est la version infixée de `seq` par exemple :
  `(2^k)$(k=0..3)= seq(2^k,k=0..3)=(1,2,4,8)` (ne pas oublier de parenthéser les arguments),
– `mod` ou `%` pour définir un nombre modulaire,
– `@` pour composer des fonctions par exemple : `(f@g)(x)=f(g(x))`,
– `@@` pour composer une fonction avec elle-même par exemple :
  `(f@@3)(x)=f(f(f(x)))`,
– `minus union intersect` pour traduire la différence, l'union et l'intersection de deux ensembles,
– `->` pour définir une fonction,
– `:= ou =>` pour affecter une variable (c'est la version infixée de `sto` avec l'ordre permuté des arguments pour `:=`), par exemple : `a:=2` or `2=>a` or

`sto(2,a)`.
- `=>` permet aussi de faire des conversions d'unité et des réécritures d'expressions, par exemples : `3_m=>_cm`, `sin(x)=>diff`, `sin(x)=>exp`, `x^2-1=>*` etc... `=<` pour stocker une expression dans une variable, avec une affectation par référence (l'ordre des arguments est le même que pour `:=`) si la cible est un élément d'une matrice ou d'une liste. Ceci est plus rapide si on modifie les élèments d'une matrice ou d'une liste existante de grande dimension, car on ne fait pas de copie. À utiliser avec précautions car tous les objets pointant sur cette matrice seront modifiés. Dans un programme il faudra utiliser `copy` lors de l'initialisation pour que les modifications se fassent sur la copie (cf 8.4.15)

6.15.3 Définition d'un opérateur : `user_operator`

`user_operator` a comme argument :
- une chaîne de caractères qui est le nom de l'opérateur,
- une fonction de deux variablesà valeur dans $\mathbb{R}$ ou dans `true, false`,
- une option `Binary` pour la définition ou `Delete` pour annuler cette définition.

`user_operator` renvoie 1 si la définition a eu lieu et 0 sinon.

Exemple 1

Soit la loi R définit sur $\mathbb{R}$ par $x\ R\ y = x * y + x + y$.
On tape pour définir la loi R :

```
user_operator("R",(x,y)->x*y+x+y,Binary)
```

On obtient :

```
1
```

On tape :

```
5 R 7
```

Bien mettre les espace autour de `R`.
On obtient :

```
47
```

Exemple 2

Soit la relation S définit sur $\mathbb{N}$ par :
pour x et y entiers, $x\ S\ y <=> x$ et y ne sont pas premiers entre eux.
On tape pour définir la relation S :

```
user_operator("S",(x,y)->(gcd(x,y))!=1,Binary)
```

On obtient :

```
1
```

On tape :

```
5 S 7
```

Bien mettre les espace autour de `S`.
On obtient :

```
0
```

On tape :

```
8 S 12
```

Bien mettre les espace autour de `S`.
On obtient :

```
1
```

6.16 Les fonctions et les expressions de variables symboliques

6.16.1 Différence entre fonction et expression

Une fonction `f` est définie par exemple par :
`f(x):=x^2-1` ou encore par `f:=x->x^2-1`
cela signifie que pour tous les x, $f(x)$ est égale à l'expression $x^2 - 1$.
On pourra ainsi taper `f(2)` pour avoir la valeur de f en $x = 2$.
Par contre si on définit :
`g:=x^2-1` cela signifie que `g` est une variable qui contient l'expression $x^2 - 1$.
Pour avoir la valeur de g en $x = 2$ il faut alors écrire :
`subst(g,x=2)` car g est une expression qui dépend de x.
Aussi, lorsque l'argument d'une commande est une fonction il faut mettre comme argument, soit par exemple `x->x^2-1`, soit `f` (si `f` est une fonction qui a été définie auparavant par exemple par `f(x):=x^2-1`) et
lorsque l'argument d'une commande est une expression on met comme argument, soit par exemple `x^2-1`, soit `g` (si `g` est une variable que l'on a définie auparavant par exemple `g:=x^2-1`), soit `f(x)` (si `f` est une fonction qui a été définie auparavant par exemple `f(x):=x^2-1`).

6.16.2 Transformer une expression en une fonction : `unapply`

Pour transformer une expression en une fonction, on utilise la commande `unapply`.
`unapply` a deux arguments une expression et le nom d'une (ou des) variable(s).
`unapply` renvoie une fonction définie à partir de cette expression et de la (ou des) variable(s) donnée(s) en argument.

Attention lorsqu'on définit une fonction, le membre de droite de l'affectation n'est pas évalué, ainsi l'écriture `g:=x^2; f(x):=g` ne définit pas la fonction $f : x \to x^2$ mais la fonction $f : x \to g$.
On tape :

```
g:= sin(x+1); f:=unapply(g,x)
```

On obtient :

```
(sin(x+1), (x)->sin(x+1))
```

On a alors la variable `g` qui contient une expression symbolique et la variable `f` qui contient une fonction.
On tape :

```
unapply(exp(x+2),x)
```

On obtient :

```
(x)->exp(x+2)
```

On tape :

```
f:=unapply(lagrange([1,2,3],[4,8,12]),x)
```

On obtient :

```
(x)->4+4*(x-1)
```

On tape :

```
f:=unapply(integrate(log(t),t,1,x),x)
```

On obtient :

```
(x)->x*log(x)-x+1
```

On tape :

```
f:=unapply(integrate(log(t),t,1,x),x)
```

```
f(x)
```

On obtient :

```
x*log(x)-x+1
```

Remarque Pour définir, à partir d'une fonction de 2 variables $f(x, w)$, la fonction g qui à w fait correspondre la fonction $g(w)$ définie par : $g(w)(x) = f(x, w)$, on utilise aussi `unapply`.
On tape :

```
f(x,w):=2*x+w
```

```
g(w):=unapply(f(x,w),x)
```

```
g(3)
```

On obtient :

```
x->2· x+3
```

6.16.3 Sommet et feuille d'une expression : `sommet feuille op`

Un opérateur est une fonction infixée : par exemple '+' est un opérateur et 'sin' est une fonction.
On peut représenter une expression par un arbre. Le sommet de l'arbre est soit un opérateur, soit une fonction et les feuilles de l'arbre sont les arguments de l'opérateur ou de la fonction (voir aussi 6.40.13).
La fonction `sommet` (resp `feuille` (ou `op`)) renvoie le sommet (resp la liste des feuilles) d'une expression.
On tape :

```
sommet(sin(x+2))
```

On obtient :

```
'sin'
```

On tape :

```
sommet(x+2*y)
```

On obtient :

```
'+'
```

On tape :

```
feuille(sin(x+2))
```

Ou on tape :

```
op(sin(x+2))
```

On obtient :

```
x+2
```

On tape :

```
feuille(x+2*y)
```

Ou on tape :

```
op(x+2*y)
```

On obtient :

```
(x,2*y)
```

Remarque
Lorsque l'utilisateur définit une fonction par un programme par exemple la fonction `pgcd`.
On tape :

```
pgcd(a,b):={local r; while (b!=0)
{r:=irem(a,b);a:=b;b:=r;} return a;}
```

Puis on tape :

`sommet(pgcd)`

On obtient :

`'program'`

Puis on tape :

`feuille(pgcd)[0]`

On obtient :

`(a,b)`

Puis on tape :

`feuille(pgcd)[1]`

On obtient :

`(0,0) ou (15,25) si l'on vient d'exécuter pgcd(15,25)`

Puis on tape :

`feuille(pgcd)[2]`

On obtient :

`Le corps du programme : {local r;....return(a);}`

6.17 Les fonctions

6.17.1 Les fonctions ayant plusieurs usages

+ et −

+ (resp −) est une fonction infixée et `'+'` (resp `'-'`) est une fonction préfixée.
Elle renvoie un résultat qui dépend de la nature de ses arguments.
Voici des exemples avec + (seul le dernier exemple n'est pas utilisable avec −) :

- On tape (1,2)+(3,4) ou (1,2,3)+4=1+2+3+4 ou'+'(1,2,3,4), on obtient 10,
- On tape 1+i+2+3*i ou '+'(1,i,2,3*i), on obtient 3+4*i,
- On tape [1,2,3]+[4,1] ou [1,2,3]+[4,1,0] ou '+'([1,2,3],[4,1]), on obtient [5,3,3],
- On tape [1,2]+[3,4] ou '+'([1,2],[3,4]), on obtient [4,6],
- On tape [[1,2],[3,4]]+ [[1,2],[3,4]], on obtient [[2,4],[6,8]],
- On tape [1,2,3]+4 ou '+'([1,2,3],4), on obtient poly1[1,2,7],
- On tape [1,2,3]+(4,1) ou '+'([1,2,3],4,1), on obtient poly1[1,2,8],
- On tape "bon"+"jour" ou '+'("bon","jour"), on obtient "bonjour".

`*`

`*` est une fonction infixée et `'*'` est une fonction préfixée. Elle renvoie un résultat qui dépend de la nature de ses arguments.
Voici des exemples avec `*` :

- On tape (1,2)*(3,4) ou (1,2,3)*4 ou 1*2*3*4 ou '*'(1,2,3,4), on obtient 24,
- On tape 1*i*2*3*i ou '*'(1,i,2,3*i), on obtient -6,
- On tape [10,2,3]*[4,1] ou [10,2,3]*[4,1,0] ou '+'([10,2,3],[4,1]), on obtient 42 (produit scalaire),
- On tape [1,2]*[3,4] ou '*'([1,2],[3,4]), on obtient 11 (produit scalaire),
- On tape [[1,2],[3,4]]* [[1,2],[3,4]], on obtient [[7,10],[15,22]],
- On tape [1,2,3]*4='*'([1,2,3],4), on obtient [4,8,12],
- On tape [1,2,3]*(4,2)ou '*'([1,2,3],4,2), on obtient [1,2,3]*8=[8,16,24],
- On tape (1,2)+i*(2,3), on obtient 1+2+i*2*3=3+6*i.

`/`

`/` est une fonction infixée et `'/'` est une fonction préfixée. Elle renvoie un résultat qui dépend de la nature de ses arguments.
Voici des exemples avec `/` :

- On tape [10,2,3]/[4,1], on obtient invalid dim,
- On tape [1,2]/[3,4] ou '/'([1,2],[3,4]), on obtient [1/3,1/2],
- On tape 1/[[1,2],[3,4]], on obtient [[-2,1],[3/2,(-1)/2]],
- On tape [[1,2],[3,4]]*1/ [[1,2],[3,4]], on obtient [[1,0],[0,1]],
- On tape [[1,2],[3,4]]/ [[1,2],[3,4]], on obtient [[1,1],[1,1]] (division terme à terme),
- On tape [1,2,3]*4 ou '*'([1,2,3],4), on obtient [4,8,12],
- On tape [1,2,3]/(4,2) ou '*'([1,2,3],4,2)=[1,2,3]*8, on obtient [8,16,24].

6.17.2 Les fonctions usuelles

`max` d'une séquence ou d'une liste de réels renvoie leur maximum,
`min` d'une séquence ou d'une liste de réels renvoie leur minimum,
`abs` d'un réel renvoie sa valeur absolue,
`sign` d'un réel renvoie son signe (+1 si il est positif, 0 si il est nul et -1 si il est négatif),
`floor` d'un réel renvoie sa partie entière,
`round` d'un réel renvoie l'entier le plus proche,
`ceil` ou `ceiling` d'un réel renvoie sa partie entière plus un,
`frac` (ou `fPart`) d'un réel renvoie sa partie fractionnaire,
`trunc` d'un réel renvoie l'entier égal au réel argument sans sa partie fractionnaire ou le réel tronqué à n décimales,
`iPart`) d'un réel renvoie le réel égal au réel argument sans sa partie fractionnaire,
`id` désigne la fonction identité,
`sq` désigne la fonction carrée,
`sqrt` désigne la fonction racine carrée,
`surd` désigne la fonction puissance $\frac{1}{n}$,
`exp` désigne la fonction exponentielle,
`log` ou `ln` désigne la fonction logarithme népérien,

`log10` désigne la fonction logarithme à base dix,
`logb` désigne la fonction logarithme à base donnée comme deuxième argument : `logb(7,10)=log10(7)=log(7)/log(10)`,
`alog10` désigne la fonction anti-logarithme à base dix : c'est la fonction $x- > 10^x$.
`sinh` désigne la fonction sinus hyperbolique,
`cosh` désigne la fonction cosinus hyperbolique,
`tanh` désigne la fonction tangente hyperbolique,
`asinh` ou `arcsinh` (respectivement `acosh` ou `arccosh`, `atanh` ou `arctanh`) désigne la fonction réciproque de `sinh` (respectivement `cosh`, `tanh`)
`sin, cos, tan, cot, sec, csc` et `asin` (ou `arcsin`),`acos` (ou `arccos`), `atan` (ou `arctan`), `acot, asec, acsc` pour les fonctions trigonométriques et pour leurs fonctions réciproques (on se reportera à la section 6.23.1 pour les commandes les concernant). On tape :

```
normal(surd(8,3))
```

On obtient :

```
2
```

On tape :

```
normal(surd(-8,3))
```

On obtient :

```
-2
```

On tape :

```
normal(8^(1/3))
```

On obtient :

```
2
```

On tape :

```
simplify((-8)^(1/3))
```

On obtient :

```
(i)*sqrt(3)+1
```

car `Xcas` prend $i\pi$ comme détermination de $\ln(-1)$ et $i\pi/3$ comme détermination de $\ln((-1)^{1/3})$

6.17.3 Définition d'une fonction

Définition d'une fonction de $\mathbb{R}^p$ dans $\mathbb{R}$

On tape pour définir la fonction $f\ :\ (x)->x*\sin(x)$:

```
f(x):=x*sin(x)
```

Ou on tape :

```
f:=x->x*sin(x)
```

On obtient :

```
(x)->x*sin(x)
```

On tape pour définir la fonction $f\ :\ (x,y)->x*\sin(y)$:

```
f(x,y):=x*sin(y)
```

Ou on tape :

```
f:=(x,y)->x*sin(y)
```

On obtient :

```
(x,y)->x*sin(y)
```

Attention !!! ce qui se trouve après `->` n'est pas évalué.

Définition d'une fonction de $\mathbb{R}^p$ dans $\mathbb{R}^q$

On tape pour définir la fonction $h\ :\ (x,y)->(x*\cos(y),x*\sin(y))$:

```
h(x,y):=(x*cos(y),x*sin(y))
```

On tape pour définir la fonction $h\ :\ (x,y)->[x*\cos(y),x*\sin(y)]$:

```
h(x,y):=[x*cos(y),x*sin(y)];
```

Ou on tape :

```
h:=(x,y)->[x*cos(y),x*sin(y)];
```

Ou on tape :

```
h(x,y):={[x*cos(y),x*sin(y)]};
```

Ou on tape :

```
h:=(x,y)->return[x*cos(y),x*sin(y)];
```

Ou on tape

```
h(x,y):={return [x*cos(y),x*sin(y)];}
```

On obtient :

```
(x,y)->{return([x*cos(y),x*sin(y)]);}
```

Attention !!! ce qui se trouve après `->` n'est pas évalué.

Définition d'une fonction de $\mathbb{R}^{p-1}$ dans $\mathbb{R}^q$ à partir d'une fonction de $\mathbb{R}^p$ dans $\mathbb{R}^q$

On définit la fonction $f(x, y) = x * \sin(y)$, puis on veut définir la famille de fonctions dépendant du paramètre t par $g(t)(y) := f(t, y)$.
Comme ce qui se trouve après `->` n'est pas évalué, on ne peut pas définir $g(t)$ par `g(t):=y->f(t,y)` et on doit utiliser la commande `unapply`.
On tape pour définir les fonctions $f(x, y) = x \sin(y)$ et $g(t) = y- > f(t, y)$:

```
f(x,y):=x*sin(y);g(t):=unapply(f(t,y),y)
```

On obtient :

```
((x,y)->x*sin(y), (t)->unapply(f(t,y),y))
```

On tape

```
g(2)
```

On obtient :

```
y->2· sin(y)
```

On tape

```
g(2)(1)
```

On obtient :

```
2· sin(1)
```

On définit la fonction $h(x, y) = (x * \cos(y), x * \sin(y))$, puis on veut définir la famille de fonctions dépendant du paramètre t par $k(t)(y) := h(t, y)$.
Comme ce qui se trouve après `->` n'est pas évalué, on ne peut pas définir $k(t)$ par $k(t) := y- > h(x, y)$ et on est obligé d'utiliser la commande `unapply`.
On tape pour définir la fonction $h(x, y)$:

```
h(x,y):=(x*cos(y),x*sin(y))
```

On tape pour définir la fonction $k(t)$:

```
k(t):=unapply(h(x,t),x)
```

On obtient :

```
(t)->unapply(h(x,t),x)
```

On tape

```
k(2)
```

On obtient :

```
(x)->(x*cos(2),x*sin(2))
```

On tape

```
k(2)(1)
```

On obtient :

```
(2*cos(1),2*sin(1))
```

Ou encore On définit la fonction $h(x, y) = [x * \cos(y), x * \sin(y)]$, puis on veut définir la famille de fonctions dépendant du paramètre t par $k(t)(y) := h(t, y)$.
Comme ce qui se trouve après $->$ n'est pas évalué, on ne peut pas définir $k(t)$ par $k(t) := y- > h(x, y)$ et on est obligé d'utiliser la commande `unapply`.
On tape pour définir la fonction $h(x, y)$:

```
h(x,y):={[x*cos(y),x*sin(y)]}
```

On tape pour définir la fonction $k(t)$:

```
k(t):=unapply(h(x,t),x)
```

On obtient :

```
(t)->unapply(h(x,t),x)
```

On tape

```
k(2)
```

On obtient :

```
(x)->{[x*cos(2),x*sin(2)];}
```

On tape

```
k(2)(1)
```

On obtient :

```
[2· cos(1),2· sin(1)]
```

6.17.4 Composition de fonctions : `@`

La composition de fonctions se fait avec `Xcas` grâce à l'opérateur `@` qui est infixé.
On tape :

```
(sq@sin+id)(x)
```

On obtient :

```
(sin(x))^2+x
```

On tape :

```
(sin@sin)(pi/2)
```

On obtient :

```
sin(1)
```

6.17.5 Puissance n-ième de composition d'une fonction : `@@`

La puissance n-ième de composition d'une fonction se fait avec `Xcas` grâce à l'opérateur `@@` qui est infixé.
On tape :

```
(sin@@3)(x)
```

On obtient :

```
sin(sin(sin(x)))
```

On tape :

```
(sin@@2)(pi/2)
```

On obtient :

```
sin(1)
```

6.17.6 Definir une fonction avec l'historique : `as_function_of`

Si on a affecté une valeur à la variable `a` et si on définit ensuite, dans une autre ligne d'entrée, la variable `b`, à partir de `a`, on utilise `c:=as_function_of(b,a)` pour définir une fonction `c` vérifiant : `c(a)=b`.
On tape :

```
a:=sin(x)
```

On obtient :

```
sin(x)
```

On tape :

```
b:=sqrt(1+a^2)
```

On obtient :

```
b:=sqrt(1+sin(x)^2)
```

On tape :

```
c:=as_function_of(b,a)
c(x)
```

On obtient :

```
sqrt(1+x^2)
```

On tape :

```
a:=2
b:=1+a^2
```

On obtient :

```
b:=5
```

On tape :

```
c:=as_function_of(b,a)
```

On obtient :

```
(a)->
{ local NULL;
return(sqrt(1+a^2));
}
```

On tape :

```
c(x)
```

On obtient :

```
1+x^2
```

Attention
Si la variable `b` a été affectée plusieurs fois c'est la première affectation de `b` après la dernière affectation de `a` qui compte.
On tape par exemple :
`a:=2` puis
`b:=2*a+1` puis
`b:=3*a+2` puis
`c:=as_function_of(b,a)`
On obtient :

```
(a)-> {local NULL; return(2*a+1);}
```

c'est à dire que `c(x)` vaut `2*x+1`.
Mais si on tape par exemple :
`a:=2` puis
`b:=2*a+1` puis
`a:=2` puis
`b:=3*a+2` puis
`c:=as_function_of(b,a)`
On obtient :

```
(a)-> {local NULL; return(3*a+2);}
```

c'est à dire que `c(x)` vaut `3*x+2`.
Il est donc préférable de valider la ligne ou se trouve `a` avant de définir le `b` qui convient.

6.18 Dérivation

6.18.1 Gńéralités

Pour définir la fonction dérivée d'une fonction on a plusieurs possibilités.
Exemple Calculer la dérivée de la fonction f définie par $f(x) = \sin(2x) + x$.
On tape, par exemple, pour défínr f :
`f(x):=sin(2x)+x`
On peut taper pour défínr la dériv´e g de f :

- `g:=f'` ou
  `g(x):=f'(x)` ou
  `g:=x->f'(x)`
- `g:=function_diff(f)` ou
  `g(x):=function_diff(f)(x)` ou
  `g:=x->function_diff(f)(x)`
- `g:=unapply(diff(f(x),x),x)` ou
  `g(x):=unapply(diff(f(x),x),x)(x)` ou
  `g:=x->unapply(diff(f(x),x),x)(x)`
- `g:=unapply(diff(f(x)),x)` ou
  `g(x):=unapply(diff(f(x)),x)(x)` ou
  `g:=x->unapply(diff(f(x)),x)(x)`
- `g:=diff(f)` ou
  `g(x):=diff(f)(x)` ou
  `g:=x->diff(f)(x)`

MAIS ATTENTION
`g(x):=diff(f(x))` ou `g(x):=diff(f(x),x)` n'est pas correct, car lors d'une affectation ce qui est à droite de := n'est pas évalué lors de la définition....il faut utiliser `unapply` ou écrire `g(x):=diff(f)(x)`.

6.18.2 Calcul du taux d'accroissement : `taux_accroissement`

`taux_accroissement` calcule le taux d'accroissement d'une expression lorsque la variable passe d'une valeur à une autre. Par défaut la variable est égale à `x`.
Permet d'introduire la notion de nombre dérivé.
On tape :

```
taux_accroissement(x^2,1,2)
```

Ou on tape :

```
taux_accroissement(y^2,y,1,2)
```

On obtient :

```
3
```

On tape :

```
taux_accroissement(x^2,1,1+h)
```

Ou on tape :

```
taux_accroissement(y^2,y,1,1+h)
```

On obtient :

```
((1+h)^2-1)/(1+h-1)
```

Après simplification, on obtient :

```
h+2
```

On tape :

```
limit(taux_accroissement(x^2,1,1+h),h,0)
```

On obtient :

```
2
```

6.18.3 Fonction dérivée d'une fonction : `function_diff` `fonction_derivee`

`function_diff` a comme argument une fonction.
`function_diff` renvoie la fonction dérivée de cette fonction.
On tape :

```
function_diff(sin)
```

On obtient :

```
(` x`)->cos(` x`)
```

On tape :

```
function_diff(sin)(x)
```

On obtient :

```
cos(x)
```

On tape :

```
f(x):=x^2+x*cos(x)
function_diff(f)
```

On obtient :

```
(` x`)->2*` x`+cos(` x`)+` x`*(-(sin(` x`)))
```

On tape :

```
function_diff(f)(x)
```

On obtient :

```
cos(x)+x*(-(sin(x)))+2*x
```

Attention!!!
Lorsqu'on est en mode `Maple`, pour des raisons de compatibilité, on peut aussi utiliser `D` à la place de `function_diff` et c'est pourquoi en géométrie on ne pourra pas avoir d'objet géométrique ayant comme nom `D`, lorsqu'on est en mode `Maple`.

6.18.4 Derivées et derivées partielles d'une expression et derivées d'une fonction : `diff derive deriver '`

Généralités

`diff` ou `derive` ou `deriver` sont des fonctions préfixées alors que `'` est la version postfixée de `diff` ou `derive` ou `deriver`.
Ces fonctions ont un, deux ou plus de 2 arguments :
- avec un argument qui peut être soit une fonction, soit une expression de la variable `x`.
  - Si cet argument est une fonction, `diff` ou `derive` ou `deriver` ou `'` renvoie la fonction dérivée de cette fonction.
    Ces fonctions sont alors equivalentes à `function_diff`.
  - Si cet argument est une expression de la variable `x`, `diff` ou `derive` ou `deriver` ou `'` renvoie la dérivée de l'expression par rapport à `x`
  - **Remarque** Dans ce cas on peut aussi utiliser `convert` (ou sa version infixée `=>`) avec l'option `diff`.
- avec 2 arguments qui peuvent être soit une expression et le nom d'une variable, soit une expression et une liste de noms de variables.
  Cela va permettre de calculer des dérivées et des derivées partielles du premier ordre et plusieurs arguments pour calculer des derivées partielles de tous les ordres d'une expression.
- avec plus que 2 arguments qui peuvent être :
  une expression et le nom des variables par rapport auxquelles il faut dériver cette expression (le nom des variables est éventuellement suivi de $$n$ pour indiquer le nombre n de fois que l'on veut dériver) ,

Derivée et fonction derivée

`diff` ou `derive` ou `deriver` ou `'` ont comme argument soit une fonction, soit une expression de la variable `x`.

- Si cet argument est une fonction, `diff` ou `derive` ou `deriver` ou `'` renvoie une fonction qui est la fonction dérivée de la fonction argument.
  `diff` (ou `derive` ou `deriver` ou `'`) est alors equivalente à `function_diff`.
  On tape :

  `f(x):=x^2+x*cos(x)+`

  Pour définir g comme f', on tape :

  `g:=f'`

  Ou :

  `g:=(f=>diff)`

  Ou :

  `g:=diff(f)`

  Ou :

  `g:=function_diff(f)`

  Ou :

  `g:=unapply(diff(f(x),x),x)`

Ou :

```
g(x):=f'(x)
```

Ou :

```
g(x):=(f(x)=>diff)
```

Ou :

```
g(x):=diff(f)(x))
```

Puis, on tape :

```
g(x)
```

On obtient :

```
cos(x)+x*(-(sin(x)))+2*x
```

Pour définir h comme f'', on tape :

```
h:=f''
```

Ou :

```
h:=diff(diff(f))
```

Ou :

```
h:=function_diff(function_diff(f))
```

Ou :

```
h:=diff(unapply(diff(f(x),x),x))
```

Ou :

```
h(x):=diff(diff(f))(x))
```

Ou :

```
h(x):=f''(x)
```

Puis, on tape :

```
h(x)
```

On obtient :

```
-sin(x)+x*(-cos(x))-sin(x)+2
```

– Si cet argument est une expression de la variable `x`, `diff` ou `derive` ou `deriver` ou `'` renvoie une expression qui est l'expression de la dérivée de l'argument par rapport à `x`. On tape :

```
f(x):=x^+x*cos(x)+
```

Pour calculer la dérivée de $f(x)$, on tape :

```
A:=f(x)'
```

Ou :

```
A:=diff(f(x))
```

On obtient :

```
cos(x)+x*(-(sin(x)))+2*x
```

Ou encore :

```
A:=diff(f(a),a)
```

On obtient :

```
cos(a)+a*(-(sin(a)))+2*a
```

MAIS ATTENTION

cela ne définit pas une fonction car le résultat est une expression.
`g(x):=diff(f(x))` ou `g(x):=diff(f(x),x)` n'est pas correct, car lors d'une affectation ce qui est à droite de := n'est pas évalué lors de la définition....il faut utiliser `unapply` (`g(x):=unapply(diff(f(x),x),x)`)

ou écrire `g(x):=diff(f)(x)`.

Derivées et derivées partielles d'ordre 1 : `diff derive deriver '`

Pour avoir des derivées partielles d'ordre 1 :
`diff` (ou `derive` ou `deriver` ou `'`) a deux arguments : une expression et une variable (resp une liste contenant le nom des variables) (voir fonctions de plusieurs variables paragraphe 6.53).
`diff` renvoie la dérivée de l'expression par rapport à la variable donnée comme deuxième paramètre, trés utile pour calculer des dérivées partielles !) (resp renvoie une liste contenant les dérivées par rapport aux variables de la liste du 2nd argument).
Exemples :

- Soit à calculer :
$$\frac{\partial(x.y^2.z^3+x.y.z)}{\partial z}$$
On tape :
`(x*y ^2*z^3+x*y*z,z)'`
ou on tape :
`diff(x*y ^2*z^3+x*y*z,z)`
On obtient :
`x*y^2*3*z^2+x*y`
- Soit à calculer les 3 derivées partielles premières de $x*y^2*z^3+x*y*z$.
On tape :
`(x*y^2*z^3+x*y,[x,y,z])'`
Ou on tape :
`diff(x*y^2*z^3+x*y,[x,y,z])`
On obtient :
`[y^2*z^3+y*z, x*2*y*z^3+x*z, x*y^2*3*z^2+x*y]`
- Soit à calculer :
$$\frac{\partial^3(x.y^2.z^3+x.y.z)}{\partial y\partial^2 z}$$
On tape :
`(x*y ^2*z^3+x*y*z,y,z$2)'`
Ou on tape :
`diff(x*y ^2*z^3+x*y*z,y,z$2)`
On obtient :
`12*x*y*z`

Derivée et derivée partielle d'ordre n : `diff derive deriver`

Lorsque `derive` (ou `diff`) a plus de deux arguments, ce sont : une expression et le nom des variables par rapport auxquelles il faut dériver cette expression (le nom des variables est éventuellement suivi de $$n$ pour indiquer le nombre n de fois que l'on veut dériver).
`diff` renvoie la dérivée de l'expression par rapport aux variables données après le premier paramètre (utile pour calculer des dérivées partielles de tous les ordres).
Donc pour dériver n fois :

`diff` (ou `derive`) a $n+1$ arguments : une expression et le nom de la variable qui sera répété n fois. Pour avoir une écriture plus facile on écrira plutôt le nom de la variable suivi de $\$n$ pour indiquer que l'on veut dériver n fois (en effet `x$3=(x,x,x)`). Par exemple pour dériver `exp(x*y)` 1 fois par rapport à x et 2 fois par rapport à y, on met comme arguments l'expression, puis, les noms des variables éventuellement suivi de $ pour indiquer le nombre de fois que l'on veut dériver et on tape `diff(exp(x*y),x,y$2)` qui est équivalent à `diff(exp(x*y),x,y,y)` (en effet `y$2=(y,y)`).

Exemples

– Soit à calculer :

$$\frac{\partial^2(x.y^2.z^3+x.y.z)}{\partial x\partial z}$$

On tape :

```
(x*y ^2*z^3+x*y*z,x,z)'
```

Ou on tape :

```
diff(x*y ^2*z^3+x*y*z,x,z)
```

On obtient :

```
y^2*3*z^2+y
```

– Soit à calculer :

$$\frac{\partial^3(x.y^2.z^3+x.y.z)}{\partial x\partial^2 z}$$

On tape :

```
(x*y ^2*z^3+x*y*z,x,z,z)'
```

Ou on tape :

```
(x*y ^2*z^3+x*y*z,x,z$2)'
```

Ou on tape :

```
diff(x*y ^2*z^3+x*y*z,x,z,z)
```

Ou on tape :

```
diff(x*y ^2*z^3+x*y*z,x,z$2)
```

On obtient :

```
6*y^2*z
```

– Soit à calculer la dérivée troisième de :

$$\frac{1}{x^2+2}$$

On tape :

```
(1/(x^2+2),x,x,x)'
```

Ou on tape :

```
(1/(x^2+2),x$3)'
```

Ou on tape :

```
diff((1)/(x^2+2),x,x,x)
```

Ou on tape :

```
diff((1)/(x^2+2),x$3)
```

On obtient :

```
(-24*x^3+48*x)/(x^8+8*x^6+24*x^4+32*x^2+16)
```

Remarque

Bien voir la différence entre `diff(Xpr,x,y)` et `diff(Xpr,[x,y])` où `Xpr` est une expression :

`diff`(Xpr,x,y) renvoie $\frac{\partial^2(Xpr)}{\partial x\partial y}$ et
`diff`$(Xpr,[x,y])$ renvoie $[\frac{\partial(Xpr)}{\partial x},\frac{\partial(Xpr)}{\partial y}]$

6.19 Intégration

6.19.1 Primitive et intégrale définie : `integrate Int integrer int integration`

`integrate` (ou `int`) permettent de calculer une primitive ou une intégrale définie. La seule différence entre ces deux commandes est que `integrate` écrit avec le symbole $\int$ la réponse de la commande `quest()` qui suit l'évaluation de `integrate`.
Par contre `Int` renvoie `integrate` sans l'évaluer : c'est pour avoir la compatibilité avec Maple, lorsque l'on fait un calcul numérique d'intégrales :
On tape :

```
Int(exp(x),x,0,1))
```

On obtient :

```
integrate(exp(x),x,0,1)
```

On tape :

```
int(exp(x),x,0,1))
```

On obtient :

```
exp(1)-1
```

On tape :

```
evalf(Int(exp(x^2),x,0,1))
```

Ou on tape :

```
evalf(int(exp(x^2),x,0,1))
```

On obtient :

```
1.46265174591
```

`integrate` (ou `int` ou `Int`) a un, deux ou quatre arguments.

– avec un argument qui est une expression de la variable `x`, (resp une fonction). `integrate` (ou `int`) renvoie alors une expression qui est une primitive de l'expression par rapport à la variable `x` (resp renvoie une fonction primitive de la fonction donnée en argument) On tape :
  ```
  integrate(x^2)
  ```
  On obtient :
  ```
  x^3/3
  ```
  On tape :

`f(t):=t^2`
`g:=integrate(f)`

On obtient :

`(t)->t^3/3`

Remarque Dans ce cas on peut aussi utiliser `convert` (ou sa version infixée =>) avec l'option `int`.
On tape :

`f(x):=x^2`
`g(x):=(f(x)=>int)`

ou

`g:=(f=>int))`

– avec deux arguments qui sont :
  une expression et une variable,
  `integrate` (ou `int`) renvoie alors une primitive de l'expression par rapport à la variable donnée comme deuxième paramètre.
  On tape :

  `integrate(x^2)`

  On obtient :

  `x^3/3`

  On tape :

  `integrate(t^2,t)`

  On obtient :

  `t^3/3`

– avec quatre arguments qui sont :
  une expression, une variable et les bornes de l'intégrale définie,
  `integrate` (ou `int`) renvoie alors la valeur de l'intégrale définie.
  On tape :

  `integrate(x^2,x,1,2)`

  On obtient :

  `7/3`

  On tape :

  `integrate(1/(sin(x)+2),x,0,2*pi)`

  On obtient après simplification (appel à `simplify`) :

  `2*pi*sqrt(3)/3`

Exercice 1
Soit

$$f(x) = \frac{x}{x^2-1} + \ln(\frac{x+1}{x-1})$$

Calculer une primitive de f.
On tape :

```
int(x/(x^2-1)+ln((x+1)/(x-1)))
```

On trouve :

```
x*log((x+1)/(x-1))+log(x^2-1)+1/2*log(2*x^2/2-1)
```

Ou bien on définit la fonction `f` en tapant :

```
f(x):=x/(x^2-1)+ln((x+1)/(x-1))
```

puis on tape :

```
int(f(x))
```

On obtient bien sûr le même résultat.
Attention
Pour `Xcas`, `log` est égal à `ln` (logarithme népérien) et `log10` est le logarithme en base 10.
Exercice 2
Calculer :

$$\int \frac{2}{x^6 + 2 \cdot x^4 + x^2}\, dx$$

On tape :

```
int(2/(x^6+2*x^4+x^2))
```

On trouve :

```
2*((3*x^2+2)/(-(2*(x^3+x)))+-3/2*atan(x))
```

Exercice 3
Calculer :

$$\int \frac{1}{\sin(x) + \sin(2 \cdot x)}\, dx$$

On tape :

```
integrate(1/(sin(x)+sin(2*x )))
```

On trouve :

```
(1/-3*log((tan(x/2))^2-3)+1/12*log((tan(x/2))^2))*2
```

6.19.2 Primitive et intégrale définie : `risch`

`risch` calcule une primitive ou une intégrale définie par l'algorithme de Risch.
On tape :

```
risch(x^2)
```

On obtient :

```
x^3/3
```

On tape :

```
risch(x^2,x,0,1)
```

On obtient :

```
1/3
```

On tape :

```
risch(exp(-x^2))
```

On obtient :

```
∫ exp(x^2) dx
```

ce qui signifie que $\exp(-x^2)$ n'a pas de primitive exprimable avec des fonctions connues.

6.19.3 Somme indicée finie et infinie et primitive discrète : `sum`

`sum` a deux, quatre ou cinq arguments :

- Avec 5 arguments `sum(Xpr,Var,a,b,p)` renvoie la somme demandée c'est à dire renvoie la somme des valeurs de l'expression `Xpr` quand la variable `Var` va de `a` à `b` avec un pas égal à `p`.
  On tape :
  `sum(x^2+1,x,1,5,1)`
  Ou on tape :
  `sum(x^2+1,x,1,5)`
  Ou on tape :
  `sum(x^2+1,x,5,1,1)`
  Ou on tape :
  `sum(x^2+1,x,5,1,-1)`
  On obtient :
  `60`
  En effet :
  $2+5+10+17+26=60$
  On tape :
  `sum(x^2+1,x,1,5,2)`
  On obtient :
  `38`
  En effet :
  $2+10+26=38$
- Avec 4 ou 2 arguments, `sum(Xpr,Var,a,b)` ou `sum(Xpr,Var=a..b)` n'a pas la même valeur selon que `a` est plus petit ou égal à `b` ou non, car on veut avoir l'égalité :
  `sum(Xpr,Var,a,b)=sum(Xpr,Var,a,c)+sum(Xpr,Var,c+1,b).`
  Aussi, lorsque le pas `p` n'est pas précisé on a :
  - si `a` est inférieur à `b`,
    `sum(Xpr,Var,a,b)` renvoie la somme des valeurs de l'expression `Xpr` quand la variable `Var` va de `a` à `b` avec un pas de `1` : cette syntaxe est compatible avec Maple.
    Ainsi si `a<= b` on a :
    `sum(Xpr,Var,a,b)=sum(Xpr,Var,a,b,1).`
    On tape :
    `sum(x^2+1,x,1,5)`
    Ou on tape :
    `sum(x^2+1,x=1..5)`
    On obtient :
    `60`
    En effet :
    $2+5+10+17+26=60$
  - si `a` est supérieur à `b+1`,
    `sum(Xpr,Var,a,b)` renvoie l'opposé de la somme des valeurs de l'expression `Xpr` quand la variable `Var` va de `b+1` à `a-1` avec un pas de `1` : cette syntaxe est compatible avec Maple.

On tape :

```
sum(x^2+1,x,5,1)
```

On obtient :

```
-32
```

En effet :
$-(5+10+17) = -32$ **Attention** on n'obtient pas la même chose si on précise le pas (avec $p = 1$ ou $p = -1$)
On tape :

```
sum(x^2+1,x,5,1,-1)
```

Ou on tape :

```
sum(x^2+1,x,5,1,1)
```

On obtient :

```
60
```

Car on a :
$2+5+10+17+26 = 60$
On tape :

```
sum(x^2+1,x,4,1)
```

On obtient :

```
-15
```

En effet :
$-(5+10) = -15$

– si `a` est égal à `b+1`,
  `sum(Xpr,Var,b+1,b)` renvoie 0.
  On tape :

```
sum(x^2+1,x,5,4)
```

  Ou on tape :

```
sum(x^2+1,x=5..4)
```

  On obtient :

```
0
```

– si `sum` a deux arguments :
  `sum` a comme premier argument une expression (par exemple $f(x)$) d'une variable (par exemple x) qui est donnée comme deuxième argument.
  `sum` renvoie la primitive discrète de cette expression, c'est à dire la fonction G verifiant $G(x+1) - G(x) = f(x)$.
  On tape :

```
sum(x,x)
```

  On obtient :

```
(x^2-x)/2
```

  Donc :
  $4+5+...19 = (20\hat{}2-20)/2 - (4\hat{}2-4)/2 = 190-6 = 184$ On tape :

```
sum(1/(x*(x+1)),x)
```

  On obtient :

```
-1/x
```

  On tape :

```
sum(cos(p*x),p)
```

  On obtient :

```
(-cos(p*x)*cos(x)+cos(p*x)-sin(p*x)*sin(x))/(2*cos(x)-2)
```

Autes Exemples
On tape :

```
sum(k,k,2,6)
```

On obtient :

```
20
```

On tape :

```
sum(k,k,7,2)
```

On obtient :

```
-18
```

On tape :

```
sum(k,k,3,2)
```

On obtient :

```
0
```

On tape :

```
sum(1,k,-2,n)
```

On obtient :

```
n+1+2
```

On tape :

```
normal(sum(2*k-1,k,1,n))
```

On obtient :

```
n^2
```

On tape :

```
sum(1/(n^2),n,1,10)
```

On obtient :

```
1968329/1270080
```

On tape :

```
sum(1/(n^2),n,1,+(infinity))
```

On obtient :

```
pi^2/6
```

On tape :

```
sum((-1)^n/(2*n+1)!,n,0,+(infinity))
```

On obtient :

```
sin(1))
```

On tape :

```
sum(1/(3*n)!,n,0,+(infinity))
```

On obtient :

```
(2*cos((sqrt(3))/2)*exp(1/-2)+exp(1))/3
```

On tape :

```
sum(1/(n*2^n),n,1,+(infinity))
```

On obtient :

```
-(ln(1/2))
```

On tape :

```
sum((-1)^n/(2*n+1),n,0,+(infinity))
```

On obtient :

```
pi/4
```

On tape :

```
assume(x>0 && x<1);
sum(x^(2*n)/(2*n+1),n,0,+(infinity))
```

On obtient :

```
x,(-ln(-x+1)+ln(x+1))/(2*x)
```

On tape :

```
sum(1/(n^3-n),n,2,10)
```

On obtient :

```
27/110
```

On tape :

```
sum(1/(n^3-n),n,1,+(infinity))
```

On obtient :

```
1/4
```

Pour justifier ce résultat on décompose `1/(n^3 - n)`, on tape :

```
partfrac(1/(n^3-n))
```

On obtient :

```
1/(2*(n+1))-1/n+1/(2*(n-1))
```

Donc quand on fait la somme de `2` à `N` on a :
$\sum_{n=2}^{N} -\frac{1}{n} = -\sum_{n=1}^{N-1} \frac{1}{n+1} = -\frac{1}{2} - \sum_{n=2}^{N-2} \frac{1}{n+1} - \frac{1}{N}$
$\frac{1}{2} * \sum_{n=2}^{N} \frac{1}{n-1} = \frac{1}{2} * (\sum_{n=0}^{N-2} \frac{1}{n+1}) = \frac{1}{2} * (1 + \frac{1}{2} + \sum_{n=2}^{N-2} \frac{1}{n+1})$
$\frac{1}{2} * \sum_{n=2}^{N} \frac{1}{n+1} = \frac{1}{2} * (\sum_{n=2}^{N-2} \frac{1}{n+1} + \frac{1}{N} + \frac{1}{N+1})$
les termes $\sum_{n=2}^{N-2}$ se détruisent et il reste :
$-\frac{1}{2} + \frac{1}{2} * (1 + \frac{1}{2}) - \frac{1}{N} + \frac{1}{2} * (\frac{1}{N} + \frac{1}{N+1}) = \frac{1}{4} - \frac{1}{2N(N+1)}$
d'ou les résultat précédents :
- pour $N = 10$ la somme vaut : $1/4 - 1/220 = 27/110$
- pour $N = +\infty$ la somme vaut : $1/4$ car $\frac{1}{2N(N+1)}$ tend vers zéro quand N tend vers l'infini.

6.19.4 Somme de Riemann : `sum_riemann`

`sum_riemann` a deux arguments : une expression `Xpr` dépendant de deux variables et la liste des noms de ces deux variables.
`sum_riemann(Xpr(n,k),[n,k])` renvoie un équivalent, au voisinage de $n = +\infty$, de $\sum_{k=1}^{n} Xpr(n,k)$ ou de $\sum_{k=0}^{n-1} Xpr(n,k)$ ou de $\sum_{k=1}^{n-1} Xpr(n,k)$, lorsque la somme considérée est une somme de Riemann associée à une fonction continue sur [0,1] ou répond quand la recherche a été infructueuse `"ce n'est probablement pas une somme de Riemann"`.
Exercice 1
Soit $S_n = \sum_{k=1}^{n} \frac{k^2}{n^3}$.
Calculer $\lim_{n\to+\infty} S_n$.
On tape :

```
sum_riemann(k^2/n^3,[n,k])
```

On obtient :

```
1/3
```

car :

$$\sum_{k=1}^{n} \frac{k^2}{n^3} = \frac{1}{n} \sum_{k=1}^{n} \frac{k^2}{n^2}$$

est la somme de riemann associée à :

$$\int_0^1 x^2 dx = \frac{1}{3}$$

Exercice 2
Soit $S_n = \sum_{k=1}^{n} \frac{k^3}{n^4}$.
Calculer $\lim_{n\to+\infty} S_n$.
On tape :

```
sum_riemann(k^3/n^4,[n,k])
```

On obtient :

```
1/4
```

car :

$$\sum_{k=1}^{n} \frac{k^3}{n^4} = \frac{1}{n}\sum_{k=1}^{n} \frac{k^3}{n^3}$$

est la somme de riemann associée à :

$$\int_0^1 x^3 dx = \frac{1}{4}$$

Exercice 3
Calculer $\lim_{n\to+\infty} (\frac{1}{n+1} + \frac{1}{n+2} + ... + \frac{1}{n+n})$.
On tape :

```
sum_riemann(1/(n+k),[n,k])
```

On obtient :

```
log(2)
```

car :

$$\sum_{k=1}^{n} \frac{1}{n+k} = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{1+(k/n)}$$

est la somme de riemann associée à :

$$\int_0^1 \frac{1}{1+x} dx = \ln(1+1) = \ln(2)$$

Exercice 4
Soit $S_n = \sum_{k=1}^{n} \frac{32n^3}{16n^4 - k^4}$.
Calculer $\lim_{n\to+\infty} S_n$.
On tape :

```
sum_riemann(32*n^3/(16*n^4-k^4),[n,k])
```

On obtient :

```
2*atan(1/2)+log(3)
```

car :

$$\sum_{k=1}^{n} \frac{32n^3}{16n^4 - k^4} = \frac{1}{n}\sum_{k=1}^{n} \frac{32}{16-(k/n)^4}$$

est la somme de riemann associée à :

$$\int_0^1 \frac{32}{16-x^4}dx = \int_0^1 \frac{1}{x+2} - \frac{1}{x-2}\frac{4}{x^2+4}$$

qui vaut donc $\ln(3) - \ln(2) + \ln(2) - \ln(1) + 2 \text{ atan}(1/2) = \ln(3) + 2 \text{ atan}(1/2)$

Exercice 5

Calculer $\lim_{n\to+\infty}(\frac{n}{n^2+1^2} + \frac{n}{n^2+2^2} + ... + \frac{n}{n^2+n^2})$.

On tape :

```
sum_riemann(n/(n^2+k^2),[n,k])
```

On obtient :

```
pi/4
```

car :

$$\sum_{k=1}^{n} \frac{n}{n^2+k^2} = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{1+(k/n)^2}$$

est la somme de riemann associée à :

$$\int_0^1 \frac{1}{1+x^2}dx = \text{ atan}(1) = \frac{\pi}{4}$$

Exercice 6

Calculer $\lim_{n\to+\infty}(\frac{1}{\sqrt{n^2+1^2}} + \frac{1}{\sqrt{n^2+2^2}} + ... + \frac{1}{\sqrt{n^2+n^2}})$.

On tape :

```
sum_riemann(1/sqrt(n^2+k^2),[n,k])
```

On obtient :

```
-ln(sqrt(2)-1)
```

car :

$$\sum_{k=1}^{n} \frac{1}{\sqrt{n^2+k^2}} = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{\sqrt{1+(k/n)^2}}$$

est la somme de riemann associée à :

$$\int_0^1 \frac{1}{\sqrt{1+x^2}}dx = \ln(1+\sqrt{1+1^2} - \ln(0+\sqrt{1+0^2} = \ln(1+\sqrt{2}$$

6.19.5 Intégration par parties : `integrer_par_parties_dv ibpdv` et `integrer_par_parties_u ibpu`

`ibpdv`

`ibpdv` permet de chercher une primitive (ou de calculer une intégrale définie) d'une expression de la forme $u(x).v'(x)$.
`ibpdv` a deux paramètres pour les primitives et cinq paramètres pour les intégrales définies :

- soit une expression de la forme $u(x).v'(x)$ et $v(x)$ (ou une liste de deux expressions $[F(x), u(x) * v'(x)]$ et $v(x)$),
- soit une expression de la forme $g(x)$ et 0 (ou une liste de deux expressions $[F(x), g(x)]$ et 0).
- pour les intégrales définies, il faut rajouter trois autres paramètres : le nom de la variable et les bornes.

Lorsque `ibpdv` a 2 arguments `ibpdv` renvoie :

- si $v(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (ou une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),
- si le deuxième argument est nul, une primitive de $g(x)$ (le premier argument) (ou $F(x)$+une primitive de $g(x)$) :
  donc, `ibpdv(g(x),0)` renvoie une primitive `G(x)` de `g(x)` ou
  `ibpdv([F(x),g(x)],0)` renvoie `F(x)+G(x)` où `diff(G(x))=g(x)`.

C'est à dire `ibpdv` renvoie les termes que l'on doit calculer quand on fait une intégration par parties, en faisant éventuellement plusieurs `ibpdv` à la suite.
Ainsi, lorsque l'on vient d'utiliser la commande `ibpdv(u(x)*v'(x),v(x))`, il reste alors à calculer l'intégrale du deuxième terme puis à faire la somme avec le premier terme pour obtenir une primitive de $u(x).v'(x)$: pour cela on peut utiliser à nouveau la commande `ibpdv` avec comme premier paramètre la liste obtenue et comme deuxième paramètre un nouveau $v(x)$ (ou 0 pour terminer l'intégration).
On tape :

```
ibpdv(ln(x),x)
```

On obtient :

```
[x.ln(x),-1]
```

puis

```
ibpdv([x.ln(x),-1],0)
```

On obtient :

```
-x+x.ln(x)
```

Lorsque `ibpdv` a 5 arguments `ibpdv(u(x)*v'(x),v(x),x,a,b)` ou `ibpdv([F(x),u(x)*v'(x),v(x),x,a,b)` renvoie :

- si $v(x) \neq 0$, une liste formée de $u(b).v(b)-u(a).v(a)$ et de $-v(x).u'(x)$ (ou une liste formée de $F(b)+u(b).v(b)-F(a)-u(a).v(a)$ et de $-v(x).u'(x)$),
- si le deuxième argument est nul, `ibpdv(g(x),0,x,a,b)` renvoie $G(b)-G(a)$ où $G(x)$ est une primitive du premier argument $g(x)$ et
  `ibpdv([F(x),g(x)],0,x,a,b)` renvoie $F(x)+G(b)-G(a)$ où $G(x)$ est une primitive de $g(x)$) de façon à pouvoir faire plusieurs `ibpdv` à la suite.

On tape :

```
ibpdv(ln(x),x,x,2,3)
```

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis

```
ibpdv([3*ln(3)-2*ln(2),-1],0,x,2,3)
```

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

Remarque
Lorsque le premier paramètre de `ibpdv` est une liste de deux éléments, `ibpdv` n'agit que sur le dernier élément de cette liste et ajoute le terme intégré au premier élément de la liste (de façon à pouvoir faire plusieurs `ibpdv` à la suite).
On a par exemple :
`ibpdv((log(x))^2,x) = [x*(log(x))^2,-(2*log(x))]`
il reste à intégrer `-(2*log(x))`, on utilise `ibpdv(ans(),x)` ou on tape :
`ibpdv([x*(log(x))^2,-(2*log(x))],x)`
On obtient :
`[x*(log(x))^2+x*(-(2*log(x))),2]`
et il reste à intégrer 2, on utilise `ibpdv(ans(),0)` :
`ibpdv([x*(log(x))^2+x*(-(2*log(x))),2],0)`.
On obtient : `x*(log(x))^2+x*(-(2*log(x)))+2*x`

`ibpu`

`ibpu` permet de chercher une primitive (ou de calculer une intégrale définie) d'une expression de la forme $u(x).v'(x)$.
`ibpu` a deux paramètres pour les primitives et cinq paramètres pour les intégrales définies :
- soit une expression de la forme $u(x).v'(x)$ et $u(x)$ (ou une liste de deux expressions $[F(x), u(x) * v'(x)]$ et $u(x)$),
- soit une expression de la forme $g(x)$ et 0 (ou une liste de deux expressions $[F(x), g(x)]$ et 0).
- pour les intégrales définies, il faut rajouter trois autres paramètres : le nom de la variable et les bornes.

Lorsque `ibpu` a 2 arguments `ibpu` renvoie : `ibpu` renvoie :
- si $u(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (ou une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),
- si le deuxième argument est nul, une primitive de $g(x)$ (le premier argument) (ou $F(x)$+une primitive de $g(x)$) :
  `ibpu(g(x),0)` renvoie `G(x)` où `diff(G(x))=g(x)` ou
  `ibpu([F(x),g(x)],0)` renvoie `F(x)+G(x)` où `diff(G(x))=g(x)`.

c'est à dire `ibpu` renvoie les termes que l'on doit calculer quand on fait une intégration par parties, en faisant éventuellement plusieurs `ibpu` à la suite.
Ainsi, lorsque l'on vient d'utiliser la commande `ibpu(u(x)*v'(x),u(x))`, il reste à calculer l'intégrale du deuxième terme puis à faire la somme avec le premier terme pour obtenir une primitive de $u(x).v'(x)$. Pour cela, on peut utiliser à nouveau la commande `ibpu` avec comme premier paramètre la liste obtenue et comme deuxième paramètre un nouveau $u(x)$ (ou 0 pour terminer l'intégration).
On tape :

```
ibpu(ln(x),ln(x))
```

On obtient :

```
[x.ln(x),-1]
```

puis

```
ibpu([x.ln(x),-1],0)
```

On obtient :

```
-x+x.ln(x)
```

Lorsque `ibpu` a 5 arguments `ibpu(u(x)*v'(x),u(x),x,a,b)` ou `ibpu([F(x),u(x)*v'`
renvoie :

- si $u(x) \neq 0$, une liste formée de $u(b).v(b)-u(a).v(a)$ et de $-v(x).u'(x)$ (ou une liste formée de $F(b)+u(b).v(b)-F(a)-u(a).v(a)$ et de $-v(x).u'(x)$),
- si le deuxième argument est nul, `ibpu(g(x),0,x,a,b)` renvoie $G(b) - G(a)$ où $G(x)$ une primitive de $g(x)$ (le premier argument) (ou `ibpu([F(x),g(x)],0,x,a,b`
  renvoie $F(x) + G(b) - G(a)$ où $G(x)$ est une primitive de $g(x)$) de façon à pouvoir faire plusieurs `ibpu` à la suite.

On tape :

```
ibpu(ln(x),ln(x),x,2,3)
```

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis

```
ibpu([3*ln(3)-2*ln(2),-1],0,x,2,3)
```

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

Remarque
Lorsque le premier paramètre de `ibpu` est une liste de deux éléments, `ibpu` n'agit que sur le dernier élément de cette liste et ajoute le terme intégré au premier élément de la liste (de façon à pouvoir faire plusieurs `ibpu` à la suite).

On a par exemple :

```
ibpu((log(x))^2,log(x)) = [x*(log(x))^2,-(2*log(x))]
```

il reste à intégrer `-(2*log(x))`, on utilise `ibpu(ans(),log(x))` ou on tape :
`ibpu([x*(log(x))^2,-(2*log(x))],log(x))`
On obtient :
`[x*(log(x))^2+x*(-(2*log(x))),2]`
et il reste à intégrer `2`, on utilise `ibpu(ans(),0)` :
`ibpu([x*(log(x))^2+x*(-(2*log(x))),2],0)`.
On obtient : `x*(log(x))^2+x*(-(2*log(x)))+2*x`

6.19.6 Changement de variables : `subst substituer`

On se reportera à la commande `subst` de la section 6.13.21.

6.19.7 Longueur d'un arc de courbe : `arcLen`

`arcLen` a soit 1 soit quatre paramètres.

- le paramètre est soit un cercle ou un arc de cercle, soit un polygone.
  On tape
  `arcLen(cercle(0,1,0,pi/2))`
  On obtient :
  `pi/4`
  On tape
  `arcLen(hexagone(0,1))`
  On obtient :
  `6`
- les 4 paramètres sont :une expression $expr$ (resp une liste de 2 expressions $[expr1, expr2]$), le nom d'un paramètre et deux valeurs a et b de ce paramètre.
  `arcLen` calcule la longueur de l'arc de courbe définie par l'équation $y = f(x) = expr$ (resp par $x = expr1, y = expr2$) pour les valeurs du paramètre comprises entre a et b.
  On a donc `arcLen(f(x),x,a,b)=:`
  `integrate(sqrt(diff(f(x),x)^2+1),x,a,b)`
  ou
  `integrate(sqrt(diff(x(t),t)^2+diff(y(t),t)^2),t,a,b)`.

Exemples

- Calculer la longueur de l'arc de cercle AB (avec $A = (0, 0)$ et $B = (0, 1)$) et d'angle au centre $\pi/2$.
  On tape
  `arcLen(arc(0,1,pi/2))`
  On obtient :
  `sqrt(2)*pi/4`
- Calculer le périmètre du triangle ABC (avec $A = (0, 0)$, $B = (0, 1)$ et $C = (1, 1)$).
  On tape
  `arcLen(triangle(0,1,1+i))`

On obtient :

`sqrt(2)+2`

On tape

On obtient :

- Calculer la longueur de l'arc de parabole $y = x^2$ pour x allant de 0 à $x = 1$.
  On tape

  `arcLen(x^2,x,0,1)`

  ou

  `arcLen([t,t^2],t,0,1)`

  On obtient :

  `(sqrt(5))/2-ln(sqrt(5)-2)/4`
- Calculer la longueur de l'arc de la courbe $y = \cosh(x)$ pour x allant de 0 à $x = \ln(2)$.
  On tape :

  `arcLen(cosh(x),x,0,log(2))`

  On obtient :

  `3/4`
- Calculer la longueur de l'arc de cercle $x = \cos(t), y = \sin(t)$ pour t allant de 0 à $t = 2 * \pi$.
  On tape

  `arcLen([cos(t),sin(t)],t,0,2*pi)`

  On obtient :

  `2*pi`

6.20 Maximum, minimum, tableau de valeurs et graph

6.20.1 Maximum et minimum d'une expression : `fMax fMin`

`fMax` et `fMin` ont comme argument : une expression d'une variable et le nom de cette variable (par défaut `x`).
`fMax` renvoie l'abscisse de la solution principale du maximum de l'expression.
`fMin` renvoie l'abscisse de la solution principale du minimum de l'expression.
On tape :

`fMax(sin(x),x)`

Ou on tape :

`fMax(sin(x))`

Ou on tape :

`fMax(sin(y),y)`

On obtient :

`pi/2`

On tape :

```
fMin(sin(x),x)
```

Ou on tape :

```
fMin(sin(x))
```

Ou on tape :

```
fMin(sin(y),y)
```

On obtient :

```
-pi/2
```

On tape :

```
fMin(sin(x)^2,x)
```

On obtient :

```
0
```

6.20.2 Tableau de valeurs et graphe : `tablefunc` `table_fonction` et `plotfunc`

On peut avoir, dans le tableur, les différentes valeurs d'une expression $f(x)$ pour $x = x_0,\ x_0 + h, ...$, grâce à la commande :
$\mathtt{tablefunc(f(x), x, x_0, h)}$ ou $\mathtt{tablefunc(f(x), x)}$.
Dans ce cas les valeurs de départ $\mathtt{x_0}$ et le pas $\mathtt{h}$, valent par défaut : $\mathtt{x_0} = -5.0$ et $\mathtt{h} = 1.0$.
On ouvre un tableur avec `Alt+t`.
Puis, on sélectionne une case du tableur (par exemple `C0`) et pour avoir une table de `"sinus"`, on tape, dans la ligne de commande du tableur :

```
tablefunc(sin(x),x)
```

On obtient deux colonnes `x` et `sin(x)` :
- dans la colonne `x` il y a la valeur du pas `h` (1.0) et le départ de l'évaluation numérique (-5.0), puis une formule, par exemple `=C2+C$1` qui a été recopiée vers le bas.
- dans la colonne `sin(x)` il y a `"Tablefunc"` puis une formule, par exemple `=evalf(subst(D$0,C$0,C2))`, qui a été aussi recopiée vers le bas.
Les valeurs de `sin(x)` s'affichent alors en face des `x` correspondants.
On peut bien sûr changer le pas ou la valeur de départ ou encore remplacer `sin(x)` par `cos(x)` en changeant la valeur de la cellule correspondante.
La représentation graphique se fait avec la commande :
`plotfunc` pour cela voir 3.7.1.

6.21 Limites

6.21.1 Limites : `limit limite`

`limit` permet de calculer **à condition d'être en radians** la limite d'une expression en un point fini (ou infini). En utilisant un paramètre supplementaire, on peut indiquer si on cherche une limite par valeurs supérieures ou par valeurs inférieures (1 pour dire "par valeurs supérieures" et -1 pour dire "par valeurs inférieures").
`limit` a trois ou quatre arguments :
une expression, le nom de la variable (par exemple `x`), le point limite (par exemple `a`) et un argument optionnel qui indique si la limite est unidirectionnelle ou bidirectionnelle (par défaut `0`). Cet argument est égal à `-1` pour une limite à gauche (`x<a`) ou est égal à `1` pour une limite à droite (`x>a`) ou à `0` pour une limite.
L'argument optionnel est donc utilisé lorsque l'on veut calculer une limite à droite (+1) ou une limite à gauche (-1).
`limit` renvoie la limite demandée (si elle existe !).
Remarque
On peut aussi mettre comme argument `x=a` à la place de `x,a` donc : `limit` a aussi comme arguments une expression dépendant d'une variable, une égalité (variable =la valeur où l'on veut calculer la limite) et éventuellement 1 ou -1 pour indiquer la direction.
Autre remarque
si on tape `limit((-1)^n,n=inf)`, alors `Xcas` répond `bounded_function(5)` ce qui veut dire que la fonction est bornée mais qu'elle n'a pas de limite à l'infini.
On tape :

```
limit(1/x,x,0,-1)
```

ou

```
limit(1/x,x=0,-1)
```

On obtient :

```
-(infinity)
```

On tape :

```
limit(1/x,x,0,1)
```

ou

```
limit(1/x,x=0,1)
```

On obtient :

```
+(infinity)
```

On tape :

```
limit(1/x,x,0,0)
```

ou

```
limit(1/x,x,0)
```

ou

```
limit(1/x,x=0)
```

On obtient :

```
infinity
```

cela veut dire que `abs(1/x)` tend vers $+\infty$ quand x tend vers 0.

Exercices :

- Trouver pour $n > 2$, la limite quand x tend vers 0 de :

  $$\frac{n\tan(x) - \tan(nx)}{\sin(nx) - n\sin(x)}$$

  On tape :
  `limit((n*tan(x)-tan(n*x))/(sin(n*x)-n*sin(x)),x=0)`
  On obtient :
  `2`
- Trouver la limite quand x tend vers $+\infty$ de :

  $$\sqrt{x + \sqrt{x + \sqrt{x}}} - \sqrt{x}$$

  On tape :
  `limit(sqrt(x+sqrt(x+sqrt(x)))-sqrt(x),x=+infinity)`
  On obtient :
  `1/2`
- Trouver la limite quand x tend vers 0 de :

  $$\frac{\sqrt{1 + x + x^2/2} - \exp(x/2)}{(1 - \cos(x))\sin(x)}$$

  On tape :

  `limit((sqrt(1+x+x^2/2)-exp(x/2))/((1-cos(x))*sin(x)),x,0)`
  On obtient :
  `-1/6`

Pour calculer quelquefois des limites plus aisément, il peut être judicieux de quoter le premier argument.
On tape par exemple :

```
limit('(2*x-1)*exp(1/(x-1))',x=+infinity)
```

On remarquera que l'on a quoté ici le premier argument pour qu'il ne soit pas évalué c'est à dire pour qu'il ne soit pas simplifié.
On obtient :

```
+(infinity)
```

6.21.2 Limite et intégrale

On ne donne ici que deux exemple :

– Déterminer la limite quand a tend vers l'infini de :

$$\int_2^a \frac{1}{x^2}\, dx$$

On tape :

`limit(integrate(1/(x^2),x,2,a),a,+(infinity))`

On obtient (vérifier que `a` est formelle sinon faire `purge(a)`) :

`1/2`

En effet $\int_2^a \frac{1}{x^2}\, dx = \frac{1}{2} - \frac{1}{a}$

Donc $\int_2^a \frac{1}{x^2}\, dx$ tend vers $\frac{1}{2}$ quand a tend vers l'infini.

– Déterminer la limite quand a tend vers l'infini de :

$$\int_2^a (\frac{x}{x^2-1} + \ln(\frac{x+1}{x-1}))\, dx$$

On tape :

```
limit(integrate(x/(x^2-1)+log((x+1)/(x-1)),x,2,a),
                a,+(infinity))
```

On obtient (vérifier que `a` est formelle sinon faire `purge(a)`) :

`+(infinity)`

En effet :

$\int_2^a \frac{x}{x^2-1}\, dx = \frac{1}{2}(\ln(a^2-1) - \ln(3))$ et

$\int_2^a \ln(\frac{x+1}{x-1})\, dx = ln(a+1) + ln(a-1) + a * ln(\frac{a+1}{a-1}) - 3\ln(3)$ Donc quand a tend vers $+\infty$ l'intégrale tend vers $+\infty$.

– Déterminer la limite quand a tend vers 0 de :

$$\int_a^{3a} \cos(x)/x\, dx$$

`limit(int(cos(x)/x,x,a,3a),a,0)`

On obtient (vérifier que `a` est formelle sinon faire `purge(a)`) :

`ln(3)`

Pour trouver cette limite on encadre $\frac{\cos(x)}{x}$ car on ne connait pas la primitive de $\frac{\cos(x)}{x}$.

On sait que :

$1 - 2\sin^2\frac{x}{2} = \cos(x) \leq 1$ et

$\sin^2\frac{x}{2} \leq \frac{x^2}{4}$ donc,

$1 - \frac{x^2}{2} = \cos(x) \leq 1$ et

$\frac{1}{x} - \frac{x}{2} \leq \frac{\cos(x)}{x} \leq \frac{1}{x}$

Donc :

$\int_a^{3a} (\frac{1}{x} - \frac{x}{2})\, dx \leq \int_a^{3a} \cos(x)/x\, dx \leq \int_a^{3a} \frac{1}{x}\, dx$.

$\ln(3) - 9a^2/4 + a^2/4 \leq \int_a^{3a} \cos(x)/x\, dx \leq \ln(3)$.

Donc $\int_a^{3a} \cos(x)/x\, dx$ tend vers $\ln(3)$ quand a tend vers 0.

6.22 Réécrire des expressions transcendantes et trigonométriques

6.22.1 Développer une expression transcendante et trigonométrique : `texpand tExpand developper_transcendant`

`texpand` ou `tExpand` a comme argument une expression transcendante et trigonométrique.
`texpand` ou `tExpand` est la généralisation de `expexpand`, `lnexpand` et `trigexpand` car elle développe les expressions transcendantes et trigonométriques.
`texpand` ou `tExpand` permet, par exemple, de transformer $\ln(x^n)$ en $n\ln(x)$, $\exp(x)^n$ en $\exp(nx)$ et $\sin(2x)$ en $2\sin(x)\cos(x)$.

- `texpand` ou `tExpand` a comme argument une expression transcendante et trigonométrique.
  Exemple :
  Développer $\exp(x+y)+\cos(x+y)+\ln(3x^2)$.
  On tape :
  ```
  texpand(exp(x+y)+cos(x+y)+ln(3*x^2))
  ```
  On obtient :
  ```
  cos(x)*cos(y)-sin(x)*sin(y)+exp(x)*exp(y)+
  ln(3)+2*ln(x)
  ```
- `texpand` ou `tExpand` a comme argument une expression trigonométrique. `texpand` ou `tExpand` développe cette expression en fonction de $\sin(x)$ et $\cos(x)$.
  Exemples
  1. Développer $\cos(x+y)$.
     On tape :
     ```
     texpand(cos(x+y))
     ```
     On obtient :
     ```
     cos(x)*cos(y)-sin(x)*sin(y)
     ```
  2. Développer $\cos(3x)$.
     On tape :
     ```
     texpand(cos(3*x))
     ```
     On obtient :
     ```
     4*(cos(x))^ 3-3*cos(x)
     ```
  3. Développer $\dfrac{\sin(3*x)+\sin(7*x)}{\sin(5*x)}$.
     On tape :
     ```
     texpand((sin(3*x)+sin(7*x))/sin(5*x))
     ```
     On obtient
     ```
     (4*(cos(x))^2-1)*(sin(x)/(16*(cos(x))^4-
     12*(cos(x))^2+1))/sin(x)+(64*(cos(x))^6-
     ```

```
80*(cos(x))^4+24*(cos(x))^2- 1)*sin(x)/
(16*(cos(x))^4-12*(cos(x))^2+1)/sin(x)
```

Et, après une simplification en tapant `normal(ans())`, on obtient :

```
4*(cos(x))^2-2
```

– `texpand` ou `tExpand` a comme argument une expression transcendante.
`texpand` ou `tExpand` développe cette expression.
Exemples

1. Développer $\exp(x+y)$.
   On tape :

   ```
   texpand(exp(x+y))
   ```

   On obtient :

   ```
   exp(x)*exp(y)
   ```

2. Développer $\ln(x+y)$.
   On tape :

   ```
   texpand(log(x*y))
   ```

   On obtient :

   ```
   log(x)+log(y)
   ```

3. Développer $\ln(x^n)$.

   ```
   texpand(ln(x^n))
   ```

   On obtient :

   ```
   n*ln(x)
   ```

4. Développer $\ln((e^2)+\exp(2*\ln(2))+exp(\ln(3)+\ln(2)))$.
   On tape :

   ```
   texpand(log(e^2)+exp(2*log(2))+exp(log(3)+log(2)))
   ```

   On obtient :

   ```
   6+3*2
   ```

   Ou on tape :

   ```
   texpand(log(e^2)+exp(2*log(2)))+
   lncollect(exp(log(3)+log(2)))
   ```

   On obtient :

   12

6.22.2 Rassembler les termes de même nature : `combine`

`combine` a deux arguments : une expression `Xpr` et une option `exp, log, ln, sin, cos, trig` le nom d'une classe de fonction.
`combine` rassemble les termes de l'expression contenant cette fonction.
`combine(Xpr, ln)` ou `combine(Xpr, log)` donne le même résultat que `lncollect(Xpr)`
`combine(Xpr,trig)` ou `combine(Xpr,sin)` ou `combine(Xpr,cos)`
donne le même résultat que `tcollect(Xpr)`.
On tape :

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),exp)
```

On obtient :

```
exp(x+y)+sin(x)*cos(x)+ln(x)+ln(y)
```

On tape :

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),trig)
```

ou

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),sin)
```

ou

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),cos)
```

On obtient :

```
exp(y)*exp(x)+(sin(2*x))/2+ln(x)+ln(y)
```

On tape :

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),ln)
```

ou

```
combine(exp(x)*exp(y)+sin(x)*cos(x)+ln(x)+ln(y),log)
```

On obtient :

```
exp(x)*exp(y)+sin(x)*cos(x)+ln(x*y)
```

6.23 Les expressions trigonométriques

6.23.1 Les différentes fonctions trigonométriques

`sin` désigne la fonction sinus,
`cos` désigne la fonction cosinus,
`tan` désigne la fonction tangente (`tan(x)= sin(x)/cos(x)`),
`cot` désigne la fonction cotangente (`cot(x)= cos(x)/sin(x)`),
`sec` désigne la fonction sécante (`sec(x)= 1/cos(x)`),
`csc` désigne la fonction cosécante (`csc(x) = 1/sin(x)`),
`asin` ou `arcsin`, `acos` ou `arccos`, `atan` ou `arctan`, `acot`, `asec`, `acsc`
désignent les fonctions réciproques des fonctions précédentes.
On a :
`asec(x) = acos(1/x)`,
`acsc(x) = asin(1/x)`,
`acot(x) = atan(1/x)`.

6.23.2 Développer une expression trigonométriques : `trigexpand`

`trigexpand` a comme argument une expression trigonométrique.
`trigexpand` développe cette expression en fonction de $\sin(x)$ et $\cos(x)$.
On tape :

```
trigexpand(cos(x+y))
```

On obtient :

```
cos(x)*cos(y)-sin(x)*sin(y)
```

6.23.3 Linéariser une expression trigonométrique : `tlin` `lineariser_trigo`

`tlin` a comme argument une expression trigonométrique.
`tlin` linéarise cette expression en fonction de $\sin(n.x)$ et $\cos(n.x)$.
Exemples

– Linéariser $\cos(x) * \cos(y)$.
  On tape :
  ```
  tlin(cos(x)*cos(y))
  ```
  On obtient :
  ```
  1/2*cos(x-y)+1/2*cos(x+y)
  ```
– Linéariser $\cos(x)^3$.
  On tape :
  ```
  tlin(cos(x)^3)
  ```
  On obtient :
  ```
  3/4*cos(x)+1/4*cos(3*x)
  ```
– Linéariser $4\cos(x)^2 - 2$.
  On tape :
  ```
  tlin(4*cos(x)^2-2)
  ```
  On obtient :
  ```
  2*cos(2*x)
  ```

6.23.4 Augmenter la phase de $\frac{\pi}{2}$ dans les expressions trigonométriques : `shift_phase`

`shift_phase` a comme argument une expression trigonométrique.
`shift_phase` permet d'augmenter la phase de $\frac{\pi}{2}$ dans les expressions trigonométriques une fois que la simplification automatique a eu lieu. On tape :

```
shift_phase(x+sin(x))
```

On obtient :

```
x-cos((pi+2*x)/2)
```

On tape :

```
shift_phase(x+cos(x))
```

On obtient :

```
x-+sin((pi+2*x)/2)
```

On tape :

```
shift_phase(x+tan(x))
```

On obtient :

```
x+1/(tan((pi+2*x)/2))
```

Si on ne veut pas que l'expression soit évaluée (i.e. qu'il n'y ait pas de simplification automatique), il faut quoter l'argument. On tape :

```
shift_phase('sin(x+pi/2)')
```

On obtient :

```
-(cos(pi+x))
```

Mais si on tape sans quoter le sinus :

```
shift_phase(sin(x+pi/2))
```

On obtient :

```
sin((pi+2*x)/2)
```

car `sin(x+pi/2)` est évaluée (i.e. simplifiée) en `cos(x)` avant que la commande `shift_phase` ne soit appelée et ensuite `shift_phase(cos(x))` renvoie `sin((pi+2*x)/2)`. **Exercice**
Calcul de $\sum_{n=1}^{+\infty} \frac{\sin(n*x)}{n}$
On tape :
`normal(sum((sin(n*x))/n,n=1..+infinity))`
On obtient :
`-atan((sin(x))/(cos(x)-1))`
On tape :
`normal(shift_phase(halftan(atan(sin(x)/(-cos(x)+1)))))`
On obtient :
`pi*floor(((pi+x)/2)/pi+1/2)+(-1)/2*pi+(-1)/2*x`
si on tape :
`tsimplify(atan((sin(x))/(-cos(x)+1)))`
On obtient car `tsimplify` n'est pas rigoureux vis a vis des $2k\pi$:
`-1/2*pi-1/2*x`

6.23.5 Rassembler les sinus et les cosinus de même angle : `tcollect` `tCollect` `rassembler_trigo`

`tcollect` ou `tCollect` a comme argument une expression trigonométrique.
`tcollect` linéarise cette expression en fonction de $\sin(n.x)$ et $\cos(n.x)$ puis rassemble les sinus et les cosinus de même angle.
On tape :

```
tcollect(sin(x)+cos(x))
```

On obtient :

```
sqrt(2)*cos(x-pi/4)
```

On tape :

```
tcollect(2*sin(x)*cos(x)+cos(2*x))
```

On obtient :

```
sqrt(2)*cos(2*x-pi/4)
```

6.23.6 Simplifier : `simplify simplifier`

`simplify` simplifie l'expression de façon automatique.
Comme toutes simplifications automatiques, il ne faut pas s'attendre à des miracles et pourtant...
On tape :

```
simplify((sin(3*x)+sin(7*x))/sin(5*x))
```

On obtient :

```
4*(cos(x))^2-2
```

Attention `simplify` est plus efficace qu'en on est en mode `radian` (pour cela on coche `radian` dans la configuration du `cas` ou bien on tape `angle_radian:=1`).

6.23.7 Transformer les arccos en arcsin : `acos2asin`

`acos2asin` a comme argument une expression trigonométrique.
`acos2asin` transforme cette expression en remplaçant :
$\arccos(x)$ par $\frac{\pi}{2} - \arcsin(x)$.
On tape :

```
acos2asin(acos(x)+asin(x))
```

On obtient après simplification :

```
pi/2
```

6.23.8 Transformer les arccos en arctan : `acos2atan`

`acos2atan` a comme argument une expression trigonométrique.
`acos2atan` transforme cette expression en remplaçant :
$\arccos(x)$ par $\frac{\pi}{2} - \arctan(\frac{x}{\sqrt{1-x^2}})$.
On tape :

```
acos2atan(acos(x))
```

On obtient :

```
pi/2-atan(x/sqrt(1-x^2))
```

6.23.9 Transformer les arcsin en arccos : `asin2acos`

`asin2acos` a comme argument une expression trigonométrique.
`asin2acos` transforme cette expression en remplaçant :
$\arcsin(x)$ par $\frac{\pi}{2} - \arccos(x)$.
On tape :

```
asin2acos(acos(x)+asin(x))
```

On obtient après simplification :

```
pi/2
```

6.23.10 Transformer les arcsin en arctan : `asin2atan`

`asin2atan` a comme argument une expression trigonométrique.
`asin2atan` transforme cette expression en remplaçant :
$\arcsin(x)$ par $\arctan(\frac{x}{\sqrt{1-x^2}})$.
On tape :

```
asin2atan(asin(x))
```

On obtient :

```
atan(x/sqrt(1-x^2))
```

6.23.11 Transformer les arctan en arcsin : `atan2asin`

`atan2asin` a comme argument une expression trigonométrique.
`atan2asin` transforme cette expression en remplaçant :
$\arctan(x)$ par $\arcsin(\frac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2asin(atan(x))
```

On obtient :

```
asin(x/sqrt(1+x^2))
```

6.23.12 Transformer les arctan en arccos : `atan2acos`

`atan2acos` a comme argument une expression trigonométrique.
`atan2acos` transforme cette expression en remplaçant :
$\arctan(x)$ par $\frac{\pi}{2} - \arccos(\frac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2acos(atan(x))
```

On obtient :

```
pi/2-acos(x/sqrt(1+x^2))
```

6.23.13 Transformer les exponentielles complexes en sin et en cos : `sincos` `exp2trig`

`sincos` ou `exp2trig` a comme argument une expression contenant des exponentielles complexes.
`sincos` ou `exp2trig` transforme cette expression en fonction de $\sin(x)$ et de $\cos(x)$.
On tape :

```
sincos(exp(i*x))
```

On obtient si `Variables_complex` n'est pas coché dans la configuration du CAS :

```
cos(x)+i*sin(x)
```

On obtient si `Variables_complex` est coché dans la configuration du CAS :

```
exp(im(x))*(cos(re(x))+(i)*sin(re(x)))
```

On tape :

```
exp2trig(exp(-i*x))
```

On obtient si `Variables_complex` n'est pas coché dans la configuration du CAS :

```
cos(x)-i*sin(x)
```

On tape :

```
simplify(sincos(((i)*(exp((i)*x))^2-i)/(2*exp((i)*x))))
```

Ou on tape :

```
simplify(exp2trig(((i)*(exp((i)*x))^2-i)/(2*exp((i)*x))))
```

On obtient :

```
-sin(x)
```

6.23.14 Transformer tan(x) en sin(x)/cos(x) : `tan2sincos`

`tan2sincos` a comme argument une expression trigonométrique.
`tan2sincos` transforme cette expression en remplaçant :
$\tan(x)$ par $\dfrac{\sin(x)}{\cos(x)}$.
On tape :

```
tan2sincos(tan(2*x))
```

On obtient :

```
sin(2*x)/cos(2*x)
```

6.23.15 Transformer sin(x) en cos(x)*tan(x) : `sin2costan`

`sin2costan` a comme argument une expression trigonométrique.
`sin2costan` transforme cette expression en remplaçant :
$\sin(x)$ par $\cos(x) * \tan(x)$.
On tape :

```
sin2costan(sin(2*x))
```

On obtient :

```
cos(2*x)*tan(2*x)
```

6.23.16 Transformer cos(x) en sin(x)/tan(x) : `cos2sintan`

`cos2sintan` a comme argument une expression trigonométrique.
`cos2sintan` transforme cette expression en remplaçant :
$\cos(x)$ par $\dfrac{\sin(x)}{\tan(x)}$.
On tape :

```
cos2sintan(cos(2*x))
```

On obtient :

```
sin(2*x)/tan(2*x)
```

6.23.17 Transformer tan(x) avec sin(2x) et cos(2x) : `tan2sincos2`

`tan2sincos2` a comme argument une expression trigonométrique.
`tan2sincos2` transforme cette expression en remplaçant :
$\tan(x)$ par $\dfrac{\sin(2.x)}{1+\cos(2.x)}$.
On tape :

```
tan2sincos2(tan(x))
```

On obtient :

```
sin(2*x)/(1+cos(2*x))
```

6.23.18 Transformer tan(x) avec cos(2x) et sin(2x) : `tan2cossin2`

`tan2cossin2` a comme argument une expression trigonométrique.
`tan2cossin2` transforme cette expression en remplaçant :
$\tan(x)$ par $\dfrac{1-\cos(2.x)}{\sin(2.x)}$.
On tape :

```
tan2cossin2(tan(x))
```

On obtient :

```
(1-cos(2*x))/sin(2*x)
```

6.23.19 Transformer une expression trigonométrique en fonction de tan(x/2) : `halftan`

`halftan` a comme argument une expression trigonométrique.
`halftan` transforme les $\sin(x)$, $\cos(x)$ et $\tan(x)$ contenus dans l'expression en fonction de $\tan(\frac{x}{2})$.
On tape :

```
halftan(sin(x))
```

On obtient :

```
2*tan(x/2)/(1+tan(x/2)^2)
```

On tape :

```
halftan(sin(2*x)/(1+cos(2*x)))
```

On obtient :

```
2*tan(2*x/2)/((tan(2*x/2))^2+1)/
(1+(1-(tan(2*x/2))^2)/((tan(2*x/2))^2+1))
```

Et, après simplification avec `simplify(ans())`, on obtient :

```
tan(x)
```

On tape :

```
halftan(sin(x)^2+cos(x)^2)
```

On obtient :

```
(2*tan(x/2)/((tan(x/2))^2+1))^2+
((1-(tan(x/2))^2)/((tan(x/2))^2+1))^2
```

On obtient, après simplification avec `normal(ans())` :

```
1
```

6.23.20 Transformer les expressions trigonomètriques et hyperboliques en tan(x/2) et en exp(x) : `halftan_hyp2exp`

`halftan_hyp2exp` a comme argument une expression trigonométrique ou hyperbolique.
`halftan_hyp2exp` transforme les $\sin(x)$, $\cos(x)$ et $\tan(x)$ contenus dans l'expression en fonction de $\tan(\frac{x}{2})$ et de $\exp(x)$.
On tape :

```
halftan_hyp2exp(tan(x)+tanh(x))
```

On obtient :

```
(2*tan(x/2))/((1-(tan(x/2))^2))+(((exp(x))^2-1))/
(((exp(x))^2+1))
```

On tape :

```
halftan_hyp2exp(sin(x)^2+cos(x)^2-sinh(x)^2+cosh(x)^2)
```

On obtient, après simplification avec `normal(ans())` :

```
2
```

6.23.21 Transformer avec des fonctions trigonométriques inverses en logarithmes : `atrig2ln`

`atrig2ln` réécrit l'expression contenant des fonctions trigonométriques inverses avec des logarithmes.
On tape :

```
atrig2ln(asin(x))
```

On obtient :

```
i*ln(x+sqrt(x^2-1))+pi/2
```

6.23.22 Transformer une expression trigonométrique en des exponentielles complexes : `trig2exp`

`trig2exp` a comme argument une expression trigonométrique.
`trig2exp` transforme les fonctions trigonométriques en exponentielles complexes SANS linéariser.
On tape :

```
trig2exp(tan(x))
```

On obtient :

```
((exp((i)*x))^2-1)/((i)*((exp((i)*x))^2+1))
```

On tape :

```
trig2exp(sin(x))
```

On obtient :

```
(exp((i)*x)-1/(exp((i)*x)))/(2*i)
```

6.23.23 Simplifier en privilégiant les sinus : `trigsin`

`trigsin` a comme argument une expression trigonométrique.
`trigsin` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les sinus.
On tape :

```
trigsin(sin(x)^4+cos(x)^2+1)
```

On obtient :

```
sin(x)^4-sin(x)^2+2
```

6.23.24 Simplifier en privilégiant les cosinus : `trigcos`

`trigcos` a comme argument une expression trigonométrique.
`trigcos` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1, \tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les cosinus.
On tape :

```
trigcos(sin(x)^4+cos(x)^2+1)
```

On obtient :

```
cos(x)^4-cos(x)^2+2
```

6.23.25 Simplifier en privilégiant les tangentes : `trigtan`

`trigtan` a comme argument une expression trigonométrique.
`trigtan` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1, \tan(x) = \dfrac{\sin(x)}{\cos(x)}$ et en privilégiant les tangentes.
On tape :

```
trigtan(sin(x)^4+cos(x)^2+1)
```

On obtient :

```
((tan(x))^2/(1+(tan(x))^2))^2+1/(1+(tan(x)^2)+1
```

et après simplification avec `normal` on a :

```
(2*tan(x)^4+3*tan(x)^2+2)/(tan(x)^4+2*tan(x))^2+1)
```

6.23.26 Réecriture d'une expression avec différentes options : `convert` `convertir` `=>`

`=>` est la version infixée de `convert`.
`convert` a deux arguments une expression et une option.
`convert` réecrit cette expression en fonction de l'option.
Voici la liste des différentes options :

- `'+'` convertit une expression comme si on appelait directement `expand` (à noter : il ne faut pas quoter + avec =>).
- `'*'` convertit une expression comme si on appelait directement `factor`(à noter : il ne faut pas quoter `*` avec =>).
- `sin` convertit une expression comme si on appelait directement `trigsin`.
- `cos` convertit une expression comme si on appelait directement `trigcos`.
- `sincos` convertit une expression comme si on appelait directement `sincos`.
- `trig` convertit une expression comme si on appelait directement `sincos`.
- `tan` convertit une expression comme si on appelait directement `halftan`.
- `exp` convertit une expression comme si on appelait directement `trig2exp`.
- `ln` convertit une expression comme si on appelait directement `trig2exp`.
- `expln` convertit une expression comme si on appelait directement `trig2exp`.
- `string` convertit une expression en une chaîne comme si on appelait directement `string`.

– `list` convertit un polynôme en une liste(cf 6.26.28).
– `polynom` convertit un développement de Taylor ou une liste en un polynôme (cf 6.26.27 et 6.26.29).
– `parfrac` ou `partfrac` ou `fullparfrac` ou `'+'` convertit une fraction rationnelle en éléments simples comme si on utilisait directement `partfrac`(6.30.9). À noter : il ne faut pas quoter + avec =>.
– `diff` calcule la dérivée d'une expression ou d'une fonction comme si on utilisait directement `diff`.
– `int` calcule l'intégrale d'une expression ou d'une fonction comme si on utilisait directement `int`.

`convert` permet aussi :
– des changements d'unité, par exemple `convert(1000_g,_kg)=1.0_kg` (cf 5.1.3).
– d'écrire un réel selon une fraction continue : `convert(a,confrac,'fc')` écrit `a` selon une fraction continue stockée dans `fc`. Ne pas oublier de quoter le dernier argument ! ! ! Par exemple, `convert(1.2,confrac,'fc')=[1,5]` et `fc` contient la fraction continue égale à 1.2 (cf 6.9.7).
– de transformer un entier en la liste de ses chiffres dans son écriture dans une base, en commencant par le chiffre des unités (et réciproquement) : `convert(n,base,b)` transforme l'entier `n` en la liste de ses chiffres dans son écriture dans la base `b` en commencant par le chiffre des unités. Par exemple, `convert(123,base,10)=[3,2,1]` et réciproquement `convert(l,base,b)` transforme la liste `l` en l'entier `n` qui a `l` pour liste de chiffres dans son écriture dans la base `b` en commencant par le chiffre des unités. Par exemple, `convert([3,2,1],base,10)=123` (cf 6.6).

6.24 Transformée de Fourier

6.24.1 Les coefficients de Fourier : `fourier_an` et `fourier_bn` ou `fourier_cn`

Si la fonction f est continue par morceaux sur $\mathbb{R}$, et est périodique de période T, alors aux points de continuité de f on a :

$$f(x) = a_0 + \sum_{n=1}^{+\infty} a_n \cos(\frac{2\pi nx}{T}) + b_n \sin(\frac{2\pi nx}{T})$$

ou

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{\frac{2i\pi nx}{T}}$$

où les coefficients a_n, b_n, $n \in N$, (ou c_n, $n \in Z$) sont les coefficients de Fourier de f et se calculent avec les fonctions :
`fourier_an` et `fourier_bn` ou `fourier_cn`.

`fourier_an`

`fourier_an` a quatre ou cinq paramètres : une expression Xpr dependant d'une variable, le nom de la variable (par exemple x), la période T, un entier n et a (a

vaut 0 par défaut).
`fourier_an(Xpr,x,T,n,a)` renvoie le coefficient de Fourier a_n d'une fonction de variable x définie sur $[a, a+T[$ par $f(x) = Xpr$ et périodique de période T.
Si f est continue par morceaux `fourier_an(Xpr,x,T,n,a)` renvoie $a_n = \frac{2}{T}\int_a^{a+T} f(x)\cos(\frac{2\pi nx}{T})dx$
Si l'on veut que les calculs soient simplifiés il faut dire que n est un entier en tapant `assume(n,integer)`.
Exemples

–
– Soit la fonction f, de période $T = 2\pi$, définie sur $[-\pi; \pi[$ par $f(x) = 1 + \cos(x) + \sin(x)$.
  On tape, pour avoir son coefficient a_0 :
  ```
  fourier_an(1+cos(x)+sin(x),x,2*pi,0,-pi)
  ```
  On obtient :
  ```
  1
  ```
  On tape, pour avoir son coefficient a_1 :
  ```
  fourier_an(1+cos(x)+sin(x),x,2*pi,1,-pi)
  ```
  On obtient :
  ```
  1
  ```
  On tape, pour avoir son coefficient a_n ($n \neq 0$ et $n \neq 1$) :
  ```
  assume(n,integer)
  fourier_bn(1+cos(x)+sin(x),x,2*pi,n,-pi)
  ```
  On obtient :
  ```
  0
  ```
– Soit la fonction f, de période $T = 2$, définie sur $[-1; 1[$ par $f(x) = x^2$.
  On tape, pour avoir son coefficient a_0 :
  ```
  fourier_an(x^2,x,2,0,-1)
  ```
  On obtient :
  ```
  1/3
  ```
  On tape, pour avoir son coefficient a_n ($n \neq 0$) :
  ```
  assume(n,integer)
  fourier_an(x^2,x,2,n,-1)
  ```
  On obtient :
  ```
  4*(-1)^n/(pi^2*n^2)
  ```

`fourier_bn`

`fourier_bn` a quatre ou cinq paramètres : une expression Xpr dependant d'une variable, le nom de la variable (par exemple x), la période T, un entier n et a (a vaut 0 par défaut).
`fourier_bn(Xpr,x,T,n,a)` renvoie le coefficient de Fourier b_n d'une fonction de variable x définie sur $[a, a+T[$ par $f(x) = Xpr$ et périodique de période T.
Si f est continue par morceaux `fourier_bn(Xpr,x,T,n,a)` renvoie :
$b_n = \frac{2}{T}\int_a^{a+T} f(x)\sin(\frac{2\pi nx}{T})dx$

Si l'on veut que les calculs soient simplifiés il faut dire que n est un entier en tapant `assume(n,integer)`.

Exemples

- Soit la fonction f, de période $T = 2\pi$, définie sur $[-\pi; \pi[$ par $f(x) = 1 + \cos(x) + \sin(x)$.
  On tape, pour avoir son coefficient b_1 :
  `fourier_bn(1+cos(x)+sin(x),x,2*pi,1,-pi)`
  On obtient :
  `1`
  On tape, pour avoir son coefficient b_n ($n \neq 1$) :
  `assume(n,integer)`
  `fourier_bn(1+cos(x)+sin(x),x,2*pi,n,-pi)`
  On obtient :
  `0`
- Soit la fonction f, de période $T = 2$, définie sur $[-1; 1[$ par $f(x) = x^2$.
  On tape, pour avoir son coefficient b_n ($n \neq 0$) :
  `assume(n,integer)`
  `fourier_bn(x^2,x,2,n,-1)`
  On obtient :
  `0`
- Soit la fonction f, de période $T = 2$, définie sur $[-1; 1[$ par $f(x) = x^3$.
  On tape, pour avoir son coefficient b_1 :
  `fourier_bn(x^3,x,2,1,-1)`
  On obtient :
  `(2*pi^2-12)/pi^3`
  On tape, pour avoir son coefficient b_n :
  `assume(n,integer)`
  `fourier_bn(x^3,x,2,n,-1)`
  On obtient :
  `(-2*n^2*pi^2*(-1)^n+12*(-1)^n)/(n^3*pi^3)`

`fourier_cn`

`fourier_cn` a quatre ou cinq paramètres : une expression Xpr, le nom de la variable (par exemple x), la période T, un entier n et a (a vaut 0 par défaut).
`fourier_cn(Xpr,x,T,n,a)` renvoie le coefficient de Fourier c_n d'une fonction de variable x définie sur $[a, a+T[$ par $f(x) = Xpr$ et périodique de période T.
Si $f(x) = Xpr$ est continue par morceaux sur $\mathbb{R}$, `fourier_cn(Xpr,x,T,n,a)` renvoie $c_n = \dfrac{1}{T}\displaystyle\int_a^{a+T} f(x)e^{\frac{-2i\pi nx}{T}}dx$

Exemples

- Soit la fonction f, de période $T = 2\pi$, définie sur $[-\pi; \pi[$ par $f(x) = 1 + \cos(x) + \sin(x)$.
  On tape, pour avoir son coefficient c_0 :
  `fourier_cn(1+cos(x)+sin(x),x,2*pi,0,-pi)`
  On obtient :
  `1`
  On tape, pour avoir son coefficient c_1 :

`fourier_cn(1+cos(x)+sin(x),x,2*pi,1,-pi)`

On obtient :

`(1-i)/2`

On tape, pour avoir son coefficient c_{-1} :

`fourier_cn(1+cos(x)+sin(x),x,2*pi,-1,-pi)`

On obtient :

`(1+i)/2`

On tape, pour avoir son coefficient c_n ($n \neq 0$ et $n \neq \pm 1$) :

`assume(n,integer)`
`fourier_cn(1+cos(x)+sin(x),x,2*pi,n,-pi)`

On obtient :

`0`

– Déterminer les coefficients c_n de Fourier de la fonction f périodique de période 2 et définie sur $[-1;1[$ par $f(x) = x^2$.
On tape, pour avoir c_0 :
`fourier_cn(x^2,x,2,0,-1)`
On obtient :
`1/3`
On tape, pour avoir c_n :
`assume(n,integer)`
`fourier_cn(x^2,x,2,n,-1)`
On obtient :
`2*(-1)^n/(pi^2*n^2)`

– Déterminer les coefficients c_n de Fourier de la fonction f périodique de période 2 et définie sur $[0;2[$ par $f(x) = x^2$.
On tape, pour avoir c_0 :
`fourier_cn(x^2,x,2,0)`
On obtient :
`4/3`
On tape, pour avoir c_n :
`assume(n,integer)`
`fourier_cn(x^2,x,2,n)`
On obtient :
`((2*i)*pi*n+2)/(pi^2*n^2)`

– Déterminer les coefficients c_n de Fourier de la fonction f périodique de période 2π, et définie sur $[0;2\pi[$ par $f(x) = x^2$.
On tape :
`assume(n,integer)`
`fourier_cn(x^2,x,2*pi,n)`
On obtient :
`((2*i)*pi*n+2)/n^2`
Si on ne met pas `assume(n,integer)` on obtient une expression non simplifiée :

```
((2*i)*pi^2*n^2*exp((-i)*n*2*pi)+2*pi*n*exp((-i)*n*2*pi)+
(-i)*exp((-i)*n*2*pi)+i)/(pi*n^3)
```

que l'on peut simplifier en remplacant `exp((-i)*n*2*pi)` par `1` :
`subst(ans(),exp((-i)*n*2*pi)=1)`

On obtient :

```
((2*i)*pi^2*n^2+2*pi*n+-i+i)/pi/n^3
```

expression que l'on peut simplifier avec `normal` et on trouve finalement :

```
((2*i)*pi*n+2)/n^2
```

Il est donc préférable d'écrire `assume(n,integer)`.
Donc si $n \neq 0$ on a :

$$c_n = \frac{2 * i * n * \pi + 2}{n^2}$$

Puis on tape :

```
fourier_cn(x^2,x,2*pi,0)
```

On obtient :

```
4*pi^2/3
```

Donc si $n = 0$ on a :

$$c_0 = \frac{4.\pi^2}{3}$$

Remarque :
Lorsque l'on ne veut plus considérer n comme un entier on doit taper :
`purge(n)`.
Pour connaître les hypothéses faites sur une variable, par exemple n, on tape :
`about(n)`

6.24.2 Transformée de Fourier discrète

Soit N un entier.
On considère une suite x périodique de période N : elle est entièrement déterminée par la liste $x = [x_0, x_1, ...x_{N-1}]$.
La transformée de Fourier discréte est une transformation F_N qui a une suite x périodique de période N fait correspondre une suite y, périodique de période N, définie pour $k = 0..N-1$ par :
$(F_N(x))_k = y_k = \sum_{j=0}^{N-1} x_j \omega_N^{-k \cdot j}$ avec ω_N racine N-ième de l'unité.
Le but est de calculer pour $k = 0..N-1$:

$$\sum_{j=0}^{N-1} x_j \omega_N^{-k \cdot j}$$

avec $\omega_N = \exp(\frac{2i\pi}{N})$
La méthode de la transformée de Fourier rapide permet de calculer rapidement ces sommes si N est une puissance de 2.

Les propriétes

La transformée de Fourier discréte F_N est une transformation bijective.
On a :
$F_N^{-1} = \frac{1}{N}\overline{F_N}$
c'est à dire :
$(F_N^{-1}(x))_k = \dfrac{1}{N} \sum_{j=0}^{N-1} x_j \omega_N^{k \cdot j}$

Notation
Avec `Xcas` on a les fonctions `fft` et `ifft` qui sont :
`fft(x)`=$F_N(x)$ et
`ifft(x)`=$F_N^{-1}(x)$
Définiton
Soient deux suites x et y périodiques de période N.
On définit :
- le produit de Hadamard, noté $\cdot$, par :
$(x \cdot y)_k = x_k y_k$
- le produit de convolution, noté $*$, par :
$(x * y)_k = \sum_{j=0}^{N-1} x_j y_{k-j}$
On a alors :
$N * F_N(x \cdot y) = F_N(x) * F_N(y)$
$F_N(x * y) = F_N(x) \cdot F_N(y)$

Où interviennent ces sommes ?

1. Valeur d'un polynôme
   Soit un polynôme $P(x) = \sum_{j=0}^{N-1} c_j x^j$ donné par la liste de ces coefficients $[c_0, c_1, ..c_{N-1}]$ complété par des zéros pour que N soit une puissance de 2.
   – On veut calculer les valeurs prises par $P(x)$ aux points $a_k = \omega_N^{-k} = \exp(\frac{-2ik\pi}{N})$ pour $k = 0..N-1$, c'est à dire les valeurs $\sum_{j=0}^{N-1} c_j (\omega_N^{-k})^j$ pour $k = 0..N-1$.
     On a :
     $[P(a_0), P(a_1), ..P(a_{N-1})] = F_N([c_0, c_1, ..c_{N-1}])$
     Par exemple on tape :
     `P(x):=x+x^2` et `w:=i`
     Les coefficients de P sont [0,1,1,0], $\omega = w = i$, $N = 4$ et $\exp(2i\pi/4) = i$.
     On a :
     `fft([0,1,1,0])=[2,-1-i,0,-1+i]` et donc
     `P(0)=2,`
     `P(-i)=P(w^-1)=-1-i,`
     `P(-1)=P(w^-2)=0,`
     `P(i)=P(w^-3)=-1+i.`
   – On veut calculer les valeurs prises par $P(x)$ aux points $b_k = \omega_N^{k} = \exp(\frac{2ik\pi}{N})$ pour $k = 0..(N-1)$, c'est à dire les valeurs $\sum_{j=0}^{N-1} c_j (\omega_N^{k})^j$ pour $k = 0..(N-1)$.
     On a :
     $[P(a_0), P(a_1), ..P(a_{(N-1)})] = N * F_N^{-1}([c_0, c_1, ..c_{(N-1)}])$
     Par exemple on tape :
     `P(x):=x+x^2` et `w:=i`
     Les coefficients de P sont [0,1,1,0], $\omega = w = i$, $N = 4$ et $\exp(2i\pi/4) = i$.
     On a :
     `4*ifft([0,1,1,0])=[2,-1+i,0,-1-i]` et donc
     `P(0)=2,`
     `P(i)=P(w^1)=-1+i,`

```
P(-1)=P(w^2)=0,
P(-i)=P(w^3)=-1-i.
```

2. Interpolation trigonométrique
Soit f une fonction 2π-périodique dont on connait les valeurs en $x_k = 2k\pi/N$, $f(x_k) = f(2k\pi/N) = f_k$ pour $k = 0..(N-1)$ avec $N = 2*m$.
On veut déterminer le polynôme p trigonométrique interpolateur de f aux points $2k\pi/N$ pour $k = 0..(N-1)$, sous la forme :
$p(x) = a_0/2 + \sum_{n=0}^{\frac{N}{2}-1} a_n\cos(nx) + b_n\sin(nx) + 1/2a_{\frac{N}{2}}cos(\frac{Nx}{2})$
on suppose donc $b_0 = b_{\frac{N}{2}} = 0$ et on choisit cette notation pour simplifier les calculs ultérieurs.
On a aussi :
$p(x) = 1/2p_{\frac{-N}{2}}\exp(-iNx/2)+\sum_{n=\frac{-N}{2}+1}^{\frac{N}{2}-1} p_n\exp(inx)+1/2p_{\frac{N}{2}}\exp(iNx/2).$
avec pour $k = 0..\frac{N}{2}$
$p_k = 1/2(a_k - ib_k)$
$p_{-k} = 1/2(a_k + ib_k)$
comme $b_{\frac{N}{2}} = 0$ on a $p_{\frac{-N}{2}} = p_{\frac{N}{2}}$
On veut déterminer les p_k pour avoir :
$p(x_k) = f_k =$
$1/2p_{\frac{-N}{2}}\exp(-ik\pi) + \sum_{n=\frac{-N}{2}+1}^{\frac{N}{2}-1} p_n\exp(in2k\pi/N) + 1/2p_{\frac{N}{2}}\exp(ik\pi)$
puisque $\exp(-ik\pi) = \exp(ik\pi) = -1$ et $p_{\frac{-N}{2}} = p_{\frac{N}{2}}$ on a
$p(x_k) = f_k = -p_{\frac{-N}{2}} + \sum_{n=\frac{-N}{2}+1}^{\frac{N}{2}-1} p_n\exp(in2k\pi/N)$
On transforme $\sum_{n=\frac{-N}{2}+1}^{-1}$ en posant $j = n + N$ on obtient :
$\sum_{n=\frac{-N}{2}}^{-1} p_n\exp(inx) = \sum_{j=\frac{N}{2}}^{N-1} p_{j-N}\exp(i(j-N)x)$
puisque pour $k = 0..(N-1)$, $\exp(i(j-N)x_k) = \exp(ijx_k)$,
on a :
$p(x_k) = \sum_{j=\frac{N}{2}}^{N-1} p_{j-N}\exp(ijx_k) + \sum_{n=0}^{\frac{N}{2}-1} p_n\exp(inx_k).$
ou encore :
$p(x_k) = \sum_{n=0}^{\frac{N}{2}-1} p_n\exp(inx_k) + \sum_{n=\frac{N}{2}}^{N-1} p_{n-N}\exp(inx_k).$
On pose :
pour $n = 0..(\frac{N}{2}-1)$, $q_n = p_n$ et
pour $n = \frac{N}{2}..(N-1)$, $q_n = p_{n-N}$.
Donc pour $k = 0..(N-1)$, $p(x_k) = \sum_{n=0}^{N-1} q_n\exp(inx_k) = f_k$.
Il suffit donc de résoudre ce système d'inconnues q_n.
On a :
$f_k = f(2k\pi/N) = p(x_k) = N*F_N^{-1}(q) = \mathtt{N}*\mathtt{ifft(q)}$
soit :
$q = [p_0, ..p_{\frac{N}{2}-1}, p_{\frac{-N}{2}}, .., p_{-1}] = \frac{1}{N}F_N([f_0, ..f_{(N-1)}]) =$
$\frac{\mathtt{1}}{\mathtt{N}}\mathtt{fft}([\mathtt{f_0}, ...\mathtt{f_{(N-1)}}])$
`Remarque`
Si la fonction f est réelle on a :
$p_{-k} = \overline{p}_k$ pour $k = 1..(\frac{N}{2}-1)$ et p_0 et $p_{\frac{-N}{2}}$ sont réels.
Donc si la fonction f est réelle on a :
$p(x) = p_0 + 2*re(\sum_{k=0}^{\frac{N}{2}-1} p_k\exp(ikx) + 1/2p_{\frac{-N}{2}}*\exp(-i\frac{Nx}{2}))$

En résumé :
pour $0 \leq n < \frac{N}{2}$, p_n est le n-ième élément de $\frac{1}{N}F_N([f_0, ..f_{(N-1)}])$ et,
pour $-\frac{N}{2} \leq n < 0$, p_n est le $(n+N)$-ième élément de $\frac{1}{N}F_N([f_0, ..f_{(N-1)}])$.

3. Série de Fourier
Soit f une fonction 2π-périodique dont on connait les valeurs en $x_k = 2k\pi/N$, $f(x_k) = f(2k\pi/N) = y_k$ pour $k = 0..(N-1)$.
On suppose que sa série de Fourier converge simplement vers f, et on veut connaître les coefficients c_n pour $-\frac{N}{2} \leq n < \frac{N}{2}$.
Il fait donc calculer de façon approchée :
$c_n = \frac{1}{2\pi}\int_0^{2\pi} f(t)\exp(-int)dt$
Remarque
Lorsque la fonction périodique f est égale à sa série de Fourier, on a :
$f(x) = a_0/2 + \sum_{k=1}^{\infty} a_k\cos(kx) + b_k\sin(kx)$ avec $c_k = \dfrac{a_k - ib_k}{2}$.
On calcule l'intégrale donnant c_n de manière approchée par la méthode des trapèzes. Ici Romberg ne sert a rien, car le développement d'Euler Mac Laurin a ses coefficients déjà nuls puisque la fonction que l'on intègre est périodique et donc toutes ses dérivées sont égales en 0 et en 2π.
Si $\tilde{c_n}$ est la valeur approchée de c_n obtenue par la méthode des trapèzes, on a pour $-\frac{N}{2} \leq n < \frac{N}{2}$:
$\tilde{c_n} = \frac{1}{2\pi}\frac{2\pi}{N}\sum_{k=0}^{N-1} y_k\exp(-2ink\pi/N)$
puisque $x_k = 2k\pi/N$ et $f(x_k) = y_k$ on a
$f(x_k)\exp(-inx_k) = y_k\exp(-2ink\pi/N)$ et
$f(0)\exp(0) = f(2\pi)\exp(-2inN\pi/N) = y_0 = y_N$
On a donc :
$[\tilde{c_0}, ..\tilde{c_{\frac{N}{2}-1}}, \tilde{c_{\frac{-N}{2}}}, ..\tilde{c_{-1}}] = \frac{1}{N}F_N([y_0, y_1...y_{(N-1)}])$ car
si $n \geq 0$, on a $\tilde{c_n} = y_n$ et
si $n < 0$, on a $\tilde{c_n} = y_{n+N}$ si $\omega_N = \exp(\frac{2i\pi}{N})$, $\omega_N^n = \omega_N^{n+N}$ puisque $\omega_N^N = 1$
Propriétés
On retrouve les coefficients du polynôme interpolateur de f :
$p_n = \tilde{c_n}$ pour $-\frac{N}{2} \leq n < \frac{N}{2}$.
Ce qui veut dire que le polynôme trigonométrique $\sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \tilde{c_n}\exp(inx)$ interpole $f(x)$ aux points $x = 2k\pi/N$.
Donc si f est un polynôme trigonométrique P de degré $m \leq \frac{N}{2}$:
$f(t) = P(t) = \sum_{k=-m}^{m-1} c_k\exp(2ik\pi t)$, le polynôme trigonométrique qui interpole $f = P$ est P lui-même, les coefficients approchés sont exacts ($\tilde{c_n} = c_n$).
On peut, plus généralement calculer l'erreur $\tilde{c_n} - c_n$.
On suppose que f est égale à sa série de Fourier, c'est à dire que l'on a :
$f(t) = \int_{m=-\infty}^{+\infty} c_m\exp(2i\pi mt)$ avec $\sum_{m=-\infty}^{+\infty} |c_m| < \infty$
On a donc :
$f(x_k) = f(2k\pi/N) = y_k = \sum_{m=-\infty}^{+\infty} c_m\omega_N^{km}$ et
$\tilde{c_n} = \frac{1}{N}\sum_{k=0}^{N-1} y_k\omega_N^{-kn}$
En remplaçant y_k on obtient :
$\tilde{c_n} = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=-\infty}^{+\infty} c_m\omega_N^{km}\omega_N^{-kn}$
Or :
$\omega_N^{km}\omega_N^{-kn} = \omega_N^{k(m-n)}$ et

puisque ω_N^{m-n} est une racine N-ième de l'unité, on a :
$\omega_N^{(m-n)N} = 1$ et $\sum_{k=0}^{N-1} \omega_N^{(m-n)k} = 0$.
Donc :
si $m-n$ est un multiple de N ($m = n + l \cdot N$) on a $\sum_{k=0}^{N-1} \omega_N^{k(m-n)} = N$ et sinon $\sum_{k=0}^{N-1} \omega_N^{k(m-n)} = 0$
En intervertisant les deux sommes on a :
$c_n^{\simeq} = \frac{1}{N} \sum_{m=-\infty}^{+\infty} c_m \sum_{k=0}^{N-1} \omega_N^{k(m-n)}$
$c_n^{\simeq} = \sum_{l=-\infty}^{+\infty} c_{(n+l \cdot N)}$
c'est à dire :
$c_n^{\simeq} = ...c_{n-2 \cdot N} + c_{n-N} + c_n c_{n+N} c_{n+2 \cdot N} + ...$
Exemple :
`f(t):=cos(t)+cos(2*t)`
`x:=f(2*k*pi/8)$(k=0..7)`
On obtient :
$\mathtt{x} = \{2, (\sqrt{2})/2, -1, -((\sqrt{2})/2), 0, -((\sqrt{2})/2), -1, (\sqrt{2})/2\}$
`fft(x)=[0.0,4.0,4.0,0.0,0.0,0.0,4.0,4.0]`
donc en divisant par $N = 8$:
$c_0 = 0, c_1 = 4.0/8, c_2 = 4.0/2, c_3 = 0.0,$
$c_{-4} = 0, c_{-3} = 0, c_{-2} = 4.0/8, = c_{-1} = 4.0/8$
donc retrouve bien :
$b_k = 0$ et $a_k = c_{-k} + c_k$ vaut 1 si $k = 1, 2$ et 0 sinon.

4. Produit de convolution
   Soient deux polynômes $P(x) = \sum_{j=0}^{n-1} a_j x^j$ et $Q(x) = \sum_{j=0}^{m-1} b_j x^j$ donné par la liste de leurs coefficients $a = [a_0, a_1, ..a_{n-1}]$ et $b = [b_0, b_1, ..b_{m-1}]$.
   On veut calculer le produit de ces deux polynômes c'est à dire le produit de convolution de leurs coefficients si on se place dans l'ensemble des suites périodiques de période supérieure ou égale à $(n+m)$ en complétant a (resp b) par $m+?$ (resp $n+?$) zéros (en pratique on choisit ? pour que $n+m+?$ soit une puissance de 2).
   On alors pour $a = [a_0, a_1, ..a_{n-1}, 0..0]$ et $b = [b_0, b_1, ..b_{m-1}, 0..0]$:
   $P(x)Q(x) = \sum_{j=0}^{n+m-1} (a * b)_j x^j$
   On calcule donc :
   $F_n(a)$, $F_n(b)$, $F_n^{-1}(F_n(a) \cdot F_n(b))$ et puisque
   $nF_n(x \cdot y) = F_n(x) * F_n(y)$
   $F_n(x * y) = F_n(x) \cdot F_n(y)$
   on a :
   $a * b = F_n^{-1}(F_n(a) \cdot F_n(b))$

6.24.3 La transformée de Fourier rapide : `fft`

`fft` a comme argument une liste (ou une séquence) $[\mathtt{a_0}, ..\mathtt{a_{N-1}}]$ où `N` est une puissance de deux.
`fft` renvoie la liste $[\mathtt{b_0}, ..\mathtt{b_{N-1}}]$ tel que pour `k=0..N-1` on ait :

$$\mathtt{fft}([\mathtt{a_0}, ..\mathtt{a_{N-1}}])[\mathtt{k}] = \mathtt{b_k} = \sum_{\mathtt{j=0}}^{\mathtt{N-1}} \mathtt{x_j} \omega_{\mathtt{N}}^{-\mathtt{k \cdot j}} \text{ avec } \omega_N \text{ racine } N\text{-ième de l'unité.}$$

On tape :

```
fft(0,1,1,0)
```

On obtient :

```
[2.0, -1-i, 0.0, -1+i]
```

On peut aussi travailler sur un corps fini $\mathbb{Z}/p\mathbb{Z}$, en indiquant une racine N-ième primitive de l'unité en 2ième argument et p en 3ième argument de `fft`. Par exemple on verifie que 22798 est une racine primitive d'ordre 128 de 1 modulo 35969. Par exemple, calculons par FFT le carré d'un polynôme à coefficients entiers aléatoire inférieur à 10, de degré 60, représenté par la liste de ses coefficients par ordre croissant complété par des 0 pour avoir une liste de taille $N = 128$:

```
P:=poly1[op(ranm(1,60,10)[0]),0$68];
p:=fft(P,22798,35969)
```

il suffit maintenant de calculer le produit terme à terme de `p` avec lui-même et d'en calculer la FFT inverse

```
Q:=ifft(p .* p,22798,35969)
```

6.24.4 L'inverse de la transformée de Fourier rapide : `ifft`

`ifft` a comme argument une liste $[\mathtt{b}_0,..\mathtt{b}_{\mathtt{N}-1}]$ (ou une séquence) où `N` est une puissance de deux.
`ifft` renvoie la liste $[\mathtt{a}_0,..\mathtt{a}_{\mathtt{N}-1}]$ tel que :
$\mathtt{fft}([\mathtt{a}_0,..\mathtt{a}_{\mathtt{N}-1}]) = [\mathtt{b}_0,..\mathtt{b}_{\mathtt{N}-1}]$.
On tape :

```
ifft([2,-1-i,0,-1+i])
```

On obtient :

```
[0.0, 1.0, 1.0, 0.0]
```

Comme pour la `fft`, on peut travailler sur un corps fini en indiquant une racine N-ième primitive de l'unité en 2ième argument et p en 3ième argument de `ifft`.

6.24.5 Un exercice utilisant `fft`

Voici un relevé des températures T, en degré Celsius, au temps t :

t	0	3	6	9	12	15	19	21
T	11	10	17	24	32	26	23	19

Quelle est la température à 13h45 ?
On a $N = 8 = 2 * m$.
Le polynôme d'interpolation est :
$p(t) = 1/2p_{-m}(\exp(-2i\pi mt/24)+\exp(2i\pi mt/24))+\sum_{k=-m+1}^{m-1} p_k \exp(2i\pi kt/24)$
et on a $p_k = 1/N * \sum_{k=j}^{N-1} T_k \exp(2i\pi k/N)$
On tape :
`q:=1/8*fft([11,10,17,24,32,26,23,19])`
On obtient :
```
[20,-4.5+1.7*i,0.37+0.88*i,-0.77+0.22*i,0.5,
-0.77-0.22*i,0.38-0.88*i,-4.5-1.7*i]
```
ou avec plus de décimales :

$p_0 = 20.25$,
$p_1 = -4.48115530061 + 1.72227182413 * i = \overline{p_{-1}}$,
$p_2 = 0.375 + 0.875 * i = \overline{p_{-2}}$,
$p_3 = -0.768844699385 + 0.222271824132 * i = \overline{p_{-3}}$,
$p_{-4} = 0.5$ car on a :
$q = [q_0, ...q_{N-1}] = [p_0, ..p_{\frac{N}{2}-1}, p_{-\frac{N}{2}}, .., p_{-1}] = \frac{1}{N}F_N([y_0, ..y_{N-1}]) = \frac{1}{N}\texttt{fft}(y)$
On calcule la valeur $T0$ du polynôme d'interpolation au point $t0 = 13,75 = 13 + 3/4 = 55/4$.
On a :

```
q:=[20.25,-4.48115530061+1.72227182413*i,-0.375+0.875*i,
-0.768844699385+0.222271824132*i,0.5,
-0.768844699385-0.222271824132*i,
-0.375-0.875*i,-4.48115530061-1.72227182413*i]
```

On pose :

```
pp:=[q[4],q[5],q[6],q[7],q[0],q[1],q[2],q[3]]
```

On a $p_k = pp[k + 4]$ pour $k = -4...3$
On tape :

```
t0(j):=exp(2*i*pi*(13+3/4)/24*j)
T0:=1/2*pp[0]*(t0(4)+t0(-4))+sum(pp[j+4]*t0(j),j,-3,3)
evalf(sincos(T0))
```

On obtient :

```
29.4863181684
```

On prévoit donc une température de 29.49 degrés Celsius.
On tape :

```
q1:=[q[4]/2,q[3],q[2],q[1],q[0]/2]
```

`a:=t0(1)` (ou `a:=-exp(i*pi*7/48)`)

```
g(x):=r2e(q1,x)
evalf(2*re(g(a)))
```

ou encore

```
2.0*re(q[0]/2+q[1]*t0(1)+q[2]*t0(2)+q[3]*t0(3)+q[4]/2*t0(4))
```

On obtient :

```
29.4863181684
```

Remarque
Si on utilise le polynôme d'interpolation de Lagrange (on interpole par un polynôme non périodique).
On tape :

```
l1:=[0,3,6,9,12,15,18,21]
l2:=[11,10,17,24,32,26,23,19]
subst(lagrange(l1,l2,13+3/4),x=13+3/4)
```

On obtient :

$$\frac{8632428959}{286654464} \simeq 30.1144061688$$

6.25 Les Exponentielles et les Logarithmes

6.25.1 Transformer les fonctions hyperboliques en exponentielles : `hyp2exp`

`hyp2exp` a comme argument an hyperbolic expression.
`hyp2exp` transforme les hyperbolic fonctions hyperboliques en exponentielles

SANS linéariser.
On tape :

`hyp2exp(sinh(x))`

On obtient :

`(exp(x)-1/(exp(x)))/2`

6.25.2 Développer les exponentielles : `expexpand`

`expexpand` a comme argument une expression contenant des exponentielles.
`expexpand` développe cette expression.
On tape :

`expexpand(exp(3*x))`

On obtient :

`exp(x)^3`

On tape :

`expexpand(exp(3*x)+exp(2*x+2))`

On obtient :

`exp(x)^3+exp(x)^2*exp(2)`

6.25.3 Développer les logarithmes : `lnexpand`

`lnexpand` a comme argument une expression contenant des logarithmes.
`lnexpand` développe cette expression.
On tape :

`lnexpand(ln(3*x^2)+ln(2*x+2))`

On obtient :

`ln(3)+2*ln(x)+ln(2)+ln(x+1)`

6.25.4 Linéariser les exponentielles : `lin lineariser`

`lin` ou `lineariser` a comme argument une expression contenant des exponentielles.
`lin` ou `lineariser` linéarise cette expression (l'exprime en fonction de $\exp(n.x)$).
Exemples

- On tape :

  `lin(sinh(x)^2)`

  On obtient :

  `1/4*exp(2*x)+1/-2+1/4*exp(-(2*x))`
- On tape :

  `lin((exp(x)+1)^3)`

  On obtient :

  `exp(3*x)+3*exp(2*x)+3*exp(x)+1`

6.25.5 Regrouper les log : `lncollect`

`lncollect` a comme argument une expression contenant des logarithmes.
`lncollect` regroupe les termes en logarithmes. Il est donc préférable de l'utiliser sur une expression factorisée (en utilisant `factor`).
On tape :

```
lncollect(ln(x+1)+ln(x-1))
```

On obtient :

```
ln((x+1)*(x-1))
```

On tape :

```
lncollect(exp(ln(x+1)+ln(x-1)))
```

On obtient :

```
(x+1)*(x-1)
```

Attention !!! Pour `CAS`, `log=ln`

6.25.6 Transformer une puissance en produit de puissances : `powexpand`

`powexpand` permet de transformer une puissance en un produit de puissances.
On tape :

```
powexpand(a^(x+y))
```

On obtient :

```
a^x*a^y
```

6.25.7 Transformer une puissance en une exponentielle : `pow2exp`

`pow2exp` permet de transformer une puissance en exponentielle.
On tape :

```
pow2exp(a^(x+y))
```

On obtient :

```
exp((x+y)*ln(a))
```

6.25.8 Transformer exp(n*ln(x)) en puissance : `exp2pow`

`exp2pow` permet de transformer une expression de la forme $\exp(n*\ln(x))$ en une puissance de x.
On tape :

```
exp2pow(exp(n*ln(x)))
```

On obtient :

```
x^n
```

Bien voir la différence avec `lncollect` :
`lncollect(exp(n*ln(x))) = exp(n*ln(x))`
`lncollect(exp(2*ln(x))) = exp(2*ln(x))`
`exp2pow(exp(2*ln(x)))= x^2`
Mais :
`lncollect(exp(ln(x)+ln(x))) = x^2`
`exp2pow(exp(ln(x)+ln(x))) = x^(1+1)`

6.25.9 Écrire avec des exponentielles complexes : `tsimplify`

`tsimplify` simplifie toutes les expressions en les transformant en exponentielles complexes.
On n'utilise `tsimplify` qu'en dernier ressort.
On tape :

```
tsimplify((sin(7*x)+sin(3*x))/sin(5*x))
```

On obtient :

```
((exp((i)*x))^4+1)/(exp((i)*x))^2
```

6.26 Les polynômes

6.26.1 Les polynômes à une variable `poly1[...]`

Un polynôme d'une variable peut être représenté, soit par la liste de ses coefficients selon les puissances décroissantes (on utilise comme délimiteurs `poly1[...]` en particulier lorsqu'il peut y avoir confusion avec un vecteur), soit par son expression symbolique. Dans les réponses affichées par un éditeur d'équations, le résultat est parenthésé avec `[]` et `[]` pour indiquer qu'il s'agit de la liste de coefficients d'un polynôme, lorsqu'on effectue une copie, c'est `poly1[...]` qui s'affiche.

Attention L'écriture est toujours selon les puissances décroissantes même si on a coché `puissance croissante` dans la configuration du `cas` (`puissance croissante` ne s'applique qu'aux expressions symboliques).

6.26.2 Les polynômes à plusieurs variables

Un monôme de plusieurs variables peut être représenté :

- soit par son expression symbolique (par exemple $3x^2y$),
- soit par son coefficent et la liste des puissances de ses variables (selon la liste de ses variables) que l'on parenthése avec `%%%{` et `%%%}` : c'est le format interne creux distribué : par exemple `symb2poly(3x^2*y,[x,y])` renvoie `%%%{3,[2,1]%%%}` qui représente $3x^2y$ si la liste des variables est `[x,y]`,
- soit par la liste des coefficients de la première variable, coefficients qui sont eux-mêmes des polynômes qui seront donnés sous la forme la liste des coefficients de la deuxième variable etc... et que l'on parenthése avec `[],[]` : c'est le format interne dense récursif : par exemple `symb2poly(3x^2*y,x,y)` renvoie `[][]`3,0`[]`,0,0`[]` qui représente $3x^2y$ si les variables sont `x,y`.

Un polynôme de plusieurs variables est représenté par la somme de ses monômes.

6.26.3 Transformer le format interne dense récursif en une écriture polynômiale : `poly2symb r2e`

`r2e` ou `poly2symb` a comme argument la liste des coefficients par puissances décroissantes d'un polynôme et un nom de variable formelle (par défaut `x`) (resp la liste récursive (c'est le format interne dense récursif) des coefficients par puissances décroissantes d'un polynôme et la séquence des variables formelles tel que `x,y,z` (par exemple `[]`[[1,0],[2,3]],`[]` `[]`4,0`[]`,5`[]` `[]` représente le polynôme $x(yz+2z+3)+4yz+5$) car `[],[]` sert de parenthèsages)
On tape :

```
r2e([1,0,-1],x)
```

Ou on tape :

```
r2e([1,0,-1])
```

Ou on tape :

```
poly2symb([1,0,-1],x)
```

On obtient :

```
x*x-1
```

Ou on peut aussi taper :

```
r2e(%%%{1,[2]%%%}+%%%{-1,[0]%%%},[x])
```

On obtient :

```
x^2-1
```

On tape :

```
r2e([1,0,-1],x)
```

Ou on tape :

```
r2e([1,0,-1])
```

Ou on tape :

```
poly2symb([1,0,-1],x)
```

On obtient :

```
x*x-1
```

On tape :

```
normal(poly2symb([[[1,0],[2,3]],[[4,0],5]],x,y,z))
```

Ou on tape :

```
normal(r2e([[[1,0],[2,3]],[[4,0],5]],x,y,z))
```

On obtient :

```
x*y*z+2*x*z+3*x+4*y*z+5
```

Remarque
Si en deuxième argument on met une valeur a (resp a, b, c), on obtient la valeur du polynôme en $x = a$ (resp en $x = a$, $y = b$ et $z = c$).
On tape :

```
poly2symb([[[1,0],[2,3]],[[4,0],5]],1,1,1)
```

On obtient car $1 + 2 + 3 + 4 + 5 = 15$:

```
15
```

6.26.4 Transformer le format interne creux distribué du polynôme en une écriture polynômiale : `poly2symb r2e`

`r2e` ou `poly2symb` a comme argument la liste des coefficients par puissances décroissantes d'un polynôme et un nom de variable formelle (par défaut `x`) (resp le format interne creux distribué du polynôme c'est à dire la somme de monômes tels que : `%%%{c,[px,py,pz] %%%}` et une liste de variables formelles tel que `[x,y,z]` ce qui représente le monôme $cx^{px}y^{py}z^{pz}$).
`r2e` ou `poly2symb` transforme la liste des coefficients par puissances décroissantes d'un polynôme (resp la somme de `%%%{c,[px,py,pz] %%%}`), en son écriture polynômiale (selon Horner), en utilisant le nom de la variable donné en deuxième argument (resp en utilisant la liste de variables donné en deuxième argument `[x,y,z]`).
On tape :

```
r2e([1,0,-1],x)
```

Ou on tape :

```
r2e([1,0,-1])
```

Ou on tape :

```
poly2symb([1,0,-1],x)
```

On obtient :

```
x*x-1
```

Ou on peut aussi taper :

```
r2e(%%%{1,[2]%%%}+%%%{-1,[0]%%%},[x])
```

On obtient :

```
x^2-1
```

On tape :

```
r2e(%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+%%%{2,[0,1]%%%},[x,y])
```

Ou on tape :

```
poly2symb(%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+
          %%%{2,[0,1]%%%},[x,y])
```

On obtient :

```
x^2-x*y+2*y
```

Remarque
Si en deuxième argument on met une valeur a (resp $[a, b]$), on obtient la valeur du polynôme en $x = a$ (resp en $x = a$ et $y = b$).
On tape :

```
poly2symb([1,0,-1],3)
```

On obtient car $3^2 - 1 = 8$:

```
8
```

On tape :

```
poly2symb(%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+
          %%%{2,[0,1]%%%},[1,2])
```

On obtient :

```
1-1*2+2*2
```

6.26.5 Transformer un polynôme en une liste (format interne récursif dense) : `symb2poly e2r`

`e2r` ou `symb2poly` a comme argument polynôme, donné avec une écriture polynômiale, d'une variable (resp plusieurs variables), et le nom de cette variable formelle (par défaut `x`) (resp la séquence des noms de ces variables).
`e2r` ou `symb2poly` transforme cette écriture polynômiale, en la liste des coefficients selon les puissances décroissantes selon le nom de la variable donné en deuxième argument (resp l'écriture récursive de la liste des coefficients selon les puissances décroissantes selon les noms des variables donnés en deuxième argument : le résultat est la liste des coefficients de la première variable, coefficients qui sont eux-mêmes des polynômes qui seront donnés sous la forme la liste des coefficients de la deuxième variable etc...).
Attention Si le deuxième argument est une liste, le résultat est l'écriture du polynôme au format interne.
On tape :

```
e2r(x^2-1)
```

Ou on tape :

```
symb2poly(x^2-1)
```

Ou on tape :

```
symb2poly(x^2-1,x)
```

Ou on tape :

```
e2r(x^2-1,x)
```

Ou on tape :

```
symb2poly(y^2-1,y)
```

Ou on tape :

```
e2r(y^2-1,y)
```

On obtient :

```
[]1,0,-1[]
```

On tape :

```
symb2poly(x*y^2+2y-1,x)
```

Ou on tape :

```
e2r(x*y^2+2y-1,x)
```

On obtient :

```
[]y^2,2y-1[]
```

On tape :

```
symb2poly(x*y^2+2y-1,y)
```

Ou on tape :

```
e2r(x*y^2+2y-1,y)
```

On obtient :

```
[]x,2,-1[]
```

On tape :

```
symb2poly(x*y^2+2y-1,x,y)
```

Ou on tape :

```
e2r(x*y^2+2y-1,x,y)
```

On obtient :

```
[][]1,0,0[],[]2,-1[][]
```

ce qui signifie que le polynôme est de degré 1 en x que le coeff de x est le polynôme en y de coefficients [1,0,0] c'est à dire y^2 et que le terme constant est le polynôme en y de coefficients [2,-1] c'est à dire $2y - 1$.
On tape :

`symb2poly(x^2*y^2-x^2+2x*y+4y^2-y+3,x,y)`

Ou on tape :

`e2r(x^2*y^2-x^2+2x*y+4y^2-y+3,x,y)`

On obtient :

`[]1,0,-1[],[]2,0[],[]4,-1,3[][]`

ce qui signifie que le polynôme est de degré 2 en x que le coeff de x^2 est le polynôme en y de coefficients `[]`1,0,-1`[]` c'est à dire $y^2 - 1$, le coeff de x est le polynôme en y de coefficients `[]`2,0`[]` c'est à dire $2y$ et que le terme constant est le polynôme en y de coefficients `[]`4,-1,3`[]` c'est à dire $4y^2 - y + 3$.

6.26.6 Transformer un polynôme au format interne : `e2r symb2poly`

`e2r` ou `symb2poly` a comme argument polynôme d'une ou plusieurs variables, donné avec une écriture polynômiale et la liste de ces variables formelles.
`e2r` ou `symb2poly` transforme cette écriture polynômiale, en l'écriture au format interne selon la liste de variables donnée en deuxième argument.
Attention Si le deuxième argument n'est pas une liste, le résultat est la liste des coefficients de la première variable, coefficients qui sont eux-mêmes des polynômes qui seront donnés sous la forme la liste des coefficients de la deuxième variable etc...
On tape :

`e2r(x^2-1,[x])`

Ou on tape :

`symb2poly(x^2-1,[x])`

On obtient :

`%%%{1,[2]%%%}+%%%{-1,[0]%%%}`

On tape :

`e2r(x^2-x*y+y, [x,y])`

Ou on tape :

`symb2poly(x^2-x*y+2*y, [x,y])`

On obtient :

`%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+%%%{2,[0,1]%%%}`

ce qui signifie que le coefficient du terme x^2y^0 est 1, celui du terme x^1y^1 est -1 et celui du terme x^0y^1 est 2.
On tape :

`symb2poly(x^2*y^2-x^2+2x*y+4y^2-y+3,[x,y])`

Ou on tape :

`e2r(x^2*y^2-x^2+2x*y+4y^2-y+3,x,y)`

On obtient :

```
%%%{1,[2,2]%%%}+%%%{-1,[2,0]%%%}+%%%{2,[1,1]%%%}+
%%%{4,[0,2]%%%}+%%%{-1,[0,1]%%%}+%%%{3,[0,0]%%%}
```

6.26.7 Transformer un polynôme au format interne en une liste et réciproquement : `convert`

Si `convert` a comme premier argument un polynôme donné a u format interne et comme deuxième argument l'option `list` alors `convert` transforme le polynôme en une liste (l'option `list` peut être omis).
Si `convert` a comme premier argument une liste et comme deuxième argument l'option `polynom` alors `convert` transforme la liste en un polynôme au format interne.
On tape :

```
p:=symb2poly(x^2-x*y+2y, [x,y])
```

On obtient :

```
%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+%%%{2,[0,1]%%%}
```

On tape :

```
l:=convert(p,list)
```

Ou on tape :

```
convert(p)
```

On obtient :

```
[[1,[2,0]],[-1,[1,1]],[2,[0,1]]]
```

ce qui est la liste des coefficients suivi de la liste des exposants.
On tape :

```
convert(l,polynom)
```

On obtient :

```
%%%{1,[2,0]%%%}+%%%{-1,[1,1]%%%}+%%%{2,[0,1]%%%}
```

6.26.8 Coefficients d'un polynôme : `coeff coeffs`

`coeff` ou `coeffs` a trois arguments : le polynôme, le nom de la variable (ou la liste des noms des variables) le degré (ou la liste des degrés des variables).
`coeff` ou `coeffs` renvoie le coefficient du polynôme de degré spécifié.
On tape :

```
coeff(-x^4+3*x*y^2+x,x,1)
```

On obtient :

```
3*y^2+1
```

On tape :

```
coeff(-x^4+3x*y^2+x,y,2)
```

On obtient :

```
3*x
```

On tape :

```
coeff(-x^4+3x*y^2+x,[x,y],[1,2])
```

On obtient :

```
3
```

6.26.9 Degré d'un polynôme : `degree`

`degree` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`degree` renvoie le degré de ce polynôme (degré du monôme de plus grand degré).
On tape :

```
degree(x^3+x)
```

On obtient :

```
3
```

On tape :

```
degree([1,0,1,0])
```

On obtient :

```
3
```

6.26.10 Valuation d'un polynôme : `valuation ldegree`

`valuation` ou `ldegre` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`valuation` ou `ldegre` renvoie la valuation de ce polynôme, c'est le degré du monôme de plus petit degré (`ldegree`=low degree).
On tape :

```
valuation(x^3+x)
```

On obtient :

```
1
```

On tape :

```
valuation([1,0,1,0])
```

On obtient :

```
1
```

6.26.11 Coefficient du terme de plus haut degré d'un polynôme : `lcoeff`

`lcoeff` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`lcoeff` renvoie le coefficient de plus haut degré de ce polynôme (`lcoeff`=leading coefficient).
On tape :

```
lcoeff([2,1,-1,0])
```

On obtient :

```
2
```

On tape :

```
lcoeff(3*x^2+5*x,x)
```

On obtient :

```
3
```

On tape :

```
lcoeff(3*x^2+5*x*y^2,y)
```

On obtient :

```
5*x
```

6.26.12 Coefficient du terme de plus bas degré d'un polynôme : `tcoeff`

`tcoeff` a comme argument un polynôme donné sous forme symbolique ou par la liste de ses coefficients.
`tcoeff` renvoie le coefficient de plus bas degré de ce polynôme (`tcoeff`=trailing coefficient).
On tape :

```
tcoeff([2,1,-1,0])
```

On obtient :

```
-1
```

On tape :

```
tcoeff(3*x^2+5*x,x)
```

On obtient :

```
5
```

On tape :

```
tcoeff(3*x^2+5*x*y^2,y)
```

On obtient :

```
3*x^2
```

6.26.13 Évaluation d'un polynôme : `peval polyEval`

`peval` ou `polyEval` a comme argument un polynôme `p` donné par la liste de ses coefficients et un réel `a`.
`peval` ou `polyEval` renvoie la valeur numérique ou exacte de `p(a)`.
On tape :

```
peval([1,0,-1],sqrt(2))
```

On obtient :

```
sqrt(2)*sqrt(2)-1
```

Puis :

```
normal(sqrt(2)*sqrt(2)-1)
```

On obtient :

```
1
```

On tape :

```
peval([1,0,-1],1.4)
```

On obtient :

```
0.96
```

6.26.14 Mise en facteur de x^n dans un polynôme : `factor_xn`

`factor_xn` a comme argument un polynôme `P`.
`factor_xn` renvoie le polynôme `P` dans lequel on a mis en facteur x^n où n est le degré de `P` (`n=degree(P)`).
On tape :

```
factor_xn(-x^4+3)
```

On obtient :

```
x^4*(-1+3*x^-4)
```

6.26.15 PGCD des coefficients d'un polynôme : `content`

`content` a comme arguments un polynôme `P` donné sous forme symbolique ou par la liste de ses coefficients et le nom de la variable (par défaut c'est x).
`content` désigne le PGCD (plus grand commun diviseur) des coefficients du polynôme `P`.
On tape :

```
content(6*x^2-3*x+9)
```

ou on tape :

```
content(6*t^2-3*t+9,t)
```

On obtient :

```
3
```

ou on tape :

```
content([6,-3,9]))
```

On obtient :

```
poly1[3]
```

On tape :

```
content(6*x^2*(y^3-y)-3*x*(y^3-1)+9*(y-1))
```

On obtient :

```
3*y-3
```

ou on tape :

```
content([6*(y^3-y),-3*(y^3-1),9*(y-1)])
```

On obtient :

```
poly1[3*y-3]
```

On tape :

```
content(6*x^2*(y^3-y)-3*x*(y^3-1)+9*(y-1),y)
```

ou :

```
content([3x,0,6*x^2+9,-9-3x])
```

On obtient :

```
3
```

6.26.16 Partie primitive d'un polynôme : `primpart`

`primpart` a comme argument un polynôme `P` donné sous forme symbolique ou par la liste de ses coefficients.
`primpart` renvoie le polynôme `P` divisé par le PGCD (plus grand commun diviseur) de ses coefficients.
On tape :

```
primpart(6x^2-3x+9)
```

ou :

```
primpart([6,-3,9],x))
```

On obtient :

```
2*x^2-x+3
```

6.26.17 Factorisation sur les entiers : `collect`

`collect` a comme paramètre un polynôme ou une liste de polynômes et éventuellement `sqrt(n)`.
`collect` factorise le polynôme (ou les polynômes de la liste) sur les entiers lorsque les coefficients du polynôme sont entiers ou sur $\mathbb{Q}(\sqrt(n))$, si les coefficients du polynôme sont dans $\mathbb{Q}(\sqrt(n))$ ou si `sqrt(n)` est le second argument.
. Exemples :

– Factoriser sur les entiers :
$$x^2 - 4$$
On tape :
`collect(x^2-4)`
On trouve en mode réel :
`(x-2)*(x+2)`
– Factoriser sur les entiers :
$$x^2 + 4$$
On tape :
`collect(x^2+4)`
On obtient en mode réel :
`x^2+4`
On obtient en mode complexe :
`(x+2*i)*(x-2*i)`
– Factoriser sur les entiers :
$$x^2 - 2$$
On tape :
`collect(x^2-2)`
On obtient :
`x^2-2`
Mais si on tape :
`collect(sqrt(2)*(x^2-2))`
On obtient :
`sqrt(2)*(x-sqrt(2))*(x+sqrt(2))`
– Factorier sur les entiers :
$$x^3 - 2x^2 + 1\text{et}x^2 - x$$
On tape :
`collect([x^3-2*x^2+1,x^2-x])`
On obtient :
`[(x-1)*(x^2-x-1),x*(x-1)]`
Mais si on tape :
`collect((x^3-2*x^2+1)*sqrt(5))`
On obtient :

`sqrt(5)*(x+(-(sqrt(5))-1)/2)*(x-1)*(x+(sqrt(5)-1)/2)`
Ou si on tape :
`collect(x^3-2*x^2+1,sqrt(5))`

On obtient :

`(x+(-(sqrt(5))-1)/2)*(x-1)*(x+(sqrt(5)-1)/2)`

6.26.18 Factorisation : `factor factoriser`

`factor` a pour argument un polynôme ou une liste de polynômes et éventuellement `sqrt(n)`.
`factor` factorise le polynôme ou la liste de polynômes sur les réels en mode réel et sur les complexes en mode complexe Les coefficients des facteurs sont dans $\mathbb{Q}(\sqrt(n))$ si `Sqrt` est coché dans la configuration du `cas` ou si `sqrt(n)` est le second argument (voir aussi 6.13.11).

Note pour être en mode réel (ou complexe) décochez (ou cochez) `Complexe` dans la configuration du `cas` que l'on ouvre avec le bouton donnant la ligne d'état.
On tape :

`factor(x^2+2*x+1)`

On obtient :

`(x+1)^2`

On tape :

`factor(x^4-2*x^2+1)`

On obtient :

`(-x+1)^2*(x+1)^2`

On tape :

`factor(x^ 3-2*x^2+1)`

On obtient si `Sqrt` n'est pas coché dans la configuration du `cas` :

`(x-1)*(x^2-x-1)`

On tape :

`factor(x^ 3-2*x^2+1)`

On obtient si `Sqrt` est coché dans la configuration du `cas` :

`(x-1)*(x+(sqrt(5)+1)/2)*(x+(-sqrt(5)+1)/2)`

On tape :

`factor(x^ 3-2*x^2+1,sqrt(5))`

On obtient si `Sqrt` est coché ou non dans la configuration du `cas` :

`((2*sqrt(5)-19)*((sqrt(5)+15)*x+7*sqrt(5)-5)*(-x+1)*((sqrt(5)+25)*x-1`

On tape :

`factor(x^2+1)`

On obtient en mode réel :

`x^2+1`

On obtient en mode complexe :

`((-i)*x+1)*((i)*x+1)`

6.26.19 Factorisation sans facteur carré : `sqrfree`

`sqrfree` a comme paramètre un polynôme.
`sqrfree` factorise ce polynôme en regroupant les termes ayant même exposant.
On tape :

```
sqrfree((x^2-1)*(x-1)*(x+2))
```

On obtient :

```
(x^2+3*x+2)*(x-1)^2
```

On tape :

```
sqrfree((x^2-1)^2*(x-1)*(x+2)^2)
```

On obtient :

```
(x^2+3*x+2)*(x-1)^3
```

6.26.20 Liste des facteurs d'un polynôme : `factors`

`factors` a pour argument un polynôme ou une liste de polynômes.
`factors` donne la liste des facteurs du polynôme avec leur multiplicité.
On tape :

```
factors(x^2+2*x+1)
```

On obtient :

```
[x+1,2]
```

On tape :

```
factors(x^4-2*x^2+1)
```

On obtient :

```
[x-1,2,x+1,2]
```

On tape :

```
factors([x^3-2*x^2+1,x^2-x])
```

On obtient :

```
[[x-1,1,x^2-x-1,1],[x,1,x-1,1]]
```

On tape :

```
factors([x^2,x^2-1])
```

On obtient :

```
[[x,2],[x+1,1,x-1,1]]
```

6.26.21 Évaluer un polynôme : `horner`

`horner` a deux paramètres : un polynôme `P` donné sous forme symbolique ou par la liste de ses coefficients et un nombre `a`.
`horner` renvoie `P(a)` calculé par la méthode de Hörner.
On tape :

```
horner(x^2-2*x+1,2)
```

ou :

```
horner([1,-2,1],2)
```

On obtient :

```
1
```

On peut mettre un troisième argument optionnel pour indiquer la variable à remplacer, par exemple

```
horner(y^2-2*x*y+1,2,y)
```

6.26.22 Écriture selon les puissances de (x-a) : `ptayl`

Il s'agit d'écrire un polynôme `P(x)` selon les puissances de `x-a)` $x - a$.
`ptayl` a deux paramètres : un polynôme `P` donné sous forme symbolique ou par la liste de ses coefficients et un nombre `a`.
`ptayl` renvoie le polynôme `Q` tel que `Q(x-a)=P(x)`.
On tape :

```
ptayl(x^2+2*x+1,2)
```

On obtient le polynôme Q(x) :

```
x^2+6*x+9
```

On tape :

```
ptayl([1,2,1],2)
```

On obtient :

```
[1,6,9]
```

Attention
On a :

```
P(x)=Q(x-a)
```

c'est à dire pour l'exemple que :
$x^2 + 2x + 1 = (x - 2)^2 + 6(x - 2) + 9$

6.26.23 Calcul avec les racines exactes d'un polynôme : `rootof`

Soient P et Q deux polynômes donnés par la liste de leurs coefficients alors, `rootof(P,Q)` désigne la valeur $P(\alpha)$ où α est la "plus grande" racine de Q (on compare d'abord les parties réelles et en cas dégalité on compare les parties imaginaires).
On peut alors faire des calculs avec cette valeur.
On tape :

```
normal(rootof([1,0],[1,2,-3]))
```

On obtient :

```
1
```

en effet $x^2 + 2x - 3 = (x - 1)(x + 3)$ a comme plus grande racine 1.

Autre exemple
Soit α la plus grande racine en norme de $Q(x) = x^4 + 10x^2 + 1$.

- Calculer $\dfrac{1}{\alpha}$
  On tape :
  ```
  normal(1/rootof([1,0],[1,0,10,0,1]))
  ```
  car $P(x) = x$ est représenté par [1,0].
  On obtient :
  ```
  rootof([[-1,0,-10,0],[1,0,10,0,1]])
  ```
  ce qui veut dire que :
  $\dfrac{1}{\alpha} = -(\alpha)^3 - 10.\alpha$
- Calculer $(\alpha)^2$.
  On tape :
  ```
  normal(rootof([1,0],[1,0,10,0,1])^2)
  ```
  On a α=`rootof([1,0],[1,0,10,0,1])` car $P(x) = x$ est représenté par [1,0], et pour avoir α^2, on élève α au carré.
  On obtient :
  ```
  -5-2*sqrt(6)
  ```
  ou pour avoir α^2 directement, on tape :
  ```
  normal(rootof([1,0,0],[1,0,10,0,1])^2)
  ```
  car $P(x) = x^2$ est représenté par [1,0,0].
  On obtient :
  ```
  -5-2*sqrt(6)
  ```

Ce résultat peut se vérifier puisque l'on a une équation bicarrée de discriminant réduit $25 - 1 = 24 = 4 * 6$. On tape :

```
csolve(x^4+10x^2+1)
```

On obtient :

```
[(i)*sqrt(-2*sqrt(6)+5),(-i)*sqrt(-2*sqrt(6)+5),
 (i)*sqrt(2*sqrt(6)+5),(-i)*sqrt(2*sqrt(6)+5)]
```

Donc $\alpha = i * \sqrt{2 * \sqrt{6} + 5}$
On tape :

```
((i)*sqrt(2*sqrt(6)+5))^2
```

On obtient :

```
-5-2*sqrt(6)
```

6.26.24 Racines exactes d'un polynôme : roots

`roots` a comme arguments une fonction polynôme et le nom de sa variable.
`roots` renvoie une matrice ayant 2 colonnes : chaque ligne est composée d'une racine du polynôme et de son ordre de multiplicité.
Exemples

- Chercher les racines de $P(x) = x^5 - 2x^4 + x^3$.
  On tape :
  ```
  roots(x^5-2*x^4+x^3)
  ```
  On obtient :
  ```
  [[8+3*sqrt(7),1],[8-3*sqrt(7),1],[0,3]]
  ```
- Chercher les racines de $x^{10}-15x^8+90x^6-270x^4+405x^2-243 = (x^2-3)^5$.
  On tape :
  ```
  roots(x^10-15*x^8+90*x^6-270*x^4+405*x^2-243)
  ```
  On obtient :
  ```
  [[sqrt(3),5],[-(sqrt(3)),5]]
  ```
- Chercher les racines de $(t^3 - 3)$.
  On tape :
  ```
  roots(t^3-1,t)
  ```
  On obtient :
  ```
  [[(-1+(i)*sqrt(3))/2,1],[(-1-(i)*sqrt(3))/2,1],[1,1]]
  ```

6.26.25 Coefficients d'un polynôme défini par ses racines : pcoeff pcoef

`pcoeff` (ou `pcoef`) a comme argument, une liste de composantes les racines d'un polynôme P.
`pcoeff` (ou `pcoef`) renvoie une liste de composantes, les coefficients du polynôme unitaire P (par ordre décroissant).
On tape :

```
pcoef([1,2,0,0,3])
```

On obtient :

```
[1,-6,11,-6,0,0]
```

c'est à dire $(x - 1)(x - 2)(x^2)(x - 3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

6.26.26 Troncature d'ordre n : truncate

`truncate` permet de tronquer un polynôme à un ordre donné `truncate` est utile quand on fait des développements limités à la main, ou pour transformer un développement limité en polynôme.

`truncate` a deux arguments : un polynôme et un entier `n`.
`truncate` renvoie le polynôme tronqué à l'ordre `n` (pas de termes d'ordre supérieur ou égal à `n+1`).
On tape :

```
truncate((1+x+x^2/2)^3,4)
```

On obtient :

```
(9*x^4+16*x^3+18*x^2+12*x+4)/4
```

On tape :

```
truncate(series(sin(x)),4)
```

On obtient :

```
(-x^3-(-6)*x)/6
```

On remarquera que le polynôme renvoyé est réduit au même dénominateur.

6.26.27 Convertir un développement limité en polynôme : `convert` `convertir`

`convert`, avec l'option `polynom`, permet de convertir un développement de Taylor en un polynôme.
`convert` est utile pour transformer un développement limité en polynôme : par exemple pour faire le tracé d'une fonction et de ses polynômes de Taylor.
`convert` a deux arguments : un développement de Taylor d'ordre n et l'option `polynom`.
`convert` renvoie le polynôme de Taylor d'ordre n tronqué à l'ordre n (on supprime le reste).
On tape :

```
convert(taylor(sin(x)),polynom)
```

On obtient :

```
x+1/-6*x^3+1/120*x^5+x^6*0
```

On tape :

```
convert(series(sin(x),x=0,6),polynom)
```

On obtient :

```
x+1/-6*x^3+1/120*x^5+x^7*0
```

6.26.28 Convertir un polynôme de n variables en une liste : `convert` `convertir`

`convert`, avec l'option `list`, permet de convertir un polynôme de n variables en une liste.
On tape :

```
p:=symb2poly((x+y)^2,[x,y])
```

```
l:=convert(p,list)
```

ou

```
l:=convert(p)
```

On obtient :

```
[[1,[2,0]],[2,[1,1]],[1,[0,2]]]
```

6.26.29 Convertir une liste en polynôme de n variables : `convert` `convertir`

`convert`, avec l'option `polynom`, permet de convertir une liste en un polynôme de n variables.
On tape :

```
l:=[[1,[2,0]],[2,[1,1]],[1,[0,2]]]
```

```
p:=convert(l,polynom)
```

On obtient :

```
%%%1,[2,0]%%%+%%%2,[1,1]%%%+%%%1,[0,2]%%%
```

On tape :

```
poly2symb(p,[x,y])
```

On obtient :

```
x^2+2*x*y+y^2
```

6.26.30 Polynômes aléatoires : `randpoly` `randPoly`

`randpoly` (ou `randPoly`) a deux paramètres le nom d'une variable (par défaut `x`) et un entier `n` (en fait l'ordre des paramètres n'est pas important).
`randpoly` renvoie un polynôme de variable le premier paramètre (ou `x`), de degré le deuxième paramètre et dont les coefficients sont des entiers aléatoires équirépartis sur -99..+99.
On tape :

```
randpoly(t,4)
```

On obtient par exemple :

```
-8*t^4-87*t^3-52*t^2+94*t+80
```

On tape :

```
randpoly(4)
```

On obtient par exemple :

```
70*x^4-46*x^3-7*x^2-24*x+52
```

On tape :

```
randpoly(4,u)
```

On obtient par exemple :

```
2*u^4+33*u^3-6*u^2-92*u-12
```

6.26.31 Changer l'ordre des variables : `reorder`

`reorder` a deux paramètres : une expression et une liste contenant les noms des variables dans un certain ordre.
`reorder` développe l'expression selon l'ordre des variables donné dans le second paramètre.
On tape :

```
reorder(x^2+2*x*a+a^2+z^2-x*z,[a,x,z])
```

On obtient :

```
a^2+2*a*x+x^2-x*z+z^2
```

Attention :
Il ne faut pas que les variables soient affectées !

6.26.32 Liste aléatoire : `ranm`

`ranm` a comme argument un entier `n`.
`ranm` renvoie une liste de n entiers aléatoires (entre -99 et +99) pouvant être considérés comme les coefficients d'un polynôme de degré `n-1` (voir aussi 6.44.3, 6.40.39 et 7.3.9).
On tape :

```
ranm(3)
```

On obtient :

```
[68,-21,56]
```

6.26.33 Interpolation de Lagrange : `lagrange interp`

`lagrange` a comme argument deux listes de longueur `n` ou une matrice de deux lignes et `n` colonnes et eventuellement le nom de la variable `var` (par défaut `x`) :
la première liste (ou ligne) correspond à des valeurs d'abscisses x_k, et la deuxième liste (ou ligne) correspond à des valeurs d'ordonnées y_k pour k allant de 1 à `n`.
`lagrange` renvoie une expression polynômiale `P(var)` de degré `n-1` tel que $P(x_k) = y_k$.
On tape :

```
lagrange([[1,3],[0,1]])
```

Ou on tape :

```
lagrange([1,3],[0,1])
```

On obtient :

```
(x-1)/2
```

en effet pour $x = 1$ on a $\frac{x-1}{2} = 0$ et pour $x = 3$ on a $\frac{x-1}{2} = 1$.
On tape :

```
lagrange([1,3],[0,1],y)
```

On obtient :

```
(y-1)/2
```

Attention `lagrange([1,2],[3,4],y)` ne renvoie pas une fonction mais une expression. mais on peut définir une fonction en mettant :
`f(x):=lagrange([1,2],[3,4],x)` ou
`f(y):=lagrange([1,2],[3,4],y)` et alors
`f(4)` renvoie `6` car `f(x)=x+2`)
Bien voir la différence entre :
`g(x):=lagrange([1,2],[3,4])` et
`f(x):=lagrange([1,2],[3,4],x)`.
`g(x):=lagrange([1,2],[3,4])` ne definit pas une fonction, par exemple, `g(2)=x-1+3` alors que `f(2)=4`.
Ceci dit, la définition of `f` n'est pas efficace car le polynôme sera recalculé depuis le début à chaque appel de `f` (quand on définit une fonction le membre de droite n'est pas évalué, l'évaluation est faite seulement quand on appelle `f`).
Pour être efficace il faut utiliser `unapply` :
`f:=unapply(lagrange([1,2],[3,4]),x)` ou
`f:=unapply(lagrange([1,2],[3,4],y),y)` **Exercice** Soient $f(x) = \frac{1}{x}$, $x_0 = 2$ $x_1 = 2.5$ et $x_2 = 4$. On demande de calculer le polynôme L d'interpolation de Lagrange et sa valeur en $x = 3$ et $x = 4.5$.
On tape :

```
f(x):=1/x
L:=unapply(normal(lagrange([2,2.5,4],[f(2),f(2.5),f(4)])),x)
```

On obtient :

```
x->0.05*x^2-0.425*x+1.15
```

On tape :
`L(3),L(4.5)`
On obtient :
`0.325,0.25`

6.26.34 Les splines naturelles : `spline`

Définition
Soit une subdivision σ_n de l'intervalle $[a, b]$:

$$a = x_0, \quad x_1, \quad ..., \quad x_n = b$$

On dit que s est une fonction spline de degré l si s est une application de $[a, b]$ dans $\mathbb{R}$ vérifiant :
- s admet des dérivées continues jusqu'à l'ordre $l - 1$,
- s restreint à chaque intervalle de la subdivision est un polynôme de degré inférieur ou égal à l.

Théorème
L'ensemble des fonctions splines de degré l sur σ_n est un $\mathbb{R}$-espace vectoriel de dimension $n + l$.
En effet :
Sur $[a, x_1]$, s est un polynôme A de degré inférieur ou égal à l, donc sur $[a, x_1]$, $s = A(x) = a_0 + a_1x + ...a_lx^l$ et A est une combinaison linéaire de $1, x, ...x^l$.
Sur $[x_1, x_2]$, s est un polynôme B de degré inférieur ou égal à l, donc sur $[x_1, x_2]$, $s = B(x) = b_0 + b_1x + ...b_lx^l$.
Puisque s admet des dérivées continues jusqu'à l'ordre $l - 1$ on doit avoir :

$$\forall 0 \le j \le l - 1, \quad B^{(j)}(x_1) - A^{(j)}(x_1) = 0$$

donc $B(x) - A(x) = \alpha_1(x - x_1)^l$ ou encore $B(x) = A(x) + \alpha_1(x - x_1)^l$.
Soit la fonction :

$$\mathrm{q}_1(x) = \begin{cases} 0 & \text{sur} \quad [a, x_1] \\ (x - x_1)^l & \text{sur} \quad [x_1, b] \end{cases}$$

Donc :
$s|_{[a,x_2]} = a_0 + a_1x + ...a_lx^l + \alpha_1 q_1(x)$.
Sur $[x_2, x_3]$, s est un polynôme C de degré inférieur ou égal à l, donc sur $[x_2, x_3]$, $s = C(x) = c_0 + c_1x + ...c_lx^l$.
Puisque s admet des dérivées continues jusqu'à l'ordre $l - 1$ on doit avoir :

$$\forall 0 \le j \le l - 1, \quad C^{(j)}(x_2) - B^{(j)}(x_2) = 0$$

donc $C(x) - B(x) = \alpha_2(x - x_2)^l$ ou encore $C(x) = B(x) + \alpha_2(x - x_2)^l$.
Soit la fonction :

$$\mathrm{q}_2(x) = \begin{cases} 0 & \text{sur} \quad [a, x_2] \\ (x - x_2)^l & \text{sur} \quad [x_2, b] \end{cases}$$

Donc : $s|_{[a,x_3]} = a_0 + a_1x + ...a_lx^l + \alpha_1 q_1(x) + \alpha_2 q_2(x)$
Et ainsi de suite, on définit les fonctions :

$$\forall 1 \le j \le n - 1, \mathrm{q}_j(x) = \begin{cases} 0 & \text{sur} \quad [a, x_j] \\ (x - x_j)^l & \text{sur} \quad [x_j, b] \end{cases}$$

ainsi,
$s|_{[a,b]} = a_0 + a_1x + ...a_lx^l + \alpha_1q_1(x) + ... + \alpha_{n-1}q_{n-1}(x)$ et
s est une combinaison linéaire des $n+l$ fonctions indépendantes $1, x, ..x^l, q_1, ..q_{n-1}$.

Interpolation avec des fonctions splines

On peut demander d'interpoler une fonction f sur σ_n par une fonction spline s de degré l, ce qui va imposer à s de vérifier $s(x_k) = y_k = f(x_k)$ pour tout $0 \geq k \geq n$. On a donc $n + 1$ conditions, il reste donc $l - 1$ degrés de liberté. On peut donc encore imposer $l - 1$ conditions supplémentaires qui seront des conditions sur les derivées de s en a et b. Il existe alors trois types d'interpolation (interpolation d'Hermite, interpolation naturelle, interpolation périodique) qui sont obtenues en rajoutant trois types de contraintes. On peut montrer que pour chacun de ces types d'interpolation la solution au problème d'interpolation est unique.

Supposons l impair, $l = 2m - 1$, il y a donc $2m - 2$ degrés de liberté. On rajoute les contraintes suivantes :

– Interpolation d'Hermite

$$\forall 1 \leq j \leq m - 1, \quad s^{(j)}(a) = f^{(j)}(a), s^{(j)}(b) = f^{(j)}(b)$$

– Interpolation naturelle

$$\forall m \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b) = 0$$

– Interpolation périodique

$$\forall 1 \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b)$$

Supposons l pair, $l = 2m$, il y a donc $2m - 1$ degrés de liberté. On rajoute les contraintes suivantes :

– Interpolation d'Hermite

$$\forall 1 \leq j \leq m - 1, \quad s^{(j)}(a) = f^{(j)}(a), s^{(j)}(b) = f^{(j)}(b)$$

et

$$s^{(m)}(a) = f^{(m)}(a)$$

– Interpolation naturelle

$$\forall m \leq j \leq 2m - 2, \quad s^{(j)}(a) = s^{(j)}(b) = 0$$

et

$$s^{(2m-1)}(a) = 0$$

– Interpolation périodique

$$\forall 1 \leq j \leq 2m - 1, \quad s^{(j)}(a) = s^{(j)}(b)$$

Une spline naturelle de degré donné passant par des points donnés est une fonction spline vérifiant l'interpolation naturelle.

L'instruction `spline` calcule une spline naturelle de degré donné passant par des points dont les listes des abscisses par ordre croissant et des ordonnées sont passées en argument. Elle renvoie la fonction spline sous forme d'une liste de polynômes, chaque polynôme étant valide dans un intervalle. On donne dans l'ordre croissant la liste des abscisses, la liste des ordonnées, le nom de variables souhaité pour les polynômes et le degré.

Par exemple, on veut une spline naturelle de degré 3, passant par les points $x_0 = 0, y_0 = 1, x_1 = 1, y_1 = 3$ et $x_2 = 2, y_2 = 0$, on tape :

```
spline([0,1,2],[1,3,0],x,3)
```

On obtient une liste de deux polynômes fonction de x :

$$[-5*x^3/4+13*x/4+1, \quad 5*(x-1)^3/4-15*(x-1)^2/4+(x-1)/-2+3]$$

valables respectivement sur les intervalles $[0,1]$ et $[1,2]$.

Par exemple, on veut une spline naturelle de degré 4, passant par les points $x_0 = 0, y_0 = 1, x_1 = 1, y_1 = 3, x_2 = 2, y_2 = 0$ et $x_3 = 3, y_3 = -1$, on tape :

```
spline([0,1,2,3],[1,3,0,-1],x,4)
```

On obtient une liste de trois polynômes fonction de x :

$$[(-62*x^4+304*x)/121+1,$$

$$(201*(x-1)^4-248*(x-1)^3-372*(x-1)^2+56*(x-1))/121+3,$$

$$(-139*(x-2)^4+556*(x-2)^3+90*(x-2)^2+-628*(x-2))/121]$$

valables respectivement sur les intervalles $[0,1]$, $[1,2]$ et $[2,3]$.

Par exemple, pour avoir l'interpolation naturelle de cos sur $[0,\pi/2,3\pi/2]$, on tape :

```
spline([0,pi/2,3*pi/2],cos([0,pi/2,3*pi/2]),x,3)
```

On obtient :

$$[((3\pi^3+(-7\pi^2)x+4x^3)1/3)/(\pi^3),$$

$$((15\pi^3+(-46\pi^2)*+36\pi x^2-8x^3)1/12)/(\pi^3)]$$

6.27 Arithmétique des polynômes

Les polynômes sont représentés par des expressions ou par la liste de leurs coefficients par ordre de puissances décroissantes. Dans le premier cas la variable utilisée par défaut est x. Pour les polynômes à coefficients dans $\mathbb{Z}/n\mathbb{Z}$, appliquez `% n` à l'expression ou à chaque coefficient de la liste.

6.27.1 Liste des diviseurs d'un polynôme : `divis`

`divis` a pour argument un polynôme symbolique (ou une liste de polynômes) et renvoie la liste des diviseurs.
On tape :

```
divis(x^2-1)
```

On obtient :

```
[1,x-1,x+1,(x-1)*(x+1)]
```

On tape :

```
divis(t^2-1)
```

On obtient :

```
[1,t-1,t+1,(t-1)*(t+1)]
```

On tape :

```
divis(x^4-1)
```

Ou on tape :

```
divis(poly2symb([1,0,0,0,-1],x))
```

On obtient :

```
[1,x^2+1,x+1,(x^2+1)*(x+1),x-1,(x^2+1)*(x-1),
(x+1)*(x-1),(x^2+1)*(x+1)*(x-1)]
```

On tape :

```
divis([t^2,x^2-1])
```

On obtient :

```
[[1,t,t^2],[1,x+1,x-1,(x+1)*(x-1)]]
```

6.27.2 Quotient euclidien de 2 polynômes : quo

quo donne le quotient de la division euclidienne de deux polynômes (division selon les puissances décroissantes).
On peut donner les polynômes soit par la liste de leurs coefficients selon les puissances décroissantes, soit sous leurs formes symboliques et dans ce cas la variable doit être rajoutée comme troisième argument (par défaut la variable est x).
On tape :

```
quo(x^2+2x+1,x+3)
```

On obtient :

```
x-1
```

On tape :

```
quo(t^2+2t+1,t+3,t)
```

On obtient :

```
t-1
```

ou on tape :

```
quo([1,2,1],[1,3])
```

On obtient :

```
[] 1,-1 []
```

c'est à dire le polynôme poly1[1,-1].
Pour avoir le quotient de $x^3 + 2x + 4$ par $x^2 + x + 2$, on tape :

`quo(x^3+2x+4,x^2+x+2)`

On obtient :

`x-1`

Ou on tape :

`quo([1,0,2,4],[1,1,2])`

On obtient :

`[] 1,-1 []`

c'est à dire le polynôme `poly1[1,-1]` ou encore le polynôme `x-1`.
On tape :

`quo(t^3+2t+4,t^2+t+2,t)`

On obtient :

`t-1`

On tape si on ne met pas la variable t comme dernier argument :

`quo(t^3+2t+4,t^2+t+2)`

On obtient :

`(t^3+2*t+4)/(t^2+t+2)`

6.27.3 Quotient euclidien : `Quo`

`Quo` est la forme inerte de `quo`.
`Quo` renvoie le quotient de la division euclidienne de deux polynômes (division selon les puissances décroissantes) sans l'évaluer et cela permet de calculer le quotient euclidien de deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ en utilisant la syntaxe Maple.
Attention il faut être en mode Maple pour que cela soit efficace.
On tape, en mode `Xcas` :

`Quo(x^2+2*x+1,x)`

On obtient :

`quo(x^2+2*x+1,x)`

On peut aussi taper pour avoir le quotient de $x^2 + 2x + 4$ par $x^2 + x + 2$:

`Quo([1,2,4],[1,1,2])`

On obtient :

`quo([1,2,4],[1,1,2])`

En mode `Maple` on tape :

```
Quo(x^3+3*x,2*x^2+6*x+5) mod 5
```

On obtient :

```
-(2)*x+1)
```

La division est faite dans $\mathbb{Z}/5\mathbb{Z}[X]$ alors que pour :

```
quo(x^3+3*x,2*x^2+6*x+5) mod 5
```

la division est faite dans $\mathbb{Z}[X]$ puis est réduite après :

```
3*x-9
```

Si `Xcas` n'est pas en mode Maple, la division des polynômes dans $\mathbb{Z}/p\mathbb{Z}[X]$ se fait en tapant :

```
quo((x^3+3*x)% 5,(2x^2+6x+5)%5)
```

6.27.4 Reste euclidien de 2 polynômes : `rem`

`rem` donne le reste de la division euclidienne de deux polynômes (division selon les puissances décroissantes).
On peut donner les polynômes soit par la liste de leurs coefficients selon les puissances décroissantes, soit sous leurs formes symboliques et dans ce cas la variable doit être rajoutée comme troisième argument (par défaut la variable est `x`).
On tape :

```
rem(x^3-1,x^2-1)
```

On obtient :

```
x-1
```

On tape :

```
rem(t^3-1,t^2-1,t)
```

On obtient :

```
t-1
```

On tape :

```
rem(x^2+2x+1,x+3)
```

Ou on tape :

```
rem(t^2+2t+1,t+3,t)
```

On obtient :

```
4
```

ou on tape :

```
rem([1,2,1],[1,3])
```

On obtient :

```
[] 4
```

c'est à dire le polynôme `poly1[4]` ou encore le polynôme `4`.
On tape pour avoir le reste de $x^3 + 2x + 4$ par $x^2 + x + 2$:

```
rem(x^3+2x+4,x^2+x+2)
```

On obtient :

```
x+6
```

Ou on tape :

```
rem([1,0,2,4],[1,1,2])
```

On obtient :

```
[] 1,6[]
```

c'est à dire le polynôme `poly1[1,6]` ou encore le polynôme `x+6`.
On tape :

```
rem(t^3+2t+4,t^2+t+2,t)
```

On obtient :

```
t+6
```

On tape si on ne met pas la variable t comme dernier argument :

```
rem(t^3+2t+4,t^2+t+2)
```

On obtient :

```
0
```

6.27.5 Reste euclidien : `Rem`

`Rem` est la forme inerte de `rem`.
`Rem` renvoie le reste de la division euclidienne de deux polynômes (division selon les puissances décroissantes) sans l'évaluer et cela permet de calculer le reste euclidien de deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ en utilisant la syntaxe Maple.
Attention il faut être en mode Maple pour que cela soit efficace.
On tape, en mode `Xcas` :

```
Rem(x^3-1,x^2-1)
```

On obtient :

```
rem(x^3-1,x^2-1)
```

On peut aussi taper pour avoir le reste de $x^2 + 2x + 4$ par $x^2 + x + 2$:

```
Rem([1,2,4],[1,1,2])
```

On obtient :

```
rem([1,2,4],[1,1,2])
```

En mode `Maple` on tape :

```
Rem(x^3+3*x,2*x^2+6*x+5) mod 5
```

On obtient :

```
2*x
```

La division est faite dans $\mathbb{Z}/5\mathbb{Z}[X]$ alors que pour :

```
rem(x^3+3*x,2*x^2+6*x+5) mod 5
```

la division est faite dans $\mathbb{Z}[X]$ puis est réduite après :

```
12*x
```

Si `Xcas` n'est pas en mode Maple, la division des polynômes dans $\mathbb{Z}/p\mathbb{Z}[X]$ se fait en tapant :

```
rem((x^3+3*x)% 5,(2x^2+6x+5)%5)
```

6.27.6 Quotient et reste euclidien : `quorem divide`

`quorem` (ou `divide`) donne la liste, du quotient et du reste de la division euclidienne (selon les puissances décroissantes) de deux polynômes.
On peut donner les polynômes soit par la liste de leurs coefficients selon les puissances décroissantes, soit sous leurs formes symboliques et dans ce cas la variable doit être rajoutée comme troisième argument (par défaut la variable est `x`).
On tape pour avoir le quotient et le reste de la division de x^3+2x+4 par x^2+x+2 :

```
quorem(x^3+2x+4,x^2+x+2)
```

On obtient :

```
[x-1,x+6]
```

Ou on tape :

```
quorem([1,0,2,4],[1,1,2])
```

On obtient :

```
[[1,-1],[1,6]]
```

c'est à dire la liste des polynômes `[poly1[1,-1],poly1[1,6]]` donc le quotient est le polynôme `x-1` et le reste est le polynôme `x+6`.
On tape :

```
quorem(t^3+2t+4,t^2+t+2,t)
```

On obtient :

```
[t-1,t+6]
```

On tape :

```
quorem(t^3+2t+4,t^2+t+2)
```

On obtient :

```
[(t^3+2*t+4)/(t^2+t+2),0]
```

On tape :

```
quorem(x^3-1,x^2-1)
```

On obtient :

```
[x,x-1]
```

On tape :

```
quorem(t^3-1,t^2-1,t)
```

On obtient :

```
[t,t-1]
```

6.27.7 PGCD de polynômes par l'algorithme d'Euclide : `gcd igcd`

`gcd` désigne le PGCD (plus grand commun diviseur) de deux polynômes pouvant avoir plusieurs variables et aussi le PGCD d'une liste de polynômes ou d'une séquence de polynômes pouvant avoir plusieurs variables (voir 6.7.2 pour le PGCD d'entiers). On peut aussi mettre comme paramètres deux listes de même longueur (ou une matrice ayant 2 lignes), dans ce cas `gcd` renvoie le PGCD des éléments de même indice (ou d'une même colonne). On tape :

```
gcd([x^2-4,x*y-y],[x^3-8,y^2-x^2*y])
```

Ou on tape :

```
gcd([[x^2-4,x*y-y],[x^3-8,y^2-x^2*y]])
```

On obtient :

```
[x-2,y]
```

Exemples
On tape :

```
gcd(x^2+2*x+1,x^2-1)
```

On obtient :

```
x+1
```

On tape :

```
gcd(x^2-2*x+1,x^3-1,x^2-1,x^2+x-2)
```

ou

```
gcd([x^2-2*x+1,x^3-1,x^2-1,x^2+x-2])
```

On obtient :

```
x-1
```

On tape :

```
A:=z^2+x^2*y^2*z^2+(-(y^2))*z^2+(-(x^2))*z^2
```

```
B:=x^3*y^3*z+(-(y^3))*z+x^3*z-z
```

```
C:=gcd(A,B)
```

On obtient :

```
z*x*y+z*x-z*y-z
```

On tape :

```
factor(A)
```

On obtient :

```
(y-1)*(y+1)*(x-1)*(x+1)*z^2
```

On tape :

```
factor(B)
```

On obtient :

```
(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z
```

On tape :

```
factor(C)
```

On obtient :

```
(y+1)*(x-1)*z
```

Pour les polynômes à coefficients modulaire, on tape par exemple : On tape :

```
gcd((x^2+2*x+2) mod 5,(x^2-1) mod 5)
```

On obtient :

```
(1 % 5)*x-1 % 5
```

Mais si on tape :

```
gcd(x^2+2*x+2,x^2-1) mod 5)
```

On obtient :

```
1%5
```

car l'opération modulaire se fait après le calcul du PGCD qui a èté calculé dans $\mathbb{Z}[X]$.

6.27.8 PGCD de deux polynômes par l'algorithme d'Euclide : `Gcd`

`Gcd` est la forme inerte de `gcd`.
`Gcd` renvoie le PGCD (plus grand commun diviseur) de deux polynômes (ou d'une liste de polynômes ou d'une séquence de polynômes) (voir 6.7.2 pour le PGCD d'entiers), sans l'évaluer et cela permet de calculer le PGCD de deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ en utilisant la syntaxe Maple.
Attention il faut être en mode Maple pour que cela soit efficace.
On tape en mode `Xcas` :

```
Gcd((x^2+2*x)mod 5,(x^2+6*x+5x) mod 5)
```

On obtient :

```
gcd((1% 5)*x^2+(2% 5)*x,(1% 5)*x^2+(1% 5)*x)
```

puis on obtient :

```
(1% 5)*x
```

Mais si on tape en mode `Xcas` :

```
Gcd((x^2+2*x),(x^2+6*x+5x)) mod 5
```

On obtient :

```
(1% 5)*gcd(x^2+2*x,x^2++6*x+5x)
```

puis :

```
(1% 5)
```

On tape en mode `Maple` :

```
Gcd(x^2+2*x,x^2+6*x+5) mod 5
```

On obtient :

```
1
```

On tape :

```
gcd(x^2+2*x,x^2+x) mod 5
```

On obtient :

```
x
```

6.27.9 Choisir l'algorithme du PGCD de deux polynômes : `ezgcd heugcd modgcd psrgcd`

`ezgcd heugcd modgcd psrgcd` désigne le PGCD (plus grand commun diviseur) de deux polynômes (ou d'une liste de polynômes ou d'une séquence de polynômes) de plusieurs variables.
`ezgcd` est calculé avec l'algorithme ezgcd,
`heugcd` est calculé avec l'algorithme dit du pgcd heuristique,
`modgcd` est calculé avec l'algorithme modulaire,
`psrgcd` est calculé avec l'algorithme du sous résultant.
On tape :

```
gcd(x^2-2*x*y+y^2-1,x-y)
```

ou

```
ezgcd(x^2-2*x*y+y^2-1,x-y)
```

ou

```
heugcd(x^2-2*x*y+y^2-1,x-y)
```

ou

```
modgcd(x^2-2*x*y+y^2-1,x-y)
```

ou

```
psrgcd(x^2-2*x*y+y^2-1,x-y)
```

On obtient :

```
1
```

On tape :

```
gcd((x+y-1)*(x+y+1),(x+y+1)^2)
```

ou On tape :

```
ezgcd((x+y-1)*(x+y+1),(x+y+1)^2)
```

ou

```
heugcd((x+y-1)*(x+y+1),(x+y+1)^2)
```

ou

```
modgcd((x+y-1)*(x+y+1),(x+y+1)^2)
```

On obtient :

```
x+y+1
```

On tape :

```
psrgcd((x+y-1)*(x+y+1),(x+y+1)^2)
```

On obtient :

```
-x-y-1
```

On tape :

```
ezgcd((x+1)^4-y^4,(x+1-y)^2)
```

On obtient :

```
"GCD not successfull Error: Bad Argument Value"
```

Mais si on tape :

```
gcd((x+1)^4-y^4,(x+1-y)^2)
```

ou

```
heugcd((x+1)^4-y^4,(x+1-y)^2)
```

ou

```
modgcd((x+1)^4-y^4,(x+1-y)^2)
```

ou

```
psrgcd((x+1)^4-y^4,(x+1-y)^2)
```

On obtient :

```
x-y+1
```

6.27.10 PPCM de deux polynômes : `lcm`

`lcm` désigne le PPCM (plus petit commun multiple) de deux polynômes pouvant avoir plusieurs variables et aussi le PPCM d'une liste de polynômes ou d'une séquence de polynômes pouvant avoir plusieurs variables (voir 6.7.5 pour le PPCM d'entiers).
On tape :

```
lcm(x^2+2*x+1,x^2-1)
```

On obtient :

```
(x+1)*(x^2-1)
```

On tape :

```
lcm(x,x^2+2*x+1,x^2-1)
```

ou

```
lcm([x,x^2+2*x+1,x^2-1])
```

On obtient :

```
(x^2+x)*(x^2-1)
```

On tape :

```
A:=z^2+x^2*y^2*z^2+(-(y^2))*z^2+(-(x^2))*z^2
```

```
B:=x^3*y^3*z+(-(y^3))*z+x^3*z-z
```

```
D:=lcm(A,B)
```

On obtient :

```
(x*y*z-x*z+y*z-z)*(x^3*y^3*z+(-(y^3))*z+x^3*z-z)
```

On tape :

```
factor(A)
```

On obtient :

```
(y-1)*(y+1)*(x-1)*(x+1)*z^2
```

On tape :

```
factor(B)
```

On obtient :

```
(x^2+x+1)*(x-1)*(y+1)*(y^2-y+1)*z
```

On tape :

```
factor(D)
```

On obtient :

```
(x-1)*(x+1)*(x^2+x+1)*(y-1)*(y+1)*(y^2-y+1)*z^2
```

6.27.11 Idendité de Bézout : `egcd gcdex`

Il s'agit de l'identité de Bézout pour les polynômes (Extended Greatest Common Divisor).
`egcd` a 2 ou 3 arguments : les polynômes A and B qui sont, soit sous la forme d'expressions d'une variable, (si la variable n'est pas spécifiée c'est x), soit donné par la liste de leurs coefficients par ordre de puissances décroissantes.
Etant donnés 2 polynômes $A(x), B(x)$, `egcd` ou `gcdex` renvoie 3 polynômes `[U(x),V(x),D(x)]` vérifiant :

```
U(x)*A(x)+V(x)*B(x)=D(x)=PGCD(A(x),B(x))
```

On tape :

```
egcd(x^2+2*x+1,x^2-1)
```

On obtient :

```
[1,-1,2*x+2]
```

On tape :

```
egcd([1,2,1],[1,0,-1])
```

On obtient :

```
[[1],[-1],[2,2]]
```

On tape :

```
egcd(t^2+2*t+1,t^2-1,t)
```

On obtient :

```
[1,-1,2*t+2]
```

On tape :

```
egcd(x^2-2*x+1,x^2-x+2)
```

On obtient :

```
[x-2,-x+3,4]
```

On tape :

```
egcd([1,-2,1],[1,-1,2])
```

On obtient :

```
[[1,-2],[-1,3],[4]]
```

On tape :

```
egcd(t^2-2*t+1,t^2-t+2,t)
```

On obtient :

```
[t-2,-t+3,4]
```

6.27.12 Résolution polynômiale de au+bv=c : `abcuv`

Il s'agit encore de l'identité de Bézout.
`abcuv` résout l'équation polynômiale

$$C(x) = U(x) * A(x) + V(x) * B(x)$$

dans laquelle les inconnues sont les polynômes U et V et les paramètres sont les trois polynômes, A, B, C où C doit être un multiple du PGCD de A et B.
`abcuv` a comme argument 3 expressions polynômiales A, B, C et le nom de leur variable (par défaut x) (resp 3 listes représentant les coefficients par puissances décroisantes de 3 polynômes A, B, C). `abcuv` renvoie la liste de 2 expressions polynômiales U et V (resp de 2 listes quisont les coefficients par puissances décroisantes de U et V).
On tape :

```
abcuv(x^2+2*x+1 ,x^2-1,x+1)
```

On obtient :

```
[1/2,1/-2]
```

On tape :

```
abcuv(x^2+2*x+1 ,x^2-1,x^3+1)
```

On obtient :

```
[1/2*x^2+1/-2*x+1/2,-1/2*x^2-1/-2*x-1/2]
```

On tape :

```
abcuv([1,2,1],[1,0,-1],[1,0,0,1])
```

On obtient :

```
[poly1[1/2,1/-2,1/2],poly1[1/-2,1/2,1/-2]]
```

6.27.13 Les restes chinois : `chinrem`

`chinrem` a comme argument deux listes ayant chacun comme composantes deux polynômes éventuellement donnés par la liste de leurs coefficients par ordre décroissant.
`chinrem` renvoie une liste de composantes deux polynômes.
`chinrem([A,R],[B,Q])` renvoie la liste des polynômes `P` et `S` vérifiant :

$$S = R.Q, \quad P = A \pmod R, P = B \pmod Q)$$

Il existe toujours une solution `P` si `R` et `Q` sont premiers entre eux, et toutes les solutions sont congrues modulo `S=R*Q`
Trouver les solutions $P(x)$ de :

$$\begin{cases} P(x) = & x & \mod (x^2+1) \\ P(x) = & x-1 & \mod (x^2-1) \end{cases}$$

On tape :

```
chinrem([[1,0],[1,0,1]],[[1,-1],[1,0,-1]])
```

On obtient :

```
[[1/-2,1,1/-2],[1,0,0,0,-1]]
```

ou on tape :

```
chinrem([x,x^2+1],[x-1,x^2-1])
```

On obtient :

```
[1/-2*x^2+x+1/-2,x^4-1]
```

donc $P(x) = -\dfrac{x^2-2.x+1}{2} \pmod{x^4-1}$
Autre exemple :
On tape :

```
chinrem([[1,2],[1,0,1]],[[1,1],[1,1,1]])
```

On obtient :

```
[[-1,-1,0,1],[1,1,2,1,1]]
```

ou on tape :

```
chinrem([x+2,x^2+1],[x+1,x^2+x+1])
```

On obtient :

```
[-x^3-x^2+1,x^4+x^3+2*x^2+x+1]
```

6.27.14 Polynôme cyclotomique : `cyclotomic`

`cyclotomic` a comme paramètre un entier n.
`cyclotomic` renvoie la liste des coefficients du polynôme cyclotomique d'ordre n. C'est le polynôme dont les zéros sont toutes les racines n-ième et primitives de l'unité (une racine n-ième de l'unité est primitive si ses puissances engendrent toutes les autres racines n-ième de l'unité).
Par exemple pour $n = 4$, les racines quatrième de l'unité sont : $\{1, i, -1, -i\}$, et les racines primitives sont : $\{i, -i\}$.
Donc le polynôme cyclotomique d'ordre 4 est $(x - i).(x + i) = x^2 + 1$.
On tape :

```
cyclotomic(4)
```

On obtient :

```
[1,0,1]
```

On tape :

```
cyclotomic(5)
```

On obtient :

```
[1,1,1,1,1]
```

Donc le polynôme cyclotomique d'ordre 5 est $x^4 + x^3 + x^2 + x + 1$ et on a $(x - 1) * (x^4 + x^3 + x^2 + x + 1) = x^5 - 1$.
On tape :

```
cyclotomic(10)
```

On obtient :

```
[1,-1,1,-1,1]
```

Donc le polynôme cyclotomique d'ordre 10 est $x^4 - x^3 + x^2 - x + 1$ et on a

$$(x^5 - 1) * (x + 1) * (x^4 - x^3 + x^2 - x + 1) = x^{10} - 1$$

On tape :

```
cyclotomic(20)
```

On obtient :

```
[1,0,-1,0,1,0,-1,0,1]
```

Donc le polynôme cyclotomique d'ordre 20 est $x^8 - x^6 + x^4 - x^2 + 1$ et on a

$$(x^{10} - 1) * (x^2 + 1) * (x^8 - x^6 + x^4 - x^2 + 1) = x^{20} - 1$$

6.27.15 Suites de Sturm et nombre de de changements de signe de P sur $]a;\ b]$: `sturm`

`sturm` a deux ou quatre paramètres : une expression polynômiale P ou une fraction rationnelle P/Q et le nom de la variable ou une expression polynômiale P, le nom de la variable et deux nombres a et b.

Lorsqu'il y a 2 paramètres `sturm` renvoie la liste des suites de Sturm et de leur multiplicité pour P ou pour P et pour Q (`sturm` est alors identique à `sturmseq`).

Lorsqu'il y a 4 paramètres `sturm`, se comporte comme `sturmab` :

– si a et b sont réels, `sturm` renvoie le nombre de changements de signe de P sur $]a;\ b]$

– si a ou b est complexe, `sturm` renvoie le nombre de racines complexes à l'intérieur du rectangle de sommets opposés a et b.

On tape :

```
sturm(2*x^3+2,x)
```

On obtient :

```
[2,[[1,0,0,1],[3,0,0],-9],1]
```

On tape :

```
sturm((2*x^3+2)/(x+2),x)
```

On obtient :

```
[2,[[1,0,0,1],[3,0,0],-9],1,[[1,2],1]]
```

On tape :

```
sturm(x^2*(x^3+2),x,-2,0)
```

On obtient :

```
1
```

6.27.16 Nombre de changements de signe sur $]a;\ b]$: `sturmab`

`sturmab` a quatre paramètres : une expression polynômiale P, le nom de la variable et deux nombres a et b.

– si a et b sont réels, `sturmab` renvoie soit un nombre strictement positif qui est le nombre de changements de signe de P sur $]a;\ b]$, soit 0 si P reste de signe constant positif ou nul sur $]a;\ b]$, soit -1 si P reste de signe constant négatif ou nul sur $]a;\ b]$. Ainsi, `sturmab` permet d'avoir le nombre de racines sur $[a, b[$ du polynôme P/G avec $G = \mathrm{gcd}(P, \mathrm{diff}(P))$.

– si a ou b est complexe, le nombre de racines complexes à l'intérieur du rectangle de sommets opposés a et b.

On tape :

```
sturmab(x^2*(x^3+2),x,-2,0)
```

On obtient :

```
1
```

On tape :

```
sturmab(x^3-1,x,-2-i,5+3i)
```

On obtient :

```
3
```

On tape :

```
sturmab(x^3-1,x,-i,5+3i)
```

On obtient :

```
1
```

Attention ! ! ! !
P doit être donné par son expression symbolique et, si on tape :
`sturmab([1,0,0,2,0,0],x,-2,0)`,
on obtient :
`Bad argument type`.

6.27.17 Suites de Sturm : `sturmseq`

`sturmseq` a comme paramètre une expression polynômiale P ou une fraction rationnelle P/Q.
`sturmseq` renvoie la liste des suites de Sturm et de leur multiplicité pour P ou pour P et pour Q.
La suite de sturm $R_1, R_2, ...$ est obtenue à partir du facteur F sans carré de P. Pour obtenir F à partir de la décomposition de P en facteurs premiers, on élimine les termes carrés et on transforme les puissances impaires en puissances 1.
R_1 est l'opposé du reste de la division euclidienne de F par F' puis, R_2 est l'opposé du reste de la division euclidienne de F' par R_1
....
et ainsi de suite jusqu'à ce que $R_k = 0$.
On tape :

```
sturmseq(2*x^3+2)
```

ou

```
sturmseq(2*y^3+2,y)
```

On obtient :

```
[2,[[1,0,0,1],[3,0,0],-9],1]
```

Le premier terme donne le PGCD des coefficients du numérateur (ici 2), le dernier terme donne le dénominateur (ici 1). Entre les deux on a la suite des polynômes $[x^3 + 1, 3x^2, -9]$.
On tape :

```
sturmseq((12*x^3+4)/(6*x^2+3),x)
```

On obtient :

```
[4,[[3,0,0,1],[9,0,0],-81],3,[[2,0,1],[4,0],-16]]
```

Le premier terme donne le PGCD des coefficients du numérateur (ici 4), puis la suite de Sturm du numérateur ([[3,0,0,1],[9,0,0],-81]), puis le le PGCD des coefficient du dénominateur (ici 3), et la suite de Sturm du dénominateur ([[2,0,1],[4,0],-16]). On a la suite des polynômes $[3x^3 + 1, 9x^2, -81]$ pour le numérateur et, $[2x^2 + 1, 4x, -16]$ pour le dénominateur.
On tape :

```
sturmseq((x^3+1)^2,x)
```

On obtient :

```
[1,1]
```

En effet les termes carrés sont éliminés et $F = 1$.
On tape :

```
sturmseq(3*(3*x^3+1)/(2*x+2),x)
```

On obtient :

```
[3,[[3,0,0,1],[9,0,0],-81],2,[[1,1],1]]
```

Le premier terme donne le PGCD des coefficients du numérateur (ici `3`),
le deuxième terme donne la suite de polynômes (ici `3x^3+1, 9x^2, -81`),
le troisième terme donne le PGCD des coefficients du dénominateur (ici `2`),
le quatrième terme indique la suite de polynômes du dénominateur (`x+1,1`).
Attention ! ! ! !
P doit être donné par son expression symbolique et,si on tape :
`sturmseq([1,0,0,1],x)`,
on obtient :
`Bad argument type`.

6.27.18 Matrice de Sylvester de deux polynômes : `sylvester`

`sylvester` a comme arguments deux polynômes.
`sylvester` renvoie la matrice S de Sylvester des deux polynômes.
Pour deux polynômes $A(x) = \sum_{i=0}^{i=n} a_i x^i$ et $B(x) = \sum_{i=0}^{i=m} b_i x^i$, la matrice `S` de Sylvester est une matrice carrée de dimensiom `m+n` dont les `m=degree(B(x))` premières lignes sont composées à partir des coefficients de $A(x)$:

$$\begin{pmatrix} s_{11} = a_n & s_{12} = a_{n-1} & \cdots & s_{1(n+1)} = a_0 & 0 & \cdots & 0 \\ s_{21} = 0 & s_{22} = a_n & \cdots & s_{2(n+1)} = a_1 & s_{2(n+2)} = a_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{m1} = 0 & s_{m2} = 0 & \cdots & s_{m(n+1)} = a_{m-1} & s_{m(n+2)} = a_{m-2} & \cdots & a_0 \end{pmatrix}$$

et les `n=degree(A(x))` lignes suivantes sont composées de la même façon à partir des coefficients de `B(x)` :

$$\begin{pmatrix} s_{(m+1)1} = b_m & s_{(m+1)2} = b_{m-1} & \cdots & s_{(m+1)(m+1)} = b_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{(m+n)1} = 0 & s_{(m+n)2} = 0 & \cdots & s_{(m+n)(m+1)} = b_{n-1} & b_{n-2} & \cdots & b_0 \end{pmatrix}$$

On tape :

```
sylvester(x^3-p*x+q,3*x^2-p,x)
```

On obtient :

```
[[1,0,-p,q,0],[0,1,0,-p,q],[3,0,-p,0,0],
          [0,3,0,-p,0],[0,0,3,0,-p]]
```

On tape :

```
det([[1,0,-p,q,0],[0,1,0,-p,q],[3,0,-p,0,0],
          [0,3,0,-p,0],[0,0,3,0,-p]])
```

On obtient :

```
-4*p^3-27*q^2
```

6.27.19 Résultant de deux polynômes : `resultant`

`resultant` a comme arguments deux polynômes.
`resultant` renvoie le résultant des deux polynômes.
Le résultant est le déterminant de la matrice S de Sylvester.
Pour les deux polynômes $A(x) = \sum_{i=0}^{i=n} a_i x^i$ et $B(x) = \sum_{i=0}^{i=m} b_i x^i$, la matrice S de Sylvester est une matrice carrée de dimensiom $m+n$ dont les m premières lignes sont composées à partir des coefficients de $A(x)$:

$$\begin{pmatrix} s_{00} = a_n & s_{01} = a_{n-1} & \cdots & s_{0n)} = a_0 & 0 & \cdots & 0 \\ s_{10} = 0 & s_{11} = a_n & \cdots & s_{1n} = a_1 & s_{1(n+1)} = a_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{(m-1)0} = 0 & s_{(m-1)1} = 0 & \cdots & s_{(m-1)n} = a_{m-1} & s_{(m-1)(n+1)} = a_{m-2} & \cdots & a_0 \end{pmatrix}$$

et les n lignes suivantes sont composées de la même façon à partir des coefficients de $B(x)$:

$$\begin{pmatrix} s_{m0} = b_m & s_{m1} = b_{m-1} & \cdots & s_{mm} = b_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{(m+n-1)0} = 0 & s_{(m+n-1)1} = 0 & \cdots & s_{(m+n-1)m} = b_{n-1} & b_{n-2} & \cdots & b_0 \end{pmatrix}$$

On tape :

```
resultant(x^3-p*x+q,3*x^2-p,x)
```

On obtient :

```
-4*p^3-27*q^2
```

On cherche si il existe 2 polynômes $U(x) = \alpha * x + \beta$ (de degré 1) et $V(x) = \gamma * x^2 + \delta * x + \epsilon$ (de degré 2) pour que $U(x)*(x^3 - p*x + q) + V(x)*(3*x^2 - p) = 1$
On doit donc résoudre un système linéaire de 5 équations à 5 inconnues qui sont $\alpha, ...\delta, \eta$ (Attention ! $\epsilon = 1e - 10$).
On tape :
`symb2poly((alpha*x+beta)*(x^3-p*x+q)+(gamma*x^2+delta*x+eta)*(3*x^2-p),x)`
On obtient :
`poly1[alpha+3*gamma,beta+3*delta,-alpha*p-p*gamma+3*eta, alpha*q-beta*p-p*delta,beta*q-p*eta]`
La matrice A de ce système est donc :

$$A = \begin{pmatrix} 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 \\ -p & 0 & -p & 0 & 3 \\ q & -p & 0 & -p & 0 \\ 0 & q & 0 & 0 & -p \end{pmatrix}$$

la matrice S de Sylvester est la transposée de A :

$$S = \begin{pmatrix} 1 & 0 & -p & q & 0 \\ 0 & 1 & 0 & -p & q \\ 3 & 0 & -p & 0 & 0 \\ 0 & 3 & 0 & -p & 0 \\ 0 & 0 & 3 & 0 & -p \end{pmatrix}$$

On a `det(A)=det(S)=-4*p^3+27*q^2`
En fait on résout $UP+VQ = C$ avec C quelconque tel que deg(C)<deg(P)+deg(Q) i.e. on cherche U et V tel que deg(U)<deg(Q) et deg(V)<deg(P) (inegalites strictes) vérifiant $UP + VQ = 1$. Lorsque le système est de Cramer, il y a une solution unique et ca correspond en arithmétique à P et Q premiers entre eux (et réciproquement). Donc si det(A)=det(S) est non nul, U et V existent et sont uniques donc les 2 polynômes $x^3 - p * x + q$ et $3 * x^2 - p$ sont premiers entre eux et réciproquement si les 2 polynômes $x^3 - p * x + q$ et $3 * x^2 - p$ sont premiers entre eux U et V tel que deg(U)<deg(Q) et deg(V)<deg(P) existent et sont uniques donc det(A)=det(S) est non nul.
Donc si ce déterminant est nul les 2 polynômes $x^3 - p * x + q$ et $3 * x^2 - p$ ne sont pas premiers entre eux.
Remarque
On a : discriminant(P)=`resultant(P,P')/lcoeff(P)`.

Un exemple d'utilisation du résultant
Soient 2 points fixes $F1$ et $F2$ et un point variable A sur le cercle de centre $F1$ et de rayon $2a$. On veut trouver l'équation cartésienne du lieu des points M intersection de $F1A$ et de la médiatrice de $F2A$: on a $MF1 + MF2 = MF1 + MA = F1A = 2a$ donc M décrit une ellipse de foyers $F1$ et $F2$ et de grand axe $2a$.
Choisisons comme repère orthonormé celui de centre $F1$ et d'axe Ox porté par le vecteur $\overrightarrow{F1F2}$. On a :
$A = (2a\cos(\theta); 2a\sin(\theta))$ où θ est l'angle (Ox, OA). On choisit comme paramètre $t = \tan(\theta/2)$ pour que les coordonnées de A soient une fonction rationnelle

du paramètre t. On a donc :
$A = (ax; ay) = (2a\dfrac{1-t^2}{1+t^2}; 2a\dfrac{2t}{1+t^2})$
On pose $F1F2 = 2c$ et on note I le milieu de $AF2$. On a :
$F2 = (2c, 0)$ et
$I = (c + ax/2; ay/2) = (c + a\dfrac{1-t^2}{1+t^2}; a\dfrac{2t1-t^2}{1+t^2})$
IM est perpendiculaire à $AF2$ donc $M = (x; y)$ vérifie l'équation $eq1 = 0$ avec :
$eq1 := (x - ix) * (ax - 2 * c) + (y - iy) * ay$
$M = (x; y)$ est sur $F1A$ donc M vérifie l'équation $eq2 = 0$ avec :
$eq2 := y/x - ay/ax$
On a :
`resultant(eq1,eq2,t)` est un polynôme $eq3$ en x et y, $eq3$ est indépendant de t et il existe des polynômes en t, U et V tels que : $U(t)*eq1+V(t)*eq2 = eq3$.
On tape :

```
ax:=2*a*(1-t^2)/(1+t^2);ay:=2*a*2*t/(1+t^2);
ix:=(ax+2*c)/2; iy:=(ay/2)
eq1:=(x-ix)*(ax-2*c)+(y-iy)*ay
eq2:=y/x-ay/ax
factor(resultant(eq1,eq2,t))
```

On obtient comme résultant :

```
-(64·(x^2+y^2)·(x^2·a^2-x^2·c^2+-2·x·a^2·c+2·x·c^3-a^4+
2·a^2·c^2+a^2· y^2-c^4))
```

Le facteur `-64·(x^2+y^2)` ne s'annule jamais donc l'équation du lieu est :

$$x^2a^2 - x^2c^2 + -2xa^2c + 2xc^3 - a^4 + 2a^2c^2 + a^2y^2 - c^4 = 0$$

En prenant l'origine du repère en O milieu de $F1F2$, on retrouve l'équation cartésienne de l'ellipse. Pour faire ce changement d'origine, on a $\overrightarrow{F1M} = \overrightarrow{F1O} + \overrightarrow{OM}$, donc on tape :

```
normal(subst(x^2·a^2-x^2·c^2+-2· x·a^2·c+2·x·c^3-a^4+
2·a^2·c^2+a^2·y^2-c^4,[x,y]=[c+X,Y]))
```

On obtient :

```
-c^2*X^2+c^2*a^2+X^2*a^2-a^4+a^2*Y^2
```

ou encore si on pose $b^2 = a^2 - c^2$

```
normal(subst(-c^2*X^2+c^2*a^2+X^2*a^2-a^4+a^2*Y^2,
c^2=a^2-b^2))
```

On obtient :

```
-a^2*b^2+a^2*Y^2+b^2*X^2
```

c'est à dire après division par a^2b^2, M vérifie l'équation :

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

Un autre exemple d'utilisation du résultant

Soient 2 points fixes $F1$ et $F2$ et un point variable A sur le cercle de centre $F1$ et de rayon $2a$. On veut trouver l'équation cartésienne de l'enveloppe de la médiatrice D de $F2A$ (on sait que la médiatrice de $F2A$ est tangente à l'ellipse de foyers $F1$ et $F2$ et de grand axe $2a$).
Choisisons comme repère orthonormé celui de centre $F1$ et d'axe Ox porté par le

vecteur $\overrightarrow{F1F2}$. On a :
$A = (2a\cos(\theta); 2a\sin(\theta))$ où θ est l'angle (Ox, OA). On choisit comme paramètre $t = \tan(\theta/2)$ pour que les coordonnées de A soient une fonction rationnelle du paramètre t. On a donc :
$A = (ax; ay) = (2a\dfrac{1-t^2}{1+t^2}; 2a\dfrac{2t}{1+t^2})$
On pose $F1F2 = 2c$ et on note I le milieu de $AF2$. On a :
$F2 = (2c, 0)$ et
$I = (c + ax/2; ay/2) = (c + a\dfrac{1-t^2}{1+t^2}; a\dfrac{2t1-t^2}{1+t^2})$
D est perpendiculaire à $AF2$ donc D a pour équation : $eq1 = 0$ avec :
$eq1 := (x - ix) * (ax - 2 * c) + (y - iy) * ay$
L'enveloppe de D est donc le lieu de M intersection de D et de D' d'équation $eq2 = 0$ avec $eq2 := diff(eq1, t)$.
On tape :

```
ax:=2*a*(1-t^2)/(1+t^2);ay:=2*a*2*t/(1+t^2);
ix:=(ax+2*c)/2; iy:=(ay/2)
eq1:=normal((x-ix)*(ax-2*c)+(y-iy)*ay)
eq2:=normal(diff(eq1,t))
factor(resultant(eq1,eq2,t))
```

On obtient comme résultant :

```
(-(64· a^2))·(x^2+y^2)·(x^2·a^2-x^2·c^2+-2·x·a^2·c+2·x·c^3-
a^4+2·a^2·c^2+a^2·y^2-c^4)
```

Le facteur `-64·(x^2+y^2)` ne s'annule jamais donc l'équation du lieu est :
$x^2a^2 - x^2c^2 + -2xa^2c + 2xc^3 - a^4 + 2a^2c^2 + a^2y^2 - c^4 = 0$
En prenant l'origine du repère en O milieu de $F1F2$, on retrouve comme précédemment l'équation cartésienne de l'ellipse :

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

6.28 Polynômes orthogonaux

6.28.1 Polynôme de Legendre : `legendre`

`legendre` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`legendre` renvoie le polynôme de Legendre de degré n : c'est le polynôme non nul, solution de l'équation différentielle :

$$(x^2 - 1).y'' - 2.x.y' - n(n + 1).y = 0$$

Le polynôme de Legendre de degré n noté $P(n, x)$ vérifie les relations :
$P(0, x) = 1$
$P(1, x) = x$
$P(n, x) = \frac{2n-1}{n}xP(n - 1, x) - \frac{n-1}{n}P(n - 2, x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$< f, g >= \int_{-1}^{+1} f(x)g(x)dx$
On tape :

```
legendre(4)
```

On obtient :

```
(35*x^4+-30*x^2+3)/8
```

On tape :

```
legendre(4,y)
```

On obtient :

```
(35*y^4+-30*y^2+3)/8
```

6.28.2 Polynôme de Hermite : `hermite`

`hermite` a comme argument un entier n et èventuellement le nom de la variable (x par défaut).
`hermite` renvoie le polynôme de Hermite de degré n.
Le polynôme de Hermite de degré n noté $P(n,x)$ vérifie les relations :
$P(0,x) = 1$
$P(1,x) = 2x$
$P(n,x) = 2xP(n-1,x) - 2(n-1)P(n-2,x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$< f, g >= \int_{-\infty}^{+\infty} f(x)g(x)e^{-x^2}dx$
On tape :

```
hermite(6)
```

On obtient :

```
64*x^6+-480*x^4+720*x^2-120
```

On tape :

```
hermite(6,y)
```

On obtient :

```
64*y^6+-480*y^4+720*y^2-120
```

6.28.3 Polynôme de Laguerre : `laguerre`

`laguerre` a comme argument un entier n et eventuellement le nom de la variable (x par défaut) et du paramètre (a par défaut).
`laguerre` renvoie le polynôme de Laguerre de degré n et de paramètre a.
Le polynôme de Laguerre de degré n de paramètre a noté $L(n,a,x)$ vérifie les relations :
$L(0,a,x) = 1$
$L(1,a,x) = 1 + a - x$
$L(n,a,x) = \frac{2n+a-1-x}{n}L(n-1,a,x) - \frac{n+a-1}{n}L(n-2,a,x)$
Ces polynômes sont orthogonaux pour le produit scalaire :
$< f, g >= \int_0^{+\infty} f(x)g(x)x^a e^{-x}dx$
On tape :

```
laguerre(2)
```

On obtient :

```
(a^2+-2*a*x+3*a+x^2+-4*x+2)/2
```

On tape :

```
laguerre(2,y)
```

On obtient :

```
(a^2+-2*a*y+3*a+y^2+-4*y+2)/2
```

On tape :

```
laguerre(2,y,b)
```

On obtient :

```
1/2*b^2+-b*y+3/2*b+1/2*y^2-2*y+1
```

6.28.4 Polynôme de Tchebychev de 1-ière espèce : `tchebyshev1`

`tchebyshev1` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`tchebyshev1` renvoie le polynôme de Tchebychev de première espèce, de degré n, noté $T(n,x)$.
On a :

$$T(n,x) = \cos(n.\arccos(x))$$

$T(n,x)$ vérifie les relations :
$T(0,x) = 1$
$T(1,x) = x$
$T(n,x) = 2xT(n-1,x) - T(n-2,x)$
Les polynômes $T(n,x)$ sont orthogonaux pour le produit scalaire :
$< f,g >= \int_{-1}^{+1} \frac{f(x)g(x)}{\sqrt{1-x^2}} dx$
On tape :

```
tchebyshev1(4)
```

On obtient :

```
8*x^4+-8*x^2+1
```

On tape :

```
tchebyshev1(4,y)
```

On obtient :

```
8*y^4+-8*y^2+1
```

et on a bien :
$\cos(4.x) = Re((\cos(x) + i.\sin(x))^4)$
$\cos(4.x) = \cos(x)^4 - 6.\cos(x)^2.(1 - \cos(x)^2) + ((1 - \cos(x)^2)^2.$
$\cos(4.x) = T(4, \cos(x))$.

6.28.5 Polynôme de Tchebychev de 2-ième espèce : `tchebyshev2`

`tchebyshev2` a comme argument un entier n et eventuellement le nom de la variable (x par défaut).
`tchebyshev2` renvoie le polynôme de Tchebychev de seconde espèce, de degré n, noté $U(n,x)$.
On a :

$$U(n,x) = \frac{\sin((n+1).\arccos(x))}{\sin(\arccos(x))}$$

ou encore

$$\sin((n+1)x) = \sin(x) * U(n, \cos(x))$$

$U(n,x)$ vérifie les relations :
$U(0,x) = 1$
$U(1,x) = 2x$
$U(n,x) = 2xU(n-1,x) - U(n-2,x)$
Les polynômes $U(n,x)$ sont orthogonaux pour le produit scalaire :
$< f, g >= \int_{-1}^{+1} f(x)g(x)\sqrt{1-x^2}dx$
On tape :

```
tchebyshev2(3)
```

On obtient :

```
8*x^3+-4*x
```

On tape :

```
tchebyshev2(3,y)
```

On obtient :

```
8*y^3+-4*y
```

en effet :
$\sin(4.x) = \sin(x) * (8 * \cos(x)^3 - 4.\cos(x)) = \sin(x) * U(3, \cos(x))$.

6.29 Base et réduction de Gröbner

6.29.1 Base de Gröbner : `gbasis`

`gbasis` a au moins deux arguments : une liste de polynômes de plusieurs variables et la liste du nom de ces variables.
`gbasis` renvoie une base de Gröbner de l'idéal polynomial engendré par les polynômes donnés dans le premier argument.
On choisit d'ordonner les monômes selon l'ordre lexicographique en accord avec la liste donnée par le dernier argument et selon les puissances décroissantes : par exemple on écrira $x^2 * y^4 * z^3$ puis $x^2 * y^3 * z^4$ si le deuxième argument est $[x, y, z]$ car $(2, 4, 3) > (2, 3, 4)$ mais on écrira $x^2 * y^3 z^4$ puis $x^2 * y^4 * z^3$ si le deuxième argument est $[x, z, y]$.
Si I est un idéal et si $(G_k)_{k\in K}$ est une base de Gröbner de l'idéal I alors, si F

est un polynôme non nul de I, le terme dominant de F est divisible par le terme dominant d'un G_k.
Propriété : Si on fait la division euclidienne de F par un des G_k puis, si on recommence avec le reste obtenu et le G_k suivant, on finit par obtenir un reste nul.
On tape :

```
gbasis([2*x*y-y^2,x^2-2*x*y],[x,y])
```

On obtient :

```
[y^3,x*y+(-1/2)*y^2,x^2-y^2]
```

On peut donner des arguments supplémentaires :

- `plex` (lexicographique pur utilisé par défaut)), `tdeg` (degré total puis ordre lexicographique), `revlex` (degré total puis ordre lexicographique inverse), pour spécifier un ordre sur les monômes différent de l'ordre par défaut (qui est `plex`),
- `with_cocoa=true` ou `with_cocoa=false`, si on veut utiliser la librairie `CoCoA` pour faire le calcul de la base de Gröbner.
- `with_f5=true` ou `with_f5=false` pour utiliser l'implémentation de l'algorithme F5 de la librairie `CoCoA`. L'ordre spécifié n'est alors pas utilisé, les polynômes étant homogénéisés.

On tape :

```
gbasis([x1+x2+x3,x1*x2+x1*x3+x2*x3,x1*x2*x3-1],
       [x1,x2,x3],tdeg,with_cocoa=false)
```

On obtient

```
[x3^3-1,-x2^2-x2*x3-x3^2,x1+x2+x3]
```

6.29.2 Réduction par rapport à une base de Gröbner : `greduce`

`greduce` a trois arguments : un polynôme de plusieurs variables, une liste de polynômes formant une base de Gröbner dépendant des mêmes variables et la liste du nom de ces variables.
`greduce` renvoie la réduction (à une constante multiplicative près) du polynôme donné dans le premier argument par rapport à la base de Gröbner donnée dans le deuxième argument.
On tape :

```
greduce(x*y-1,[x^2-y^2,2*x*y-y^2,y^3],[x,y])
```

On obtient :

```
1/2*y^2-1
```

ce qui veut dire que $xy - 1 = \frac{1}{2}(y^2 - 2) \mod I$ où I est l'idéal engendré par la base de Gröbner $[x^2 - y^2, 2xy - y^2, y^3]$, puisque $y^2 - 2$ est le reste de la division euclidienne de $2(xy - 1)$ par $G_2 = 2xy - y^2$.
Remarque

La constante multiplicative peut être déterminnée en regardant comment le coefficient constant est transformé. Dans l'exemple, le terme constant `-1` est transformé en le terme constant `-2`, donc le coefficient multiplicatif est `1/2`.

On peut donner des arguments supplémentaires à `greduce` comme pour `gbasis` (`plex`(par défaut),`tdeg`,`plex`...cf. 6.29.1), c'est d'ailleurs nécessaire si on a calculé une base de Gröbner avec un ordre différent de celui par défaut, dans ce cas `greduce` doit utiliser le même ordre.
On tape :

```
greduce(x1^2*x3^2,[x3^3-1,-x2^2-x2*x3-x3^2,x1+x2+x3],
                        [x1,x2,x3],tdeg)
```

On obtient

```
x2
```

6.29.3 Test d'appartenance d'un polynôme ou d'une liste de polynômes à un idéal donné par une base de Groebner : `in_ideal`

`in_ideal` a trois (ou quatre) arguments : un polynôme ou d'une liste de polynômes, une liste donnant une base de Groebner, la liste des variables des polynômes.
On peut donner des arguments supplémentaires à `in_ideal` comme pour `gbasis` (`plex`(par défaut),`tdeg`,`plex`...cf. 6.29.1), c'est d'ailleurs nécessaire si on a calculé une base de Gröbner avec un ordre différent de celui par défaut (qui est l'ordre lexicographique pur `plex`), dans ce cas `in_ideal` doit utiliser le même ordre.
`in_ideal` teste si le polynôme ou les polynômes du 1ier argument sont dans l'idéal engendré par la base de Groebner,par rapport à une liste de variables et renvoie `vrai` ou `faux` ou une liste composée de `vrai` ou de `faux`.
On tape :

```
in_ideal((x+y)^2,[y^2,x^2+2*x*y],[x,y])
```

On obtient

```
[vrai]
```

On tape :

```
in_ideal([(x+y)^2,x+y],[y^2,x^2+2*x*y],[x,y])
```

On obtient

```
[vrai,faux]
```

On tape :

```
in_ideal(x+y,[y^2,x^2+2*x*y],[x,y])
```

On obtient

```
[faux]
```

On tape :

```
|gbasis([x1+x2+x3,x1*x2+x1*x3+x2*x3,x1*x2*x3-1],|
                    [x1,x2,x3],tdeg)
in_ideal([(x1+x2+x3)^3,x1+x2+x3],[x1+x2+x3,-x2^2-x2*x3-x3^2,x3^3-1],
                    [x1,x2,x3],tdeg)
```

On obtient

```
[vrai,vrai]
```

6.29.4 Construire un polynôme de n variables : `genpoly`

`genpoly` a trois arguments : un polynôme P de $n-1$ variables, un entier b et le nom d'une variable `var`.
`genpoly` renvoie le polynôme Q de n variables (celles de P et celle donnée dans l'argument), construit à partir de P pour avoir :
`subst(Q,var=b)=P` et de plus les coefficients de Q sont dans l'intervalle $]-b/2\ ;\ b/2]$
On tape :

```
genpoly(61,6,x)
```

On obtient :

```
2*x^2-2*x+1
```

En effet, on a : $61 = 6^2+4*6+1$ mais les coefficients des puissances de 6 doivent être dans l'intervalle $]-3\ ;\ 3]$ donc on écrit $61 = 2*6^2-2*6+1$
On tape :

```
genpoly(5,6,x)
```

On obtient :

```
x-1
```

En effet : $5 = 6-1$
On tape :

```
genpoly(7,6,x)
```

On obtient :

```
x+1
```

En effet : $7 = 6+1$
On tape :

```
genpoly(7*y+5,6,x)
```

On obtient :

```
x*y+x+y-1
```

En effet : $x*y+x+y-1 = y(x+1)+(x-1)$
On tape :

```
genpoly(7*y+5*z^2,6,x)
```

On obtient :

```
x*y+x*z+y-z
```

En effet : $x*y+x*z+y-z = y*(x+1)+z*(x-1)$

6.30 Les fractions rationnelles

6.30.1 Numérateur : `getNum`

`getNum` a comme argument une fraction rationnelle et renvoie le numérateur de cette fraction non simplifiée.
On tape :

```
getNum((x^2-1)/(x-1))
```

On obtient :

```
x^2-1
```

On tape :

```
getNum((x^2+2*x+1)/(x^2-1))
```

On obtient :)

```
x^2+2*x+1
```

6.30.2 Numérateur après simplification : `numer`

`numer` a comme argument une fraction rationnelle et renvoie le numérateur de cette fraction simplifiée (voir aussi 6.9.3).
On tape :

```
numer((x^2-1)/(x-1))
```

On obtient :

```
x+1
```

On tape :

```
numer((x^2+2*x+1)/(x^2-1))
```

On obtient :

```
x+1
```

6.30.3 Dénominateur : `getDenom`

`getDenom` a comme argument une fraction rationnelle et renvoie le dénominateur de cette fraction non simplifiée.
On tape :

```
getDenom((x^2-1)/(x-1))
```

On obtient :

```
x-1
```

On tape :

```
getDenom((x^2+2*x+1)/(x^2-1))
```

On obtient :

```
x^2-1
```

6.30.4 Dénominateur après simplification : denom

denom (ou getDenom) a comme argument une fraction rationnelle et renvoie le dénominateur de cette fraction simplifiée (voir aussi 6.9.4).
On tape :

```
denom((x^2-1)/(x-1))
```

On obtient :

```
1
```

On tape :

```
denom((x^2+2*x+1)/(x^2-1))
```

On obtient :

```
x-1
```

6.30.5 Numérateur et dénominateur : f2nd fxnd

f2nd (ou fxnd) a comme argument une fraction rationnelle et renvoie la liste formée par le numérateur et le dénominateur de cette fraction simplifiée (voir aussi 6.9.5).
On tape :

```
f2nd((x^2-1)/(x-1))
```

On obtient :

```
[x+1,1]
```

On tape :

```
f2nd((x^2+2*x+1)/(x^2-1))
```

On obtient :

```
[x+1,x-1]
```

6.30.6 Simplifier : simp2

simp2 a comme paramètre deux polynômes (ou deux entiers voir 6.9.6). Ces deux polynômes sont considèrés comme représentant une fraction rationnelle.
simp2 renvoie la fraction rationnelle simplifiée sous la forme d'une liste de deux polynômes.
On tape :

```
simp2(x^3-1,x^2-1)
```

On obtient :

```
[x^2+x+1,x+1]
```

6.30.7 Réduire au même dénominateur : `comDenom`

`comDenom` a comme paramètre une somme de fractions rationnelles.
`comDenom` renvoie cette somme sous la forme d'une fraction rationnelle c'est à dire renvoie cette somme après réduction au même dénominateur des fractions rationnelles la composant.
On tape :

```
comDenom(x-1/(x-1)-1/(x^2-1))
```

On obtient :

```
(x^3+-2*x-2)/(x^2-1)
```

6.30.8 Partie entière et fractionnaire : `propfrac`

`propfrac` a comme argument une fraction rationnelle.
`propfrac` renvoie cette fraction rationnelle écrite de manière à mettre en évidence sa partie entière.
`propfrac(A(x)/B(x))` écrit la fraction rationnelle $\frac{A(x)}{B(x)}$ après simplification sous la forme :

$$Q(x) + \frac{R(x)}{B(x)}$$

avec $R(x) = 0$ ou $0 \leq \text{degree}(R(x)) < \text{degree}(B(x))$.
On tape :

```
propfrac((5*x+3)*(x-1)/(x+2))
```

On obtient :

```
5*x-12+21/(x+2)
```

6.30.9 Décomposition en éléments simples : `partfrac =>+`

`=>+` est la version postfixée de `partfrac` `partfrac` a comme argument une fraction rationnelle.
`partfrac` renvoie sa décomposition en éléments simples.
Exemple :
Décomposer en éléments simples la fraction rationnelle :

$$\frac{x^5 - 2x^3 + 1}{x^4 - 2x^3 + 2x^2 - 2x + 1}$$

On utilise la commande `partfrac` ou `=>+`.
On peut aussi utiliser `convert` avec l'option `parfrac` ou `partfrac` ou `fullparfrac` (voir aussi 6.23.26).
On tape :

```
(x^5-2*x^3+1)/(x^4-2*x^3+2*x^2-2*x+1)=>+
```

Ou on tape :

```
partfrac((x^5-2*x^3+1)/(x^4-2*x^3+2*x^2-2*x+1))
```

On obtient en mode réel :

```
x+2-1/(2*(x-1))+(x-3)/(2*(x^2+1))
```

On obtient en mode complexe :

```
x+2+1/((x-1)*-2)+(1-3*i)/((x+i)*4)+(1+3*i)/((x-i)*4)
```

6.30.10 Décomposition en éléments simples sur C : `cpartfrac`

`cpartfrac` a comme argument une fraction rationnelle.
`cpartfrac` renvoie sa décomposition en éléments simples sur C que l'on soit en mode réel ou complexe.
Exemple :
Décomposer en éléments simples la fraction rationnelle :

$$\frac{x^5-2x^3+1}{x^4-2x^3+2x^2-2x+1}$$

On utilise la commande `cpartfrac`.
On tape :

```
cpartfrac((x^5-2*x^3+1)/(x^4-2*x^3+2*x^2-2*x+1))
```

On obtient en mode réel ou en mode complexe :

```
x+2+(-1+2*i)/((2-2*i)*((i)*x+1))+1/(2*(-x+1))+

(-1-2*i)/((2-2*i)*(x+i))
```

6.31 Racines exactes d'un polynôme

6.31.1 Encadrement exact des racines complexes d'un polynôme : `complexroot`

`complexroot` a 2 ou 4 arguments : un polynôme et un nombre rèel ϵ et éventuellement deux complexes α, β.

- Si `complexroot` a 2 arguments, `complexroot` renvoie la liste des intervalles complexes contenant la valeur des racines complexes et exactes du polynôme et leur multiplicité (par exemple i[1,1.1]+i[3.1,3.2]*i pour dire que le rectangle [1,1.1]x[3.1,3.2] contient une racine complexe du polynôme) et la multiplicité de cette racine.
  Si l'intervalle est $[a_1 + ib_1, a_2 + ib_2]$ on a $|a_1 - a_2| < \epsilon$ et $|b_1 - b_2| < \epsilon$ et la racine $a + ib$ vérifie $a_1 \leq a \leq a_2$ et $b_1 \leq b \leq b_2$.
- Si `complexroot` a 4 arguments, `complexroot` ne renvoie que les racines situées dans le rectangle de côtés parallèles aux axes et de sommets opposés α, β.

On tape pour avoir les racines de $x^3 + 1$:

```
complexroot(x^3+1,0.1)
```

On obtient :

[[i[-1.0000000000036,-0.99999999999633],1],
[i[0.49999999999909,0.50000000000091] -
i[0.86602540378354,0.86602540378536]*i,1],[i[0.49999999999909,0.50000000000091]
+i[0.86602540378354,0.86602540378536]*i,1]]

Donc pour $x^3 + 1$:
-1 est une racine de multiplicité 1,
1/2i*b est une racine de multiplicité 1 avec $-7/8 \leq b \leq -13/16$,
1/2i*c est racine de multiplicité1 avec $13/1 \leq c \leq 7/8$.
On tape pour avoir les racines de $x^3 + 1$ dans le rectangle de sommets opposés $-1, 1 + 2 * i$:

```
complexroot(x^3+1,0.1,-1,1+2*i)
```

On obtient :

[[i[-1.0000000000036,-0.99999999999633],1],
[i[0.49999999999909,0.50000000000091]
+i[0.86602540378354,0.86602540378536]*i,1]]

On tape pour avoir les racines de $x^3 + 1$ dans le rectangle de sommets opposés $0, 1 + 2 * i$:

```
complexroot(x^3+1,0.1,0,1+2*i)
```

On obtient :

[[i[0.49999999999909,0.50000000000091]
+i[0.86602540378354,0.86602540378536]*i,1]]

6.31.2 Encadrement exact des racines réelles d'un polynôme avec leur multiplicité : `realroot`

`realroot` a 1 ou 4 arguments : un polynôme et un nombre rèel l et éventuellement deux réels a, b et cherche les racines rèelles du polynôme en utilisant l'algorithme de Vincent-Akritas-Strzebonski (VAS).
On peut aussi utiliser un argument supplémentaire pour dire que l'algorithme utilise les suites de Sturm : on met alors `sturm` comme premier argument. Mais les résultats seront moins précis.
`realroot(P,l,a,b)` et `realroot(sturm,P,l,a,b)` renvoient la liste des intervalles de longueur <=`l` où se trouvent les racines réelles de `P` situées dans `a..b` avec leur multiplicité. `realroot(P,l,a,b)` utilise l'algorithme de Vincent-Akritas-Strzebonski (VAS) alors que `realroot(sturm,P,l,a,b)` utilise les suites de Sturm qui sont moins efficaces.

– Si `realroot` a 1 argument (resp 2 arguments), `realroot(P)` (resp `realroot(sturm,P)`) renvoie la liste des intervalles où se trouvent les racines réelles de `P` avec leur multiplicité en utilisant l'algorithme de Vincent-Akritas-Strzebonski (VAS) (resp en utilisant les suites de Sturm). L'algorithme s'arrête dès qu'il a trouvé un intervalle qui contient une racine.
On tape :

`realroot(x^3-7*x+7)`
On obtient :
`[[[-4,0],1],[[1,3/2],1],[[3/2,2],1]]`
Ce qui veut dire qu'il y a une racine dans l'intervalle [-4,0] qui est de multiplicité 1, une racine dans l'intervalle [1,3/2] qui est de multiplicité 1 et une racine dans l'intervalle [3/2,2] qui est de multiplicité 1.
On tape :
`realroot(x^5+2*x^4-6*x^3-7*x^2+7*x+7)`
On obtient :
`[[[-5,-1],1],[-1,2],[[1,3/2],1],[[3/2,2],1]]`
Ce qui veut dire que -1 est une racine de multiplicité 2, les 3 autres racines sont de multiplicité 1 et se trouve dans les intervalles [-5,-1],[1,3/2],[3/2,2].
On tape :
`realroot(125*x^3-700*x^2+1225*x-686)`
On obtient :
`[[[1,2],2],[[2,4],1]]`
Ce qui veut dire qu'il y a une racine de multiplicité 2 dans l'intervalle [1,2] et une racine de multiplicité 1 dans l'intervalle [2,4].

– Si `realroot` a 2 arguments (resp 3 arguments), `realroot(P,l)` (resp `realroot(sturm,P,l)`) renvoie la liste des vecteurs de coordonnées la valeur des racines réelles et exactes du polynôme et leur multiplicité ou des vecteurs de coordonnées un intervalle contenant une racine réelle du polynôme et la multiplicité de cette racine.
  Si l'intervalle renvoyé est $[a_1, a_2]$ on a $|a_1 - a_2| < l$ et la racine x_1 vérifie $a_1 \leq x_1 \leq a_2$. On tape :
  `realroot(x^5+2*x^4-6*x^3-7*x^2+7*x+7,1e-5)`
  On obtient :
  ```
  [[[-1598513/524288,-1598509/524288],1],[-1,2],
  [[2845609/2097152,2845625/2097152],1],
  [[3548417/2097152,3548433/2097152],1]]
  ```
  ce qui donne en valeurs approchèes :
  ```
  [[[-3.04892158508,-3.04891395569],1.0],[-1.0,2.0],
  [[1.35689210892,1.35689973831],1.0],
  [[1.6920170784,1.69202470779],1.0]]
  ```
  On tape :
  `realroot(sturm,x^5+2*x^4-6*x^3-7*x^2+7*x+7,1e-5)`
  On obtient :
  ```
  [[[-99907/32768,-399627/131072],1],
  [[177851/131072,44463/32768],1],
  [[13861/8192,221777/131072],1],[-1,2]]
  ```
  ce qui donne en valeurs approchèes :
  ```
  [[[-3.04891967773,-3.04891204834],1.0],
  [[1.35689544678,1.35690307617],1.0],
  [[1.69201660156,1.69202423096],1.0],[-1.0,2.0]]
  ```
  On tape :
  `realroot(x^3-7*x+7,1e-5)`
  On obtient :
  ```
  [[[-1598513/524288,-1598509/524288],1], [[2845609/2097152,2845625/
  [[3548417/2097152,3548433/2097152],1]]
  ```

ce qui donne en valeurs approchèes :
`[[[-3.04892158508,-3.04891395569],1.0],`
`[[1.35689210892,1.35689973831],1.0],`
`[[1.6920170784,1.69202470779],1.0]]`

– Si `realroot` a 4 arguments (resp 5 arguments), `realroot(P,l,a,b)` (resp `realroot(sturm,P,l,a,b)`) ne renvoie que les racines situées dans l'intervalle $[a, b]$.

On tape pour avoir les racines réelles de $x^3 + 1$:

```
realroot(x^3+1, 0.1)
```

On obtient :

```
[[-1,1]]
```

On tape pour avoir les racines réelles de $x^3 - x^2 - 2x + 2$:

```
realroot(x^3-x^2-2*x+2, 0.1)
```

On obtient :

```
[[[-3/2,-45/32],1],[1,1],[[11/8,23/16],1]]
```

ou bien

```
realroot(sturm,x^3-x^2-2*x+2, 0.1)
```

On obtient :

```
[[1,1],[[(-3)/2,(-45)/32],1],[[45/32,3/2],1]]
```

On tape pour avoir les racines réelles de $x^3 - x^2 - 2x + 2$ dans l'intervalle $[0; 2]$:

```
realroot(x^3-x^2-2*x+2, 0.1,0,2)
```

ou

```
realroot(sturm,x^3-x^2-2*x+2, 0.1,0,2)
```

On obtient :

```
[[1,1],[[11/8,23/16],1]]
```

6.31.3 Encadrement exact des racines réelles d'un polynôme : `VAS`

`VAS(P)` renvoie une liste d'intervalles d'isolation des racines réelles de`P` par l'algorithme de Vincent-Akritas-Strzebonski. Cela ne fait que calculer des intervalles d'isolation des racines.
On tape :
`VAS(x^3-7*x+7)`
On obtient :
`[[-4,0],[1,3/2],[3/2,2]]`
On tape :

`VAS(x^5+2*x^4-6*x^3-7*x^2+7*x+7)`
On obtient :
`[[-5,-1],-1,[1,3/2],[3/2,2]]`
On tape :
`VAS(x^3-x^2-2*x+2)`
On obtient :
`[[-3,0],1,[1,3]]`

6.31.4 Encadrement exact des racines réelles positives d'un polynôme : VAS_positive

`VAS_positive(P)` renvoie une liste d'intervalles d'isolation des racines réelles positives de `P` par l'algorithme de Vincent-Akritas-Strzebonski. Cela ne fait que calculer des intervalles d'isolation des racines.
On tape :
`VAS_positive(x^3-7*x+7)`
On obtient :
`[[1,3/2],[3/2,2]]`
On tape :
`VAS_positive(x^5+2*x^4-6*x^3-7*x^2+7*x+7)`
On obtient :
`[[1,3/2],[3/2,2]]`
On tape :
`VAS_positive(x^3-x^2-2*x+2)`
On obtient :
`[1,[1,3]]`

6.31.5 Borne supérieure des racines réelles positives d'un polynôme : posubLMQ

`posubLMQ(P)` renvoie une borne supérieure pour les racines positives de `P` par l'algorithme Akritas-Strzebonski-Vigklas' Local Max Quadratic (LMQ). Cette borne n'est pas optimale.
On tape :
`posubLMQ(x^3-7*x+7)`
On obtient :
`4`
On tape :
`posubLMQ(x^5+2*x^4-6*x^3-7*x^2+7*x+7)`
On obtient :
`4`
On tape :
`posubLMQ(x^3-x^2-2*x+2)`
On obtient :
`3`

6.31.6 Borne inférieure des racines réelles positives d'un polynôme : poslbdLMQ

`poslbdLMQ(P)` renvoie une borne inférieure pour les racines positives de `P` par l'algorithme Akritas-Strzebonski-Vigklas' Local Max Quadratic (LMQ). Cette borne n'est pas optimale.
On tape :
`poslbdLMQ(x^3-7*x+7)`
On obtient :
`1/2`
On tape :
`poslbdLMQ(x^5+2*x^4-6*x^3-7*x^2+7*x+7)`
On obtient :
`1/2`
On tape :
`poslbdLMQ((x^3-x^2-2*x+2)`
On obtient :
`1/2`

6.31.7 Valeurs exactes des racines rationnelles d'un polynôme : rationalroot

`rationalroot` a 1 ou 3 arguments : un polynôme et éventuellement deux réels α, β.

- Si `rationalroot` a 1 argument, `rationalroot` renvoie la liste formée par la valeur des racines rationnelles du polynôme sans indiquer la multiplicité de ces racines.
- Si `rationalroot` a 3 arguments, `rationalroot` ne renvoie que les racines rationnelles situées dans l'intervalle $[\alpha, \beta]$.

On tape pour avoir les racines rationnelles de $2 * x^3 - 3 * x^2 - 8 * x + 12$:

```
rationalroot(2*x^3-3*x^2-8*x+12)
```

On obtient :

```
[2,3/2,-2]
```

On tape pour avoir les racines rationnelles de $2 * x^3 - 3 * x^2 - 8 * x + 12$ dans $[1; 2]$:

```
rationalroot(2*x^3-3*x^2-8*x+12,1,2)
```

On obtient :

```
[2,3/2]
```

On tape pour avoir les racines rationnelles de $2 * x^3 - 3 * x^2 + 8 * x - 12$:

```
rationalroot(2*x^3-3*x^2+8*x-12)
```

On obtient :

```
[3/2]
```

On tape pour avoir les racines rationnelles de $2*x^3-3*x^2+8*x-12$:

```
rationalroot(2*x^3-3*x^2+8*x-12)
```

On obtient :

```
[3/2]
```

On tape pour avoir les racines rationnelles de $(3*x-2)^2*(2x+1)) = 18*x^3-15*x^2-4*x+4$:

```
rationalroot(18*x^3-15*x^2-4*x+4)
```

On obtient :

```
[(-1)/2,2/3]
```

6.31.8 Valeurs exactes des racines complexes rationnelles d'un polynôme : `crationalroot`

`crationalroot` a 1 ou 3 arguments : un polynôme et éventuellement deux complexes α, β.

- Si `crationalroot` a 1 argument, `crationalroot` renvoie la liste des valeurs des racines complexes rationnelles du polynôme sans indiquer la multiplicité de ces racines.
- Si `crationalroot` a 3 arguments, `crationalroot` ne renvoie que les racines complexes rationnelles situées dans le rectangle de sommets opposés $[\alpha, \beta]$.

On tape pour avoir les racines rationnelles et complexes de $(x^2+4)*(2x-3) = 2*x^3-3*x^2+8*x-12$:

```
crationalroot(2*x^3-3*x^2+8*x-12)
```

On obtient :

```
[2*i,3/2,-2*i]
```

6.32 Fraction rationnelle, ses racines et ses pôles exacts

6.32.1 Racines et pôles exacts d'une fraction rationnelle : `froot`

`froot` a comme argument une fraction rationnelle $F(x)$.
`froot` renvoie un vecteur de composantes les racines et les pôles de $F(x)$ suivis de leur multiplicité.
`Xcas` renvoie les valeurs exactes de ces racines ou pôles quand cela est possible et sinon renvoie leur valeurs numériques.
On tape :

```
froot((x^5-2*x^4+x^3)/(x-2))
```

On obtient :

```
[1,2,0,3,2,-1]
```

donc pour $F(x) = \dfrac{x^5 - 2.x^4 + x^3}{x-2}$:
1 est racine double,
0 est racine triple et
2 est un pôle d'ordre 1.
On tape :

```
froot((x^3-2*x^2+1)/(x-2))
```

On obtient :

```
[1,1,(1+sqrt(5))/2,1,(1-sqrt(5))/2,1,2,-1]
```

Remarque : pour avoir les racines et les pôles complexes il faut avoir coché `Complexe` dans la configuration du `cas` (bouton donnant la ligne d'état).
On tape :

```
froot((x^2+1)/(x-2))
```

On obtient :

```
[-i,1,i,1,2,-1]
```

6.32.2 Coefficients d'une fraction rationnelle définie par ses racines et ses pôles : `fcoeff`

`fcoeff` a comme argument un vecteur de composantes les racines et les pôles d'une fraction rationnelle $F(x)$ suivis de leur multiplicité.
`fcoeff` renvoie la fraction rationnelle $F(x)$.
On tape :

```
fcoeff([1,2,0,3,2,-1])
```

On obtient :

```
(x-1)^2*x^3/(x-2)
```

6.33 Le calcul modulaire dans $\mathbb{Z}/p\mathbb{Z}$ ou dans $\mathbb{Z}/p\mathbb{Z}[x]$

On peut faire des calculs modulo `p` c'est à dire dans $\mathbb{Z}/p\mathbb{Z}$ ou dans $\mathbb{Z}/p\mathbb{Z}[x]$ et la façon de s'y prendre dépends de la syntaxe choisie :

- En mode `Xcas`, les nombres n de $\mathbb{Z}/p\mathbb{Z}$ sont notés $n\%\ p$.
  Exemples de notation
  - un entier `n` de $\mathbb{Z}/13\mathbb{Z}$
    `n:=12% 13`.
  - un vecteur `V` de coordonnées dans $\mathbb{Z}/13\mathbb{Z}$
    `V:=[1,2,3]% 13` ou `V:=[1% 13,2% 13,3% 13]`.
  - une matrice `A` de coefficients dans $\mathbb{Z}/13\mathbb{Z}$
    `A:=[[1,2,3],[2,3,4]]% 13` ou
    `A:=[[1% 13,2% 13,3% 13],[2% 13,3% 13,4% 13]]`.
  - un polynôme `A` de $\mathbb{Z}/13\mathbb{Z}[x]$ en représentation symbolique
    `A:=(2*x^2+3*x-1)%13` ou
    `A:=2%13*x^2+3%13*x-1%13`.

  - un polynôme `A` de $\mathbb{Z}/13\mathbb{Z}[x]$ en représenté avec une liste `A:=poly1[1,2,3]%13` ou `A:=poly1[1%13,2%13,3%13]`.

  Pour transformer un objet `o` à coefficients modulaires en un objetà coefficients entiers tapez `o % 0`. Par exemple, si on tape `o:=4% 7` puis `o% 0`, on obtient `-3`.
- En mode `Maple`, on n'utilise pas % pour représenter les entiers modulo p : ces entiers sont représentés comme les entiers usuels et pour eviter les confusions on utilise les commandes commençant par une majuscule que l'on fait suivre par la commande mod (on se repotera à la section suivante).

Remarques

- Pour certaines commandes dans $\mathbb{Z}/p\mathbb{Z}$ ou dans $\mathbb{Z}/p\mathbb{Z}[x]$, il faut choisir un nombre `p` premier.

- La représentation choisie est la représentation symétrique : `11%13 = -2%13`.

6.33.1 Développer et réduire : `normal`

`normal` a comme argument une expression polynomiale.
`normal` développe et réduit cette expression dans $Z/pZ[x]$.
On tape :

```
normal(((2*x^2+12)*( 5*x-4))% 13)
```

On obtient :

```
(-3% 13)*x^3+(5% 13)*x^2+(-5%13)*x+4% 13
```

6.33.2 Addition dans $\mathbb{Z}/p\mathbb{Z}$ ou dans $\mathbb{Z}/p\mathbb{Z}[x]$: +

Pour réaliser une addition dans $\mathbb{Z}/p\mathbb{Z}$, on utilise le + habituel et, pour les polynômes de $\mathbb{Z}/p\mathbb{Z}[x]$, on utilise le + habituel et la commande `normal` pour simplifier.
Pour les entiers dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
3% 13+10% 13
```

On obtient :

```
0%13
```

Pour les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
normal(11% 13*x+5% 13+8% 13*x+6% 13)
```

ou encore

```
normal((11*x+5 )% 13+(8*x+6)% 13)
```

On obtient :

```
(6%13)*x+-2% 13
```

6.33.3 Soustraction dans $\mathbb{Z}/p\mathbb{Z}$ ou $\mathbb{Z}/p\mathbb{Z}[x]$: -

Pour réaliser une soustraction dans $\mathbb{Z}/p\mathbb{Z}$, on utilise le $-$ habituel et, pour les polynômes de $\mathbb{Z}/p\mathbb{Z}[x]$, on utilise le $-$ habituel et la commande `normal` pour simplifier.
Pour les entiers dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
31% 13-10% 13
```

On obtient :

```
-5% 13
```

Pour les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
normal(11% 13*x+5% 13-8% 13*x+6% 13)
```

ou encore

```
normal((11*x+5)% 13-(8*x+6)% 13)
```

On obtient :

```
(3% 13)*x+-1% 13
```

6.33.4 Multiplication dans $\mathbb{Z}/p\mathbb{Z}$ ou $\mathbb{Z}/p\mathbb{Z}[x]$: $\star$

Pour réaliser une multiplication dans $\mathbb{Z}/p\mathbb{Z}$, on utilise le $\star$ habituel et, pour les polynômes de $\mathbb{Z}/p\mathbb{Z}[x]$, on utilise le $\star$ habituel puis la commande `normal` pour simplifier.
Pour les entiers dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
31% 13*10% 13
```

On obtient :

```
-2% 13
```

Pour les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
normal((11% 13*x+5% 13)*(8% 13*x+6% 13))
```

ou encore on tape :

```
normal((11*x+5)% 13*(8*x+6 )% 13)
```

On obtient :

```
(-3% 13)*x^2+(2% 13)*x+4% 13
```

6.33.5 Quotient : `quo`

`quo` a comme arguments deux polynômes A et B à coefficients dans $\mathbb{Z}/p\mathbb{Z}$. A et B peuvent être donnés par une expression polynômiale symbolique (de x ou du nom de variable donné comme troisième argument) ou par la liste de leur coefficients.
`quo` renvoie le quotient de la division euclidienne de A par B dans $\mathbb{Z}/p\mathbb{Z}[x]$.
On tape :

```
quo((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Ou on tape :

```
quo((x^3+x^2+1,2*x^2+4)% 13)
```

On obtient :

```
(-6% 13)*x+-6% 13
```

en effet $x^3 + x^2 + 1 = (2x^2 + 4)(\frac{x+1}{2}) + \frac{5x-4}{4}$
et que $-3 * 4 = -6 * 2 = 1 \mod 13$

6.33.6 Remainder : `rem`

`rem` a comme arguments deux polynômes A et B à coefficients dans $\mathbb{Z}/p\mathbb{Z}$. A et B peuvent être donnés par une expression polynômiale symbolique (de x ou du nom de variable donné comme troisième argument) ou par la liste de leur coefficients.
`rem` renvoie le reste de la division euclidienne de A par B dans $\mathbb{Z}/p\mathbb{Z}[x]$.
On tape :

```
rem((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Ou on tape :

```
rem((x^3+x^2+1,2*x^2+4)% 13)
```

On obtient :

```
(-2% 13)*x+-1% 13
```

en effet $x^3 + x^2 + 1 = (2x^2 + 4)(\frac{x+1}{2}) + \frac{5x-4}{4}$
et que $-3 * 4 = -6 * 2 = 1 \mod 13$

6.33.7 Quotient and remainder : `quorem`

`quorem` a comme arguments deux polynômes A et B à coefficients dans $\mathbb{Z}/p\mathbb{Z}$. A et B peuvent être donnés par une expression polynômiale symbolique (de x ou du nom de variable donné comme troisième argument) ou par la liste de leur coefficients.
`quorem` renvoie la liste du quotient et du reste de la division euclidienne de A par B dans $\mathbb{Z}/p\mathbb{Z}[x]$ (voir aussi 6.7.13 et 6.27.6).
On tape :

```
quorem(5% 13,2% 13)
```

Ou on tape :

```
quorem((5,2)% 13)
```

et puisque $2 * -4 = 5 - 13$
On obtient :

```
[-4% 13,0]
```

On tape :

```
quorem((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

Ou on tape :

```
quorem((x^3+x^2+1,2*x^2+4)% 13)
```

puisque $x^3 + x^2 + 1 = (2x^2 + 4)(\frac{x+1}{2}) + \frac{5x-4}{4}$
et que $-3 * 4 = -6 * 2 = 1 \mod 13$
On obtient :

```
[(-6% 13)*x+-6% 13,(-2% 13)*x+-1% 13]
```

6.33.8 Division dans $\mathbb{Z}/p\mathbb{Z}$ ou $\mathbb{Z}/p\mathbb{Z}[x]$: /

/ divise deux entiers dans $\mathbb{Z}/p\mathbb{Z}$, ou divise deux polynômes A et B dans $\mathbb{Z}/p\mathbb{Z}[x]$.
Pour les polynômes, le résultat est la fraction rationnelle $\frac{A}{B}$ simplifiée dans $\mathbb{Z}/p\mathbb{Z}[x]$.
Pour les entiers dans $\mathbb{Z}/p\mathbb{Z}$, on tape :

```
5% 13/2% 13
```

On obtient :

```
-4% 13
```

puisque 2 est inversible dans $\mathbb{Z}/13\mathbb{Z}$.
Pour les polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
On tape :

```
(2*x^2+5)% 13/(5*x^2+2*x-3)% 13
```

On obtient :

```
((6% 13)*x+1% 13)/((2% 13)*x+2% 13)
```

6.33.9 Puissance dans $\mathbb{Z}/p\mathbb{Z}$ et dans $\mathbb{Z}/p\mathbb{Z}[x]$: ^

Pour calculer `a` à la puissance `n` dans $\mathbb{Z}/p\mathbb{Z}$ on utilise l'opérateur `^`.
On tape :

```
(5% 13)^2)
```

On obtient :

```
-1% 13
```

Pour calculer `A` à la puissance `n` dans $\mathbb{Z}/p\mathbb{Z}[x]$ on utilise l'opérateur `^` et la commande `normal`.
On tape :

```
normal(((2*x+1)% 13)^5)
```

On obtient :

```
(6% 13)*x^5+(2% 13)*x^4+(2% 13)*x^3+(1% 13)*x^2+(-3%
13)*x+1% 13
```

car :
$10 = -3 \pmod{13}$ $40 = 1 \pmod{13}$ $80 = 2 \pmod{13}$ $32 = 6 \pmod{13}$.

6.33.10 Calcul de $a^n \mod p$ ou de $A(x)^n \mod ¶(x), p$: `powmod powermod`

- Pour calculer dans $[0; p-1]$ $a^n \mod p$ on utilise la commande `powmod` ou `powermod` avec comme argument a, n, p. On tape :

  ```
  powmod(5,21,13)
  ```

  On obtient :

  ```
  5
  ```

  On tape :

  ```
  powmod(5,21,8)
  ```

  On obtient :

  ```
  5
  ```

- Pour calculer $A(x)^n \mod ¶(x), p$ avec en réponse un polynôme à coefficients dans $\mathbb{Z}$ (qui seront des restes symétriques de division par p), on utilise la commande `powmod` ou `powermod` avec comme argument $A(x), n, p, P(x)$.

On tape :

```
powmod(x+1,17,5,x^4+x+1)
```

On obtient :

```
-x^3-x^2
```

On a en effet :

```
rem((x+1)^17,x^4+x+1)
```

qui renvoie :
`29144*x^3+36519*x^2+12270*x-4185` et
`(29144*x^3+36519*x^2+12270*x-4185)% 5` qui renvoie :
`(-1 % 5)*x^3+(-1 % 5)*x^2`
et
`((-1 % 5)*x^3+(-1 % 5)*x^2)% 0`
qui renvoie :
`-x^3-x^2`
Remarque (cf section 6.33.9)
Si on peut calculer une puissance dans $\mathbb{Z}/p\mathbb{Z}$ on tape par exemple :
:

```
(5% 13)^21)
```

On obtient :

```
5% 13
```

On tape :

```
(5% 8)^21)
```

On obtient :

```
-3% 8
```

6.33.11 Inverse dans $\mathbb{Z}/p\mathbb{Z}$: `inv` ou `/`

On calcule l'inverse d'un entier `n` dans $\mathbb{Z}/p\mathbb{Z}$ en tapant `1/n% p` ou `inv(n% p)` ou `inverse(n% p)`.
On tape :

```
inv(3% 13)
```

On obtient :

```
-4% 13
```

En effet : $3 \times -4 = -12 = 1 \pmod{13}$

6.33.12 Transformer un entier en sa fraction modulo p : `fracmod` `iratrecon`

`fracmod` (ou `iratrecon` pour compatibilité Maple) a deux arguments, un entier `n` (ou une expression entière) et un nombre entier `p`.
`fracmod` renvoie une fraction `a/b` vérifiant :

$$-\frac{\sqrt{p}}{2} < a \leq \frac{\sqrt{p}}{2}, \quad 0 \leq b < \frac{\sqrt{p}}{2}, \quad n \times b = a \pmod{p}$$

En d'autres termes $n = a/b \pmod{p}$.
On tape :

```
fracmod(3,13)
```

On obtient :

```
-1/4
```

En effet on a : `-1/4 mod 13` renvoie `3 % 13` i.e.
$3 * -4 = -12 = 1 \pmod{13}$ donc $3\%\ 13 = -1/4\%\ 13$. On tape :

```
fracmod(13,121)
```

On obtient :

```
-4/9
```

En effet on a : `-4/9 mod 121` renvoie `13 % 121` i.e.
: $13 \times -9 = -117 = 4 \pmod{121}$ donc $13\%\ 121 = -4/9\%\ 121$.

6.33.13 PGCD dans $\mathbb{Z}/p\mathbb{Z}[x]$: gcd

Lorsque `gcd` a deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ comme arguments (p doit être premier).
`gcd` calcule le PGCD des deux polynômes dans $\mathbb{Z}/p\mathbb{Z}[x]$ (voir aussi 6.27.7 pour les polynômes à coefficients non modulaires).
On tape :

```
gcd((2*x^2+5)% 13,(5*x^2+2*x-3)% 13)
```

On obtient :

```
(-4% 13)*x+5% 13
```

On tape :

```
gcd(x^2+2*x+1,x^2-1) mod 5
```

On obtient :

```
1
```

Mais si on tape :

```
gcd((x^2+2*x+1,x^2-1)) mod 5)
```

`gcd` est calculé dans $\mathbb{Z}[X]$ puis le calcul modulaire est effectué, on obtient :

```
x% 5
```

6.33.14 Factorisation dans $\mathbb{Z}/p\mathbb{Z}[x]$: factor factoriser

`factor` a comme argument un polynôme à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
`factor` factorise ce polynôme dans $\mathbb{Z}/p\mathbb{Z}[x]$ (p doit être premier).
On tape :

```
factor((-3*x^3+5*x^2-5*x+4)% 13)
```

On obtient :

```
((1% 13)*x+-6% 13)*((-3% 13)*x^2+-5% 13)
```

6.33.15 Déterminant d'une matrice de $\mathbb{Z}/p\mathbb{Z}$: `det`

`det` a comme argument une matrice A à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
`det` renvoie le déterminant de cette matrice A.
On tape :

```
det([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Ou on tape :

```
det([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

On obtient :

```
5% 13
```

donc, dans $\mathbb{Z}/13\mathbb{Z}$, le déterminant de la matrice $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ est `5% 13` (on a `det(A)=31`).

6.33.16 Inverse d'une matrice de $\mathbb{Z}/p\mathbb{Z}$: `inv inverse`

`inverse` (ou `inv`) a comme argument une matrice A à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
`inverse` (ou `inv`) renvoie, l'inverse de la matrice A dans $\mathbb{Z}/p\mathbb{Z}$.
On tape :

```
inverse([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Ou on tape :

```
inv([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Ou on tape :

```
inverse([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

Ou on tape :

```
inv([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

On obtient :

```
[[2% 13,-4% 13,-5% 13],[2% 13,0% 13,-5% 13], [-2%
13,-1% 13,6% 13]]
```

c'est l'inverse de la matrice $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ dans $Z/13Z$.

6.33.17 Résolution d'un système linéaire de $\mathbb{Z}/p\mathbb{Z}$: `rref`

`rref` permet de résoudre, dans $\mathbb{Z}/p\mathbb{Z}$, un système d'équations linéaires de la forme : $Ax = B$ (voir aussi 6.55.3).
L'argument est une matrice formée par A bordée avec B comme dernier vecteur colonne. Le résultat est une matrice formée de A_1 et de B_1 où, A_1 a des zéros de part et d'autre de la diagonale et où, le système $A_1x = B_1$ est équivalent à $Ax = B$.
Résoudre dans $Z/13Z$

$$\left\{\begin{array}{rcl} x + 2\cdot y &=& 9 \\ 3\cdot x + 10\cdot y &=& 0 \end{array}\right.$$

On tape :

```
rref([[1, 2, 9]% 13,[3,10,0]% 13])
```

Ou on tape :

```
rref([[1, 2, 9],[3,10,0]])%13
```

On obtient :

```
[[1% 13,0% 13,3% 13],[0% 13,1% 13,3% 13]]
```

ce qui veut dire que `x=3% 13` et `y=3% 13`.

6.33.18 Construction d'un corps de Galois : `GF`

Dans sa forme la plus simple, `GF` a comme arguments un nombre premier p et un entier $n > 1$ ou la puissance d'un nombre premier p^n et un argument optionnel qui est le nom de variable choisi pour le générateur du corps (la variable doit être purgée au préalable).
`GF` crée un corps de Galois de caractéristique p et ayant p^n éléments, les éléments du corps sont alors 0 et les puissances de 0 à $p^n - 2$ du générateur. Le corps lui-même est stocké dans une variable libre (par défaut K, cette variable est affichée par le système, en même temps que le nom du générateur et de la variable libre, par défaut k, servant à représenter les élements du corps comme le quotient $\mathbb{Z}/p\mathbb{Z}[k]/P(k)$ où P est un polynôme irréductible et primitif).
Par exemple :

- `GF(3,5)` ou `GF(3^5)` crée un corps ayant 3^5 éléments dont le générateur est `g` (ou `h`, ... si g est affectée). On peut créer un élément du corps en prenant un polynôme en fonction de g, par exemple $g^{10} + 5g + 1$.
- `GF(2,8,a)` crée un corps ayant 2^8 éléments, et utilise la variable `a` pour en désigner le générateur (attention, faire `purge(a)` auparavant si nécessaire).
- La commande `pmin` permet de connaitre le polynôme minimal d'un élément du corps.

On peut ensuite créer des polynômes ou des matrices ayant des coefficients dans le corps, et les manipuler avec les instructions habituelles `+ - * / inv sqrt, quo, rem, quorem, diff, factor, gcd. egcd, ...` par exemple :

- `GF(3,5,b); A:=[[1,b],[b,1]]; inv(A)` calcule l'inverse d'une matrice à coefficients dans le corps à 3^5 éléments
- `GF(5,3,c); p:=x^2-c-1; factor(p)` factorise le polynôme p comme polynôme à coefficients dans le corps à 5^3 éléments, on en déduit une valeur de racine carrée de $c + 1$.
- `p:=randpoly(x,5,g); q:=diff(p); gcd(p,q)` génere un polynôme à coefficients aléatoires puis calcule sa dérivée et le PGCD ce qui permet de savoir si `p` a des racines multiples.

Il y a encore quelques limitations dues à une implémentation incomplète de certains algorithmes (par exemple factorisation à plusieurs variables lorsque le polynôme n'est pas unitaire).

Dans sa forme la plus complète (mais plus difficile à manipuler et moins lisible), les éléments de ce corps et le corps lui-même sont représentés par `GF(...)` où `...` est une séquence composée de :

- la caractéristique p ($px = 0$),

– le polynôme minimal irréductible (primitif s'il est créé par giac) engendrant un idéal I dans $\mathbb{Z}/p\mathbb{Z}[X]$, le corps de Galois est alors le quotient de $\mathbb{Z}/p\mathbb{Z}[X]$ par I,
– le nom de la variable du polynôme, par défaut `x`,
– un polynôme (un reste modulo le polynôme minimal) pour désigner un élément du corps (ces éléments ont une représentation additive) ou `undef` pour désigner tout le corps qui est le quotient des polynômes à coefficients dans Z/pZ par I.

Habituellement on donne un nom au corps crée (par exemple `G:=GF(p,n)`), afin de construire un élément particulier du groupe à partir d'un polynôme de $\mathbb{Z}/p\mathbb{Z}[X]$, on écrira par exemple `G(x^3+x)`. Notez que `G(x)` est un générateur du groupe multiplicatif G^* lorsque le polynôme minimal est généré par giac.
On tape :

```
G:=GF(2,8)
```

On obtient (par exemple) :

```
GF(2,k^8-k^7-k^6-k-1,k,undef)
```

Le corps G a $2^8 = 256$ éléments et $g = G(k)$ engendre le groupe multiplicatif de ce corps ($\{1, g, g^2, ...g^{254}\}$).
On tape :

```
K(k^9)
```

On obtient :

```
g^6+g^2+1)
```

On tape :

```
K(k)^255
```

On obtient 1. Comme vous pouvez le constater sur les exemples précédents, lorsque l'on travaille avec le même corps, les réponses contiennent des informations redondantes. C'est pourquoi la définition d'un corps peut avoir un troisième argument : le nom du générateur ou une liste contenant deux noms ou trois noms de variable formelle, (le nom de l'indéterminée du polynôme irréductible et le nom du corps de Galois que l'on doit mettre entre quote pour que ces variables ne soient pas évaluées ainsi que le nom du générateur). Cela permet d'obtenir un affichage plus compact des éléments du corps.
On tape :

```
G:=GF(2,2,['w','G']):; G(w^2)
```

On obtient :

```
Done, G(w+1)
```

On tape :

```
G(w^3)
```

On obtient :

`G(1)`

Les éléments de `GF(2,2)` sont donc : `0,1,w,w^2=w+1`.

On peut enfin indiquer quel polynôme irréductible on souhaite utiliser, en l'indiquant en 2-ième paramètre (au lieu de n), par exemple :

`G:=GF(2,w^8+w^6+w^3+w^2+1,['w','G'])`

Si on donne un polynôme irréductible non primitif, `Xcas` l'indique et propose un remplacement par un polynôme primitif, par exemple :

`G:=GF(2,w^8+w^7+w^5+w+1,['w','G'])`

On obtient :

`G:=GF(2,w^8-w^6-w^3-w^2-1,['w','G'],undef)`

6.33.19 Factorisation d'un polynôme à coefficients dans un corps de Galois : `factor`

On peut factoriser un polynôme à coefficients dans un corps de Galois avec `factor`.
On tape par exemple pour avoir G=$\mathbb{F}_4$:

`GF(2,2,a)`

On obtient :

`GF(2,k^2+k+1,[k,K,a],undef)`

On tape par exemple :

`factor(a^2*x^2+1))`

On obtient :

`(a+1)*(x+a+1)^2`

6.34 Le calcul modulaire comme Maple dans $\mathbb{Z}/p\mathbb{Z}[x]$

6.34.1 Quotient euclidien : `Quo`

`Quo` est la forme inerte de `quo`.
`Quo` renvoie `quo`, le quotient de la division euclidienne de deux polynômes sans l'évaluer.
On utilise `Quo` et `mod` pour calculer en mode `Maple` le quotient de la division euclidienne de deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
On tape, en mode `Xcas` :

`Quo((x^3+x^2+1) mod 13,(2*x^2+4) mod 13)`

On obtient :

```
quo((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

puis en utilisant `eval(ans())`, on obtient :

```
(-6% 13)*x+-6% 13
```

Attention `Quo` est surtout utile en mode `Maple`.
On tape, en mode `Maple` :

```
Quo(x^3+x^2+1,2*x^2+4) mod 13
```

On obtient :

```
(-6)*x-6
```

On tape, en mode `Maple` :

```
Quo(x^2+2*x,x^2+6*x+5) mod 5
```

On obtient :

```
1
```

6.34.2 Reste euclidien : `Rem`

`Rem` est la forme inerte de `rem`.
`Rem` renvoie `rem`, le reste de la division euclidienne de deux polynômes sans l'évaluer.
On utilise `Rem` et `mod` pour calculer en mode `Maple` le reste de la division euclidienne de deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
On tape, en mode `Xcas` :

```
Rem((x^3+x^2+1) mod 13,(2*x^2+4) mod 13)
```

On obtient :

```
rem((x^3+x^2+1)% 13,(2*x^2+4)% 13)
```

puis en utilisant `eval(ans())`, on obtient :

```
(-2% 13)*x+-1% 13
```

Attention `Rem` est surtout utile en mode `Maple`.
On tape alors, en mode `Maple` :

```
Rem(x^3+x^2+1,2*x^2+4) mod 13
```

On obtient :

```
(-2)*x-1
```

On tape, en mode `Maple` :

```
Rem(x^2+2*x,x^2+6*x+5) mod 5
```

On obtient :

```
1*x
```

6.34.3 PGCD dans $\mathbb{Z}/p\mathbb{Z}[x]$: `Gcd`

`Gcd` est la forme inerte de `gcd`.
`Gcd` renvoie le `gcd` (greatest common divisor) de deux polynômes (ou d'une liste de polynômes ou d'une suite de polynômes) sans l'évaluer.
On utilise `Gcd` et `mod` pour calculer en mode `Maple` le PGCD des deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$ lorsque p est premier (voir aussi 6.27.7).
On tape en mode `Xcas` :

```
Gcd((2*x^2+5,5*x^2+2*x-3)% 13)
```

On obtient :

```
gcd((2*x^2+5)% 13,(5*x^2+2*x-3)% 13)
```

puis avec `eval(ans())` on obtient :

```
(1% 13)*x+2% 13
```

Attention `Gcd` est surtout utile en mode `Maple`.
On tape alors en mode `Maple` :

```
Gcd(2*x^2+5,5*x^2+2*x-3) mod 13
```

On obtient :

```
1*x+2
```

```
Gcd(x^2+2*x,x^2+6*x+5) mod 5
```

On obtient :

```
1*x
```

6.34.4 Factorisation dans $\mathbb{Z}/p\mathbb{Z}[x]$: `Factor`

`Factor` a comme argument un polynôme.
`Factor` renvoie `factor` sans l'évaluer. Ensuite, `factor` factorise ce polynôme dans $\mathbb{Z}/p\mathbb{Z}[x]$ à condition que l'on ait p premier.
On tape en mode `Xcas` :

```
Factor((-3*x^3+5*x^2-5*x+4)% 13)
```

On obtient :

```
factor((-3*x^3+5*x^2-5*x+4)% 13)
```

```
((1% 13)*x+-6% 13)*((-3% 13)*x^2+-5% 13)
```

Attention `Factor` est surtout utile en mode `Maple`.
On tape alors en mode `Maple` :

```
Factor(-3*x^3+5*x^2-5*x+4) mod 13
```

On obtient :

```
-3*(1*x-6)*(1*x^2+6)
```

6.34.5 Déterminant d'une matrice de Z/pZ : `Det`

`Det` est la forme inerte de `det`.
`Det` a pour argument une matrice A de Z/pZ.
`Det` renvoie `det` sans l'évaluer. Ensuite, `det` calcule, dans Z/pZ, le déterminant de la matrice A.
On tape en mode `Xcas` :

```
Det([[1,2,9] mod 13,[3,10,0] mod 13,[3,11,1] mod 13])
```

Ou on tape en mode `Xcas` :

```
Det([[1,2,9],[3,10,0],[3,11,1]] mod 13)
```

On obtient :

```
det([[1% 13,2% 13,-4% 13],[3% 13,-3% 13,0% 13], [3%
13,-2% 13,1% 13]])
```

puis :

```
5% 13
```

donc, dans $\mathbb{Z}/13\mathbb{Z}$, le déterminant de la matrice $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ est `5% 13` (on a `det(A)=31`).
Attention `Det` est surtout utile en mode `Maple`.
On tape alors en mode `Maple` :

```
Det([[1,2,9],[3,10,0],[3,11,1]]) mod 13
```

On obtient :

```
5
```

6.34.6 Inverse d'une matrice de $\mathbb{Z}/p\mathbb{Z}$: `Inverse`

`Inverse` a pour argument une matrice A de Z/pZ.
`Inverse` renvoie `inverse` sans l'évaluer. Ensuite, `inverse` calcule, dans Z/pZ, l'inverse de la matrice A.
On tape :

```
Inverse([[1,2,9] mod 13,[3,10,0] mod 13,[3,11,1]
mod13])
```

Ou on tape :

```
Inverse([[1,2,9],[3,10,0],[3,11,1]] mod 13)
```

Ou on tape :

```
Inverse([[1,2,9]% 13,[3,10,0]% 13,[3,11,1]% 13])
```

Ou on tape :

```
Inverse([[1,2,9],[3,10,0],[3,11,1]]% 13)
```

On obtient :

```
inverse([[1% 13,2% 13,9% 13],[3% 13,10% 13,0% 13], [3%
13,11% 13,1% 13]])
```

puis :

```
[[2% 13,-4% 13,-5% 13],[2% 13,0% 13,-5% 13], [-2%
13,-1% 13,6% 13]]
```

c'est l'inverse de la matrice $A = [[1, 2, 9], [3, 10, 0], [3, 11, 1]]$ dans $Z/13Z$.
Attention en mode `Maple` on tape :

```
Inverse([[1,2,9],[3,10,0],[3,11,1]]) mod 13
```

On obtient :

```
[[2,-4,-5],[2,0,-5],[-2,-1,6]]
```

6.34.7 Résolution d'un système linéaire de $\mathbb{Z}/p\mathbb{Z}$: `Rref`

`Rref` renvoie `rref` sans l'évaluer. Ensuite, `rref` résout, dans $\mathbb{Z}/p\mathbb{Z}$, un système d'équations linéaires de la forme : $Ax = B$ (voir aussi 6.55.3).
Résoudre dans $\mathbb{Z}/13\mathbb{Z}$

$$\left\{\begin{array}{rcl} x + \ 2\cdot y & = & 9 \\ 3\cdot x + 10\cdot y & = & 0 \end{array}\right.$$

On tape :

```
Rref([[1,2,9] mod 13,[3,10,0] mod 13])
```

Ou on tape :

```
Rref([[1,2,9],[3,10,0]] mod 13)
```

Ou on tape :

```
Rref([[1,2,9]% 13,[3,10,0]% 13])
```

Ou on tape :

```
Rref([[1,2,9],[3,10,0]]% 13)
```

On obtient :

```
rref([[1% 13, 2% 13, 9% 13],[3% 13,10% 13,0% 13]])
```

puis :

```
[[1% 13,0% 13,3% 13],[0% 13,1% 13,3% 13]]
```

ce qui veut dire que `x=3% 13` et `y=3% 13`.
Attention en mode `Maple` on tape :

```
Rref([[1,2,9],[3,10,0],[3,11,1]]) mod 13
```

On obtient :

```
[[1,0,0],[0,1,0],[0,0,1]]
```

6.35 Développements limités et asymptotiques

6.35.1 Division selon les puissances croissantes : `divpc`

`divpc` a trois arguments : deux polynômes $A(x), B(x)$ (avec $B(0) \neq 0$) et un entier n.
`divpc` renvoie le quotient $Q(x)$ de la division de $A(x)$ par $B(x)$ selon les puissances croissantes avec `degree`$(Q) \leq n$ ou $Q = 0$.
$Q(x)$ est donc le développement limité, d'ordre n, de $\frac{A(x)}{B(x)}$ au voisinage de $x = 0$.
On tape :

```
divpc(1+x^2+x^3,1+x^2,5)
```

On obtient :

```
-x^5+x^3+1
```

Attention ! ! ! cette commande ne marche pas si les polynômes sont écrits avec la liste de leurs coefficients.

6.35.2 Développement limité : `taylor`

`taylor` peut avoir de un à quatre paramètres :
l'expression à développer, `x=a` (par défaut `x=0`), l'ordre du développement (par défaut `5`), ou encore :
l'expression à développer, `x`, l'ordre du développement (par défaut `5`) et le point au voisinage duquel on veut le développement (par défaut `0`).
Remarque on peut aussi mettre `x`, a, n au lieu de `x=`a, n
`taylor` renvoie un polynôme en `x-a`, plus un reste que `Xcas` écrit :
`(x-a)^n*order_size(x-a)`
cela signifie que l'on a un développement limité à l'ordre $n-1$ (ou à l'ordre $p < n$).
En effet `order_size` désigne une fonction telle que, quelque soit `r` positif :
`x^r*order_size(x)` tend vers zéro quand `x` tend vers zéro.
Par exemple, les fonctions constantes, la fonction $\log$ (ou $\ln$), sont des fonctions `order_size`.
On tape :

```
taylor(sin(x),x=1,2)
```

Ou on tape (attention à l'ordre des arguments !) :

```
taylor(sin(x),x,2,1)
```

On obtient :

```
sin(1)+cos(1)*(x-1)+(-(1/2*sin(1)))*(x-1)^2+
(x-1)^3*order_size(x-1)
```

Attention ! ! !
L'ordre que l'algorithme utilise pour les développements limités peut être plus petit que celui demandé : l'ordre peut diminuer si il y des compensations par exemple :
développement de $\frac{x^3 + \sin(x)^3}{x - \sin(x)}$ au voisinage de `x=0`
On tape :

```
taylor(x^3+sin(x)^3/(x-sin(x)))
```

On obtient seulement un développement à l'ordre 2 :

```
6+-27/10*x^2+x^3*order_size(x)
```

On tape :

```
taylor(x^3+sin(x)^3/(x-sin(x)),x=0,7)
```

On obtient seulement un développement à l'ordre 4 :

```
6+-27/10*x^2+x^3+711/1400*x^4+x^5*order_size(x)
```

6.35.3 Développement limité : `series`

`series` permet de faire le développement limité d'une expression au voisinage d'un point à un ordre donné.
`series` peut avoir de un à quatre paramètres :
l'expression à développer, `x=a` (par défaut `x=0`), l'ordre du développement (par défaut `5`), la direction `-1, 1` (pour un développement unidirectionel) ou `0` (pour un développement bidirectionel) (par défaut `0`).
Remarque on peut aussi mettre `x`, a, n au lieu de `x=`a, n
`series` renvoie un polynôme en `x-a`, plus un reste que `Xcas` écrit :
`(x-a)^n*order_size(x-a)`
cela signifie que l'on a un développement limité à l'ordre $n-1$ (ou à l'ordre $p < n$).
En effet `order_size` désigne une fonction telle que, quelque soit `r` positif :
`x^r*order_size(x)` tend vers zéro quand `x` tend vers zéro.
Par exemple, les fonctions constantes, la fonction $\log$ (ou $\ln$), sont des fonctions `order_size`.
Attention ! ! !
L'ordre que l'algorithme utilise pour les développements limités peut être plus petit que celui demandé : l'ordre peut diminuer si il y a des compensations (voir les exemples qui suivent)

- développement au voisinage de `x=0`
  Donner le développement de $\dfrac{x^3+\sin(x)^3}{x-\sin(x)}$ au voisinage de `x=0`
  On tape :
  ```
  series(x^3+sin(x)^3/(x-sin(x)))
  ```
  On obtient seulement un développement à l'ordre 2 :
  ```
  6+-27/10*x^2+x^3*order_size(x)
  ```
  On tape :
  ```
  series(x^3+sin(x)^3/(x-sin(x)),x=0,7)
  ```
  On obtient seulement un développement à l'ordre 4 :
  ```
  6+-27/10*x^2+x^3+711/1400*x^4+x^5*order_size(x)
  ```
- développement au voisinage de `x=a`
  Exemple :
  Donner un développement limité à l'ordre 4 au voisinage de $x = \frac{\pi}{6}$ de $\cos(2x)^2$.
  On tape :
  ```
  series(cos(2*x)^2,x=pi/6, 4)
  ```

On obtient :

```
1/4+(-(4*sqrt(3)))/4*(x-pi/6)+(4*3-4)/4*(x-pi/6)^2+
32*sqrt(3)/3/4*(x-pi/6)^3+(-16*3+16)/3/4*(x-pi/6)^4+
            (x-pi/6)^5*order_size(x-pi/6)
```

– développement au voisinage de x=+∞ ou x=-∞

  Exemple 1 :

  Donner un développement de $\arctan(x)$ à l'ordre 5 au voisinage de x=+∞ en prenant comme infiniment petit $h = \frac{1}{x}$.

  On tape :

```
series(atan(x),x=+infinity,5)
```

  On obtient :

```
pi/2-1/x+1/3*(1/x)^3-1/5*(1/x)^5+
      (1/x)^6*order_size(1/x)
```

  Donner un développement de $\arctan(x)$ à l'ordre 5 au voisinage de x=-∞ en prenant comme infiniment petit $h = \frac{1}{x}$.

  On tape :

```
series(atan(x),x=-infinity,5)
```

  On obtient :

```
-pi/2-1/x-1/3*(-1/x)^3+1/5*(-1/x)^5+
      (-1/x)^6*order_size(-1/x)
```

  Exemple 2 :

  Donner un développement de $(2x-1)e^{\frac{1}{x-1}}$ à l'ordre 2 au voisinage de x=+∞ en prenant comme infiniment petit $h = \frac{1}{x}$.

  On tape :

```
series((2*x-1)*exp(1/(x-1)),x=+infinity,3)
```

  On obtient seulement l'ordre 1 :

```
2*x+1+2/x+(1/x)^2*order_size(1/x)
```

  On tape pour avoir le développement à l'ordre 2 en $1/x$:

```
series((2*x-1)*exp(1/(x-1)),x=+infinity,4)
```

  On obtient :

```
2*x+1+2/x+17/6*(1/x)^2+(1/x)^3*order_size(1/x)
```

  Exemple 3 :

  Donner un développement de $(2x-1)e^{\frac{1}{x-1}})$ à l'ordre 2 au voisinage de x=-∞ en prenant comme infiniment petit $h = -\frac{1}{x}$.

  On tape :

```
series((2*x-1)*exp(1/(x-1)),x=-infinity,4)
```

  On obtient :

```
-2*(-x)+1-2*(-1/x)+17/6*(-1/x)^2+
      (-1/x)^3*order_size(-1/x)
```

– développement unidirectionnel

  Il faut utiliser un paramètre supplémentaire pour indiquer la direction :

  – 1 pour faire un développement au voisinage de $x = a$ avec $x > a$,
  – -1 pour faire un développement au voisinage de $x = a$ avec $x < a$,
  – 0 pour faire un développement au voisinage de $x = a$ avec $x \neq a$.

  Exemple

Donner un développement de $\frac{(1+x)^{\frac{1}{x}}}{x^3}$ à l'ordre 2, au voisinage de $x = 0^+$.
On tape :

```
series((1+x)^(1/x)/x^3,x=0,2,1)
```

On obtient :

```
exp(1)/x^3+(-(exp(1)))/2/x^2+1/x*order_size(x)
```

6.35.4 Développement réciproque d'un développement en séries en 0 : `revert`

`revert` a comme argument une expression qui est le début du développement en séries en 0 d'une fonction f.
`revert` renvoie le développement en séries en 0 de $g(f(0)+x)$ où g vérifie $g(f(x)) = x$.
On tape :

```
revert(x+x^2+x^4)
```

On obtient :

```
x-x^2+2*x^3-6*x^4
```

En effet la fonction f vérifie : $f(0) = 0, f'(0) = 1, f"(0) = 2, f'"(0) = 0, f^{(4)}(0) = 24$ et si $g(f(x)) = x$ on en déduit en dérivant cette identité que : $g(0) = 0, g'(0) = 1/f'(0) = 1, g''(0) = -2, g'''(0) = 12, g^{(4)}(0) = -15*8-24 = -144 = -24*6$
On tape le début du développement en 0 de $\exp(x)$:

```
revert(1+x+x^2/2+x^3/6+x^4/14)
```

On obtient le début du développement en $x = 0$ de $\ln(1+x)$:

```
x-x^2/2+2*x^3/3-x^4/4
```

6.35.5 Résidu d'une expression en un point : `residue`

`residue` a comme argument une expression dépendant d'une variable, le nom de cette variable et un complexe a ou bien une expression dépendant d'une variable et l'égalité : nom_de_variable=a. `residue` renvoie le résidu de cette exppression au point a.
On tape :

```
residue(cos(x)/x^3,x,0)
```

Ou on tape :

```
residue(cos(x)/x^3,x=0)
```

On obtient :

```
(-1)/2
```

On tape :

```
int(exp(i*t)/(2*exp(i*t)-1),t=0..2*pi)
```

On obtient :
Searching int of $1/(2*t-1)$ where t is on the unit circle, using residues

```
(2*pi)/2
```

On tape :

```
int(exp(2*i*t)/(2*exp(i*t)-1))^2,t=0..2*pi)
```

On obtient :
Searching int of $t/(4*t^2-4*t+1)$ where t is on the unit circle, using residues

```
(2*pi)/4
```

6.35.6 Développement de Padé : `pade`

`pade` a 4 arguments
- une expression,
- le nom de la variable utilisée,
- un entier n ou un polynôme N,
- un entier p.

`pade` renvoie une fraction rationnelle P/Q (avec le degré de $P < p$) qui a, au voisinage de 0, le même développement de Taylor à l'ordre n que l'expression, ou qui est égal à l'expression modulo x^{n+1} (resp modulo N).
On tape :

```
pade(exp(x),x,5,3)
```

Ou on tape :

```
pade(exp(x),x,x^6,3)
```

On obtient :

```
(3*x^2+24*x+60)/(-x^3+9*x^2-36*x+60)
```

On vérifie en tapant :

```
taylor((3*x^2+24*x+60)/(-x^3+9*x^2-36*x+60))
```

On obtient :

```
1+x+1/2*x^2+1/6*x^3+1/24*x^4+1/120*x^5+x^6*order_size(x)
```

On reconnait le développement de Taylor à l'ordre 5 de `exp(x)` au voisinage de 0.
On tape :

```
pade((x^15+x+1)/(x^12+1),x,12,3)
```

Ou on tape :

```
pade((x^15+x+1)/(x^12+1),x,x^13,3)
```

On obtient :

```
x+1
```

On tape :

```
pade((x^15+x+1)/(x^12+1),x,14,4)
```

Ou on tape :

```
pade((x^15+x+1)/(x^12+1),x,x^15,4)
```

On obtient :

```
(-2*x^3-1)/(-x^11+x^10-x^9+x^8-x^7+x^6-x^5+x^4-
x^3-x^2+x-1)
```

On vérifie en tapant :

```
series(ans(),x=0,15)
```

On obtient :

```
1+x-x^12-x^13+2x^15+x^16*order_size(x)
```

puis en tapant :

```
series((x^15+x+1)/(x^12+1),x=0,15)
```

On obtient :

```
1+x-x^12-x^13+x^15+x^16*order_size(x)
```

Les deux expressions ont même développement de Taylor à l'ordre 14 au voisinage de 0.

6.36 Les plages de valeurs

6.36.1 Définition d'une plage de valeurs : `a1..a2`

Un intervalle est défini par deux nombres séparés par `..` on écrit :
`1..3` ou
`1.2..sqrt(2)`
On tape :

```
A:=1..4
```

```
B:=1.2..sqrt(2)
```

Attention !
l'ordre est important : si `B:=2..3` et `C:=3..2` la réponse de `B==C` est `0`.

6.36.2 Pour accéder aux bornes d'une plage de valeurs : `left` `gauche` `right` `droit`

`left` (resp `right`) a comme argument un intervalle.
`left` (resp `right`) permet d'accéder à la partie gauche (resp droite) de l'intervalle.
On tape :

```
(3..5)[0]
```

Ou on tape :

```
sommet(3..5)
```

On obtient :

```
'..'
```

On tape :

```
left(3..5)
```

Ou on tape :

```
(3..5)[1]
```

Ou on tape :

```
feuille(3..5)[0]
```

Ou on tape :

```
op(3..5)[0]
```

On obtient :

```
3
```

On tape :

```
right(3..5)
```

Ou on tape :

```
(2..5)[2]
```

Ou on tape :

```
feuille(3..5)[1]
```

Ou on tape :

```
op(3..5)[1]
```

On obtient :

```
5
```

Remarque
`left` et `right` permettent aussi d'accéder aux membres d'une équation (par exemple `left(2*x+1=x+2)` renvoie `2*x+1`).

6.36.3 Centre d'une plage de valeurs : `interval2center`

`interval2center` a comme argument un intervalle ou une liste d'intervalles.
`interval2center` renvoie le centre de l'intervalle ou la liste des centres de ces intervalles.
On tape :

```
interval2center(3..5)
```

On obtient :

```
4
```

On tape :

```
interval2center([2..4,4..6,6..10])
```

On obtient :

```
[3,5,8]
```

6.36.4 Plages de valeurs définies par leur centre : `center2interval`

`center2interval` a comme argument un vecteur `V` de réels et éventuellement un réel comme deuxième argument (par défaut `(3*V[0]-V[1])/2`).
`center2interval` renvoie un vecteur d'intervalles ayant pour centres les réels de l'argument : ces intervalles sont définis en commençant par la valeur donnée par le deuxième argument.
On tape :

```
center2interval([3,5,8],2)
```

Ou on tape car la valeur par défaut vaut (3*3-5)/2=2 :

```
center2interval([3,5,8])
```

On obtient :

```
[2..4,4..6,6..10]
```

On tape :

```
center2interval([3,5,8],2.5)
```

On obtient :

```
[2.5..3.5,3.5..6.5,6.5..9.5]
```

6.37 Les intervalles

6.37.1 Définition : `i[]`

Un intervalle est un ensemble de 2 nombres flottants de $n \geq 15$ chiffres significatifs séparés par une virgule et parenthésés par `i[]`.
On utilise le mot reservé `i[]`), on écrit pour désigner l'intervalle $[1, 13/11]$: `i[1,13/11]`.
L'intervalle vide s'écrira `i[]`
Par exemple, on tape :

```
i[1,13/11]
```

On obtient :

```
i[1.0000000000000,1.1818181818182]
```

Remarque Si `a>b` alors `i[a,b]` renvoie
`i[evalf(b,15)-epsilon,evalf(a,15)+epsilon]` avec `epsilon` Par exemple, on tape :

```
i[pi,sqrt(3)]
```

On obtient :

```
i[1.7320508075689,3.1415926535898]
```

Autre notation On peut aussi désigner un intervalle par un nombre décimal `a` suivi d'un point d'interrogation. Si la partie décimale de `a` contient `n` chiffres , l'intervalle a pour milieu `a` et pour longueur $2 * 10^{-n}$ Par exemple, on tape :
`0.123?`
On obtient :
`i[0.122,0.123]`
On tape : `789.123456?`
On obtient :
`i[0.789123455e3,0.789123457e3]`
Attention
On tape :

```
a:=i[1,13/11]
```

On obtient :

```
i[1.0000000000000,1.1818181818182]
```

```
b:=i[1,13/11]
```

On obtient :

```
i[1.0000000000000,1.1818181818182]
```

```
a==b
```

On obtient :

```
faux
```

En fait, `a` ou `b` représente un nombre décimal entre 1 et 1.1818181818182 car l'intervalle sert à donner une approximation du nombre `a` ou `b` i.e `a` et `b` sont dans le même intervalle.
On tape :

```
b:=i[pi,pi+10^-5]
```

On obtient :

```
i[3.1415926535898,3.1416026535898]
```

En fait `b` représentent un nombre décimal entre 3.1415926535898 et 3.1416026535898.

6.37.2 Somme de 2 intervalles

La somme de 2 intervalles est un intervalle qui a pour borne inférieure (resp supérieure) la somme des bornes inférieures (resp supérieures).
On tape :

```
i[1,4]+i[2,3]
```

On obtient :

```
i[3.0000000000000,7.0000000000000]
```

6.37.3 Opposé d'un intervalle

L'opposé d'un intervalle est un intervalle qui a pour borne inférieure (resp supérieure) l'opposé de la borne supérieure du (resp inférieure).
On tape :

```
-i[2,3]
```

On obtient :

```
i[-3.0000000000000,-2.0000000000000]
```

6.37.4 Produit de 2 intervalles

Le produit de 2 intervalles est un intervalle qui a pour borne inférieure (resp supérieure) le produit des bornes inférieures (resp supérieures).
On tape :

```
i[1,4]*i[2,3]
```

On obtient :

```
i[2.0000000000000,0.12000000000000e2]
```

6.37.5 Inverse d'un intervalle

L'inverse d'un intervalle est un intervalle qui a pour borne inférieure (resp supérieure) l'inverse de la borne supérieure du (resp inférieure).
On tape :

```
1/i[2,3]
```

On obtient :

```
i[0.33333333333333,0.50000000000000]
```

6.37.6 Pour accéder aux bornes d'un intervalle : `left gauche right droit`

On tape :

```
left(i[2,5])
```

On obtient :

```
2.0000000000000
```

On tape :

```
right(i[2,5])
```

On obtient :

```
5.0000000000000
```

6.37.7 Milieu d'un intervalle : `midpoint milieu`

`union` est un opérateur infixé.
On tape :

```
milieu(i[2,3])
```

Ou on tape :

```
midpoint(i[2,3])
```

On obtient :

```
2.5000000000000
```

6.37.8 Union de 2 intervalles : `union`

`union` est un opérateur infixé.
`union` renvoie l'enveloppe convexe des 2 intervalles donnés en argument.
On tape :

```
i[1,3] union i[2,4]
```

On obtient :

```
i[1.0000000000000,4.0000000000000]
```

6.37.9 Intersection de 2 intervalles : `intersect`

`intersect` est un opérateur infixé.
`intersect` renvoie l'intersection des 2 intervalles donnés en argument.
On tape :

```
i[1,3] intersect i[2,4]
```

On obtient :

```
i[2.0000000000000,3.0000000000000]
```

6.37.10 Tester si un élément est dans un intervalle : `contains`

`contains` a deux paramètres : un intervalle `I` et un élément `c`.
`contains` est une fonction qui teste si l'élément `c` appartient à l'intervalle `L`.
`contains` renvoie `0` si `c` n'est pas dans `L`, et sinon renvoie `1`.
Attention, à l'ordre des paramètres, c'est pour des raisons de compatibilité !
On tape :

```
contains(i[0,2],1)
```

On obtient :

```
1
```

On tape :

```
contains(i[0,2],3)
```

On obtient :

```
0
```

6.37.11 Convertir un nombre en un intervalle : `convert`

`convert` a deux ou trois paramètres : une expression symbolique, le mot reservé `interval` et eventuellement un entier $n > 15$ pour donner le nombre de chiffres significatifs désirés.
`convert` renvoie le plus petit intervalle dans lequel se trouve la valeur de l'expression ou la valeur de l'expression ? ? ? ? ? ? ? ?.
On tape :

```
convert(sin(3)+1,interval)
```

On obtient :

```
i[1.1411200080599,1.1411200080599]
```

On tape :

```
convert(sin(3)+1,interval,20)
```

On obtient :

```
i[1.14112000805986722221,1.14112000805986722221]
```

6.38 Les séquences

6.38.1 Définition : `seq[] ()`

Une séquence est une suite d'éléments séparés par une virgule et parenthésés par `( )` ou on utilise le mot reservé `seq[...]` donc on écrit :
`(1,2,3,4)` ou
`seq[1,2,3,4]`
On tape :

`A:=(1,2,3,4)` ou `A:=seq[1,2,3,4]`

`B:=(5,6,3,4)` ou `B:=seq[5,6,3,4]`

Attention !
l'ordre est important : si `B:=(5,6,3,4)` et `C:=(3,4,5,6)` la réponse de `B==C` est `0`.
Attention ! (voir 6.38.5)
`seq([0,2])=(0,0)` et `seq([0,1,1,5])=[0,0,0,0,0]` alors que
`seq[0,2]=(0,2)` et `seq[0,1,1,5]=(0,1,1,5)`

6.38.2 Concaténer deux séquences : ,

La concaténation de deux séquences est simple puisqu'il suffit d'écrire les séquences en les séparant par une virgule (`,`).
On tape :

`A:=(1,2,3,4)`

`B:=(5,6,3,4)`

`A,B`

On obtient :

`(1,2,3,4,5,6,3,4)`

6.38.3 Pour accéder à un élément d'une séquence : `[]`

Pour accéder à un élément d'une séquence, on tape l'indice de cet élément entre des crochets pour des indices qui commencent à 0
ou bien
on tape son indice entre des doubles crochets pour des indices qui commencent à 1.
On tape :

`(0,1,2)[1]`

On obtient car l'indice du premier élément est 0 :

`1`

Ou bien
On tape :

`(0,1,2)[[1]]`

On obtient car l'indice du premier élément est 1 :

`0`

6.38.4 Pour extraire une sous-séquence d'une séquence : `[]`

Pour extraire une sous-séquence d'une séquence, on tape entre des crochets l'indice de début, puis `..` et l'indice de fin de la sous-séquence.
On tape :

`(0,1,2,3,4)[1..3]`

On obtient :

`(1,2,3)`

6.38.5 Pour fabriquer une séquence ou une liste : `seq $`

`seq` peut renvoyer une séquence (avec la même syntaxe que `Maple`) ou une liste (avec la même syntaxe que `TI`) selon la forme de ses arguments qui sont : une expression dépendant d'un paramètre (par exemple j) et des paramètres décrivant la variation de j.
`$` renvoie une séquence et c'est la version infixée de `seq` lorsque `seq` a deux arguments : une expression dépendant d'un paramètre (par exemple j) et $(j = a..b)$ où a et b sont des nombres réels (par exemple `j^2$ (j=-1..3)`), ou une expression constante et un nombre n (par exemple `4$3`).
`seq` a deux, trois, quatre ou cinq arguments car on peut exprimer la variation de j de a à b à l'aide d'un argument $(j = a..b)$ ou de deux arguments $(j, a..b)$ (syntaxe `Maple` où il n'y a pas la possibilité de mettre un paramètre de saut) ou encore à l'aide de trois arguments (j, a, b) ou de quatre arguments (j, a, b, p) (syntaxe `TI` avec la possibilité de mettre p comme paramètre de saut).

- **Syntaxe** `Maple`
  `seq` a deux arguments une expression dépendant d'un paramètre (par exemple j) et $j = a..b$ où a et b sont des nombres réels (ou une expression constante et un nombre n). On peut ajouter pour `seq` un troisième argument, le pas (1 par défaut).
  `$` a les même arguments mais est une fonction infixée et il faut parenthéser les arguments.
  `seq` (ou `$`) définit la séquence obtenue en remplacant dans l'expression j par a, $a + 1..b$ si $b > a$ et par a, $a - 1..b$ si $b < a$ (ou `seq` renvoie la séquence formée par n fois la constante).
  -`seq` a trois arguments une expression dépendant d'un paramètre (par exemple j), le nom du paramètre (par exemple j) et $a..b$ où a et b sont des nombres réels, ou
  `seq` définit la séquence obtenue en remplacant dans l'expression j par a, $a + 1..b$ si $b > a$ et par a, $a - 1..b$ si $b < a$ le pas est donc soit égal à 1, soit

égal à -1.

– **Syntaxe** `TI`
-`seq` a quatre arguments une expression dépendant d'un paramètre (par exemple j), le nom du paramètre (par exemple j), puis a et b où a et b sont des nombres réels. `seq` définit la liste obtenue en remplaçant dans l'expression j par a, $a+1..b$ si $b > a$ ou par a, $a-1..b$ si $b < a$).
-`seq` a cinq arguments une expression dépendant d'un paramètre (par exemple j), le nom du paramètre (par exemple j), puis a et b (où a et b sont des nombres réels) et p le pas (où p est un nombre réel positif ou négatif).
`seq` définit la liste obtenue en remplaçant dans l'expression j par a, $a+p..a+k*p$ ($a+k*p \leq b < a+(k+1)*p$ ou $a+k*p \geq b > a+(k+1)*p$).
Par défaut, on a p=1 si $b > a$ et p=-1 si $b < a$. Si p n'a pas le bon signe, ce signe est rectifié par le logiciel !

Remarque :
Dans la syntaxe `Maple`, `seq` renvoie une séquence et il n'y a pas la possibilité de mettre un paramètre de saut, contrairement à la syntaxe `TI` où `seq` renvoie une liste avec la possibilité de mettre un paramètre de saut.
On tape pour avoir une séquence d'éléments identiques :

```
seq(t,4)
```

Ou on tape

```
seq(t,k=1..4)
```

Ou on tape

```
t$4
```

On obtient :

```
(t,t,t,t)
```

On tape pour avoir une séquence :

```
seq(j^3,j=1..4)
```

Ou on tape

```
(j^3)$(j=1..4)
```

Ou on tape :

```
seq(j^3,j,1..4)
```

On obtient :

```
(1,4,9,16)
```

Ou on tape pour avoir une liste :

```
seq(j^3,j,1,4)
```

On obtient :

```
[1,4,9,16]
```

On tape :

```
seq(j^3,j,0,5,2)
```

On obtient :

```
[0,8,64]
```

On tape :

```
seq(j^3,j,5,0,-2)
```

ou

```
seq(j^3,j,5,0,2)
```

On obtient :

```
[125,27,1]
```

On tape :

```
seq(j^3,j,1,3,0.5)
```

On obtient :

```
[1,3.375,8,15.625,27]
```

On tape :

```
seq(j^3,j,1,3,1/2)
```

On obtient :

```
[1,27/8,8,125/8,27]
```

Exemples d'utilisation

- On tape pour avoir la dérivée troisième de $\ln(t)$:
  ```
  diff(log(t),t$3)
  ```
  On obtient :
  ```
  -((-(2*t))/t^4)
  ```
  On tape :
  ```
  l:=[[2,3],[5,1],[7,2]]
  seq((l[k][0])$(l[k][1]),k=0 .. size(l)-1)
  ```
  On obtient :
  ```
  2,2,2,seq[5],7,7
  2,2,2,5,7,7
  ```
- On tape pour transformer une chaîne en la liste de ces caractères :
  ```
  f(chn):={ local l; l:=size(chn); return
  seq(chn[j],j,0,l-1); }
  ```
  puis,
  ```
  f("abracadabra")
  ```

On obtient :
`["a","b","r","a","c","a","d","a","b","r","a"]`

– On tape pour transformer une chaîne en la séquence de ces caractères :

```
f(chn):={ local l; l:=size(chn); return
        seq(chn[j],j,0..l-1); }
```

puis,

```
f("abracadabra")
```

On obtient :
`("a","b","r","a","c","a","d","a","b","r","a"]))`

6.38.6 Pour transformer une séquence en liste : `[]` `nop`

Pour transformer une séquence en liste il suffit d'entourer la séquence par des crochets (`[]`) ou on utilise la fonction `nop`.
On tape :

```
[seq(j^3,j=1..4)]
```

ce qui équivaut à :

```
seq(j^3,j,1,4)
```

ou à :

```
[(j^3)$(j=1..4)]
```

On obtient :

```
[1,4,9,16]
```

On tape :

```
nop(1,4,9,16)
```

On obtient :

```
[1,4,9,16]
```

6.38.7 L'effet de l'opérateur + sur deux séquences

L'opérateur infixé + ayant comme argument deux séquences, renvoie la somme des éléments des deux séquences.
Bien voir la différence avec les listes car si les arguments de l'opérateur infixé + sont deux listes, il renvoie la somme termes à terme des éléments des deux listes.
On tape :

```
(1,2,3,4,5,6)+(4,3,5)
```

Ou on tape :

```
'+'((1,2,3,4),(5,6,4,3,5))
```

On obtient :

Mais si on tape :

`[1,2,3,4,5,6]+[4,3,5]`

On obtient :

`[5,5,8,4,5,6]`

`bf Attention`
Quand l'opérateur + est préfixé il doit être quoté c'est à dire écrit `'+'`.

6.39 Les ensembles

6.39.1 Définition : `set[]`

Un ensemble est un ensemble d'éléments séparés par une virgule et parenthésés par `% { % }`, ou on utilise le mot reservé `set[]`, on écrit :
`% {1,2,3,4% }` ou
`set[1,2,3,4]`
L'ensemble vide s'écrira `%{%}` ou `set[]`
En réponse on utilise **[** et **]** comme délimiteurs pour distinguer les ensembles des listes.
On tape :

`A:=% {1,2,3,4% }` ou `A:=set[1,2,3,4]`

On obtient :

[1,2,3,4**]**

On tape :

`B:=% {5,5,6,3,4% }` ou `B:=set[5,5,6,3,4]`

On obtient :

[5,6,3,4**]**

Attention !
L'ordre n'est pas important et il n'y a que des éléments différents : si `B:=% {5,5,6,3,4% }` et `C:=% {3,4,5,3,6% }` la réponse de `B==C` est `1`.

6.39.2 Tester si 2 ensembles (ou listes) sont inclus(e) l'un(e) dans l'autre : `is_included`, `est_inclus`

`est_inclus` ou `is_included` teste si l'ensemble (ou la liste considérée comme un ensemble) mis comme premier argument est contenu dans l'ensemble (ou la liste) mis comme deuxième argument en renvoyant `0` ou `1`.
On tape :

`est_inclus(set[1,2,3,4],set[1,2,5,6,3,4])`

Ou on tape :

```
est_inclus(% {1,2,3,4% },% {1,2,5,6,3,4% })
```

On obtient :

```
1
```

On tape :

```
est_inclus([1,2,3,4],[1,2,5,6,3,4])
```

On obtient :

```
1
```

Attention, `set[1,2,3,4,2]` renvoie `set[1,2,3,4,]`, d'où les résultats suivants.
On tape :

```
est_inclus(set[1,2,3,4,2],set[1,2,5,6,3,4])
```

Ou on tape :

```
est_inclus(% {1,2,3,4,2% },% {1,2,5,6,3,4% }
```

On obtient :

```
1
```

On tape :

```
est_inclus([1,2,3,4,2],[1,2,5,6,3,4])
```

On obtient :

```
1
```

6.39.3 Union de deux ensembles ou de deux listes : `union`

`union` renvoie l'ensemble qui est l'union de deux ensembles ou de deux listes.
`union` est un opérateur infixé.
On tape :

```
set[1,2,3,4] union set[5,6,3,4]
```

Ou on tape :

```
% {1,2,3,4% } union % {5,6,3,4% }
```

On obtient :

```
[1,2,3,4,5,6]
```

On tape :

```
[1,2,3] union [2,5,6]
```

On obtient :

```
[1,2,3,5,6]
```

6.39.4 Intersection de deux ensembles, de deux listes : `intersect`

`intersect` renvoie l'ensemble qui est l'intersection de deux ensembles ou de deux listes.
`intersect` est un opérateur infixé.
On tape :

```
set[1,2,3,4] intersect set[5,6,3,4]
```

Ou on tape :

```
% {1,2,3,4% } intersect % {5,6,3,4% }
```

On obtient :

[3,4]

On tape :

```
[1,2,3,4] intersect [5,6,3,4]
```

On obtient :

[3,4]

6.39.5 Différence de deux ensembles ou de deux listes : `minus`

`minus` renvoie l'ensemble qui est la différence de deux ensembles ou de deux listes.
`minus` est un opérateur infixé.
On tape :

```
set[1,2,3,4] minus set[5,6,3,4]
```

Ou on tape :

```
% {1,2,3,4% } minus % {5,6,3,4% }
```

On obtient :

[1,2]

On tape :

```
[1,2,3,4] minus [5,6,3,4]
```

On obtient :

[1,2]

6.40 Les listes ou les vecteurs

6.40.1 Les différentes listes

Une liste peut contenir des listes (c'est le cas des matrices) et peut désigner un vecteur (liste des coordonnées) ou un polynôme (liste des coefficients des puissances décroissantes), alors que une séquence est plate car on ne peut pas avoir un élément d'une séquence qui soit une séquence.
L'ordre est important dans une liste alors que l'ordre n'a pas d'importance pour un ensemble, il peut y avoir plusieurs fois le même objet dans une liste alors que chaque objet est unique dans un ensemble.

Dans les réponses de `Xcas`,
- les coordonnées d'un vecteur sont parenthésées avec `[` et `]`,
- une matrice avec [et],
- un polynôme avec `[]` et `[]`,
- un ensemble avec **[** et **]**.

Un vecteur peut donc être représentés par une liste contenant les composantes du vecteur (on met les composantes du vecteur entre crochets).
Attention !
Pour accéder à un élément d'une liste, on tape son indice entre des crochets pour des indices qui commencent à 0
ou bien
on tape son indice entre des doubles crochets pour des indices qui commencent à 1.
Pour toutes les autres fonctions de `Xcas` (autres que l'accés à un élément), l'indice du premier élément est 0.
Exemple
On tape : `L:=[2,5,1,3]`
`L[1]`
On obtient :
`5`
On tape : `L[[1]]`
On obtient :
`2`
On tape : `L:=[2,5,1,3]`
`L[1]:=4`
`L`
On obtient :
`[2,4,1,3]`
On tape : `L:=[2,5,1,3]`
`L[[1]]:=4`
`L`
On obtient :
`[4,5,1,3]`
Pour dessiner ou définir un vecteur géométrique voir 9.10.5 et 10.5.5.

6.40.2 Aplatir une liste : `flatten`

`flatten` a comme argument une liste de listes.

`flatten` transforme cette liste de listes en une liste.
On tape : `flatten([[1,[2,3],4],[5,6]])`
On obtient :
`[1,2,3,4,5,6]`
Remarque si la liste de listes est une matrice on peut aussi utiliser la commande `mat2list`.

6.40.3 Accès à un élément ou à une sous-liste d'une liste : `at []`

Accès à un élément

On utilise la fonction `at` pour accéder à un élément d'une liste, ou on tape son indice entre des crochets pour des indices qui commencent à 0
ou bien
on tape son indice entre des doubles crochets ou entre des parenthèses pour des indices qui commencent à 1.
Pour toutes les autres fonctions de `Xcas` (autres que l'accés à un élément), l'indice du premier élément est 0.

Exemples
Soit la liste : `[0,1,2]` et on veut désigner l'élément `1`.
On tape son indice `1` entre des crochets :

```
[0,1,2][1]
```

ou

```
at([0,1,2],1)
```

On obtient puisque les indices commencent à 0 :

```
1
```

ou bien 0n tape son indice `2` entre des doubles crochets ou entre des parenthèses :

```
[0,1,2][[2]]
```

ou

```
[0,1,2](2)
```

On obtient puisque avec cette notation, les indices commencent à 1 :

```
1
```

Soit la matrice : `A:=[[4,5],[2,6]]` et on veut désigner la ligne `[4,5]`.
On tape son indice `0` entre des crochets, si on veut que les indices commencent à 0 :

```
A[0]
```

ou

```
at(A,0)
```

On obtient puisque les indices commencent à 0 :

`[4,5]`

Ou bien si on veut que les indices commencent à 1, on tape son indice `1` entre des doubles crochets ou entre des parenthèses :

`A[[1]]`

ou

`A(1)`

On obtient puisque avec cette notation, les indices commencent à 1 :

`[4,5]`

Extraire une sous-liste

Voir aussi : 6.40.4 la fonction `mid`.
On utilise aussi la fonction `at` pour extraire une sous-liste d'une liste. On tape alors, entre des crochets, l'indice de début, puis `..` et l'indice de fin de la sous-liste ou on utilise la fonction `at`.
Attention ! l'indice du premier élément est 0.

On tape :

`[0,1,2,3,4][1..3]`

ou

`at([0,1,2,3,4],1..3)`

On obtient :

`[1,2,3]`

Attention, la commande `at` ne peut pas être utilisée pour les séquences : il faut utiliser la syntaxe `(0,1,2,3,4,5)[2..3]`.

6.40.4 Extraire une sous-liste d'une liste : `mid`

Voir aussi : 6.40.3 la fonction `at`.
`mid` pour extraire une sous-liste d'une liste.
`mid` a trois paramètres : la liste, l'indice du début de la sous-liste et la longueur de la sous-liste.
Attention ! l'indice du premier élément est 0.
On tape :

`mid([0,1,2,3,4,5],2,3)`

On obtient :

`[1,2,3]`

Attention, la commande `mid` ne peut pas être utilisée pour les séquences : il faut utiliser `(0,1,2,3,4,5)[2..3]`, syntaxe valable aussi pour les listes (voir ci-dessus).

6.40.5 Avoir le premier élément d'une liste : `head`

`head` renvoie le premier élément d'une liste.
On tape :

```
head([0,1,2,3])
```

On obtient :

```
0
```

`a:=head([0,1,2,3])` est équivalent à `a:=[0,1,2,3][0]`

6.40.6 Supprimer un élément dans une liste : `suppress`

`suppress` supprime dans une liste l'élément d'indice donné.
Attention ! l'indice du premier élément est 0.
On tape :

```
suppress([3,4,2],1)
```

On obtient :

```
[3,2]
```

6.40.7 Avoir la liste privée de son premier élément : `tail`

`tail` renvoie la liste privée de son premier élément.
On tape :

```
tail([0,1,2,3])
```

On obtient :

```
[1,2,3]
```

`l:=tail([0,1,2,3])` est équivalent à `l:=suppress([0,1,2,3],0)`

6.40.8 Partie droite et gauche d'une liste : `droit ou right, gauche ou left`

– `droit(l,n)` ou `right(l,n)` renvoie les n derniers éléments d'une liste `l`.
  On tape :
  ```
  droit([0,1,2,3,4,5,6,7,8],4)
  ```
  Ou on tape :
  ```
  right([0,1,2,3,4,5,6,7,8],4)
  ```
  On obtient :
  ```
  [5,6,7,8]
  ```
– `gauche(l,n)` ou `left(l,n)` renvoie les n premiers éléments d'une liste `l`.
  On tape :

`gauche([0,1,2,3,4,5,6,7,8],3)`

Ou on tape :

`left([0,1,2,3,4,5,6,7,8],3)`

On obtient :

`[0,1,2]`

6.40.9 Avoir la liste permutée : `revlist`

`revlist` a comme argument une liste (resp séquence).
`revlist` renvoie la liste (resp séquence) dans l'ordre inverse.
On tape :

`revlist([0,1,2,3,4])`

On obtient :

`[4,3,2,1,0]`

On tape :

`revlist([0,1,2,3,4],3)`

On obtient :

`3,[0,1,2,3,4]`

6.40.10 Avoir la liste permutée à partir de son n-ième élément : `rotate`

`rotate` a comme argument une liste et un nombre entier relatif (par défaut `n=-1`).
`rotate` renvoie :

- si `n>0`, la liste obtenue en permuttant les `n` premiers éléments avec la fin de la liste,
- si `n<0` en permuttant les `-n` derniers éléments avec le début de la liste. Par défaut `n=-1` et on met le dernier élément en premier.

On tape :

`rotate([0,1,2,3,4])`

On obtient :

`[4,0,1,2,3]`

On tape :

`rotate([0,1,2,3,4],2)`

On obtient :

`[2,3,4,0,1]`

On tape :

`rotate([0,1,2,3,4],-2)`

On obtient :

`[3,4,0,1,2]`

6.40.11 Avoir la liste permutée à partir de son n-ième élément : `shift`

`shift` a comme argument une liste et un nombre entier relatif (par défaut `n=-1`).
`shift` renvoie :

- si `n>0` la liste obtenue en remplaçant les `n` premiers éléments de la liste par `undef`, puis en en permuttant ces `n` premiers éléments avec la fin de la liste,
- si `n<0` en remplaçant les `-n` derniers éléments de la liste par `undef`, puis en permuttant les `-n` derniers éléments avec le début de la liste. Par défaut (`n=-1`) le premier élément vaut `undef` et il est suivi par la liste privée de son dernier élément.

On tape :

```
shift([0,1,2,3,4])
```

On obtient :

```
[undef,0,1,2,3]
```

On tape :

```
shift([0,1,2,3,4],2)
```

On obtient :

```
[2,3,4,undef,undef]
```

On tape :

```
shift([0,1,2,3,4],-2)
```

On obtient :

```
[undef,undef,0,1,2]
```

6.40.12 Modifier un élément d'une liste : `subsop`

`subsop` permet de modifier un élément dans une liste sans avoir à la mettre dans une variable.
Remarque Si le second argument est `'k=NULL'`, l'élément d'indice `k` est enlevé de la liste.
On tape en mode `Xcas` (les indices commencent à 0) :

```
subsop([0,1,2],1=5)
```

Ou on tape :

```
L:=[0,1,2];L[1]:=5
```

On obtient :

```
[0,5,2]
```

On tape en mode `Xcas` (les indices commencent à 0)

```
subsop([0,1,2],'1=NULL')
```

On obtient :

```
[0,2]
```

On tape en mode `Mupad TI` (les indices commencent à 1) :

```
subsop([0,1,2],2=5)
```

Ou on tape :

```
L:=[0,1,2];L[2]:=5
```

On obtient :

```
[0,5,2]
```

Attention, en mode `Maple` les arguments sont permutés et les indices commencent à 1.
On tape :

```
subsop(2=5,[0,1,2])
```

Ou on tape :

```
L:=[0,1,2];L[2]:=5
```

On obtient :

```
[0,5,2]
```

6.40.13 Transformer une liste en séquence : `op makesuite`

`op` ou `makesuite` permet de transformer une liste en séquence.
Remarque :
`op` est une fonction plus générale (cf 6.16.3).
On tape :

```
op([0,1,2])
```

Ou on tape :

```
makesuite([0,1,2])
```

On obtient :

```
(0,1,2)
```

Remarque `op` transforme une liste `L` en séquence, donc `op(op(L))` est équivalent à `op(L)`. Pour différencier une séquence de longueur 1 de l'élément de cette séquence, `Xcas` affiche une séquence de longueur 1 avec les délimiteurs `seq[ ]`. Toutefois, l'évaluation d'une séquence de longueur 1 renvoie son élément.

6.40.14 Transformer une séquence en liste : `makevector []`

`makevector` permet de transformer une séquence en une liste.
On peut aussi mettre simplement des crochets autour d'une séquence pour transformer cette séquence en une liste.
On tape :

```
makevector(0,1,2)
```

On obtient :

```
[0,1,2]
```

On tape :

```
a:=(0,1,2)
```

On tape :

```
[a]
```

Ou on tape :

```
makevector(a)
```

On obtient :

```
[0,1,2]
```

6.40.15 Longueur d'une liste : `size nops length`

`size` ou `nops` ou `length` renvoie la longueur d'une liste (ou d'une chaîne).
On tape :

```
nops([3,4,2])
```

ou

```
size([3,4,2])
```

ou

```
length([3,4,2])
```

On obtient :

```
3
```

6.40.16 Longueur d'une liste de listes : `sizes`

`sizes` permet d'avoir la liste des longueurs d'une liste de listes.
On tape :

```
sizes([[3,4],[2]])
```

On obtient :

```
[2,1]
```

6.40.17 Concaténer deux listes ou une liste et un élément : `concat` `augment`

`concat` (ou `augment`) concaténe deux listes, ou concaténe une liste et un élément.
On tape :

```
concat([3,4,2],[1,2,4])
```

Ou on tape :

```
augment([3,4,2],[1,2,4])
```

On obtient :

```
[3,4,2,1,2,4]
```

On tape :

```
concat([3,4,2],5)
```

Ou on tape :

```
augment([3,4,2],5)
```

On obtient :

```
[3,4,2,5]
```

Attention

```
concat([[3,4,2]],[[1,2,4]])
```

On obtient :

```
[[3,4,2,1,2,4]]
```

6.40.18 Rajouter un élément à la fin d'une liste : `append`

`append` rajoute un élément à la fin d'une liste.
On tape :

```
append([3,4,2],1)
```

On obtient :

```
[3,4,2,1]
```

On tape :

```
append([1,2],[3,4])
```

On obtient :

```
[1,2,[3,4]]
```

6.40.19 Rajouter un élément au début d'une liste : `prepend`

`prepend` rajoute un élément au début d'une liste.
On tape :

```
prepend([3,4,2],1)
```

On obtient :

```
[1,3,4,2]
```

On tape :

```
prepend([1,2],[3,4])
```

On obtient :

```
[[3,4],1,2]
```

6.40.20 Trier : `sort`

`sort` a comme argument une liste ou une expression ou une chaine de caractères.

– Pour une liste, `sort` renvoie la liste triée selon l'ordre croissant.
  On tape :
  ```
  sort([3,4,2])
  ```
  On obtient :
  ```
  [2,3,4]
  ```
  Remarque Si la liste est une matrice (resp une liste de chaines de caractères), `sort` renvoie la matrice dont les lignes sont triées selon l'ordre croissant en triant les lignes selon l'ordre $[a_0, a_1, ..a_n] < [b_0, b_1, ..b_n]$ si $a_0 < b_0$ ou si il existe $k \leq n$ tel que $a_0 = b_0$,...$a_{k-1} < b_{k-1}$ et $a_k < b_k$ (resp la liste de chaines de caractères triées par l'ordre lexicographique).
  On tape :
  ```
  sort([[3,4,2],[4,2,3],[2,3,4],[2,4,3]])
  ```
  On obtient :
  ```
  [[2,3,4],[2,4,3],[3,4,2],[4,2,3]]
  ```
  Pour trier par ordre décroissant il faut mettre en second argument une fonction de tri, par exemple on tape la fonction booléenne `superoueg` qui renvoie `1` si `L1>=L2` et `0` sinon :

```
superoueg(L1,L2):={
  local s,j;
  s:=min(size(L1),size(L2))-1;
  j:=0;
  tantque L1[j]==L2[j] and j<s faire
    j:=j+1;
  ftantque;
  //si L2[j]>L1[j] alors return 0 sinon return 1; fsi;
  si [sort(L1[j],L2[j])]==[L1[j],L2[j]] alors
     return 0
```

```
    sinon
     return 1;
  fsi;
}:;
```

On a remplacé `L2[j]>L1[j]` par `[sort(L1[j],L2[j])]==[L1[j],L2[j]]` pour que `superoueg` soit aussi valable pour les chaines de caractères. On tape :

`sort([[3,4,2],[4,2,3],[2,3,4],[2,4,3]],superoueg)`

ou on tape

`sort([[3,4,2],[4,2,3],[2,3,4],[2,4,3]],(x,y)->superoueg(x,y))`

On obtient :

`[[2,3,4],[2,4,3],[3,4,2],[4,2,3]][[2,3,4],[2,4,3],[3,4,2],[4,2,3]]`

On tape :

`sort(["dac","bac","sac","asc","cab"])`

On obtient :

`["asc","bac","cab","dac","sac"]`

On tape :

`sort(["dac","bac","sac","asc","cab"],superoueg)`

On obtient :

`["sac","dac","cab","bac","asc"]`

– Pour une expression, `sort` trie et collecte les termes égaux dans les sommes et produits.
  On tape :

  `sort(exp(2*ln(x))+x*y-x+y*x+2*x)`

  On obtient :

  `2*x*y+exp(2*ln(x))+x`

  On tape :

  `simplifier(exp(2*ln(x))+x*y-x+y*x+2*x)`

  On obtient :

  `x^2+2*x*y+x`

Remarque

`sort` accepte un 2-ième argument après une liste qui est la fonction de tri, par exemple (x,y)->x>y pour avoir la liste triée selon l'ordre décroissant.

Attention La fonction de tri f doit définir un ordre strict faible c'est à dire que

– $f(x, y)$ est un fonction renvoyant 0 (faux) ou 1 (vrai) qui est toujours définie (f doit renvoyer 0 si 2 éléments ne sont pas comparables) et doit vérifier :
– f doit définir une relation transitive (si $f(x, y)$ et $f(y, z)$ sont vrais alors $f(x, z)$ est vrai)
– on ne peut pas avoir $f(x, y)$ et $f(y, x)$ vrai en même temps (antisymétrie **non réflexive**)
– f ne définit pas forcément une relation d'ordre total, on peut avoir $f(x, y)$ et $f(y, x)$ simultanément faux. Si on définit la relation E par $x\ E\ y$ est vrai lorsque $f(x, y)$ et $f(y, x)$ sont simultanément faux alors E doit être une relation d'équivalence.

Sinon l'algorithme employé risque de boucler.... Par exemple, on ne peut pas mettre comme fonction de tri : `(x,y)->x[1]>=y[1]`. On tape :

```
sort([3,4,2],(x,y)->x>y)
```

On obtient :

```
[4,3,2]
```

Pour trier des listes de listes
`sort` trie les listes de listes par ordre croissant.

```
sort([[1,3],[2,4],[2,3],[1,4]])
```

On obtient :

```
[[1,3],[1,4],[2,3],[2,4]]
```

Pour un ordre différent, il faut mettre une fonction de tri comme 2ième argument.
Par exemple :
Si on veut trier par ordre décroissant la première colonne ou en cas d'égalité par ordre décroissant la 2ième colonne (ordre lexicographique).
On tape :

```
sort([[1,3],[2,4],[2,3],[1,4]],(x,y)->when(x[0]==y[0],x[1]>y[1],x[0]>
```

On obtient :

```
[[2,4],[2,3],[1,4],[1,3]]
```

Et si on veut trier par ordre décroissant la 2ième colonne ou en cas d'égalité par ordre décroissant la première colonne.
On tape :

```
sort([[1,3],[2,4],[2,3],[1,4]],
(x,y)->when(x[1]==y[1],x[0]>y[0],x[1]>y[1]))
```

On obtient :

```
[[2,4],[1,4],[2,3],[1,3]]
```

Attention Dans l'exemple précédent,

– on ne peut pas mettre comme fonction de tri `(x,y)->x[1]>=y[1]` car l'ordre n'est pas strict
– on peut mettre comme fonction de tri `f:=(x,y)->x[1]>y[1]` bien que l'ordre ne soit pas total.

Soient :
`L1:=[[1,2],[2,3],[4,3]]`
`L2:=[[1,2],[4,3],[2,3]]`
Dans ce cas `sort(L1,(x,y)->x[1]>y[1])` et `sort(L2,(x,y)->x[1]>y[1])` renvoient des réponses différentes parce que l'ordre n'est pas total et que `[2,3]` et `[4,3]` sont considérés comme équivalents.

6.40.21 Trier une liste selon l'ordre croissant : `SortA et sorta`

Attention `SortA` et `sorta` ne sont pas des synonymes : `SortA` s'utilise avec la syntaxe TI i.e. sans avoir besoin de mettre des parenthèses et modifie la valeur de l'argument alors que `sorta` s'utilise comme des fonctions normales de `Xcas` (parenthèses obligatoires et sans changer l'argument !).
`SortA` ou `sorta` a comme argument une liste, une séquence ou une matrice.
`SortA` ou `sorta` renvoie la liste ou la séquence triée selon l'ordre croissant.
Si l'argument est une matrice, `SortA` ou `sorta` trie la 1-ière ligne de la matrice selon l'ordre croissant, et reporte les manœuvres de tri de la 1-ière ligne sur les autres lignes c'est à dire le tri conserve les colonnes de la matrice.
On tape :

```
sorta([3,4,2])
```

ou

```
SortA([3,4,2])
```

ou

```
SortA [3,4,2]
```

ou

```
SortA 3,4,2
```

On obtient :

```
[2,3,4]
```

On tape :

```
sorta([[3,4,2],[6,4,5]])
```

ou

```
SortA([[3,4,2],[6,4,5]])
```

ou

```
SortA [[3,4,2],[6,4,5]]
```

ou

```
SortA [3,4,2],[6,4,5]
```

On obtient :

```
[[2,3,4],[5,6,4]]
```

On tape :

```
A:=[[3,4,2],[6,4,5]]
```

```
SortA(A)
```

On obtient :

`[[2,3,4],[5,6,4]]`

et maintenant `A` vaut `[[2,3,4],[5,6,4]]` Mais, on tape :

`A:=[[3,4,2],[6,4,5]]`

`sorta(A)`

On obtient :

`[[2,3,4],[5,6,4]]`

et maintenant `A` vaut toujours `[[3,4,2],[6,4,5]]`

Attention La syntaxe sans parenthèse peut vous jouer des tours !
Par exemple, on tape :
`L1:=[1,3,2];L2:=[2,3,1];`
`SortA L1,SortA L2`
On obtient la matrice ! : `[[1,2,3],[3,2,1]]`
qui est le résultat de :
`SortA(L1,SortA(L2))`

6.40.22 Trier une liste selon l'ordre décroissant : `SortD sortd`

Attention et `sortd` ne sont pas des synonymes : `SortD` s'utilise avec la syntaxe TI i.e. sans avoir besoin de mettre des parenthèses et modifie la valeur de l'argument alors que `sortd` s'utilise comme des fonctions normales de `Xcas` (parenthèses obligatoires et sans changer l'argument !).
`SortD` ou `sortd` a comme argument une liste, une séquence ou une matrice.
`SortA` ou `sorta` renvoie la liste ou la séquence triée selon l'ordre décroissant.
Si l'argument est une matrice, `SortD` ou `sortd` trie la 1-ière ligne de la matrice selon l'ordre décroissant, et reporte les manœuvres de tri de la 1-ière ligne sur les autres lignes c'est à dire le tri conserve les colonnes de la matrice.
On tape :

`sortd([3,4,2])`

ou

`SortD([3,4,2])`

ou

`SortD [3,4,2]`

ou

`SortD 3,4,2`

On obtient :

`[4,3,2]`

On tape :

```
sortd([[3,4,2],[6,4,5]])
```

ou

```
SortD([[3,4,2],[6,4,5]])
```

ou

```
SortD [[3,4,2],[6,4,5]]
```

ou

```
SortD [3,4,2],[6,4,5]
```

On obtient :

```
[[4,3,2],[4,6,5]]
```

On tape :

```
A:=[[3,4,2],[6,4,5]]
SortD(A)
```

On obtient :

```
[[4,3,2],[4,6,5]]
```

et maintenant `A` vaut `[[4,3,2],[4,6,5]]` Mais, on tape :

```
A:=[[3,4,2],[6,4,5]]
sortd(A)
```

On obtient :

```
[[4,3,2],[4,6,5]]
```

et maintenant `A` vaut toujours `[[3,4,2],[6,4,5]]`

Attention La syntaxe sans parenthèse peut vous jouer des tours !
Par exemple, on tape :
`L1:=[1,3,2];L2:=[2,3,1];`
`SortD L1,SortD L2`
On obtient la matrice ! : `[[3,2,1],[2,1,3]]`
qui est le résultat de :
`SortD(L1,SortD(L2))`

6.40.23 Sélectionner des éléments d'une liste : `select`

`select` a deux paramètres : une fonction booléenne `f` et une liste `L`.
`select` sélectionne les éléments `c` de la liste `L`, qui vérifie `f(c)=true`.
On tape :

```
select(x->(x>=2),[0,1,2,3,4,5])
```

On obtient :

```
[2,3,4,5]
```

6.40.24 Supprimer des éléments d'une liste : `remove`

`remove` a deux paramètres : une fonction booléenne `f` et une liste `L`.
`remove` enlève les éléments `c` de la liste `L`, qui vérifie `f(c)=true`.
On tape :

```
remove(x->(x>=2),[0,1,2,3,4,5])
```

On obtient :

```
[0,1]
```

Remarque Pour faire la même chose avec une chaine de catactère, par exemple, enlever tous les "a" d'une chaine :
On tape :

```
ord("a")
```

On obtient :

```
97
```

On tape :

```
f(chn):={local l:=length(chn)-1; return
remove(x->(ord(x)==97),seq(chn[k],k,0,l));}
```

Puis on tape :

```
f("abracadabra")
```

On obtient :

```
["b","r","c","d","b","r"]
```

Puis on tape :

```
char(ord(["b","r","c","d","b","r"]))
```

On obtient :

```
"brcdbr"
```

6.40.25 Tester si un élément est dans une liste : `member`

`member` a deux paramètres : un élément `c` et une liste (ou un ensemble) `L`.
`member` est une fonction qui teste si l'élément `c` est dans la liste `L`.
`member` renvoie `0` si `c` n'est pas dans `L`, et sinon renvoie :
`1+"l'indice de sa première apparition"`.
Attention à l'ordre des paramètres, c'est pour des raisons de compatibilité !
On tape :

```
member(2,[0,1,2,3,4,2])
```

On obtient :

```
3
```

On tape :

```
member(2,% {0,1,2,3,4,2% })
```

On obtient :

```
3
```

6.40.26 Tester si un élément est dans une liste : `contains`

`contains` a deux paramètres : une liste (ou un ensemble) `L` et un élément `c`.
`contains` est une fonction qui teste si l'élément `c` est dans la liste `L`.
`contains` renvoie `0` si `c` n'est pas dans `L`, et sinon renvoie :
`1+"l'indice de sa première apparition"`.
Attention, à l'ordre des paramètres, c'est pour des raisons de compatibilité !
On tape :

```
contains([0,1,2,3,4,2],2)
```

On obtient :

```
3
```

On tape :

```
contains(% {0,1,2,3,4,2% },2)
```

On obtient :

```
3
```

6.40.27 Compter les éléments d'une liste ou d'une matrice vérifiant une propriété : `count`

`count` a un, deux ou trois paramètres :

1. une liste d'entiers L
2. – une fonction réelle `f`,
   – liste `L` de longueur `n` ou une matrice `A` de dimension `p*q`,
   – un argument optionnel `row` ou `col`, dans le cas où le deuxième paramètre est une matrice `A`.

Lorsque `count` a :

– un paramètre qui est une liste d'entiers L, `count(L)` compte le nombre d'occurrences en renvoyant une matrice de 1ière colonne les éléments de la liste L triée et de 2ième colonne l'effectif de cet élément dans la liste.
– deux paramètres, `count` applique la fonction aux éléments de la liste (ou de la matrice) et en renvoie la somme, c'est à dire,
`count(f,L)` renvoie le nombre `f(L[0])+f(L[1])+..f(L[n-1])`
ou
`count(f,A)` renvoie le nombre `f(A[0,0])+....+f(A[p-1,q-1])`.
Si `f` est une fonction boolénne `count` renvoie le nombre d'éléments de la liste (ou de la matrice) pour lesquels la fonction boolénne est vraie.
– ou trois paramètres, `count` applique la fonction aux éléments de chaque ligne (resp colonne) de la matrice `A` si l'argument optionnel est `row` (resp `col`) et renvoie une liste de longueur p ayant comme $k$ième élément :
`f(A[k,0])+...f(A[k,q-1])` (resp une liste de longueur q ayant comme $k$ième élément : `f(A[0,k])+..f(A[p-1,k])`).

On tape :

```
count([1,3,1,1,2,10,3])
```

On obtient :

```
[[1,3],[2,1],[3,2],[10,1]]
```

On tape :

```
count((x)->x,[2,12,45,3,7,78])
```

Ou on tape :

```
count((x)->x,[[2,12,45],[3,7,78]])
```

On obtient :

```
147
```

car on a : 2+12+45+3+7+78=147.
On tape :

```
count((x)->x,[[2,12,45],[3,7,78]],row)
```

On obtient :

```
[59,88]
```

car on a : 2+12+45=59 et 3+7+78=88.
On tape :

```
count((x)->x,[[2,12,45],[3,7,78]],col)
```

On obtient :

```
[5,19,123]
```

car on a : 2+3=5,12+7=10,45+78=123.
On tape :

```
count((x)->x<12,[2,12,45,3,7,78])
```

On obtient :

```
3
```

On tape :

```
count((x)->x==12,[2,12,45,3,7,78])
```

Ou on tape :

```
count((x)->x==12,[[2,12,45],[3,7,78]])
```

On obtient :

```
1
```

On tape :

```
count((x)->x>12,[2,12,45,3,7,78])
```

On obtient :

```
2
```

On tape :

```
count(x->x^2,[3,5,1])
```

On obtient :

```
35
```

En effet on a : $3^2 + 5^2 + 1^1 = 35$.
On tape :

```
count(id,[3,5,1])
```

On obtient :

```
9
```

En effet, `id` est la fonction identité et on a : 3+5+1=9.
On tape :

```
count(1,[3,5,1])
```

On obtient :

```
3
```

En effet, `1` est la fonction constante égale à 1 et on a : 1+1+1=3.

6.40.28 Nombre d'éléments ayant une valeur donnée : `count_eq`

`count_eq` a deux ou trois et paramètres : une nombre et une liste (ou une matrice) et dans le cas où le deuxième paramètre est une matrice, un argument optionnel `row` ou `col`.
`count_eq` renvoie le nombre d'éléments de la liste (ou de la matrice) qui sont égaux au premier argument. Dans le cas où il y a un argument optionnel `row` (resp `col`) `count_eq` agit sur chacune des lignes (resp colonnes) de la matrice et renvoie alors une liste.
On tape :

```
count_eq(12,[2,12,45,3,7,78])
```

Ou on tape :

```
count_eq(12,[[2,12,45],[3,7,78]])
```

On obtient :

```
1
```

On tape :

```
count_eq(12,[[2,12,45],[3,7,78]],row)
```

On obtient :

`[1,0]`

On tape :

`count_eq(12,[[2,12,45],[3,7,78]],col)`

On obtient :

`[0,1,0]`

Remarque
Les deux paramètres de `count_eq` ne sont pas forcément numériques : `count_eq(ab,[[-ab,1,ab`
renvoie 4 si la variable `ab` n'est pas affectée, mais renverra 6 si il y a `0` dans `ab`
`count_eq("ab",["ab",1,"ab",1,"ab",-3])` renvoie 3. Attention ! ! !
si la variable `ab` n'est pas affectée,
`count_eq(ab+1-1,[[-ab,1,ab,1,ab,-3],[-ab,1,ab,1,ab,-3]])`
renvoie 4,
mais `count_eq(ab+1,[-ab,1,ab+1,1,1+ab,-3])` renvoie 1.

6.40.29 Nombre d'éléments inférieurs à une valeur : `count_inf`

`count_inf` a deux ou trois paramètres : une nombre et une liste (ou une matrice) et dans le cas où le deuxième paramètre est une matrice, un argument optionnel `row` ou `col`.
`count_inf` renvoie le nombre d'éléments de la liste (ou de la matrice) qui sont strictement inférieurs au premier argument. Dans le cas où il y a un argument optionnel `row` (resp `col`) `count_inf` agit sur chacune des lignes (resp colonnes) de la matrice et renvoie alors une liste.
On tape :

`count_inf(12,[2,12,45,3,7,78])`

Ou on tape :

`count_inf(12,[[2,12,45],[3,7,78]])`

On obtient :

`3`

On tape :

`count_inf(12,[[2,12,45],[3,7,78]],row)`

On obtient :

`[1,2]`

On tape :

`count_inf(12,[[2,12,45],[3,7,78]],col)`

On obtient :

`[2,1,0]`

6.40.30 Nombre d'éléments supérieurs à une valeur : count_sup

count_sup a deux paramètres : une nombre et une liste réelle (ou une matrice réelle) et dans le cas où le deuxième paramètre est une matrice, un argument optionnel row ou col.
count_sup renvoie le nombre d'éléments de la liste (ou de la matrice) qui sont strictement supérieurs au premier argument. Dans le cas où il y a un argument optionnel row (resp col) count_sup agit sur chacune des lignes (resp colonnes) de la matrice et renvoie alors une liste.
On tape :

```
count_sup(12,[2,12,45,3,7,78])
```

Ou on tape :

```
count_sup(12,[[2,12,45],[3,7,78]])
```

On obtient :

```
2
```

On tape :

```
count_sup(12,[[2,12,45],[3,7,78]],row)
```

On obtient :

```
[1,1]
```

On tape :

```
count_sup(12,[[2,12,45],[3,7,78]],col)
```

On obtient :

```
[0,0,2]
```

6.40.31 Somme des éléments d'une liste : sum add

sum ou add a comme paramètre l une liste (ou une séquence) de nombres réels ou de décimaux.
sum ou add renvoie la somme des éléments de l.
On tape :

```
sum(2,3,4,5,6)
```

On obtient :

```
20
```

6.40.32 Somme cumulée des éléments d'une liste : `cumSum`

`cumSum` a comme paramètre `l` une liste (ou une séquence) de nombres réels ou de décimaux ou de chaîne de caractères.
`cumSum` renvoie la liste (ou la séquence) de même longueur que `l` avec comme k-ième élément la somme (ou la concaténation) des éléments `l[0]`, .., `l[k]`.
On tape :

```
cumSum(sqrt(2),3,4,5,6)
```

On obtient :

```
sqrt(2),3+sqrt(2),3+sqrt(2)+4,3+sqrt(2)+4+5,
3+sqrt(2)+4+5+6
```

On tape :

```
normal(cumSum(sqrt(2),3,4,5,6))
```

On obtient :

```
sqrt(2),sqrt(2)+3,sqrt(2)+7,sqrt(2)+12,sqrt(2)+18
```

On tape :

```
cumSum(1.2,3,4.5,6)
```

On obtient :

```
1.2,4.2,8.7,14.7
```

On tape :

```
cumSum([0,1,2,3,4])
```

On obtient :

```
[0,1,3,6,10]
```

On tape :

```
cumSum("a","b","c","d")
```

On obtient :

```
"a","ab","abc","abcd"
```

On tape :

```
cumSum("a","ab","abc","abcd")
```

On obtient :

```
"a","aab","aababc","aababcabcd"
```

6.40.33 Produit indicé : `product` `mul`

Voir aussi 6.40.33, 6.45.6 et 6.45.8).

Produit des valeurs d'une expression : `product`

`product` ou `mul` a 4 ou 5 arguments.

- Avec 5 arguments `product(Xpr,Var,a,b,p)` ou `mul(Xpr,Var,a,b,p)` renvoie le produit demandé c'est à dire renvoie le produit des valeurs de l'expression `Xpr` quand la variable `Var` va de `a` à `b` avec un pas égal à `p`.
  On tape :
  `product(x^2+1,x,1,5,2)`
  Ou on tape :
  `mul(x^2+1,x,1,5,2)`
  On obtient :
  `520`
  En effet :
  $2 * 10 * 26 = 520$
- Avec 4 arguments, `product(Xpr,Var,a,b)` ou `mul(Xpr,Var,a,b)` n'a pas la même valeur selon que `a` est plus petit ou égal à `b` ou non car on veut avoir l'égalité :
  `product(Xpr,Var,a,b)= product(Xpr,Var,a,c)*product(Xpr,Var,c+1,b)`.
  Aussi, lorsque le pas `p` n'est pas précisé on a :
  - si `a` est inférieur à `b`,
    `product(Xpr,Var,a,b)` renvoie le produit des valeurs de l'expression `Xpr` quand la variable `Var` va de `a` à `b` avec un pas de 1 : cette syntaxe est compatible avec Maple.
    Ainsi si `a<= b` on a :
    `product(Xpr,Var,a,b)=product(Xpr,Var,a,b,1)`.
    On tape :
    `product(x^2+1,x,1,4)`
    Ou on tape :
    `mul(x^2+1,x,1,4)`
    On obtient :
    `1700`
    En effet :
    $2 * 5 * 10 * 17 = 1700$
  - si `a` est supérieur à `b+1`,
    `product(Xpr,Var,a,b)` renvoie l'inverse du produit des valeurs de l'expression `Xpr` quand la variable `Var` va de `b+1` à `a-1` avec un pas `1` : cette syntaxe est compatible avec Maple.
    On tape :
    `product(x^2+1,x,4,1)`
    Ou on tape :
    `mul(x^2+1,x,4,1)`
    On obtient :
    `1/50`
    En effet :

$1/(5*10) = 1/50$

- si `a` est égal à `b+1`,
  `product(Xpr,Var,b+1,b)` renvoie 1.
  On tape :
  `product(x^2+1,x,5,4)`
  Ou on tape :
  `mul(x^2+1,x,5,4)`
  On obtient :
  `1`

Produit des éléments d'une liste : `product`

Voir aussi `.*` (cf 6.41.5) et `hadamard` pour les matrices (cf 6.45.8).
`product` ou `mul` peut aussi avoir 1 ou 2 arguments :
une liste de nombres réels (ou de décimaux) ou
deux listes de même longueur.
(voir aussi 6.40.33, 6.45.6 et 6.45.8).

- si `product` ou `mul` a un argument `l`, `product` ou `mul` renvoie le produit des éléments de `l`.
  On tape :
  `product([2,3,4])`
  On tape :
  `mul([2,3,4])`
  On obtient :
  `24`
  On tape :
  `product([[2,3,4],[5,6,7]])`
  On obtient :
  `[10,18,28]`
  -
- si `product` ou `mul` a deux arguments `l1` et `l2` (qui sont deux listes ou deux matrices), `product` ou `mul` renvoie le produit terme à terme des éléments de `l1` et des éléments de `l2`.
  On tape :
  `product([2,3,4],[5,6,7])`
  Ou on tape :
  `mul([2,3,4],[5,6,7])`
  On obtient :
  `[10,18,28]`
  On tape :
  `product([[2,3,4],[5,6,7]],[[2,3,4],[5,6,7]])`
  Ou on tape :
  `mul([[2,3,4],[5,6,7]],[[2,3,4],[5,6,7]])`
  On obtient :
  `[[4,9,16],[25,36,49]]`

6.40.34 Appliquer une fonction d'une variable aux éléments d'une liste : `map apply of`

`map` ou `apply` ou `of` sert à appliquer une fonction aux éléments d'une liste. Mais ces trois instructions ne sont pas des synonymes. On a :

- `of` a 2 paramètres une fonction `f` et une expression `E` ou une liste `L`. `of(f,E)` renvoie `f(E)` et `of(f,L)` renvoie `f(L)`. En effet, `of` est la traduction interne des parenthèses : `f(x)` est traduit en interne par `of(f,x)`. On peut donc utiliser directement `f(x)`,
- `apply` a 2 paramètres une fonction `f` et une liste `L`. `apply(f,L)` renvoie `[f(L[0]),f(L[1]),...f(L[size(L)-1])]`. **Attention** `apply` rèpond `[]` si le deuxième élément n'est pas une liste.

- `map` a 2 paramètres une expression `E` ou une liste `L` et une fonction `f`. `map(E,f)` renvoie `f(E)` et `map(L,f)` renvoie `[f(L[0]),f(L[1]),...f(L[size(L)-1])]`. **Attention** à l'ordre des paramètres qui n'est pas le même pour `map` et pour `apply` , c'est pour des raisons de compatibilité ! ! !.
  Lorsque la liste est une matrice et que la fonction doit s'appliquer à chaque élément d'une matrice il faut mettre `matrix` comme argument optionnel à `map`

On tape :

```
apply(x->x+1,[3,5,1])
```

ou

```
map([3,5,1],x->x+1)
```

cela ajoute 1 à chaque élément de la liste et on obtient :

```
[4,6,2]
```

On tape

```
of(x->x+1,[3,5,1])
```

et puisque `[3,5,1]+1=[3,5,2]`, on obtient :

```
[3,5,2]
```

Exemple avec une matrice
On tape :

```
apply(x->x+1,[[3,5,1],[3,5,1],[3,5,1]])
```

ou

```
map([[3,5,1],[3,5,1],[3,5,1]],x->x+1)
```

cela ajoute 1 à chaque élément de la liste c'est à dire à chaque ligne de la matrice et comme `[3,5,1]+1=[3,5,2]`, on obtient :

```
[[3,5,2],[3,5,2],[3,5,2]]
```

On tape :

```
of(x->x+1,[[3,5,1],[3,5,1],[3,5,1]])
```

cela ajoute 1 c'est à dire la matrice identité à la matrice et on obtient :

```
[[4,5,1],[3,6,1],[3,5,2]]
```

On tape :

```
map([[3,5,1],[3,5,1],[3,5,1]],x->x+1,matrix)
```

cela ajoute 1 à chaque élément de la matrice et on obtient :

```
[[4,6,2],[4,6,2],[4,6,2]]
```

Autres exemples On tape :

```
apply(x->x^2,[3,5,1])
```

ou

```
of(x->x^2,[3,5,1])
```

ou

```
map([3,5,1],x->x^2)
```

ou on définit la fonction $h(x) = x^2$ en tapant :

```
h(x):=x^2
```

puis

```
apply(h,[3,5,1])
```

ou

```
of(h,[3,5,1])
```

ou

```
map([3,5,1],h)
```

On obtient :

```
[9,25,1]
```

On tape :

```
apply(h,[[3,5,1],[3,5,1],[3,5,1]])
```

ou

```
map([[3,5,1],[3,5,1],[3,5,1]],h)
```

ou

```
map([[3,5,1],[3,5,1],[3,5,1]],h,matrix)
```

On obtient chaque èlement au carré :

```
[[9,25,1],[9,25,1],[9,25,1]]
```

On tape :

```
of(h,[[3,5,1],[3,5,1],[3,5,1]])
```

On obtient le carré de la matrice :

```
[[27,45,9],[27,45,9],[27,45,9]]
```

On définit la fonction $g(x) = [x, x^2, x^3]$ en tapant :

```
g(x):=[x,x^2,x^3]
```

ou

```
g:=(x)->[x,x^2,x^3]
```

puis, on tape :

```
apply(g,[3,5,1])
```

ou

```
map([3,5,1],g)
```

On fait agir g sur 3, puis sur 5, puis sur 1 et on obtient :

```
[[3,9,27],[5,25,125],[1,1,1]]
```

On tape :

```
of(g,[3,5,1])
```

On fait agir g sur la liste [3,5,1] et on obtient :

```
[[3,5,1],[9,25,1],[27,125,1]]
```

Remarque
Si `l1,l2,l3` sont des listes :
`sizes([l1,l2,l3])=map(size,[l1,l2,l3])`

6.40.35 Appliquer une fonction de plusieurs variables à un polynôme donné au format interne :`map`

Le polynôme $x^2 + 2xy + y^2$ s'écrit au format interne :
`% % % {1,[2,0]% % % }+% % % {2,[1,1]% % % }+% % % {1,[0,2]% % % }`
On peut transformer $x^2+2xy+y^2$ au format interne avec la commande `symb2poly` (voir 6.26.6). Par exemple `p:=symb2poly((x+y)^2,[x,y])` renvoie `% % % {1,[2,0]% % % }+% % % {2,[1,1]% % % }+% % % {1,[0,2]% % % }`
Pour un polynôme de n variables, f va agir sur les coefficients de chaque monôme du polynôme.

Pour agir sur un monôme de n variables, f doit être une fonction de $n+1$ variables, on considère que ces variables représentent le coefficient, l'exposant de la première variable,...,l'exposant de la n-ième variable et f transforme le coefficient de ce monôme.
Par exemple, si le monôme est $5x^2y^3$, au format interne il s'écrit : `% % % {5,[2,3]% % % }`
et si f est dédfinie par :
`f(a,b,c):=a*(2*b+3*c)`
alors
`map(% % % {5,[2,3]% % % },f)` renvoie le monôme $f(5,2,3)x^2y^3 = 65x^2y^3$ écrit au format interne, c'est à dire `% % % {65,[2,3]% % % }`
On tape :
`map(% % % {5,[2,3]% % % },(a,b,c)->a*(2b+3c))`
On obtient :
`% % % {65,[2,3]% % % }`
On tape :
`map(% % % {1,[2,0]% % % }+% % % {2,[1,1]% % % }+% % % {1,[0,2]% % % },(a,b,c)->a*(b+2c))`
On obtient :
`% % % {2,[2,0]% % % }+% % % {6,[1,1]% % % }+% % % {4,[0,2]% % % }` car si f est la fonction $(a,b,c)- > a*(b+2c)$ alors $f(1,2,0) = 2, f(2,1,1) = 6, f(1,0,2) = 4$

6.40.36 Appliquer une fonction de 2 variables aux éléments de 2 listes : `zip`

`zip` sert à appliquer une fonction de 2 variables aux éléments de 2 listes.
On tape :

```
zip('sum',[a,b,c,d],[1,2,3,4])
```

On obtient :

```
[a+1,b+2,c+3,d+4]
```

On tape :

```
zip((x,y)->x^2+y^2,[4,2,1],[3,5,1])
```

Ou on tape :

```
f:=(x,y)->x^2+y^2
```

puis,

```
zip(f,[4,2,1],[3,5,1])
```

On obtient :

```
[25,29,2]
```

On tape :

```
f:=(x,y)->[x^2+y^2,x+y]
```

puis,

```
zip(f,[4,2,1],[3,5,1])
```

On obtient :

```
[[25,7],[29,7],[2,2]]
```

6.40.37 Faire une liste de zéros : `newList`

`newList(n)` fabrique une liste formée de `n` zéros.
On tape :

```
newList(3)
```

On obtient :

```
[0,0,0]
```

6.40.38 Faire une liste avec une fonction : `makelist`

`makelist` fabrique une liste à l'aide d'une fonction, en donnant les bornes de la variable et le pas de cette variable qui par défaut vaut 1 ou -1 selon que l'ordre des bornes.
On tape :

```
makelist(x->x^2,3,5)
```

ou bien

```
makelist(x->x^2,3,5,1)
```

ou on définit la fonction $h(x) = x^2$ en tapant : `h(x):=x^2` puis

```
makelist(h,3,5,1)
```

On obtient :

```
[9,16,25]
```

On tape :

```
makelist(x->x^2,3,6,2)
```

On obtient :

```
[9,25]
```

Attention ! ! ! il faut purger `x` si `x` est affecté.

6.40.39 Faire une liste aléatoire : `randvector`

`randvector` fabrique une liste de nombres aléatoires.
`randvector` a comme argument un entier n et éventuellement un deuxième argument, soit un entier k, soit le nom quoté ou non quoté de la loi de distribution des nombres aléatoires de la liste (voir aussi 6.26.32, 6.44.3 et 7.3.9).
`randvector` renvoie une liste d'ordre n constituée d'entiers aléatoires uniformément distribués entre -99 et 99 (par défaut) ou entre 0 et $k-1$ ou uneliste d'ordre n de nombres aléatoires distribués selon la loi mise entre-quote ou en paramètre.
Lorsque `randvector` a comme argument un entier `n` et une loi aléatoire de `Xcas` qu'il faut quoter ou pas dans ce cas, `randvector` renvoie une liste de dimension `n` dont les éléments sont pris au hasard selon la fonction donnée en troisième argument.
Les fonctions données en deuxième argument qui doivent être quoter ou non et peuvent être :
`'rand(n)'`
`'binomial(n,p)'` ou `binomial,n,p` ou `'randbinomial(n,p)'`
`'multinomial(P,K)'` ou `multinomial,P,K` ou `'randmultinomial(P,K)'`
`'poisson(`λ`)'` ou `poisson,` λ ou `'randpoisson(`λ`)'`
`'normald(`μ,σ`)'` ou `normald,`μ,σ ou `'randnorm(`μ,σ`)'`
`'exponential(a)'` ou `exponential,a` ou `'randexp(a)'`
`'fisher(n,m)'` ou `fisher,n,m` ou `'randfisher(n,m)'`
Attention la syntaxe sans quote marche avec les lois mais pas avec la commande `rand...` correspondante, donc par exemple les commandes `randvector(3,normald,0,1)` ou `randvector(3,'normald(0,1)')` ou `randvector(3,'randnorm(0,1)')` sont valables mais `randvector(3,randnorm,0,1)` n'est pas valable.
On tape :

```
randvector(3)
```

On obtient par exemple :

```
[-54,78,-29]
```

On tape :

```
randvector(3,5)
```

On tape :

```
randvector(3,'rand(5)')
```

On obtient par exemple :

```
[1,2,4]
```

On tape :

```
randvector(3,normald,0,1)
```

Ou on tape :

```
randvector(3,'normald(0,1)')
```

On obtient par exemple :

```
[1.39091705476,-0.136794772167,0.187312440336]
```

On tape :

```
randvector(3,2..4)
```

On obtient par exemple :

```
[3.92450003885,3.50059241243,2.7322040787]
```

On tape :

```
randvector(6,binomial,4,0.2)
```

Ou on tape :

```
randvector(6,'binomial(4,0.2)')
```

On obtient par exemple :

```
[0,1,0,2,2,0]
```

On effectue 6 fois 4 tirages avec une probabilité de succés de 0.2 et à chaque fois le nombre de succés a été de :
0 puis 1, puis 0, puis 2, puis 2, puis 0.
On tape :

```
randvector(6,multinomial,[1/2,1/3,1/6])
```

Ou on tape :

```
randvector(6,'multinomial([1/2,1/3,1/6])')
```

On obtient par exemple :

```
[3,2,1]
```

On effectue 6 fois le tirage d'un objet parmi 3 objets (tirage avec remise). Chaque objet a la probabilité [1/2,1/3,1/6] d'être tiré et ici on a obtenu :
3 fois l'objet ayant la probabilité 1/2 dêtre tirés, 2 fois l'objet ayant la probabilité 1/3 d'être tirés et 1 fois l'objet ayant la probabilité 1/6 d'être tirés. On tape :

```
randvector(10,multinomial,[1/2,1/3,1/6],[A,B,C])
```

Ou on tape :

```
randvector(10,'multinomial([1/2,1/3,1/6],[A,B,C])')
```

On obtient par exemple :

```
[A,C,A,A,B,B,C,A,A,B]
```

c'est à dire la liste des objets qui ont été tirés. On tape :

```
randvector(10,multinomial,[1/2,1/3,1/6],["R","V","B"])
```

Ou on tape :

```
randvector(10,'multinomial([1/2,1/3,1/6],["R","V","B"])')
```

On obtient par exemple :

```
["R","R","B","V","R","V","B","B","R","R"]
```

On tape :

```
randvector(6,poisson,1.3)
```

Ou on tape :

```
randvector(6,'poisson(1.3)')
```

On obtient par exemple :

```
[1,0,1,1,1,1]
```

On tape :

```
randvector(4,exponential,1.2)
```

Ou on tape :

```
randvector(4,'exponential(1.2)')
```

On obtient par exemple :

```
[1.67683756526,0.192937941271,0.580820253805,0.709352619633]
```

On tape :

```
randvector(5,fisher,4,6)
```

Ou on tape :

```
randvector(5,'fisher(4,6)')
```

On obtient par exemple :

```
[0.17289703163,1.03709368317,0.161051043162,1.4407877128,0.3586901042
```

6.40.40 Liste des différences de termes consécutifs : `deltalist`

`deltalist` permet d'obtenir la liste des différences de deux termes consécutifs de la liste donnée comme argument.
On tape :

```
deltalist([5,8,1,9])
```

On obtient :

```
[3,-7,8]
```

6.40.41 Faire une matrice avec une liste : `list2mat`

`list2mat` permet d'obtenir la matrice des termes de la liste donnée comme argument en scindant la liste selon le nombre de colonnes spécifiées. Si il manque des termes, la liste est complétée par des `0`.
On tape :

```
list2mat([5,8,1,9,5,6],2)
```

On obtient :

```
[[5,8],[1,9],[5,6]]
```

On tape :

```
list2mat([5,8,1,9],3)
```

On obtient :

```
[[5,8,1],[9,0,0]]
```

Remarque
En réponse, les délimiteurs d'une matrice sont [et] alors que les délimiteurs d'une liste sont `[` et `]` (la barre verticale des crochets est plus eppaisse pour les matrices).

6.40.42 Faire une liste avec une matrice : `mat2list`

`mat2list` permet d'obtenir la liste des termes de la matrice donnée comme argument.
On tape :

```
mat2list([[5,8],[1,9]])
```

On obtient :

```
[5,8,1,9]
```

6.41 Fonctions utiles pour les vecteurs et les composantes d'un vecteur

6.41.1 Les normes d'un vecteur : `maxnorm l1norm l2norm norm`

Voir aussi 6.48 pour les différentes instructions pour les normes d'une matrice.
Les différentes instructions pour les normes d'un vecteur sont :

- `maxnorm` pour calculer la norme l^∞ d'un vecteur : c'est le maximum des valeurs absolues de ses coordonnées.
  On tape :
  ```
  maxnorm([3,-4,2])
  ```
  Ou on tape :
  ```
  maxnorm(vecteur(3,-4,2))
  ```
  On obtient :

En effet : `x=3, y=-4, z=2` et `4=max(|x|,|y|,|z|)`.
Pour les matrices voir 6.48.4.

- `l1norm` pour calculer la norme l^1 d'un vecteur : c'est la somme des valeurs absolues de ses coordonnées.
On tape :

```
l1norm([3,-4,2])
```

Ou on tape :

```
l1norm(vecteur(3,-4,2))
```

On obtient :

```
9
```

En effet : `x=3, y=-4, z=2` et `9=|x|+|y|+|z|`.
Pour les matrices voir 6.48.4.

- `norm` ou `l2norm` pour calculer la norme l^2 d'un vecteur : c'est la racine carrée de la somme des carrés de ses coordonnées.
On tape :

```
norm([3,-4,2])
```

Ou on tape :

```
norm(vecteur(3,-4,2))
```

On obtient :

```
sqrt(29)
```

En effet : `x=3, y=-4, z=2` et $29 = |x|^2 + |y|^2 + |z|^2$.
Pour les matrices voir 6.48.4 et 6.48.1.

6.41.2 Pour normaliser les composantes d'un vecteur : `normalize` `unitV`

`normalize` ou `unitV` normalise les composantes d'un vecteur et renvoie les composantes d'un vecteur de norme 1 selon la norme l^2 (la racine carrée de la somme des carrés de ses coordonnées).
On tape :

```
normalize([3,4,5])
```

On obtient :

```
[3/(5*sqrt(2)),4/(5*sqrt(2)),5/(5*sqrt(2))]
```

En effet : `x=3, y=4, z=5` et $50 = |x|^2 + |y|^2 + |z|^2$.

6.41.3 Somme terme à terme de deux listes : `+` `.+`

La somme terme à terme de deux listes se fait avec l'opérateur infixé `+` ou `.+` et aussi avec l'opérateur prefixé `'+'`.
Si les deux listes n'ont pas la même longueur la liste la plus petit est complétée par des zéros.
Bien voir la différence avec les séquences car si l'opérateur infixé `+` a comme arguments deux séquences, il renvoie la somme des termes des deux séquences.
On tape :

```
[1,2,3]+[4,3,5]
```

Ou on tape :

`[1,2,3] .+[4,3,5]`

Ou on tape :

`'+'([1,2,3],[4,3,5])`

Ou on tape :

`'+'([[1,2,3],[4,3,5]])`

On obtient :

`[5,5,8]`

On tape :

`[1,2,3,4,5,6]+[4,3,5]`

Ou on tape :

`'+'([1,2,3,4,5,6],[4,3,5])`

Ou on tape :

`'+'([[1,2,3,4,5,6],[4,3,5]])`

On obtient :

`[5,5,8,4,5,6]`

Attention
Quand l'opérateur + est préfixé il doit être quoté c'est à dire écrit `'+'`.
Si on tape : On tape :

`[1,2,3,4,5,6]+4`

On obtient, car la liste est considèrée comme les coefficients d'un polynôme :

`[1,2,3,4,5,10]`

6.41.4 Différence terme à terme de deux listes : `-` `.-`

La différence terme à terme de deux listes se fait avec l'opérateur infixé `-` ou `.-` et aussi avec l'opérateur prefixé `'-'`.
Si les deux listes n'ont pas la même longueur la liste la plus petit est complétée par des zéros.
Bien voir la différence avec les séquences car si l'opérateur infixé `-` a comme arguments deux séquences, il renvoie la différence des sommes des termes de chacune des séquences.
On tape :

`[1,2,3]-[4,3,5]`

Ou on tape :

```
[1,2,3] .+ [4,3,5]
```

Ou on tape :

```
'-'([1,2,3],[4,3,5])
```

Ou on tape :

```
'-'([[1,2,3],[4,3,5]])
```

On obtient :

```
[-3,-1,-2]
```

Attention
Quand l'opérateur − est préfixé il doit être quoté c'est à dire écrit `'-'`.

6.41.5 Produit terme à terme de deux listes : `.*`

Voir aussi 6.45.9, `hadamard` pour les matrices (cf 6.45.8) et `product` pour les listes et les matrices (cf 6.45.6 et 6.40.33)
Le produit terme à terme de deux listes de même longueur se fait avec l'opérateur infixé `.*`.
On tape :

```
[1,2,3] .* [4,3,5]
```

On obtient :

```
[4,6,15]
```

On tape :

```
[[1,2],[4,3]] .* [[4,3],[5,6]]
```

On obtient :

```
[[4,6],[20,18]]
```

6.41.6 Quotient terme à terme de deux listes : `./`

Le quotient terme à terme de deux listes de même longueur se fait avec l'opérateur infixé `./`.
On tape :

```
[1,2,3] ./ [4,3,5]
```

On obtient :

```
[1/4,2/3,3/5]
```

6.41.7 Le produit scalaire : `scalar_product dotprod dot dotP * scalar_Product produit_scalaire`

`*`, opérateur infixé, calcule le produit scalaire des deux vecteurs dont les composantes sont données en argument.
`dot` ou `dotP` ou `dotprod` ou `scalar_product` ou `scalarProduct` calcule le produit scalaire des deux vecteurs dont les composantes sont données en argument.
On tape :

```
dot([1,2,3],[4,3,5])
```

ou :

```
scalar_product([1,2,3],[4,3,5])
```

ou :

```
produit_scalaire([1,2,3],[4,3,5])
```

ou :

```
produit_scalaire(vecteur([1,2,3]),vecteur([4,3,5]))
```

ou :

```
[1,2,3]*[4,3,5]
```

ou encore :

```
'*'([1,2,3],[4,3,5])
```

On obtient :

```
25
```

En effet `25=1*4+2*3+3*5`.
Attention
Le produit de deux listes de longueur n renvoie le prduit scalaire de deux vecteurs de $\mathbb{R}^n$ mais une liste élevée au carré renvoie la liste des carrées terme à terme : On tape :

```
[1,2,3]*[1,2,3]
```

On obtient :

```
25
```

On tape :

```
[1,2,3]^2
```

On obtient :

```
[1,4,9]
```

6.41.8 Le produit vectoriel : `cross crossP crossproduct`

`cross` ou `crossP` ou `crossproduct` calcule les composantes du produit vectoriel des deux vecteurs dont les composantes sont données en argument.
On tape :

```
cross([1,2,3],[4,3,2])
```

Ou on tape :

```
cross(vecteur([1,2,3]),vecteur([4,3,2]))
```

On obtient :

```
[-5,10,-5]
```

En effet : $-5 = 2*2-3*3$, $10 = -1*2+4*3$, $-5 = 1*3-2*4$.

6.42 Fonctions utiles pour les statistiques : `mean moyenne, variance, stddev ecart_type, stddevp, ecart_type_population, stdDev, median, quantile, quartiles, quartile1, quartile3, boxwhisker, moustache`

Voir aussi 6.45.37 and 7.
Fonctions utiles pour les statistiques dont les donn'ees sont des listes :

– `mean` ou `moyenne` pour calculer la moyenne des éléments d'une liste.
  On tape :
  ```
  mean([3,4,2])
  ```
  On obtient :
  ```
  3
  ```
  On tape :
  ```
  mean([1,0,1])
  ```
  On obtient
  ```
  2/3
  ```
– `stddev` ou `ecart_type` pour calculer l'écart type numèrique des éléments d'une liste.
  On tape :
  ```
  stddev([3,4,2])
  ```
  On obtient :
  ```
  sqrt(2/3)
  ```
  On a en effet la moyenne qui vaut 3 et l'écart type qui vaut :
  $\sqrt{((3-3)^2+(4-3)^2+(2-3)^2)/3} = \sqrt{2/3}$
– `stddevp` ou `stdDev` ou `ecart_type_population` pour calculer une estimation de l'écart type numèrique de la population à partir d'un échantillon dont les éléments sont donnés dans une liste.
  On tape :
  ```
  stddevp([3,4,2])
  ```
  On obtient :
  ```
  1
  ```

On a en effet la moyenne qui vaut 3 et l'écart type qui vaut :
$\sqrt{((3-3)^2+(4-3)^2+(2-3)^2)/2} = \sqrt{2/2} = 1$
On a la relation :
`stddevp(l)^2=size(l)*stddev(l)^2/(size(l)-1).`

– `variance` pour calculer la variance numèrique des éléments d'une liste.
  On tape :

```
variance([3,4,2])
```

  On obtient :

```
2/3
```

– `median` pour calculer la médiane des éléments d'une liste.
  On tape :

```
median([0,1,3,4,2,5,6])
```

  On obtient :

```
3.0
```

– `quantile` pour calculer les déciles des éléments d'une liste.
  On tape :

```
quantile([0,1,3,4,2,5,6],0.25)
```

  On obtient le premier quartile :

```
[1.0]
```

  On tape :

```
quantile([0,1,3,4,2,5,6],0.5)
```

  On obtient la médiane :

```
[3.0]
```

  On tape :

```
quantile([0,1,3,4,2,5,6],0.75)
```

  On obtient le troisième quartile :

```
[5.0]
```

– `quartiles` calcule le minimum, le premier quartile, la médiane, le troisième quartile et le maximum d'une serie statistique.
  On tape :

```
quartiles([0,1,3,4,2,5,6])
```

  On obtient :

```
[[0.0],[1.0],[3.0],[5.0],[6.0]]
```

– `quartile1` calcule le premier quartile d'une serie statistique.
  On tape :

```
quartile1([0,1,3,4,2,5,6])
```

  On obtient :

```
1.0
```

– `quartile3` calcule le troisième quartile d'une serie statistique.
  On tape :

```
quartile3([0,1,3,4,2,5,6])
```

  On obtient :

```
5.0
```

– `boxwhisker` ou `moustache` pour afficher la boite à moustaches des éléments d'une liste.
  On tape :

```
moustache([0,1,3,4,2,5,6])
```

On obtient le dessin de la boite à moustaches de la liste mise comme paramètre.
Soit `A` la liste `[0,1,2,3,4,5,6,7,8,9,10,11]`.
On tape :
`A:=[0,1,2,3,4,5,6,7,8,9,10,11]`
On obtient :
`11/2` pour `mean(A)`
`sqrt(143/12)` pour `stddev(A)`
`0` pour `min(A)`
`[1.0]` pour `quantile(A,0.1)`
`[2.0]` pour `quantile(A,0.25)`
`[5.0]` pour `median(A)` ou pour `quantile(A,0.5)`
`[8.0]` pour `quantile(A,0.75)`
`[9.0]` pour `quantile(A,0.9)`
`11` pour `max(A)`
`[[0.0],[2.0],[5.0],[8.0],[11.0]]` pour `quartiles(A)`
Voir aussi ces fonctions pour les matrices à la section 6.45.37 et pour les listes pondérées au chapitre 7.

6.43 Les tableaux indicés par des chaînes : `table`

Une table est une liste indicée par quelque chose de plus général que des entiers.
Une table peut être utilisée, par exemple, pour stocker des numèros de téléphone indicés par des noms.
Dans `Xcas`, les indices d'une table peuvent être n'importe quels objets de `Xcas`.
L'accés se fait par un algorithme qui trie par `type` puis utilise l'ordre de chaque type (par exemple $<$ pour le type numérique, l'ordre lexicographique pour les chaines etc...).
`table` a comme argument une liste ou une séquence d'égalité de la forme :
`"nom_index"=valeur_element`.
`table` renvoie cette table.
On tape :

```
T:=table(3=-10,"a"=10,"b"=20,"c"=30,"d"=40)
```

On tape :

```
T["b"]
```

On obtient :

```
20
```

On tape :

```
T[3]
```

On obtient :

```
-10
```

Exemple On veut coder les lettres "a","b",.."z" par 1,2,....26.
On tape :
`alphab:="abcdefghijklmnopqrstuvwxyz";`
puis :
`code:=table(seq(alphab[j]=j+1,j=0..25));`
On tape `code["c"]`
On obtient `3`
ou bien on écrit une fonction :

```
Code(a):={
local code,alphab,j;
alphab:="abcdefghijklmnopqrstuvwxyz";
code:=table(seq(alphab[j]=j+1,j=0..25));
return code(a);
};
```

On tape `Code("c")`
On obtient `3`
Remarque
Si on fait une affectation du type `T[n]:= ...` où `T` est le nom d'une variable et `n` un entier

- si la variable `T` contient une liste ou une séquence, alors le n-ième élément de `T` est modifié,
- si la variable `T` n'est pas assignée, une table `T` est créée avec une entrée (correspondant à l'indice n). Notez qu'après cette assignation `T` n'est pas une liste, bien que n soit un entier.

6.44 Les matrices particulières

Une matrice est representée par une liste de listes de même longueur. Dans les réponses de `Xcas`, les matrices sont parenthésées avec **[]**. Par exemple, **[**1,2,3**]** désigne la matrice [[1,2,3]] qui a une seule ligne, alors que [1,2,3] désigne la liste [1,2,3].
Dans ce document, on utilise la notation habituelle ([[1,2,3]]) pour les matrices renvoyées comme réponses.

6.44.1 Matrice identité : `idn identity`

`idn` a comme argument un entier n.
`idn` renvoie la matrice identité d'ordre n.
On tape :

`idn(2)`

On obtient :

`[[1,0],[0,1]]`

On tape :

`idn(3)`

On obtient :

`[[1,0,0],[0,1,0],[0,0,1]]`

6.44.2 Matrice de zéros : `newMat` `matrix`

`newMat(n,p)` ou `matrix(n,p)` renvoie la matrice de `n` lignes et de `p` colonnes formée par des zéros.
On tape :

`newMat(4,3)`

Ou on tape :

`matrix(4,3)`

On obtient :

`[[0,0,0],[0,0,0],[0,0,0],[0,0,0]]`

6.44.3 Matrice aléatoire : `ranm` `randMat` `randmatrix`

`ranm` ou `randMat` ou `randmatrix` a comme argument un entier n ou deux entiers n, m et éventuellement un troisième argument soit un entier k, soit le nom quoté ou non quoté de la loi de distribution des nombres aléatoires de la matrice (voir aussi 6.26.32, 6.40.39 et 7.3.9).
`ranm` renvoie un vecteur d'ordre n ou une matrice $n \times m$ constituée d'entiers aléatoires uniformément distribués entre -99 et 99 (par défaut) ou entre 0 et $k - 1$ ou une matrice $n \times m$ de nombres aléatoires distribués selon la loi mise entre-quote.
Attention la syntaxe sans quote marche avec les lois mais pas avec la commande `rand...` correspondante, donc par esxemple les commandes `ranm(3,4,normald,0,1)` ou `ranm(3,4,'normald(0,1)')` ou `ranm(3,4,'randnorm(0,1)')` sont valables mais `ranm(3,4,randnorm,0,1)` n'est pas valable.
On tape :

`ranm(3)`

On obtient :

`[-54,78,-29]`

On tape :

`ranm(2,4)`

On obtient :

`[[27,-29,37,-66],[-11,76,65,-33]]`

On tape :

`ranm(2,4,3)`

Ou on tape :

```
ranm(2,4,'rand(3)')
```

On obtient :

```
[[0,1,1,0],[0,1,2,0]]
```

On tape :

```
ranm(2,4,'randnorm(0,1)')
```

On obtient :

```
[[1.83785427742,0.793007112053,-0.978388964902,-1.88602023857],
[-1.50900874199,-0.241173369698,0.311373795585,-0.532752431454]]
```

On tape :

```
ranm(2,4,2..4)
```

On obtient :

```
[[2.00549363438,3.03381264955,2.06539073586,2.04844321217],
[3.88383254968,3.28664474655,3.76909781061,2.39113253355]]
```

On tape :

```
ranm(4,5,binomial,4,0.2)
```

Ou on tape :

```
ranm(4,5,'binomial(4,0.2)')
```

On obtient :

```
[[1,1,1,0,2],[2,1,0,0,0],[1,2,2,1,1],[0,2,1,1,0]]
```

On effectue 4*5 fois 4 tirages avec une probabilité de succés de 0.2 et à chaque fois le nombre de succés a été de :
1 puis 1, puis 1, puis 0, puis 2, puis 2, puis 1 etc...
On tape :

```
ranm(4,5,multinomial,[1/2,1/3,1/6])
```

Ou on tape :

```
ranm(4,5,'multinomial([1/2,1/3,1/6])')
```

On obtient par exemple :

```
[[4,1,0],[2,2,1],[3,2,0],[4,1,0]]
```

Dans ce cas on obitient une matrice ayant 4 lignes et 3 colonnes car sur chaque ligne on a le décompte du nombre de tirages par type, ici pour le premier tirage on a obtenu 4 objets ayant la probabilité 1/2 dêtre tirés, 1 objet de probabilité ayant la probabilité 1/3 dêtre tirés, 0 objet de probabilité ayant la probabilité 1/6 dêtre tirés, etc... C'est pratique car par exemple `ranm(4,200,multinomial,[1/2,1/3,1/6])` va renvoyer une matrice 4*3 qui donne la répartition.
On tape :

```
ranm(3,5,multinomial,[1/2,1/3,1/6],["R","V","B"])
```

Ou on tape :

```
ranm(3,5,'multinomial([1/2,1/3,1/6],["R","V","B"])')
```

On obtient par exemple :

```
[["B","B","V","V","R"],["B","R","R","R","R"],["R","R","R","R","V"]]
```

On tape :

```
ranm(3,4,poisson,1.3)
```

Ou on tape :

```
ranm(3,4,'poisson(1.3)')
```

On obtient par exemple :

```
[[2,2,0,4],[0,0,1,1],[0,4,1,0]]
```

On tape :

```
ranm(2,3,exponential,1.2)
```

Ou on tape :

```
ranm(2,3,'exponential(1.2)')
```

On obtient par exemple :

```
[[1.39292444066,0.214488721628,0.607596751757],
 [0.58087800165,0.662248573431,0.385110606536]]
```

On tape :

```
ranm(2,3,fisher,6,5)
```

Ou on tape :

```
ranm(2,3,'fisher(6,5)')
```

On obtient par exemple :

```
[[0.580815888368,0.43932104968,3.00433399184],[0.546184604298,2.07846
```

6.44.4 Diagonale d'une matrice ou matrice d'une diagonale : `diag` `BlockDiagonal`

`diag` ou `BlockDiagonal` a comme argument une matrice A ou une liste l.
`diag` renvoie la diagonale de A ou la matrice diagonale ayant pour diagonale la liste l.
On tape :

```
diag([[1,2],[3,4]])
```

On obtient :

```
[1,4]
```

On tape :

```
diag([1,4])
```

On obtient :

```
[[1,0],[0,4]]
```

6.44.5 Bloc de Jordan : `JordanBlock`

`JordanBlock` a comme argument une expression a et un entier n.
`JordanBlock` renvoie une matrice carrée d'ordre n avec a sur la diagonale principale, 1 au-dessus de la diagonale et 0 ailleurs.
On tape :

```
JordanBlock(7,3)
```

On obtient :

```
[[7,1,0],[0,7,1],[0,0,7]]
```

6.44.6 Matrice de Hilbert : `hilbert`

`hilbert` a comme argument un entier n.
`hilbert` renvoie la matrice de Hilbert.
C'est une matrice carrée d'ordre n d'éléments : $a_{j,k} = \dfrac{1}{j+k+1}$
On tape :

```
hilbert(4)
```

On obtient :

```
[[1,1/2,1/3,1/4],[1/2,1/3,1/4,1/5],[1/3,1/4,1/5,1/6],
[1/4,1/5,1/6,1/7]]
```

6.44.7 Matrice de Vandermonde : `vandermonde`

`vandermonde` a comme argument un vecteur de composantes x_j.
`vandermonde` renvoie la matrice de Vandermonde correspondante : elle a pour k^{ime} ligne est le vecteur de composantes x_j^k ($k = 0..n-1$).
Attention !
`Xcas` numérote les lignes et les colonnes à partir de 0.
On tape :

```
vandermonde([a,2,3])
```

On obtient (si `a` n'est pas affecté) :

```
[[1,1,1],[a,2,3],[a*a,4,9]]
```

6.45 Création et arithmétique des matrices

6.45.1 Pour évaluer une matrice : `evalm`

`evalm` sert à évaluer une matrice en mode Maple, par contre `Xcas` évalue toujours les matrices, sans avoir besoin de la commande `evalm`.

6.45.2 Addition et soustraction de deux matrices : `+ - .+ .-`

L'addition (resp la soustraction) de deux matrices se fait à l'aide de l'opérateur infixé + ou `.+` (resp − ou `.-`).
On tape :

```
[[1,2],[3,4]] + [[5,6],[7,8]]
```

On obtient :

```
[[6,8],[10,12]]
```

On tape :

```
[[1,2],[3,4]] - [[5,6],[7,8]]
```

On obtient :

```
[[-4,-4],[-4,-4]]
```

Remarque
+ peut aussi être préfixé, dans ce cas il doit être quoté.
On tape :

```
'+'([[1,2],[3,4]],[[5,6],[7,8]],[[2,2],[3,3]])
```

On obtient :

```
[[8,10],[13,15]]
```

6.45.3 Multiplication de deux matrices : `*` `&*`

La multiplication de deux matrices se fait à l'aide de l'opérateur infixé `*` (ou `&*`).
On tape :

```
[[1,2],[3,4]] * [[5,6],[7,8]]
```

Ou on tape :

```
[[1,2],[3,4]] &* [[5,6],[7,8]]
```

On obtient :

```
[[19,22],[43,50]]
```

6.45.4 Addition des éléments d'une même colonne d'une matrice : `sum`

`sum` a comme argument une matrice A.
`sum` renvoie la liste dont les éléments sont les sommes des éléments de chaque colonne de la matrice A.
On tape :

```
sum([[1,2],[3,4]])
```

On obtient :

```
[4,6]
```

6.45.5 Somme cumulée des éléments d'une même colonne d'une matrice : `cumSum`

`cumSum` a comme argument une matrice A.
`cumSum` renvoie la matrice dont les colonnes sont les sommes cumulées deséléments d'une même colonne de la matrice A.
On tape :

```
cumSum([[1,2],[3,4],[5,6]])
```

On obtient :

```
[[1,2],[4,6],[9,12]]
```

puisque les sommes cumulées sont 1, 1+3=4, 1+3+5=9 et 2, 2+4=6, 2+4+6=12.

6.45.6 Multiplication des éléments d'une même colonne d'une matrice : `product`

`product` a comme argument une matrice A.
`product` renvoie la liste des produits des éléments d'une même colonne de la matrice A (voir aussi 6.40.33 et 6.45.8).
On tape :

```
product([[1,2],[3,4]])
```

On obtient :

```
[3,8]
```

6.45.7 Elévation d'une matrice à une puissance entière : ˆ &ˆ

L'élévation d'une matrice à une puissance se fait à l'aide de l'opérateur infixé `^` (ou `&^`).
On tape :

```
[[1,2],[3,4]] ^ 5
```

Ou on tape :

```
[[1,2],[3,4]] &^ 5
```

On obtient :

```
[[1069,1558],[2337,3406]]
```

On tape :

```
normal([[1,2],[3,4]] ^ n)
```

Ou on tape :

```
normal([[1,2],[3,4]] &^ n)
```

On obtient :

```
[[(11-sqrt(33))/22*((sqrt(33)+5)/2)^n+(11+sqrt(33))/22*((-sqrt(33)+5)/
```

6.45.8 Produit de Hadamard : `hadamard product`

Voir aussi `.*` pour les listes (cf 6.41.5)
`hadamard` (ou `product`) a comme arguments deux matrices A et B de même ordre.
`product` est une fonction plus générale (voir aussi 6.40.33 et 6.45.6).
`hadamard` (ou `product`) renvoie la matrice constituée par le produit terme à terme des éléments de A et B.
On tape :

```
hadamard([[1, 2],[3,4]],[[5, 6],[7, 8]])
```

On obtient :

```
[[5,12],[21,32]]
```

Si on tape :

```
hadamard([1,2],[3,4])
```

ou

```
hadamard([[1,2],[3,4]])
```

On obtient :

```
5*sqrt(5)
```

Ici [1,2],[3,4] n'est pas considéré comme 2 vecteurs (car le produit de Hadamard ne marche que sur des matrices) mais comme une matrice à 2 lignes.
`hadamard` calcule prend le produit des normes des vecteurs colonnes et des vecteurs lignes et renvoie le plus petit des 2 :
$\sqrt{1+4} * \sqrt{9+16} = 5\sqrt{5}$ qui est plus petit que $\sqrt{1+9} * *\sqrt{4+16} = 10\sqrt{2}$.
Mais si on tape :

```
product([1,2],[3,4])
```

ou

```
product([[1,2],[3,4]])
```

On obtient :

```
[3,8]
```

6.45.9 Produit de Hadamard (version infixée) : `.*`

Voir aussi 6.41.5 et 6.45.8.
`.*` a comme arguments deux matrices ou deux listes A et B de même ordre.
`.*` est un opérateur infixé qui renvoie la matrice ou la liste constituée par le produit terme à terme des éléments de A et B.
On tape :

```
[[1, 2],[3,4]] .* [[5, 6],[7, 8]]
```

On obtient :

```
[[5,12],[21,32]]
```

6.45.10 Division de Hadamard (version infixée) : `./`

`./` a comme arguments deux matrices ou deux listes A et B de même ordre.
`./` est un opérateur infixé qui renvoie la matrice ou la liste constituée par la division terme à terme des éléments de A et B.
On tape :

```
[[1, 2],[3,4]] ./ [[5, 6],[7, 8]]
```

On obtient :

```
[[1/5,1/3],[3/7,1/2]]
```

6.45.11 Puissance de Hadamard (version infixée) : `.^`

`.^` a comme arguments une matrices A et un nombre réel b.
`.^` est un opérateur infixé qui renvoie la matrice constituée par les puissances b de chaque élément de A.
On tape :

```
[[1, 2],[3,4]] .^ 2
```

On obtient :

```
[[1,4],[9,16]]
```

6.45.12 Extraire un ou des élément(s) d'une matrice : `at`

Un matrice est une liste de listes de même longueur.
On tape :

```
A:=[[3,4,5],[1,2,6]]
```

On obtient :

```
[[3,4,5],[1,2,6]]
```

- Pour obtenir un élément, on met deux arguments entre des crochets : l'indice de la ligne et l'indice de la colonne séparé par une virgule si on veut que les indices commencent à 0.
  On tape :
  `[[3,4,5],[1,2,6]][1,2]`
  ou
  `A[1,2]`
  ou
  `A[1][2]`
  ou
  `at(A,[1,2])`
  On obtient :
  `6`
  Ou bien On tape des doubles crochets ou des parenthèses si on veut que les indices commencent à 1 :
  `[[3,4,5],[1,2,6]][[1,2]]`
  ou
  `A[[1,2]]`
  ou
  `A[[1]][[2]]`
  ou
  `[[3,4,5],[1,2,6]](1,2)`
  ou
  `A(1,2)`
  ou
  `A(1)(2)`
  On obtient :
  `4`
- Pour obtenir une ligne de la matrice `A`, on met l'indice de la ligne entre des crochets, on tape :
  `[[3,4,5],[1,2,6]][1]`
  ou
  `A[1]`
  ou
  `at(A,1)`
  On obtient :
  `[1,2,6]`
  Ou bien On tape des doubles crochets ou des parenthèses si on veut que les indices commencent à 1 :

`[[3,4,5],[1,2,6]][[1]]`

ou

`A[[1]]`

ou

`[[3,4,5],[1,2,6]](1)`

ou

`A(1)`

On obtient :

`[3,4,5]`

– Pour obtenir une sous-ligne de la matrice `A`, on met deux arguments entre des crochets : l'indice de la ligne et un intervalle pour désigner les indices des colonnes formant la sous-ligne.
  On tape :

  `A[1,0..2]`

  On obtient :

  `[1,2,6]`

  On tape :

  `A[1,1..2]`

  On obtient :

  `[2,6]`

– Pour obtenir, sous la forme d'une liste, une colonne de la matrice `A`, on utilise `tran(A)` qui désigne la transposée de `A` et on tape l'indice de la colonne entre des crochets si les indices commencent à 0 :

  `tran([[3,4,5],[1,2,6]])[[1]]`

  ou

  `tran(A)[1]`

  ou

  `at(tran(A),1)`

  On obtient :

  `[4,2]`

  Ou bien On tape l'indice de la colonne entre des doubles crochets ou des parenthèses si on veut que les indices commencent à 1 :

  `tran([[3,4,5],[1,2,6]])[[1]]`

  ou

  `tran(A)[[1]]`

  ou

  `tran([[3,4,5],[1,2,6]])(1)`

  ou

  `tran(A)(1)`

  On obtient :

  `[3,1]`

  ou encore on met deux arguments entre des crochets : un intervalle pour désigner toutes les lignes et l'indice de la colonne.
  On tape :

  `A[0..1,1]`

  On obtient :

  `[4,2]`

– Pour avoir une liste représentant une sous-colonne, on met deux arguments

entre des crochets : un intervalle pour désigner les indices des lignes formant la sous-colonne et l'indice de la colonne.
On tape :

`A[0..0,1]`

On obtient :

`[4]`

Cela peut être utiliser pour extraire une colonne, en mettant comme premier indice, l'intervalle désignant toutes les lignes.
On tape :

`A[0..1,1]`

On obtient :

`[4,2]`

– Pour extraire une sous-matrice d'une matrice, on met deux arguments entre des crochets : un intervalle pour désigner les lignes et un intervalle pour désigner les colonnes.
On définit la matrice `A`, on tape :

`A:=[[3,4,5],[1,2,6]]`

On tape :

`A[0..1,1..2]`

On obtient :

`[[4,5],[2,6]]`

On tape :

`A[0..1,1..1]`

On obtient :

`[[4],[2]]`

Remarque
Si on veut une sous matrice constituée de lignes consécutives complètes, on peut ommettre le deuxième argument et ne mettre que l'intervalle pour désigner les lignes entre des crochets.
On tape :

`A[1..1]`

On obtient :

`[[1,2,6]]`

6.45.13 Modifier un élément ou une ligne d'une matrice contenue dans une variable : := et =<

Si la matrice a un nom, on peut assigner un élément d'une matrice en utilisant son indice entouré de crochets si les indices commencent à 0 et entouré de doubles crochets ou des parenthèses si les indices commencent à 1.
Attention
On ne peut pas utiliser les parenthèses avec une matrice formelle car pour `Xcas` cette notation sera la définition d'une fonction !
Si on veut que les indices commencent à 1 il faut utiliser les doubles crochets. Par exemple :

```
A:=idn(3);
pour j de 1 jusque 3 faire
  pour k de 1 jusque 3 faire
```

```
    A(j,k):=j*k;
  fpour;
fpour;
B:=idn(3);
pour j de 1 jusque 3 faire
  pour k de 1 jusque 3 faire
    B[[j,k]]:=j*k;
  fpour;
fpour;
```

On a alors
`A` renvoie la fonction `(j,k)->j*k`
`B` renvoie la matrice `[[1,2,3],[2,4,6],[3,6,9]]`
Si on assigne avec `:=`, une nouvelle copie de la matrice est créée et l'élément est modifié, et si on assigne avec `=<` la matrice est modifiée sans faire de copie ce qui est plus rapide lorque la matrice est de grande taille.
Par exemple :
Si `A:=[[4,5],[2,6]]`, pour modifier `A` en la matrice `[[4,5],[3,6]]` on peut taper :
`A[1,0]:=3` ou
`A[1,0]=<3` ou
`A[[2,1]]:=3` ou
`A(2,1):=3` ou
`A[[2,1]]=<3`
Puis on tape :
`A`
On obtient la nouvelle valeur de `A` :
`[[4,5],[3,6]]`
On peut aussi modifier une ligne, par exemple, si `A:=[[4,5],[2,6]]` pour modifier en `A` la matrice `[[4,5],[3,7]]`, on peut taper :
`A:=[[4,5],[2,6]] A[1]:=[3,7]` ou
`A[1]=<[3,7]` ou
`A[[2]]:=[3,7]` ou
`A(2):=[3,7]` ou
`A[[2]]=<[3,7]` ou
`A(2)=<[3,7]`
Puis on tape :
`A`
On obtient la nouvelle valeur de `A` :
`[[4,5],[3,7]]`.

Remarque
Il faut utiliser `=<` avec précautions car tous les objets pointant sur cette matrice seront modifiés. Dans un programme il faudra utiliser `A:=copy(B)` lors de l'initialisation pour que les modifications faites avec `=<` sur `B` ne se fassent pas sur la copie `A` ou que les modifications faites avec `=<` sur la copie `A` ne se fassent pas sur `B` (cf 8.4.15).
Par exemple :
On tape :
`B:=[[4,5],[2,6]]`

`A=<B` ou `A:=B`
`A,B`
On obtient :
`[[4,5],[2,6]],[[4,5],[2,6]]`
On tape :
`B[1]=<[3,7]` ou `A[1]=<[3,7]`
`A,B`
On obtient :
`[[4,5],[3,7]],[[4,5],[3,7]]`
Mais si on tape :
`B:=[[4,5],[2,6]]`
`A:=copy(B)`
`A,B`
On obtient :
`[[4,5],[2,6]],[[4,5],[2,6]]`
On tape :
`B[1]=<[3,7]`
`A,B`
On obtient :
`[[4,5],[2,6]],[[4,5],[3,7]]`
Ou si on tape :
`B:=[[4,5],[2,6]]`
`A:=copy(B)`
`A,B`
On obtient :
`[[4,5],[2,6]],[[4,5],[2,6]]`
On tape :
`A[1]=<[3,7]`
`A,B`
On obtient :
`[[4,5],[3,7]],[[4,5],[2,6]]`

6.45.14 Modifier un élément ou une ligne d'une matrice : `subsop`

`subsop` permet de modifier un élément ou une ligne d'une matrice sans avoir à stocker la matrice dans une variable.

1. Modification d'un élément
   `subsop` a deux ou trois arguments.
   Attention
   En mode `Maple` les arguments sont permutés.

   – En mode `Xcas`, les indices commence à 0 :
     `subsop` a deux (resp trois) arguments : une matrice `A` et une égalité `[r,c]=v` (resp une matrice `A`, la liste des indices de la ligne et de la colonne `[r,c]`, la nouvelle valeur `v`).
     On tape en mode `Xcas` :
     `subsop([[4,5],[2,6]],[1,0]=3)`
     Ou on tape :

`subsop([[4,5],[2,6]],[1,0],3)`

On obtient :

`[[4,5],[3,6]]`

– En mode `Mupad, TI`, les indices commencent à 1 :
  `subsop` a deux (resp trois) arguments : une matrice `A` et une égalité `[r,c]=v` (resp une matrice `A`, la liste des indices de la ligne et de la colonne `[r,c]`, la nouvelle valeur `v`).
  On tape en mode `Mupad, TI` :

  `subsop([[4,5],[2,6]],[2,1]=3)`

  Ou on tape :

  `subsop([[4,5],[2,6]],[2,1],3)`

  On obtient :

  `[[4,5],[3,6]]`

  Remarque
  Si la matrice a un nom par exemple :
  `A:=[[4,5],[2,6]]`, on peut taper directement `A[2,1]:=3` pour modifier `A` en la matrice `[[4,5],[3,6]]`.
– En mode `Maple`, les arguments sont permutés et les indices commencent à 1 :
  `subsop` a deux arguments : une égalité `[r,c]=v` et une matrice `A`.
  On tape :

  `subsop([2,1]=3,[[4,5],[2,6]])`

  On obtient :

  `[[4,5],[3,6]]`

  Remarque
  Si la matrice à un nom par exemple :
  `A:=[[4,5],[2,6]]`, on peut taper directement `A[2,1]:=3` pour modifier `A` en la matrice `[[4,5],[3,6]]`.

2. Modification d'une ligne
   `subsop` permet de modifier une ligne d'une matrice sans avoir à stocker la matrice dans une variable.
   `subsop` a alors deux arguments :

   – En mode `Xcas`, les indices commence à 0 :
     `subsop` a deux arguments une matrice et une égalité, à savoir l'indice de la ligne à modifier, suivi du signe = puis la liste des nouvelles valeurs de cette ligne.
     On tape en mode `Xcas` :

     `subsop([[4,5],[2,6]],1=[3,3])`

     On obtient :

     `[[4,5],[3,3]]`

     Remarque
     Si la matrice à un nom par exemple :
     `A:=[[4,5],[2,6]]`, on peut taper directement `A[1]:=[3,3]` pour modifier `A` en la matrice `[[4,5],[3,3]]`.
   – En mode `Mupad, TI`, les indices commencent à 1 :
     `subsop` a deux arguments une matrice et une égalité, à savoir l'indice de la ligne à modifier, suivi du signe = puis la liste des nouvelles valeurs de cette ligne.

On tape en mode `Mupad, TI` :

`subsop([[4,5],[2,6]],2=[3,3])`

On obtient :

`[[4,5],[3,3]]`

Remarque

Si la matrice à un nom par exemple :
`A:=[[4,5],[2,6]]`, on peut taper directement `A[2]:=[3,3]` pour modifier `A` en la matrice `[[4,5],[3,3]]`.

– En mode `Maple` les arguments sont permutés et les indices commencent à 1 :
`subsop` a deux arguments une égalité, à savoir l'indice de la ligne à modifier, suivi du signe = puis la liste des nouvelles valeurs de cette ligne et une matrice.
On tape :

`subsop(2=[3,3],[[4,5],[2,6]])`

On obtient :

`[[4,5],[3,3]]`

Remarque

Si la matrice à un nom par exemple :
`A:=[[4,5],[2,6]]`, on peut taper directement `A[2]:=[3,3]` pour modifier `A` en la matrice `[[4,5],[3,3]]`.

Remarque

On peut aussi supprimer la ligne n d'une matrice avec `subsop` en mettant comme second argument `'n=NULL'`.
On tape en mode `Xcas` :

`subsop([[4,5],[2,6]],'1=NULL')`

On obtient :

`[[4,5]]`

6.45.15 Redimensionner une matrice ou un vecteur : `REDIM`

`REDIM` a comme argument une matrice A (resp un vecteur) et une liste de 2 entiers (resp 1 entier).
`redim` redimensionner cette matrice (resp ce vecteur) selon le deuxième argument soit on la (resp le)raccourcissant, soit en l'augmentant avec des 0.
On tape :

`REDIM([[4,1,-2],[1,2,-1],[2,1,0]],[5,4])`

On obtient :

`[[4,1,-2,0],[1,2,-1,0],[2,1,0,0],[0,0,0,0],[0,0,0,0]]`

On tape :

`REDIM([[4,1,-2],[1,2,-1],[2,1,0]],[2,1])`

On obtient :

On tape :

```
REDIM([4,1,-2,1,2,-1],10)
```

On obtient :

```
[[4],[1]]
```

On tape :

```
REDIM([4,1,-2,1,2,-1],3)
```

On obtient :

```
[4,1,-2]
```

6.45.16 Remplacer une partie d'une matrice ou d'un vecteur : REPLACE

REPLACE a comme argument une matrice A (resp un vecteur) et une liste de 2 indices (resp 1 entier) et la matrice (resp le vecteur) qui doit être mis en remplacement à partir de ces 2 indices.
REPLACE effectue ce remplacement en élaguant éventuellement la matrice (resp le vecteur) si elle (resp il) est sur dimensionnée.
On tape :

```
REPLACE([[1,2,3],[4,5,6]],[0,1],[[5,6],[7,8]])
```

On obtient :

```
[[1,5,6],[4,7,8]]
```

On tape :

```
REPLACE([[1,2,3],[4,5,6]],[1,2],[[7,8],[9,0]])
```

On obtient :

```
[[1,2,3],[4,5,7]]
```

On tape :

```
REPLACE([4,1,-2,1,2,-1],2,[10,11])
```

On obtient :

```
[4,1,10,11,2,-1]
```

On tape :

```
REPLACE([4,1,-2,1,2,-1],1,[10,11,13])
```

On obtient :

```
[4,10,11,13,2,-1]
```

6.45.17 Extraire des lignes ou des colonnes d'une matrice (compatibilité Maple) : `row col`

`row` (resp `col`) permet d'extraire une ou plusieurs lignes (resp colonnes) d'une matrice.
`row` (resp `col`) a 2 arguments : une matrice A, et un entier n ou un intervalle $n_1..n_2$.
`row` (resp `col`) renvoie la ligne (resp colonne) d'indice n de la matrice A, ou la séquence des lignes (resp colonnes) d'indice allant de n_1 à n_2 de la matrice A.
On tape :

```
row([[1,2,3],[4,5,6],[7,8,9]],1)
```

On obtient :

```
[4,5,6]
```

On tape :

```
row([[1,2,3],[4,5,6],[7,8,9]],0..1)
```

On obtient :

```
([1,2,3],[4,5,6])
```

On tape :

```
col([[1,2,3],[4,5,6],[7,8,9]],1)
```

On obtient :

```
[2,5,8]
```

On tape :

```
col([[1,2,3],[4,5,6],[7,8,9]],0..1)
```

On obtient :

```
([1,4,7],[2,5,8])
```

6.45.18 Supprimer des lignes ou des colonnes d'une matrice : `delrows delcols`

`delrows` (resp `delcols`) permet de supprimer une ou plusieurs lignes (resp colonnes) d'une matrice.
`delrows` (resp `delcols`) a 2 arguments : une matrice A, et un entier n ou un intervalle $n_1..n_2$.
`delrows` (resp `delcols`) renvoie la matrice obtenue en supprimant la ligne (resp colonne) n ou les lignes (resp colonnes) n_1 jusqu'à n_2 de la matrice A.
On tape :

```
delrows([[1,2,3],[4,5,6],[7,8,9]],1..1)
```

On obtient :

```
[[1,2,3],[7,8,9]]
```

On tape :

```
delrows([[1,2,3],[4,5,6],[7,8,9]],0..1)
```

On obtient :

```
[[7,8,9]]
```

On tape :

```
delcols([[1,2,3],[4,5,6],[7,8,9]],1..1)
```

On obtient :

```
[[1,3],[4,6],[7,9]]
```

On tape :

```
delcols([[1,2,3],[4,5,6],[7,8,9]],0..1)
```

On obtient :

```
[[3],[6],[9]]
```

6.45.19 Extraire une sous-matrice d'une matrice (compatibilité TI) : `subMat`

`subMat` a 5 arguments : une matrice A, et 4 entiers $nl1, nc1, nl2, nc2$. Ces indices sont : $nl1$ est l'indice du début de ligne ,$nc1$ est l'indice du début de colonne, $nl2$ est l'indice de fin de ligne et $nc2$ est l'indice de fin de colonne.
`subMat(A,nl1,nc1,nl2,nc2)` extrait la sous-matrice de la matrice `A` de premier élément `A[nl1,nc1]` et de dernier élément `A[nl2,nc2]`.
On définit la matrice `A`, on tape :

```
A:=[[3,4,5],[1,2,6]]
```

On tape :

```
subMat(A,0,1,1,2)
```

ou

```
subMat(A,[0,1],[1,2])
```

On obtient :

```
[[4,5],[2,6]]
```

On tape :

```
subMat(A,0,1,1,1)
```

ou

```
subMat(A,[0,1],[1,1])
```

On obtient :

`[[4],[2]]`

Par défaut $nl1 = 0$, $nc1 = 0$, $nl2$=nrows-1 et $nc2$=ncols-1
On tape :

`A:=[[3,4,5],[1,2,6]]`

`subMat(A)`

On obtient :

`[[3,4,5],[1,2,6]]`

On tape :

`subMat(A,1)`

Ou :

`subMat(A,1,0)`

Ou :

`subMat(A,1,0,1)`

Ou :

`subMat(A,1,0,1,2)`

On obtient :

`[[1,2,6]]`

6.45.20 Redimensionner une matrice ou un vecteur : `redim`

`redim` a comme argument une matrice A (resp un vecteur) et une liste de 2 entiers (resp 1 entier).
`redim` redimensionner cette matrice (resp ce vecteur) soit en la (resp le) raccourcissant, soit en l'augmentant avec des 0.
On tape :

`redim([[4,1,-2],[1,2,-1]],[3,4])`

On obtient :

`[[4,1,-2,0],[1,2,-1,0],[0,0,0,0]]`

On tape :

`redim([[4,1,-2],[1,2,-1],[2,1,0]],[2,1])`

On obtient :

`[[4],[1]]`

On tape :

```
redim([4,1,-2,1,2,-1],8)
```

On obtient :

```
[4,1,-2,1,2,-1,0,0]
```

On tape :

```
redim([4,1,-2,1,2,-1],3)
```

On obtient :

```
[4,1,-2]
```

6.45.21 Remplacer une partie d'une matrice ou d'un vecteur : `replace`

`replace` a comme argument une matrice A (resp un vecteur) et une liste de 2 indices (resp 1 entier) et la matrice (resp le vecteur) qui doit être mis en remplacement à partir de ces 2 indices.
`replace` effectue ce remplacement en élaguant éventuellement la matrice (resp le vecteur) si elle (resp il) est sur dimensionnée.
On tape :

```
replace([[1,2,3],[4,5,6]],[1,1],[[5,6],[7,8]])
```

On obtient :

```
[[5,6,3],[7,8,6]]
```

On tape :

```
replace([[1,2,3],[4,5,6]],[1,2],[[7,8,10],[9,0,11]])
```

On obtient :

```
[[1,7,8],[4,9,0]]
```

On tape :

```
replace([1,2,3,4],2,[5,6])
```

On obtient :

```
[1,5,6,4]
```

On tape :

```
replace([1,2,3,4],2,[5,6,7,8])
```

On obtient :

```
[1,5,6,7]
```

6.45.22 Ajouter une ligne à une autre : `rowAdd`

`rowAdd` a trois arguments : une matrice A et deux entiers $n1$ et $n2$.
`rowAdd` renvoie la matrice obtenue en remplacant dans A la ligne $n2$ par la somme des lignes $n1$ et $n2$.
On tape :

```
rowAdd([[1,2],[3,4]],0,1)
```

On obtient :

```
[[1,2],[4,6]]
```

6.45.23 Multiplier une ligne par une expression : `mRow` et `scale` `SCALE`

`mRow` a trois arguments : une expression, une matrice A et un entier n.
`scale` (ou `SCALE`) a trois arguments : une matrice A, une expression et un entier n.
Attention à l'ordre des arguments !
`mRow` et `scale` (ou `SCALE`) renvoie la matrice obtenue en remplacant dans A la ligne n par la multiplication de la ligne n par l'expression.
On tape :

```
mRow(12,[[1,2],[3,4]],1)
```

Ou on tape :

```
scale([[1,2],[3,4]],12,1)
```

On obtient :

```
[[1,2],[36,48]]
```

6.45.24 Ajouter k fois une ligne à une autre : `mRowAdd` et `scaleadd` `SCALEADD`

`mRowAdd` a quatre arguments : un réel k, une matrice A et deux entiers $n1$ et $n2$.
`scale` (ou `SCALE`) a quatre arguments : une matrice A, un réel k et deux entiers $n1$ et $n2$.
Attention à l'ordre des arguments !
`mRowAdd` et `scale` (ou `SCALE`) renvoie la matrice obtenue en remplacant dans A la ligne $n2$ par la somme de la ligne $n2$ et de k fois la ligne $n1$.
On tape :

```
mRowAdd(1.1,[[5,7],[3,4],[1,2]],1,2)
```

Ou on tape :

```
scaleadd([[5,7],[3,4],[1,2]],1.1,1,2)
```

On obtient :

```
[[5,7],[3,4],[4.3,6.4]]
```

6.45.25 Échanger deux lignes : `rowSwap swaprow rowswap`

`rowSwap` a trois arguments : une matrice A et deux entiers $n1$ et $n2$.
`rowSwap` renvoie la matrice obtenue en échangeant dans A les lignes $n1$ et $n2$.
On tape :

```
rowSwap([[1,2],[3,4]],0,1)
```

On obtient :

```
[[3,4],[1,2]]
```

6.45.26 Échanger deux colonnes : `colSwap swapcol colswap`

`rowSwap` a trois arguments : une matrice A et deux entiers $n1$ et $n2$.
`rowSwap` renvoie la matrice obtenue en échangeant dans A les lignes $n1$ et $n2$.
On tape :

```
colSwap([[1,2],[3,4]],0,1)
```

On obtient :

```
[[2,1],[4,3]]
```

6.45.27 Faire une matrice avec une liste de matrices : `blockmatrix`

`blockmatrix` a comme arguments deux entiers n, m et une liste de longueur $n * m$ formée de matrices (de même dimension $p \times q$ ou de taille différentes : les m premières matrices ont le même nombre de lignes et forment un bloc de c colonnes, les m suivantes ont le même nombre de lignes et forment un bloc de c colonnes, etc...). On forme ainsi n blocs de c colonnes.
`blockmatrix` renvoie la matrice de c colonnes obtenue en mettant ces n blocs les uns sous les autres. Si les matrices de l'argument ont même dimension $p \times q$, la matrice résultat a pour dimension $p * n \times q * m$.
On tape :

```
blockmatrix(2,3,[idn(2),idn(2),idn(2),
      idn(2),idn(2),idn(2)])
```

On obtient :

```
[[1,0,1,0,1,0],[0,1,0,1,0,1],
 [1,0,1,0,1,0],[0,1,0,1,0,1]]
```

On tape :

```
blockmatrix(3,2,[idn(2),idn(2),
 idn(2),idn(2),idn(2),idn(2)])
```

On obtient :

```
[[1,0,1,0],[0,1,0,1],
[1,0,1,0],[0,1,0,1],[1,0,1,0],[0,1,0,1]]
```

On tape :

```
blockmatrix(2,2,[idn(2),newMat(2,3),
        newMat(3,2),idn(3)])
```

On obtient :

```
[[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],
      [0,0,0,1,0],[0,0,0,0,1]]
```

On tape :

```
blockmatrix(3,2,[idn(1),newMat(1,4),
newMat(2,3),idn(2),newMat(1,2),[[1,1,1]]])
```

On obtient :

```
[[1,0,0,0,0],[0,0,0,1,0],[0,0,0,0,1],[0,0,1,1,1]]
```

On tape :

```
A:=[[1,1],[1,1]];B:=[[1],[1]]
```

puis :

```
blockmatrix(2,3,[2*A,3*A,4*A,5*B,newMat(2,4),6*B])
```

On obtient :

```
[[2,2,3,3,4,4],[2,2,3,3,4,4],
 [5,0,0,0,0,6],[5,0,0,0,0,6]]
```

6.45.28 Faire une matrice avec deux matrices : `semi_augment`

`semi_augment` concaténe deux matrices ayant le même nombre de colonnes.
On tape :

```
semi_augment([[3,4],[2,1],[0,1]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4],[2,1],[0,1],[1,2],[4,5]]
```

On tape :

```
semi_augment([[3,4,2]],[[1,2,4]])
```

On obtient :

```
[[3,4,2],[1,2,4]]
```

Attention Comparer `semi_augment` et `concat`.
On tape :

```
concat([[3,4,2]],[[1,2,4]])
```

On obtient :

```
[[3,4,2,1,2,4]]
```

En effet, quand les deux matrices A et B ont la même dimension, `concat` fabrique une matrice ayant même nombre de lignes que A et B en accolant A et B.
On tape :

```
concat([[3,4],[2,1],[0,1]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4],[2,1],[0,1],[1,2],[4,5]]
```

alors que :

```
concat([[3,4],[2,1]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4,1,2],[2,1,4,5]]
```

6.45.29 Faire une matrice avec deux matrices : `augment concat`

`augment` ou `concat` concaténe deux matrices A et B ayant le même nombre de lignes (resp de colonnes) c'est à dire `augment` ou `concat` fabrique une matrice ayant même nombre de lignes (resp de colonnes) que A et B en accolant A et B.
On tape :

```
augment([[3,4,5],[2,1,0]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4,5,1,2],[2,1,0,4,5]]
```

On tape :

```
augment([[3,4],[2,1],[0,1]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4],[2,1],[0,1],[1,2],[4,5]]
```

On tape :

```
augment([[3,4,2]],[[1,2,4]]
```

On obtient :

```
[[3,4,2,1,2,4]]
```

Quand les deux matrices A et B ont le même nombre de lignes que de colonnes, `augment` fabrique une matrice ayant même nombre de lignes que A et B en accolant A et B.
On tape :

```
augment([[3,4],[2,1]],[[1,2],[4,5]])
```

On obtient :

```
[[3,4,1,2],[2,1,4,5]]]
```

6.45.30 Faire une matrice avec une fonction : `makemat`

`makemat` a trois arguments :
- une fonction de deux variables `j` et `k` égale à la valeur de $a_{j,k}$ (`j` représente un numéro de ligne et `k` un numéro de colonne et ces indices commencent '=à 0),
- deux entiers n et p.

`makemat` renvoie la matrice de coefficients $a_{j,k}$ ($j = 0..n-1$ et $k = 0..p-1$) de dimension $n \times p$.
On tape :

```
makemat((j,k)->j+k,4,3)
```

ou on tape pour définir la fonction h :

```
h(j,k):=j+k
```

puis, on tape :

```
makemat(h,4,3)
```

On obtient :

```
[[0,1,2],[1,2,3],[2,3,4],[3,4,5]]
```

Attention ! à la dimension et aux indices qui partent de 0 quelquesoit le mode choisi.

6.45.31 Définir une matrice : `matrix`

`matrix` a trois arguments :
- deux entiers n et p.
- une fonction de deux variables `j` et `k` égale à la valeur de $a_{j,k}$ (`j` représente un numéro de ligne et `k` un numéro de colonne).

`matrix` renvoie la matrice de coefficients $a_{j,k}$ ($j = 0..n-1$ et $k = 0..p-1$) de dimension $n \times p$ si on est en Mode (syntaxe) `xcas` ou renvoie la matrice de coefficients $a_{j,k}$ ($j = 1..n$ et $k = 1..p$) de dimension $n \times p$ si on est en Mode (syntaxe) `maple, mupad, ti89/92`.
On tape :

```
matrix(4,3,(j,k)->j+k)
```

ou on définit la fonction h par : `h(j,k):=j+k` et,
on tape :

```
matrix(4,3,h)
```

On obtient en Mode (syntaxe) `xcas` :

```
[[0,1,2],[1,2,3],[2,3,4],[3,4,5]]
```

On obtient en Mode (syntaxe) `maple, mupad, ti89/92` :

```
[[2,3,4],[3,4,5],[4,5,6],[5,6,7]]
```

Attention ! à l'ordre des variables dans la définition de $a_{j,k}$ et à j and k qui partent de `0` si on est en Mode (syntaxe) `xcas` et qui partent de `1` si on est en Mode (syntaxe) `maple, mupad, ti89/92` .

6.45.32 Rajouter une colonne à une matrice : `border`

`border` a comme argument une matrice `A` de dimension $p * q$ et une liste `b` de dimension p (c'est à dire `nrows(A)=size(b)`).
`border` renvoie la matrice obtenue à partir de `A` en lui rajoutant comme dernière colonne `tran(b)`.
On a :
`border(A,b)=tran([op(tran(A)),b])=tran(append(tran(A),b))`
On tape :

```
border([[1,2,4],[3,4,5]],[6,7])
```

On obtient :

```
[[1,2,4,6],[3,4,5,7]]
```

On tape :

```
border([[1,2,3,4],[4,5,6,8],[7,8,9,10]],[1,3,5])
```

On obtient :

```
[[1,2,3,4,1],[4,5,6,8,3],[7,8,9,10,5]]
```

6.45.33 Compter les éléments d'une matrice vérifiant une propriété : `count`

`count` a deux ou trois paramètres : une fonction réelle `f` et une matrice réelle `A` de dimension `p*q` (resp une liste réelle `l` de longueur `n`) et éventuellement un paramètre optionnel `row` ou `col`.
`count` applique la fonction aux éléments de la matrice (ou liste) et en renvoie la somme, c'est à dire,
`count` renvoie `f(A[0,0])+..f(A[p-1,q-1])` (resp `f(l[0])+..f(l[n])`).
Si il y a `row` (resp `col`) comme troisième paramètre `count` agit sur chaque ligne (resp colonne) de la matrice et renvoie une liste.
Si `f` est une fonction boolénne `count` renvoie le nombre d'éléments de la matrice (ou liste) pour lesquels la fonction boolénne est vraie.
On tape :

```
count((x)->x,[[2,12],[45,3],[7,78]])
```

On obtient :

```
147
```

car on a : 2+12+45+3+7+78=147.
On tape :

```
count((x)->x,[[2,12],[45,3],[7,78]],row)
```

On obtient :

```
[14,48,85]
```

car on a : 2+12=14,45+3=48,7+78=85.
On tape :

```
count((x)->x,[[2,12],[45,3],[7,78]],col)
```

On obtient :

```
[54,93]
```

car on a : 2+45+7=54, 12+3+78=93.
On tape :

```
count(x->x<10,[[2,12],[45,3],[7,78]])
```

On obtient :

```
3
```

6.45.34 Compter les éléments ayant une valeur donnée : `count_eq`

`count_eq` a deux ou trois paramètres : un réel et une matrice (ou une liste) réelle et éventuellement un paramètre optionnel `row` ou `col`.
`count_eq` renvoie le nombre d'éléments de la matrice (ou de la liste) qui sont égaux au premier argument.
Si il y a `row` (resp `col`) comme troisième paramètre `count_eq` agit sur chaque ligne (resp colonne) de la matrice et renvoie une liste.
On tape :

```
count_eq(12,[[2,12,45],[3,7,78]])
```

On obtient :

```
1
```

On tape :

```
count_eq(12,[[2,12,45],[3,7,78]],row)
```

On obtient :

```
[1,0]
```

On tape :

```
count_eq(12,[[2,12,45],[3,7,78]],col)
```

On obtient :

```
[0,1,0]
```

6.45.35 Compter les éléments plus petits qu'une valeur donnée : `count_inf`

`count_inf` a deux ou trois paramètres : une nombre et une matrice (ou liste) réelle et éventuellement un paramètre optionnel `row` ou `col`.
`count_inf` renvoie le nombre d'éléments de la matrice (ou liste) qui sont strictement inférieurs au premier argument.
Si il y a `row` (resp `col`) comme troisième paramètre `count_inf` agit sur chaque ligne (resp colonne) de la matrice et renvoie une liste.
On tape :

```
count_inf(12,[2,12,45,3,7,78])
```

Ou on tape :

```
count_inf(12,[[2,12],[45,3],[7,78]])
```

On obtient :

```
3
```

On tape :

```
count_inf(12,[[2,12],[45,3],[7,78]],row)
```

On obtient :

```
[1,1,1]
```

On tape :

```
count_inf(12,[[2,12],[45,3],[7,78]],col)
```

On obtient :

```
[2,1]
```

6.45.36 Compter les éléments plus grands qu'une valeur donnée : `count_sup`

`count_sup` a deux ou trois paramètres : une nombre et une matrice (ou liste) réelle et éventuellement un paramètre optionnel `row` ou `col`.
`count_sup` renvoie le nombre d'éléments de la matrice (ou liste) qui sont strictement supérieurs au premier argument.
Si il y a `row` (resp `col`) comme troisième paramètre `count_sup` agit sur chaque ligne (resp colonne) de la matrice et renvoie une liste.
On tape :

```
count_sup(12,[[2,12,45],[3,7,78]])
```

On obtient :

```
2
```

On tape :

`count_sup(12,[[2,12,45],[3,7,78]],row)`

On obtient :

`[1,1]`

On tape :

`count_sup(12,[[2,12,45],[3,7,78]],col)`

On obtient :

`0,0,2]`

6.45.37 Fonctions utiles pour les colonnes d'une matrice : `mean` ou `moyenne`, `stddev` ou `ecart_type`, `variance`, `median`, `quantile`, `quartiles`, `boxwhisker` ou `moustache`

Voir aussi 6.42 and 7.

Fonctions utiles pour les statistiques dont les donn'ees sont les colonnes d'une matrice :

- `mean` ou `moyenne` pour calculer la moyenne numèrique de series statistiques qui sont les colonnes d'une matrice.
  On tape :
  `mean([[3,4,2],[1,2,6]])`
  On obtient un vecteur de composantes la moyenne des colonnes :
  `[2,3,4]`
  On tape :
  `mean([[1,0,0],[0,1,0],[0,0,1]])`
  On obtient
  `[1/3,1/3,1/3]`
- `stddev` ou `ecart_type` pour calculer l'écart type numèrique de series statistiques qui sont les colonnes d'une matrice.
  On tape :
  `stddev([[3,4,2],[1,2,6]])`
  On obtient un vecteur de composantes l'écart type des colonnes :
  `[1,1,2]`
- `variance` pour calculer la variance numèrique de series statistiques qui sont les colonnes d'une matrice.
  On tape :
  `variance([[3,4,2],[1,2,6]])`
  On obtient un vecteur de composantes la variance des colonnes :
  `[1,1,4]`
- `median` pour calculer la médiane de series statistiques qui sont les colonnes d'une matrice.
  On tape :

  ```
  median([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],[1,3,4,2,5,6,0],
   [3,4,2,5,6,0,1],[4,2,5,6,0,1,3],[2,5,6,0,1,3,4]])
  ```
  On obtient un vecteur de composantes la médiane des colonnes :

`[2.0,2.0,3.0,3.0,2.0,2.0,3.0]`

- `quantile` pour calculer le décile selon le second argument, de series statistiques qui sont les colonnes d'une matrice.
  On tape :

  ```
  quantile([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],[1,3,4,2,5,6,0],
  [3,4,2,5,6,0,1],[4,2,5,6,0,1,3],[2,5,6,0,1,3,4]],0.25)
  ```
  On obtient un vecteur de composantes le premier quartile des colonnes :
  `[1.0,1.0,2.0,2.0,1.0,1.0,1.0]`
  On tape :

  ```
  quantile([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],[1,3,4,2,5,6,0],
  [3,4,2,5,6,0,1],[4,2,5,6,0,1,3],[2,5,6,0,1,3,4]],0.75)
  ```
  On obtient un vecteur de composantes le troisième quartile des colonnes :
  `[4.0,4.0,5.0,5.0,5.0,5.0,5.0]`
- `quartiles` pour calculer le minimum, le premier quartile, la médiane, le troisième quartile et le maximum de series statistiques qui sont les colonnes d'une matrice.
  On tape :

  ```
  quartiles([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],[1,3,4,2,5,6,0],
            [3,4,2,5,6,0,1], [4,2,5,6,0,1,3],
                     [2,5,6,0,1,3,4]])
  ```
  On obtient la matrice, de premiére ligne le minimum de chaque colonne, de deuxième ligne le premier quartile de chaque colonne, de troisième ligne la médiane de chaque colonne, de quatrième ligne le troisième quartile de chaque colonne et de dernière ligne le maximum de chaque colonne :

  ```
  [[0.0,0.0,1.0,0.0,0.0,0.0,0.0],[1.0,1.0,2.0,2.0,1.0,1.0,1.0],
  [2.0,2.0,3.0,3.0,2.0,2.0,3.0],[4.0,4.0,5.0,5.0,5.0,5.0,5.0],
              [6.0,5.0,6.0,6.0,6.0,6.0,6.0]]
  ```
- `boxwhisker` ou `moustache` pour afficher les boites à moustaches de series statistiques qui sont les colonnes d'une matrice.
  On tape :
  ```
  moustache([[6,0,1,3,4,2,5],[0,1,3,4,2,5,6],
        [1,3,4,2,5,6,0], [3,4,2,5,6,0,1],
        [4,2,5,6,0,1,3],[2,5,6,0,1,3,4]])
  ```
  On obtient les dessins des boites à moustaches des series statistiques qui sont les colonnes de la matrice mise comme paramètre.

6.45.38 Dimension d'une matrice : `dim`

`dim` a comme argument une matrice A.
`dim` renvoie la dimension de la matrice A sous la forme d'une liste formée par son nombre de lignes et par son nombre de colonnes.
On tape :

```
dim([[1,2,3],[3,4,5]])
```

On obtient :

```
[2,3]
```

6.45.39 Nombre de lignes : `rowdim rowDim nrows`

`rowdim` (ou `rowDim` ou `nrows`) a comme argument une matrice A.
`rowdim` (ou `rowDim` ou `nrows`) renvoie le nombre de lignes de la matrice A.
On tape :

```
rowdim([[1,2,3],[3,4,5]])
```

ou

```
nrows([[1,2,3],[3,4,5]])
```

On obtient :

```
2
```

6.45.40 Nombre de colonnes : `coldim colDim ncols`

`coldim` (ou `colDim` ou `ncols`) a comme argument une matrice A.
`coldim` (ou `colDim` ou `ncols`) renvoie le nombre de colonnes de la matrice A.
On tape :

```
coldim([[1,2,3],[3,4,5]])
```

ou

```
ncols([[1,2,3],[3,4,5]])
```

On obtient :

```
3
```

6.46 Algèbre linéaire

6.46.1 Transposée d'une matrice : `tran transpose`

`tran` ou `transpose` a comme argument une matrice A.
`tran` ou `transpose` renvoie la matrice transposée de A.
On tape :

```
tran([[1,2],[3,4]])
```

On obtient :

```
[[1,3],[2,4]]
```

6.46.2 Inverse d'une matrice : `inv inverse /`

`inv` a comme argument une matrice carrée A.
`inv` renvoie la matrice inverse de A.
On tape :

```
inv([[1,2],[3,4]])
```

ou

```
inverse([[1,2],[3,4]])
```

ou

```
1/[[1,2],[3,4]])
```

ou

```
A:=[[1,2],[3,4]];1/A
```

On obtient :

```
[[-2,1],[3/2,1/-2]]
```

6.46.3 Trace d'une matrice : `trace`

`trace` a comme argument une matrice A (voir aussi `trace` d'un objet géométrique 9.21.3).
`trace` renvoie la trace de A : c'est la somme des éléments de la diagonale.
On tape :

```
trace([[1,2],[3,4]])
```

On obtient :

```
5
```

6.46.4 Déterminant d'une matrice : `det`

`det` a comme argument une matrice A.
`det` renvoie le déterminant de la matrice A.
On tape :

```
det([[1,2],[3,4]])
```

On obtient :

```
-2
```

On tape :

```
det(idn(3))
```

On obtient :

```
1
```

On peut spécifier l'algorithme de calcul du déterminant en ajoutant un argument optionnel

- `lagrange` : lorsque les coefficients de la matrice sont des polynômes ou des fractions rationnelles, calcule le déterminant par évaluation des variables de ces polynômes ou fractions et interpolation de Lagrange
- `rational_det` : l'algorithme du pivot de Gauss est appliqué sans conversion au format interne pour les fractions, et sans réduction préalable vers des coefficients sans dénominateurs
- `bareiss` : l'algorithme de Gauss-Bareiss est utilisé (réduction sans fraction et division par le pivot de l'étape précédente).
- `minor_det` : l'algorithme utilisé est le développement des mineurs. Ceci nécessite 2^n opérations, mais est parfois plus rapide que les autres algorithmes pour des matrices de taille moyenne (jusquà $n = 20$ environ) à coefficients polynomiaux.

Par défaut, l'algorithme utilisé est choisi parmi Bareiss et Lagrange, celui donnant à priori en fonction des coefficients le temps de calcul le plus rapide est utilisé. Pour les matrices à coefficients entiers, l'algorithme utilisé par défaut est un mix d'une méthode p-adique et d'une méthode modulaire. On commence par résoudre un système linéaire modulo un premier p pour trouver un grand facteur du déterminant, puis on termine le calcul en complétant par les restes chinois avec quelques nombres premiers. On arrête l'algorithme de manière probabiliste (en fonction de la valeur de ϵ=`proba_epsilon`) lorsque le déterminant reconstruit par les restes chinois reste constant pour un produit de nombres premiers supérieur à $1/\epsilon$. Si $\epsilon = 0$, l'algorithme est déterministe, le test d'arrêt utilise la borne de Hadamard du déterminant. Le temps de calcul est en $O(n4\ln(n))$ mais pour des valeurs de n pas trop grandes il ressemble plutot à un $O(n3)$.

6.46.5 Déterminant d'une matrice creuse : `det_minor`

`det_minor` a comme argument une matrice A.
`det_minor` renvoie le déterminant de la matrice A calculé à l'aide du développement du déterminant selon la première ligne en utilisant l'algorithme de Laplace.
On tape :

```
det_minor([[1,2],[3,4]])
```

On obtient :

```
-2
```

On tape :

```
det_minor(idn(3))
```

On obtient :

```
1
```

6.46.6 Rang d'une matrice : rank

rank a comme argument une matrice A.
rank renvoie le rang de la matrice A.
On tape :

```
rank([[1,2],[3,4]])
```

On obtient :

```
2
```

On tape :

```
rank([[1,2],[2,4]])
```

On obtient :

```
1
```

6.46.7 Matrice adjointe : trn

trn a comme argument une matrice A.
trn renvoie la matrice adjointe (transposée de la conjuguée) de A.
On tape :

```
trn([[i, 1+i],[1, 1-i]])
```

On obtient après simplification :

```
[[-i,1],[1-i,1+i]]
```

6.46.8 Matrice équivalente : changebase

changebase a comme argument une matrice A et une matrice de changement de base P.
changebase renvoie la matrice B telle que $B = P^{-1}AP$.
On tape :

```
changebase([[1,2],[3,4]],[[1,0],[0,1]])
```

On obtient :

```
[[1,2],[3,4]]
```

On tape :

```
changebase([[1,1],[0,1]],[[1,2],[3,4]])
```

On obtient :

```
[[-5,-8],[9/2,7]]
```

En effet :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} * \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} -5 & -8 \\ \frac{9}{2} & 7 \end{bmatrix}$$

.

6.46.9 Base d'un sous espace vectoriel : `basis`

`basis` a comme argument la liste des composantes des vecteurs qui engendrent un sous espace vectoriel de $\mathbb{R}^n$.
`basis` renvoie une liste constituée des vecteurs d'une base de ce sous espace vectoriel.
On tape :

```
basis([[1,2,3],[1,1,1],[2,3,4]])
```

On obtient :

```
[[1,0,-1], [0,1,2]]
```

6.46.10 Base de l'intersection de deux sous espaces vectoriels : `ibasis`

`ibasis` a comme argument deux listes de vecteurs qui engendrent deux sous espaces vectoriels de $\mathbb{R}^n$.
`ibasis` renvoie une liste constituée de vecteurs formant une base de l'intersection de ces sous espaces vectoriels.
On tape :

```
ibasis([[1,2]],[[2,4]])
```

On obtient :

```
[[1,2]]
```

6.46.11 Image d'une application linéaire : `image`

`image` a comme argument la matrice d'une application linéaire f dans la base canonique.
`image` renvoie une liste de vecteurs formant une base de l'image de f.
On tape :

```
image([[1,1,2],[2,1,3],[3,1,4]])
```

On obtient :

```
[[-1,0,1],[0,-1,-2]]
```

6.46.12 Noyau d'une application linéaire : `kernel nullspace ker`

`ker` (ou `kernel` ou `nullspace`) a comme argument la matrice d'une application linéaire f dans la base canonique.
`ker` (ou `kernel` ou `nullspace`) renvoie une liste de vecteurs formant une base du noyau de f.
On tape :

```
ker([[1,1,2],[2,1,3],[3,1,4]])
```

On obtient :

```
[[1,1,-1]]
```

Le noyau est donc engendré par le vecteur `[1,1,-1]`.

6.46.13 Noyau d'une application linéaire : `Nullspace`

Attention `Nullspace` n'est utilisable qu'en mode Maple (bouton donnant la ligne d'état puis `Prog style` et choisir `Maple` puis `OK`).
`Nullspace` est la forme inerte de `nullspace`.
`Nullspace` a comme argument la matrice d'une application linéaire f dans la base canonique.
`Nullspace`) suivi de `mod p`renvoie une liste de vecteurs formant une base du noyau de f calculés dans $\mathbb{Z}/p\mathbb{Z}[X]$.
On tape :

```
Nullspace([[1,1,2],[2,1,3],[3,1,4]])
```

On obtient :

```
nullspace([[1,1,2],[2,1,3],[3,1,4]])
```

On tape (en mode Maple) :

```
Nullspace([[1,2],[3,1]]) mod 5
```

On obtient :

```
[2,-1]
```

En mode `Xcas` la commande équivalent est :

```
nullspace([[1,2],[3,1]] % 5)
```

On obtient :

```
[2% 5,-1]
```

6.46.14 Espace engendré par les colonnes d'une matrice : `colspace`

`colspace` a comme argument la matrice A d'une application linéaire f dans la base canonique.
`colspace` renvoie une matrice dont les colonnes sont des vecteurs formant une base de l'espace engendré par les colonnes de A.
`colspace` peut avoir un deuxième argument, le nom d'une variable qui donnera la dimension de l'espace engendré par les colonnes de A.
On tape :

```
colspace([[1,1,2],[2,1,3],[3,1,4]])
```

On obtient :

```
[[-1,0],[0,-1],[1,-2]]
```

On tape :

```
colspace([[1,1,2],[2,1,3],[3,1,4]],dimension)
```

On obtient :

```
[[-1,0],[0,-1],[1,-2]]
```

Puis, on tape :

```
dimension
```

On obtient :

```
2
```

6.46.15 Espace engendré par les lignes d'une matrice : `rowspace`

`rowspace` a comme argument la matrice A d'une application linéaire f dans la base canonique.
`rowspace` renvoie une liste de vecteurs formant une base de l'espace engendré par les lignes de A.
`rowspace` peut avoir un deuxième argument le nom d'une variable qui donnera la dimension de l'espace engendré par les lignes de A.
On tape :

```
rowspace([[1,1,2],[2,1,3],[3,1,4]])
```

On obtient :

```
[[-1,0,-1],[0,-1,-1]]
```

On tape :

```
rowspace([[1,1,2],[2,1,3],[3,1,4]],dimension)
```

On obtient :

```
[[-1,0,-1],[0,-1,-1]]
```

Puis, on tape :

```
dimension
```

On obtient :

```
2
```

6.47 Programmation linéaire

6.47.1 La commande `Xcas` : `simplex_reduce`

Cas le plus simple
La fonction `simplex_reduce` effectue la réduction par l'algorithme du simplexe pour trouver :

$$\max(c.x) \quad \text{avec} \quad A.x \leq b,\ x \geq 0,\ b \geq 0$$

où c, x sont des vecteurs de $\mathbb{R}^n$, $b \geq 0$ est un vecteur de $\mathbb{R}^p$ et A est une matrice de p lignes et de n colonnes.

`simplex_reduce` a comme argument `A,b,c` et renvoie `max(c.x)`, la solution augmentée de `x` et la matrice réduite.
Exemple
Chercher

$$\max(X+2Y) \text{ lorsque } \left\{ \begin{array}{rcl} (X,Y) & \geq & 0 \\ -3X+2Y & \leq & 3 \\ X+Y & \leq & 4 \end{array} \right.$$

On tape :

```
simplex_reduce([[-3,2],[1,1]],[3,4],[1,2])
```

On obtient :

```
7,[1,3,0,0],[[0,1,1/5,3/5,3],[1,0,(-1)/5,2/5,1],
                [0,0,1/5,8/5,7]]
```

Ce qui veut dire que le maximum de `X+2Y` sous ces conditions est `7`, il est obtenu pour `X=1,Y=3` car `[1,3,0,0]` est la solution augmentée et la matrice réduite est :
`[[0,1,1/5,3/5,3],[1,0,(-1)/5,2/5,1], [0,0,1/5,8/5,7]]`.

Un cas plus compliqué qui se ramène au cas simple
`simplex_reduce` oblige à réécrire les contraintes impliquant une seule variable pour qu'elles soient sous la forme $x_k \geq 0$, puis à éliminer les variables sans contraintes puis à ajouter des variables afin d'avoir comme contraintes : toutes les composantes des éléments du simplexe sont positives. Par exemple, si on part du problème :

$$\min(2x+y-z+4) \text{ lorsque } \left\{ \begin{array}{rcl} x & \leq & 1 \\ y & \geq & 2 \\ x+3y-z & = & 2 \\ 2x-y+z & \leq & 8 \\ -x+y & \leq & 5 \end{array} \right.$$

on pose $x = 1-X$, $y = Y+2$, $z = 5-X+3Y$ le problème devient chercher le minimum de $(-2X+Y-(5-X+3Y)+8)$ lorsque

$$\left\{ \begin{array}{rcl} X & \geq & 0 \\ Y & \geq & 0 \\ 2(1-X)-(Y+2)+5-X+3Y & \leq & 8 \\ -(1-X)+(Y+2) & \leq & 5 \end{array} \right.$$

donc chercher le minimum de :

$$(-X-2Y+3) \text{ lorsque } \left\{ \begin{array}{rcl} X & \geq & 0 \\ Y & \geq & 0 \\ -3X+2Y & \leq & 3 \\ X+Y & \leq & 4 \end{array} \right.$$

ce qui revient à chercher le maximum de $-(-X-2Y+3) = X+2Y-3$ sous les mêmes conditions, on est donc ramené au problème précédent (le maximum est donc de 7-3=4).

Cas général
Tous les cas ne se ramènent pas directement au cas simple ci-dessus. On verra plus loin comment les traiter, cela nécessitera d'utiliser une autre forme d'appel de `simplex_reduce`, que l'on peut d'ailleurs aussi utiliser dans le cas simple de la manière suivante : si `A` a p lignes et n colonnes et si on définit :

```
B:=augment(A,idn(p)); C:=border(B,b);
d:=append(-c,0$(p+1)); D:=augment(C,[d]);
```

`simplex_reduce` accepte aussi en argument `D`.
Pour l'exemple précédent, on tape :

```
A:=[[-3,2],[1,1]];B:=augment(A,idn(2));
C:=border(B,[3,4]); D:=augment(C,[[-1,-2,0,0,0]])
```

On a `C=[[-3,2,1,0,3],[1,1,0,1,4]]`
et `D=[[-3,2,1,0,3],[1,1,0,1,4],[-1,-2,0,0,0]]`
On tape :

```
simplex_reduce(D)
```

On obtient le même résultat que précédemment.

6.47.2 Écriture matricielle et algorithme du simplexe

Un simplexe est une portion de $\mathbb{R}^n$ délimitée par des hyperplans.
L'algorithme du simplexe sert à maximiser une fonction linéaire sous des contraintes d'égalité ou d'inégalité linéaire. Le principe de l'algorithme consiste à trouver un sommet du simplexe défini par les contraintes, puis à sélectionner une arête et la suivre jusqu'au sommet suivant, la sélection se faisant de sorte à ne jamais diminuer la valeur de la fonction linéaire. On effectue donc les étapes suivantes

- réécrire les contraintes impliquant une seule variable sous la forme $x_k \geq 0$, éliminer si possible par pivot de Gauss les variables sans contraintes (ou ajouter des variables). À la fin de cette étape, on doit avoir pour contraintes que toutes les composantes des éléments du simplexe sont positives.
- transformer si nécessaire les contraintes d'inégalité restante en égalité (par ajout de variables d'écart) avec un second membre positif,
- construire un sommet s'il en existe (ce qui dans les cas non évidents se fait en optimisant une fonction artificielle),
- passer de sommet à sommet.

Un sommet va être caractérisé dans la suite par ses composantes nulles et ses composantes non nulles. La représentation matricielle associée à un sommet des contraintes d'égalité fait apparaitre une sous-matrice identité dans les colonnes correspondant aux composantes non nulles du sommet. Par exemple, si on part du problème, chercher :

$$\max(2x+y-z+4) \text{ lorsque } \left\{ \begin{array}{rcl} x & \leq & 1 \\ y & \geq & 2 \\ x+3y-z & = & 2 \\ 2x-y+z & \leq & 8 \end{array} \right.$$

on pose $x = 1 - X$, $y = Y + 2$, donc on cherche :

$$\max(-2X+Y-z+8) \text{ lorsque } \left\{ \begin{array}{rcl} X & \geq & 0 \\ Y & \geq & 0 \\ 1-X+3(Y+2)-z & = & 2 \\ 2(1-X)-(Y+2)+z & \leq & 8 \end{array} \right.$$

puis on élimine z par la contrainte d'égalité $z = 5 - X + 3Y$ et on cherche :

$$\max(-2X+Y-(5-X+3Y)+8) \text{ lorsque } \left\{ \begin{array}{rcl} X & \geq & 0 \\ Y & \geq & 0 \\ -2X-Y+5-X+3Y & \leq & 8 \end{array} \right.$$

cela revient donc à chercher :

$$\max(-X-2Y+3) \text{ lorsque } \left\{ \begin{array}{rcl} (X,Y) & \geq & 0 \\ -3X+2Y & \leq & 3 \end{array} \right.$$

Il faut ici ajouter une variable d'écart $t \geq 0$ pour transformer la dernière inégalité en égalité :
on pose $t = 3 - (-3X + 2Y)$ et on a alors
$t \geq 0$ et $-3X + 2Y + t = 3$ est équivalent à $-3X + 2Y \leq 3$, et cela donne un sommet de départ évident. La matrice correspondant à ces conditions est alors la ligne

$$(-3, 2, 1, 3)$$

le sommet de départ associé est $(X, Y, t) = (0, 0, 3)$ la sous-matrice identité utilise la 3ème colonne (t est le coefficient non nul). On pourrait passer le coefficient non nul en 2ème colonne (sommet $(X, Y, t) = (0, 3/2, 0)$) en réécrivant l'égalité sous la forme

$$(-3/2, 1, 1/2, 3/2)$$

mais on ne pourrait passer le coefficient non nul en 1ère colonne (il n'y a en effet que 2 sommets à ce simplexe).
Avec `Xcas`, on tape :

```
simplex_reduce([[-3,2]],[3],[-1,-2])
```

On obtient :

```
(0,[0,0,3],[[-3,2,1,3],[1,2,0,0]])
```

En général, le passage d'un sommet à un autre sommet consiste alors à effectuer une opération de réduction de Gauss qui fait sortir une colonne et entrer une autre colonne dans les composantes non nulles du sommet, ce qui revient à déplacer la sous-matrice identité. On doit prendre garde à effectuer l'opération de réduction de Gauss en conservant la positivité du membre de droite des contraintes de l'égalité, et ce sans diminuer la valeur de la fonction à optimiser. Pour pouvoir réaliser cette dernière condition, on augmente la matrice des contraintes d'égalité par une ligne formée des opposés des coefficients de la fonction à maximiser, sauf en dernière colonne (on y met le coefficient constant de la fonction à maximiser). Dans l'exemple ce serait

$$\left(\begin{array}{cccc} -3 & 2 & 1 & 3 \\ 1 & 2 & 0 & 3 \end{array} \right)$$

On crée (si nécessaire) des zéros dans cette dernière ligne dans les colonnes correspondant à la sous-matrice identité (donc aux composantes non nulles du sommet), de sorte que le coefficient constant de la ligne (après cette réduction) représente la valeur de la fonction à optimiser en ce sommet. Plus générallement pour un point du simplexe, au cours de la réduction la somme de la fonction à maximiser et du produit scalaire des coordonnés du point avec cette dernière ligne (privé du dernier coefficient) vaut le dernier coefficient. On aura donc un sommet réalisant le maximum si tous les coefficients de la ligne sont positifs (sauf le dernier). C'est le cas ici, le maximum est 3, atteint pour $X = Y = 0$ (donc $x = 1$, $y = 2$ et $z = 5$).

L'instruction `simplex_reduce` de `Xcas` applique l'algorithme du simplexe dans deux situations distinctes : soit le problème est posé sous forme "canonique" et on lui donne 3 arguments (cf. infra), soit on lui donne un seul argument qui doit être une matrice "associée à un sommet" au sens de ce paragraphe.

6.47.3 Premier cas : 3 arguments

Lorsqu'on lui passe 3 arguments A, b, c, la fonction `simplex_reduce` calcule le maximum (s'il existe) de $c.x$ pour c vecteur fixé de $\mathbb{R}^n$ et x variable, sous les conditions $x \geq 0$ et $A.x \leq b$ (avec A et $b \geq 0$ fixés). Ce problème est appelé "forme canonique".

`Xcas` ajoute m variables d'écart $y_1, ..., y_m$ (m=nombre de lignes de A) pour transformer les inégalités en égalités, puis choisit comme sommet de départ évident toutes les variables de départ nulles et les variables d'écart valant b. Il construit donc la matrice

$$\begin{pmatrix} A & I & b^t \\ -c & 0 & 0 \end{pmatrix}$$

Ensuite il se déplace en suivant des arêtes du simplexe défini par les conditions $Ax + y = b, x, y \geq 0$ en augmentant le plus possible la valeur de $c.x$. Cela se fait en cherchant dans la dernière ligne un coefficient négatif strict (soit le plus négatif possible, soit le premier négatif), qui représentera une colonne entrant dans la sous-matrice identité (en rendant son coefficient non nul dans les composantes du sommet, on augmentera au sens large la valeur de $c.x$). S'il n'y a pas de coefficient négatif, on arrête l'algorithme (on verra que le maximum est le coefficient en bas à droite de la matrice). S'il existe, on sélectionne cette colonne comme colonne du pivot (colonne entrante dans la matrice identité), il nous reste à déterminer la ligne du pivot utilisé (c'est la colonne sortant de la matrice identité) :

- d'une part, le pivot utilisé doit être positif, en effet on va diviser la ligne du pivot par la valeur du pivot, et le coefficient constant de cette ligne (qui sera la valeur d'une coordonnée d'un sommet) doit rester positif.
- d'autre part les autres coefficients du sommet doivent aussi rester positifs. Pour réaliser cela, on calcule de la ligne 1 à m les rapport des coefficients de cette ligne dernière colonne avec le coefficient de cette ligne, colonne du pivot, en cherchant la ligne qui donne un rapport positif le plus petit possible. S'il n'existe pas de telle ligne, le maximum est alors $+\infty$ (car on peut indéfiniment augmenter la valeur de la composante ayant ce numéro de colonne en restant dans le domaine). Si une telle ligne existe, on se sert du coefficient de cette ligne/colonne comme d'un pivot, et on crée un 1 à cette ligne et des 0 ailleurs dans cette colonne par combinaisons linéaires de lignes.

Au cours de l'algorithme, les m premières lignes de la matrice contiennent toujours une sous-matrice identité m, m (puisqu'on fait du pivot de Gauss), et les coefficients de la dernière ligne qui correspondent à cette sous-matrice identité sont nuls (pour la même raison). On a donc une matrice de la forme

$$\begin{pmatrix} B^{-1}A & B^{-1} & B^{-1}b \\ -c + c_B B^{-1}A & c_B B^{-1} & c_B B^{-1}b \end{pmatrix}$$

où B est une sous-matrice extraite de A, I (correspondant à des colonnes de la dernière ligne ayant pour coefficients 0) et c_B est la liste des coefficients de c correspondant aux mêmes colonnes de A, I que B. Pour éviter de boucler indéfiniment si le coefficient en bas à droite est constant, on peut garder en mémoire dans une table les colonnes correspondant à l'identité et se refuser à revenir à une configuration précédente.

Si on ne quitte pas l'algorithme (maximum=$+\infty$), à la fin, la dernière ligne ne contient que des coefficients positifs ou nuls. On a de plus l'identité fonction à optimiser + produit scalaire entre la dernière ligne et (x, y) est égal au coefficient en bas à droite (=$c_B B^{-1} b$). Comme les coefficients de la dernière ligne sont positifs ou nuls, de même que les composantes de x et y, on en déduit que la fonction à optimiser est inférieure au coefficient en bas à droite. D'autre part, cette valeur est atteinte au sommet correspondant.

Le résultat renvoyé par `simplex_reduce` est une séquence composée de la valeur du maximum, d'une solution augmentée (les premières composantes sont celles de la solution, les composantes suivantes celles des variables ajoutées artificiellement pour transformer $Ax \leq b$ en une égalité) et de la matrice de l'algorithme du simplexe après réduction.

Exemple : si on cherche le maximum de $3x_1 + x_2 + 3x_3$ sous les conditions $x_1, x_2, x_3 \geq 0$ et

$$\left\{ \begin{array}{rcl} 2x_1 + x_2 + x_3 & \leq & 2 \\ x_1 + 2x_2 + 3x_3 & \leq & 5 \\ 2x_1 + 2x_2 + x_3 & \leq & 6 \end{array} \right.$$

on prend `A:=[[2,1,1],[1,2,3],[2,2,1]]`,`b:=[2,5,6]` et `c:=[3,1,3]`, donc on tape :

```
simplex_reduce([[2,1,1],[1,2,3],[2,2,1]],[2,5,6],[3,1,3])
```

et on obtient `27/5` comme maximum, `[1/5,0,8/5,0,0,4]` comme solution augmentée (donc `[1/5,0,8/5]` est solution), ainsi que la matrice réduite.

6.47.4 Deuxième cas : un argument

On peut aussi passer un seul argument à `simplex_reduce`, cet argument étant du type des matrices construites précédemment, c'est-à-dire que si on enlève la dernière ligne et la dernière colonne, on doit pouvoir extraire une sous-matrice identité de taille maximale.

On va voir comment on peut utiliser cette forme d'appel de `simplex_reduce` pour résoudre des problèmes de programmation linéaire plus généraux que ceux de la section précédente.

6.47.5 Passage de la forme standard à la forme canonique

La forme standard d'un problème d'optimisation linéaire est analogue à la forme canonique, mais en remplacant $Ax \leq b$ par $Ax = b$ avec $b \geq 0$ et où on cherche $x \geq 0$ (on peut s'y ramener si nécessaire en changeant le signe d'une ligne).

Le problème pour appliquer l'algorithme précédent est qu'on ne peut pas ajouter de variables d'écarts et donc qu'on n'a pas de valeur x évidente dans le domaine de maximisation (dit autrement on n'a pas de sous-matrice identité dans la formulation matricielle du problème et il n'y a pas d'opération de lignes évidente qui permette de le faire).

On va se ramener à des problèmes sous forme "canonique" par la méthode dite en 2 phases. Soit m le nombre de lignes de A (nombre de conditions dans $Ax = b$).

On ajoute des variables artificielles $y_1, ..., y_m$ et on maximise $-\sum y_i$ sous condition $Ax = b, x \geq 0, y \geq 0$ en partant de la valeur initiale 0 pour les variables non artificielles et b pour y (on appelle donc `simplex_reduce` avec un argument matrice obtenue en augmentant A par l'identité, b inchangé et un c artificiel formé de 0 au début et de 1 en dessous de l'identité (que `simplex_reduce` va commencer par annuler)). Si le maximum existe et est 0, on obtiendra une sous-matrice identité dans les colonnes correspondants à x, et on pourra éliminer les variables artificielles (dont la valeur sera 0 pour atteindre l'optimum). Il restera à appliquer à nouveau l'algorithme du simplexe mais avec le c original (comme les coefficients de c correspondant à l'identité n'ont pas de raison d'être nuls, on doit commencer par les rendre nuls, ce que `simplex_reduce` fait si on permute les colonnes pour placer la sous-matrice identité à droite).

Exemple : on cherche le minimum de $2x + 3y - z + t$ avec les $x, y, z, t \geq 0$ et :

$$\left\{ \begin{array}{rcl} -x - y + t & = & 1 \\ y - z + t & = & 3 \end{array} \right.$$

Ceci revient à calculer l'opposé du maximum de $-(2x+3y-z+t)$. On ajoute donc deux variables artificielles y_1 et y_2, on tape directement la matrice de l'algorithme

```
simplex_reduce([[-1,-1,0,1,1,0,1],
[0,1,-1,1,0,1,3],
[0,0,0,0,1,1,0]])
```

On obtient comme optimum 0, avec 0 comme valeurs pour les variables artificielles, et la matrice

$$\begin{pmatrix} -1/2 & 0 & -1/2 & 1 & 1/2 & 1/2 & 2 \\ 1/2 & 1 & -1/2 & 0 & -1/2 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Ceci signifie aussi qu'une valeur initiale dans le domaine est $(0, 1, 0, 2)$. Il nous reste à optimiser le problème de départ en remplacant les lignes de $Ax = b$ par les 2 premières lignes de la matrice réponse ci-dessus dont on enlève les variables artificielles et en ajoutant la ligne de la fonction à optimiser. On prend donc la matrice

```
simplex_reduce([[-1/2,0,-1/2,1,2],
[1/2,1,-1/2,0,1],[2,3,-1,1,0]])
```

On obtient comme maximum -5, donc le minimum de l'opposé est 5, obtenu pour $(0, 1, 0, 2)$ soit après replacement des colonnes $t = 2$ et $y = 1$, les autres nuls.

Pour plus de détails, chercher sur google `simplex algorithm` ou dans une bibliothèque Ciarlet "Introduction à l'analyse matricielle et à l'optimisation".

6.48 Les différentes norme d'une matrice

Voir aussi 6.41.1 pour les différentes norme d'un vecteur.

6.48.1 Norme de Frobenius d'une matrice : `frobenius_norm`

`frobenius_norm` a comme argument une matrice A (voir aussi 6.41.1).
`frobenius_norm` renvoie la norme de Frobenius de A i.e.
$\sqrt{\sum_{j,k} a_{j,k}^2}$ si l'argument est $A = a_{j,k}$.
On tape :

```
B:=[[1,2,3],[3,-9,6],[4,5,6]]
```

puis,

```
frobenius_norm(B)
```

On obtient :

```
sqrt(217)
```

En effet : $\sqrt{1+4+9+9+81+36+16+25+36} = \sqrt{217} \simeq 14.7309198627$

6.48.2 Norme d'une matrice avec la norme des lignes : `rownorm` `rowNorm`

`rownorm` (ou `rowNorm`) a comme argument une matrice A.
`rownorm` (ou `rowNorm`) renvoie $\max_k(\sum_j |a_{j,k}|)$ si l'argument est $A = a_{j,k}$.
On tape :

```
rownorm([[1,2],[3,-4]])
```

ou

```
rowNorm([[1,2],[3,-4]])
```

On obtient :

```
7
```

En effet : $\max(1+2, 3+4) = 7$

6.48.3 Norme d'une matrice avec la norme des colonnes : `colnorm` `colNorm`

`colnorm` (ou `colNorm`) a comme argument une matrice A.
`colnorm` (ou `colNorm`) renvoie $\max_j(\sum_k(|a_{j,k}|))$ si l'argument est $A = a_{j,k}$.
On tape :

```
colnorm([[1,2],[3,-4]])
```

ou

```
colNorm([[1,2],[3,-4]])
```

On obtient :

```
6
```

En effet : $\max(1+3, 2+4) = 6$

6.48.4 La triple norme d'une matrice : `matrix_norm` et `l1norm`, `l2norm` ou `norm` ou `specnorm`, `linfnorm`

`matrix_norm` a deux arguments : une matrice `A` et `1` ou `2` ou `inf` et par défaut `1`.
`matrix_norm` renvoie la triple norme subordonnée à l_1 (resp l_2 ou l^∞) si il n'y a pas de second argument ou si le second argument est `1` (resp `2` ou `inf`).
`matrix_norm(A)` ou `matrix_norm(A,1)` c'est aussi `l1norm(A)` ou `colnorm(A)`,
`matrix_norm(A,2)` c'est aussi `l2norm(A)` ou `SPECNORM(A)` ou `max(SVL(A))`,
`matrix_norm(A,inf)` c'est aussi `linfnorm(A)` ou `rownorm(A)`.
Pour les différentes normes de vecteurs (voir aussi 6.41.1, 6.41.1 et 6.41.1).
Pour plus de détails sur la triple norme voir le **Rappel** situé après les exemples.
On tape :

```
B:=[[1,2,3],[3,-9,6],[4,5,6]]
```

Puis

```
matrix_norm(B)
```

ou

```
matrix_norm(B,1)
```

ou

```
l1norm(B)
```

ou

```
colnorm(B)
```

On obtient :

```
16
```

En effet $\texttt{max}(1+3+4, 2+9+5, 3+6+6) = 16$
On tape :

```
matrix_norm(B,2)
```

ou

```
l2norm(B)
```

ou

```
SPECNORM(B)
```

ou

```
max(SVL(B))
```

On obtient :

```
11.2449175989
```

En effet `max(SVL(B))` renvoie la plus grande racine carrée des valeurs propres de `trn(B)*B`.
`sqrt(proot(pcar(trn(B)*B)))` ou `sqrt(EIGENVAL(trn(B)*B))` renvoie :
`[9.48552308331,0.759394515579,11.2449175989]`
On tape :

```
matrix_norm(B,inf)
```

ou

```
linfnorm(B)
```

ou

```
rownorm(B)
```

On obtient :

```
18
```

En effet $\mathtt{max}(1+2+3, 3+9+6, 4+5+6) = 18$

Rappel
En mathématiques, et plus particulièrement en analyse fonctionnelle, une norme d'opérateur ou norme subordonnée est une norme définie sur l'espace des opérateurs bornés entre deux espaces vectoriels normés. Entre deux tels espaces, les opérateurs bornés ne sont autres que les applications linéaires continues.
On va considérer ici que les applications linéaires sur des espaces vectoriels de dimension finie.
Théorème
Soient E et F 2 espaces vectoriels normés (de norme $\| \|_E$ et $\| \|_F$) de dimension finie et f une application linéaire de E dans F.
Alors il existe une constante réelle K tel que pour tout $x \in E$ on ait :
$\|f(x)\|_F \leq K\|x\|_E$
f est donc lipschitzienne sur E et continue de E dans F.
Définition de la triple norme
D'après ce qui précéde, on a :
pour tout $x \in E$ si $\|x\|_E \leq 1$ alors on a $\|f(x)\|_F \leq K$.
Donc l'ensemble $\{\|f(x)\|_F \text{ pour} \|x\|_E \leq 1\}$ est une partie non vide et majorée de $\mathbb{R}$. Cet ensemble admet donc une borne supérieure que l'on appelle **la triple norme**.
Ainsi

$$|\!|\!|f|\!|\!| = \sup_{\|x\|_E \leq 1} \|f(x)\|_F$$

Attention La valeur de $|\!|\!|f|\!|\!|$ dépend des normes $\| \|_E$ et $\| \|_F$ utilisées.
la triple norme est donc une norme subordonnée aux normes de E et F.

Dans le cas où $E = F$, on choisit usuellement $\| \|_E = \| \|_F$ (même si ce n'est pas obligatoire).
Pour les normes usuelles, on dispose de formules pratiques : prenons $E = \mathbb{R}^n$ et $f \in L(E)$. Notons $x = (x_1, ..., x_n)$ un vecteur quelconque de $\mathbb{R}^n$ et $A = (a_{jk})$ la matrice de f dans la base canonique. On a alors :

– Si $\|x\|_E = \max_{0\leq j\leq n-1}|x_j|$ (c'est la norme l^∞), alors la norme de f vaut :

$$\max_{0\leq j\leq n-1}\sum_{0\leq k\leq n-1}|a_{jk}|$$

C'est ce qui est défini dans `Xcas` par `rownorm(A)`

– Si $\|x\|_E = \sum_{1\leq i\leq n}|x_i|$ (c'est la norme l_1), alors la norme de f vaut :

$$\max_{0\leq k\leq n-1}\sum_{0\leq j\leq n-1}|a_{jk}|$$

C'est ce qui est noté dans `Xcas` : `colnorm(A)`

– Si $\|x\|_E = \sqrt{\sum_{0\leq j\leq n-1}x_j^2}$ (c'est la norme l_2 ou euclidienne, associée au produit scalaire canonique), alors la norme triple de f est la racine carrée de la plus grande valeur propre de $f^*\circ f$, où f^* désigne l'adjoint de f.
La norme triple de f est donc sa plus grande valeur singulière. Ceci se généralise en remplaçant $\mathbb{R}^n$ par n'importe quel espace de Hilbert.
C'est ce qui est noté dans `Xcas` : `max(SVL(A))` : c'est la plus grande valeur singulière de `A` i.e la plus grande valeur de la racine carrée des valeurs propres de `trn(A)*A`.
Pour tout endomorphisme symétrique g (en particulier pour $g = f^*\circ f$), la norme de g est égale à son rayon spectral, qui est la plus grande des valeurs absolues de ses valeurs propres.

Démonstration Soient $E = R^n$ et $F = R^p$ munis de leur base canonique
Soit $e_1..e_n$ est la base canonique de E
Soit $a_{j,k}$ la matrice associée à f dans les bases canoniques de E et F.

Montrons que pour la norme `l1norm`, la triple norme de `A` nommée ici `matrix_norm(A,1)` c'est `colnorm`.
Soit $x = \sum_k(x_k * e_k)$
On a donc :
`l1norm`$(f(x))$ =`l1norm`$(\sum_k x_k * f(e_k)) \leq \sum_k |x_k|*$`l1norm`$(f(e_k))$
`l1norm`$(f(x)) \leq \sum_k |x_k|*\max_k($`l1norm`$(f(e_k)))$ =`l1norm`$(x)*\max_k(\sum_j |a_{j,k}|)$
donc
`matrix_norm(A,1)<=colnorm(A)`
Montrons que ce maximum est atteint.
Soit k_0 tel que :
$\max_k(\sum_j |a_{j,k}|) = \sum_j |a_{j,k_0}|$ =`colnorm(A)`
On a alors `l1norm`$(e_{k_0}) = 1$ et `l1norm`$(f(e_{k_0})) = \sum_j |a_{j,k_0}|$
donc `matrix_norm(A,1)=colnorm(A)`.

Montrons que pour la norme du maximum la triple norme de `A` nommée ici `matrix_norm(A,inf)` c'est `rownorm`.
Soit $x = \sum_k x_k * e_k$ avec $\max_k(|x_k|) \leq 1$
On a :
`maxnorm`$(f(x))$=`maxnorm`$(\sum_k x_k * f(e_k)) =$
`maxnorm`$(f(x)) = \max_j(|\sum_k x_k * ajk|) \leq \max_j(\sum_k |x_k| * |a_{j,k}|)$ puisque $\max_k |x_k) \leq 1$
`maxnorm`$(f(x)) \leq \max_j(\sum_k |a_{j,k}|)$ =`colrow(A)`
Montrons que ce maximum est atteint.
Soit j_0 tel que $\max_j(\sum_k |a_{j,k}|) = \sum_k |aj_0,k|$ =`rownorm(A)`.

Soit $x_O = \sum_k \texttt{sign}(a_{j_0,k}) * e_k$ alors $\texttt{maxnorm}(x_0) = 1$ et
$\texttt{maxnorm}(f(x_0)) = \max_j(f(x_0)) = \max_j(|\sum_k \texttt{sign}(a_{j_0,k}) * a_{j,k}|) =$
$\max(\sum_k |a_{j_0,k}|, \max_{j!=j_0}(|\sum_k \texttt{sign}(a_{j_0,k}) * a_{j,k}|))$
si $j! = j_0$ on a :
$|\sum_k \texttt{sign}(a_{j_0,k}) * a_{j,k})| \leq \sum_k |a_{j,k}| \leq \sum_k |a_{j_0,k}|$
Donc
$\texttt{maxnorm}(f(x_0)) = \sum_k |a_{j_0,k}|$ =`rownorm(A)`
Donc `matrix_norm(A,inf)=rownorm(A)`.

Montrons que pour la norme `l2norm`, la triple norme de la matrice `A` nommée ici `matrix_norm(A,2)` c'est `max(SVL(A))` c'est à dire la plus grande racine carrée des valeurs propres de `trn(A)*A`.
Soit $x = \sum_k (x_k * e_k)$
On a :
`l2norm(x)=sqrt(scalar_product(x,x))`
0n note $< x, x >= \sqrt{\sum_k x_k^2}$=`scalar_product(x,x)`
`l2norm(f(x))=sqrt(scalar_product(A*x,A*x))`
On a : $< Ax, Ax >=< x, (A)^t Ax >$ donc
`l2norm(f(x))=sqrt(scalar_product(x,trn(A)*A*x))`
La matrice $M = trn(A) * A$ est symétrique donc diagonalisable et ses valeurs propres λ_k sont réelles.
Il existe une matrice B diagonale et P une matrice de passage orthogonale (i.e. `inv(P)=trn(P)`) tels que :
$B = tran(P) * M * P$ i.e. $M = trn(A) * A = P * B * trn(P)$
On a de plus :
$Mv_k = \lambda_k v_k$ et
$< Av_k, Av_k >=< v_k, Mv_k >=< v_k, \lambda_k v_k >= \lambda_k < v_k, v_k >_k$
Comme $< Av_k, Av_k >\geq 0$ on en déduit que les λ_k sont positifs.
Donc `l2norm(f(x))=sqrt(scalar_product(trn(P)*x,B*trn(P)*x))`
Si $y = trn(P) * x = (y_0...y_{n-1})$ on a :
$< y, y >=< x, P * trn(P)x >=< x, x >$
$B * y = \sum_k \lambda_k * y_k$ et $< y, B * y >= \sum_k \lambda_k * y_k^2$
Donc :
`l2norm(f(x))`$= \sqrt{\sum_k \lambda_k * y_k^2}$
Or puisque $\lambda_k > 0$ on a :
$\sum_k \lambda_k * y_k^2 \leq max_k(\lambda_k) \sum_k y_k^2$ et
$\sqrt{\lambda_k}$ existe
et
$\sqrt{\sum_k y_k^2} =< y, y >=< x, x >= \sqrt{\sum_k x_k^2}$
Donc :
`l2norm(f(x))`$\leq \max_k(\sqrt{\lambda_k}) * \sqrt{\sum_k x_k^2}$
Montrons que ce maximum est atteint.
Soit $\lambda_m = max_k(\lambda_k)$ et v_m le vecteur propre associé.
On a alors :
`l2norm(f(vm))= sqrt(scalar_product(A*vm,A*vm))`
`l2norm(f(vm))`$= \sqrt{\lambda_m} * < v_m, v_m > = \sqrt{\lambda_m}$*`l2norm(vm)`. Donc puisque $\sqrt{\lambda_m}$ est la plus grande racine carrée des valeurs propres de `trn(A)*A`, on a :

$\sqrt{\lambda_m}$=`max(SVL(A))`.
Donc : `matrix_norm(A,2)=max(SVL(A))`

6.48.5 Norme : `COND cond`

`COND` ou `cond` a a deux arguments : une matrice `A` inversible et `1` ou `2` ou `inf` et par défaut `1`.
`COND` renvoie le nombre de condition de cette matrice `A` c'est à dire :
$\|A\| * \|A^{-1}\|$ où $\|\ \|$ désigne la norme triple d'une matrice inversible qui est subordonnée à l_1 (resp l_2 ou l^∞) si il n'y a pas de second argument ou si le second argument est `1` (resp `2` ou `inf`) On a donc :
`COND(A)` ou `cond(A)` ou `cond(A,1)` renvoie le nombre de condition de `A` pour la norme l_1,
`COND(A,2)` ou `cond(A,2)` renvoie le nombre de condition de `A` pour la norme l_2.
`COND(A,inf)` ou `cond(A,inf)` renvoie le nombre de condition de `A` pour la norme l^∞.

- La norme triple de la matrice `A` subordonnée à l_1 est `colnorm` qui est :
  `max(sum(abs(A[j,k]),j=0..size(A)-1)$(k=0..size(A)-1))`
  i.e. le maximum des sommes des valeurs absolues des éléments situés sur les colonnes de la matrice.
  `COND(A)` ou `cond(A)` ou `cond(A,1)` renvoie donc :
  `colnorm(A)*colnorm(inv(A))`.
- La norme triple de la matrice `A` subordonnée à l_2 est `max(SVL(A))` :
  c'est la plus grande valeur singulière de `A` i.e la plus grande valeur de la racine carrée des valeurs propres de `A*trn(A)`.
  `COND(A,2)` ou `cond(A,2)` renvoie donc :
  `max(SVL(A))*max(SVL(inv(A)))`
- La norme triple de la matrice `A` subordonnée à l^∞ est `rownorm` qui est :
  `max(sum(abs(A[j,k]),k=0..size(A)-1)$(j=0..size(A)-1))`
  i.e. le maximum des sommes des valeurs absolues des éléments situés sur les lignes de la matrice.
  `COND(A,inf)` ou `cond(A,inf)` renvoie donc :
  `rownorm(A)*rownorm(inv(A))`.

On tape :

```
COND([[1,2,3],[3,-9,6],[4,5,6]])
```

On obtient :

```
27.8518518519
```

En effet, si `B:=[[1,2,3],[3,-9,6],[4,5,6]]` on tape :
`colnorm(B),colnorm(inv(B)),colnorm(B)*colnorm(inv(B))`
On obtient :
`16,47/27,752/27` et $752/27 \simeq 27.8518518519$
On tape :

```
COND([[1,2,3],[3,-9,6],[4,5,6]],2)
```

On obtient :

```
14.807741389
```

En effet, si `B:=[[1,2,3],[3,-9,6],[4,5,6]]` on tape :
`max(SVL(B)),max(SVL(inv(B))),max(SVL(B))*max(SVL(inv(B)))`
On obtient :
`11.2449175989,1.31683858585,14.807741389`
On tape :

```
COND([[1,2,3],[3,-9,6],[4,5,6]],inf)
```

On obtient :

```
28
```

En effet, si `B:=[[1,2,3],[3,-9,6],[4,5,6]]` on tape :
`rownorm(B),rownorm(inv(B)),rownorm(B)*rownorm(inv(B))`
On obtient :
`18,14/9,28`

6.49 Réduction des matrices

6.49.1 Valeurs propres : `eigenvals eigenvalues`

`eigenvals` ou `eigenvalues`) a comme argument une matrice A d'ordre n.
`eigenvals` ou `eigenvalues`) renvoie une suite constitué par les n valeurs propres de A.
Remarque : Si A est symbolique, on peut avoir en réponse des valeurs propres numériques car il faut que `Xcas` puisse factoriser le polynôme caractéristique formellement !
On tape :

```
eigenvals([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
(2,2,2)
```

On tape :

```
eigenvals([[4,1,0],[1,2,-1],[2,1,0]])
```

On obtient :

```
(0.324869129433,4.21431974338,1.46081112719)
```

6.49.2 Matrice de Jordan : `egvl eigVl`

`egvl` (ou `eigVl`) a comme argument une matrice A d'ordre n.
`egvl` (ou `eigVl`) renvoie la matrice de Jordan associée à A.
Remarque : Si A est symbolique, on peut avoir en réponse des valeurs propres numériques car il faut que `Xcas` puisse factoriser le polynôme caractéristique formellement !
On tape :

```
egvl([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
[[2,1,0],[0,2,1],[0,0,2]]
```

On tape :

```
egvl([[4,1,0],[1,2,-1],[2,1,0]])
```

On obtient :

```
[[0.324869129433,0,0],[0,4.21431974338,0],[0,0,1.46081112719]]
```

6.49.3 Vecteurs propres : `egv eigenvectors eigenvects eigVc`

`egv` (ou `eigenvectors eigenvects eigVc`) a comme argument une matrice A d'ordre n.
`egv` (ou `eigenvectors eigenvects eigVc`) renvoie la matrice de passage d'une matrice diagonalisable (les colonnes du résultat sont les vecteurs propres de la matrice résultat).
On tape :

```
egv([[1,1,3],[1,3,1],[3,1,1]])
```

On obtient :

```
[[-1,1,1],[2,1,0],[-1,1,-1]]
```

On tape :

```
egv([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
"Not diagonalizable at eigenvalue 2"
```

En mode complexe on tape :

```
egv([[2,0,0],[0,2,-1],[2,1,2]])
```

On obtient :

```
[0,1,0],[-1,-2,-1],[i,0,-i]]
```

6.49.4 Matrice de Jordan rationnelle : `rat_jordan`

`rat_jordan` a comme argument une matrice A d'ordre n à coefficients rationnels.
`rat_jordan` renvoie :

– en mode `Xcas`, `Mupad` ou `TI`
  une séquencee composée de deux matrices : une matrice de changement de base P, suivie de la matrice de Jordan J associée à A si celle-ci est rationnelle ou de la matrice dont les blocs de Jordan sont remplacés par les matrices compagnons associées au facteur non rationnellement reductible du polynôme caractéristique de A (c'est la matrice la plus reduite à coefficient dans le corps de A, ou le corps complexifié de A si on est en mode complexe).

– en mode `Maple`
  la matrice de Jordan J associée à A si celle-ci est rationnelle ou de la matrice dont les blocs de Jordan sont remplacés par les matrices compagnons associées au facteur non rationnellement reductible du polynôme caractéristique de A. On peut cependant obtenir la matrice de changement de base P, dans une variable passée en second argument, par exemple
  `rat_jordan([[1,0,0],[1,2,-1],[0,0,1]],'P')`

On a :
$J = P^{-1}AP$ avec J est à coefficients rationnels.

Remarques

– La syntaxe `Maple` est aussi valable dans les autres modes, par exemple, en mode `Xcas`, on tape :
  `rat_jordan([[4,1,1],[1,4,1],[1,1,4]],'P')`
  On obtient :
  `[[1,-1,1/2],[1,0,-1],[1,1,1/2]]`
  puis `P` renvoie :
  `[[6,0,0],[0,3,0],[0,0,3]]`
  ,
– Les coefficients de P et de J appartiennent au corps des coefficients de A.
  Par exemple, en mode `Xcas`, on tape :
  `rat_jordan([[1,0,1],[0,2,-1],[1,-1,1]])`
  On obtient :

  `[[1,1,2],[0,0,-1],[0,1,2]],[[0,0,-1],[1,0,-3],[0,1,4]]`
  On tape (on met `-pcar(...)` car l'argument de `companion` doit être un polynôme unitaire (cf 6.49.11)
  `companion(-pcar([[1,0,1],[0,2,-1],[1,-1,1]],x),x)`
  On obtient :
  `[[0,0,-1],[1,0,-3],[0,1,4]]`
  On tape :
  `rat_jordan([[1,0,0],[0,1,1],[1,1,-1]])`
  On obtient :

  `[[-1,0,0],[1,1,1],[0,0,1]],[[1,0,0],[0,0,2],[0,1,0]]`

On tape :

`factor(pcar([[1,0,0],[0,1,1],[1,1,-1]],x))`

On obtient :

`-(x-1)*(x^2-2)`

On tape :

`companion((x^2-2),x)`

On obtient :

`[[0,2],[1,0]]`

– Lorsque A est symétrique et a des valeurs propres d'ordre multiple, `Xcas` renvoie des vecteurs propres orthogonaux mais pas forcément normés i.e. `tran(P)*P` est une matrice diagonale de diagonale le carré de la norme des vecteurs propres.

  On tape :

  `rat_jordan([[4,1,1],[1,4,1],[1,1,4]])`

  On obtient :

  `[[1,-1,1/2],[1,0,-1],[1,1,1/2]],[[6,0,0],[0,3,0],[0,0,3]]`

On tape en mode `Xcas`, `Mupad` et `TI` :

`rat_jordan([[1,0,0],[1,2,-1],[0,0,1]])`

On obtient :

`[[0,1,0],[1,0,1],[0,1,1]],[[2,0,0],[0,1,0],[0,0,1]]`

On tape en mode `Xcas`, `Mupad` et `TI` :

`rat_jordan([[4,1,-2],[1,2,-1],[2,1,0]])`

On obtient :

`[[[1,2,1],[0,1,0],[1,2,0]],[[2,1,0],[0,2,1],[0,0,2]]]`

En mode complexe et en mode `Xcas`, `Mupad` et `TI`, on tape :

`rat_jordan([[2,0,0],[0,2,-1],[2,1,2]])`

On obtient :

`[[1,0,0],[-2,-1,-1],[0,-i,i]],[[2,0,0],[0,2-i,0],[0,0,2+i]]`

On tape en mode `Maple` :

`rat_jordan([[1,0,0],[1,2,-1],[0,0,1]],'P')`

On obtient :

`[[2,0,0],[0,1,0],[0,0,1]]`

puis on tape :

`P)`

On obtient :

`[[0,1,0],[1,0,1],[0,1,1]]`

6.49.5 Matrice de passage et matrice de Jordan : `jordan`

`jordan` a comme argument une matrice A d'ordre n.
`jordan` renvoie

- en mode `Xcas` (il faut être en mode complexe c'est à dire avoir coché `Complexe` dans la configuration de CAS lorsqu'il y a des valeurs propres complexes), `Mupad` ou `TI`
  une séquencee composée de deux matrices : une matrice de changement de base P de colonnes les vecteurs propres et caractéristiques de A, suivie de la matrice de Jordan J associée à A dans la nouvelle base,
- en mode `Maple`
  la matrice de Jordan J associée à A dans la nouvelle base. On peut cependant obtenir la matrice de changement de base P, dans une variable passée en second argument, par exemple
  ```
  jordan([[1,0,0],[0,1,1],[1,1,-1]],'P')
  ```

On a :
$J = P^{-1}AP$.
Remarques Pour `rat_jordan` :

- La syntaxe `Maple` est aussi valable dans les autres modes, par exemple, en mode `Xcas`, on tape :
  ```
  rat_jordan([[4,1,1],[1,4,1],[1,1,4]],'P')
  ```
  On obtient :
  ```
  [[1,-1,1/2],[1,0,-1],[1,1,1/2]]
  ```
  puis `P` renvoie :
  ```
  [[6,0,0],[0,3,0],[0,0,3]]
  ```
  , On tape :
  ```
  rat_jordan([[-1,1,0],[0,-1,1],[1,0,-1]])
  ```
  On obtient :
  ```
  [[1,1,-1],[1,0,-1],[1,-1,2]],[[0,0,0],[0,0,-3],[0,1,-3]]
  ```
  On tape (en mode complexe) :
  ```
  jordan([[-1,1,0],[0,-1,1],[1,0,-1]])
  ```
  On obtient :
  ```
  [[1,(-i)*sqrt(3)-1,(i)*sqrt(3)-1],[1,2,2],
  [1,(i)*sqrt(3)-1, (-i)*sqrt(3)-1]], [[0,0,0],
  [0,((i)*sqrt(3)-3)/2,0],[0,0,((-i)*sqrt(3)-3)/2]]
  ```
- Lorsque A est symétrique et a des valeurs propres d'ordre multiple, `Xcas` renvoie des vecteurs propres orthogonaux (pas forcément normés i.e. `tran(P)*P`) et une matrice diagonale ayant pour diagonale le carré de la norme des vecteurs propres. par exemple :
  ```
  rat_jordan([[4,1,1],[1,4,1],[1,1,4]])
  ```
  renvoie :
  ```
  [[1,-1,1/2],[1,0,-1],[1,1,1/2]],[[6,0,0],[0,3,0],[0,0,3]]
  ```

Pour avoir la matrice de Jordan de :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

on tape en mode `Xcas`, `Mupad` et `TI` :

```
jordan([[1,0,0],[0,1,1],[1,1,-1]])
```

On obtient :

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 1 & 1 \\ \sqrt{2}-1 & 0 & -\sqrt{2}-1 \end{bmatrix}, \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -\sqrt{2} \end{bmatrix}$$

On tape en mode `Maple` :

```
jordan([[1,0,0],[0,1,1],[1,1,-1]])
```

On obtient :

```
[[1,0,0],[0,-(sqrt(2)),0],[0,0,sqrt(2)]]
```

On tape en mode `Maple` :

```
jordan([[1,0,0],[0,1,1],[1,1,-1]],'P')
```

On obtient :

```
[[1,0,0],[0,-(sqrt(2)),0],[0,0,sqrt(2)]]
```

puis on tape :

```
P)
```

On obtient :

```
[[-1,0,0],[1,1,1],[0,-sqrt(2)-1,sqrt(2)-1]]
```

On tape en mode `Xcas`, `Mupad` et `TI` :

```
jordan([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
[[[1,2,1],[0,1,0],[1,2,0]],[[2,1,0],[0,2,1],[0,0,2]]]
```

En mode complexe et en mode `Xcas`, `Mupad` et `TI`, on tape :

```
jordan([[2,0,0],[0,2,-1],[2,1,2]])
```

On obtient :

```
[[1,0,0],[-2,-1,-1],[0,i,-i]],[[2,0,0],[0,2+i,0],[0,0,2-i]]
```

6.49.6 Puissance n d'une matrice carrée : `matpow`

`matpow` élève une matrice carrée à la puissance n en la jordanisant On tape :

```
matpow([[1,2],[2,1]],n)
```

On obtient :

```
[[((-1)^n+3^n)/2,(-(-1)^n+3^n)/2],[(-(-1)^n+3^n)/2,((-1)^n+3^n)/2]]
```

On a en effet :
`jordan([[1,2],[2,1]])` renvoie :
`[[1,-1],[1,1]],[[3,0],[0,-1]]`

6.49.7 Polynôme caractéristique : `pcar` `charpoly`

`pcar` (ou `charpoly`) a un (resp deux) argument(s).
`pcar` (ou `charpoly`) a comme argument une matrice A d'ordre n (resp une matrice A d'ordre n et un nom de variable formelle).
`pcar` (ou `charpoly`) renvoie le polynôme caractéristique P de A écrit selon la liste de ses coefficients (resp le polynôme caractéristique P de A écrit sous forme symbolique en utilisant le nom de variable donnée en argument).
Le polynôme caractéristique P de A est défini par

$$P(x) = \det(x.I - A)$$

On tape :

```
pcar([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
[1,-6,12,-8]
```

Donc le polynôme caractéristique de [[4,1,-2],[1,2,-1],[2,1,0]] est

$$x^3 - 6x^2 + 12x - 8$$

On peut aussi avoir la forme symbolique en tapant :
`normal(poly2symb([1,-6,12,-8]))`.
On tape :

```
purge(X):;pcar([[4,1,-2],[1,2,-1],[2,1,0]],X)
```

On obtient :

```
X^3-6*X^2+12*X-8
```

On peut spécifier par un argument optionnel l'algorithme utilisé pour faire ce calcul, parmi :

- `lagrange` : calcul par interpolation de Lagrange, en donnant à x les valeurs comprises entre 0 et la dimension.
  On tape :
  ```
  pcar([[4,1,-2],[1,2,-1],[2,1,0]],lagrange)
  ```
  On obtient :
  ```
  x*((x-1)*(-x+5)-7)+8
  ```
  et après simplification :
  ```
  -x^3+6*x^2-12*x+8
  ```
- `hessenberg` : calcul par réduction sous forme tri-diagonale puis formule de récurrence, efficace sur un corps fini.
  On tape :
  ```
  pcar([[4,1,-2],[1,2,-1],[2,1,0]],hessenberg)
  ```
  On obtient :
  ```
  [1,-6,12,-8]
  ```
- `fadeev` : calcul simultané du polynôme caractéristique et de la comatrice de $xI - A$ On tape :
  ```
  pcar([[4,1,-2],[1,2,-1],[2,1,0]],fadeev)
  ```

On obtient :

`[1,-6,12,-8]`

– `pmin` : calcul du polynôme minimal relatif à un vecteur pris au hasard, c'est le polynôme caractéristique s'il est de degré maximal.
  On tape :
  `pcar([[4,1,-2],[1,2,-1],[2,1,0]],pmin)`
  On obtient :
  `[1,-6,12,-8]`

Pour les matrices à coefficients entiers, l'algorithme utilisé par défaut est modulaire, on calcule le polynôme caractéristique modulo plusieurs nombres premiers, soit par le polynôme minimal, soit par Hessenberg, et on reconstruit par les restes chinois coefficient par coefficient. Le test d'arrêt est probabiliste, lorsque le polynôme reconstruit ne varie plus pour des nombres premiers dont le produit est supérieur à l'inverse de la valeur de `proba_epsilon` (que l'on peut modifier dans la configuration du cas). Si `proba_epsilon` est nul, le résultat est déterministe (une majoration à priori des coefficients est alors utilisée). Dans tous les cas, le temps de calcul est en $O(n4\ln(n))$, mais il est plus rapide avec la méthode probabiliste.

6.49.8 Polynôme caractéristique d'une matrice creuse de grande dimension : `pcar_hessenberg`

`pcar_hessenberg` a comme argument une matrice A d'ordre n.
`pcar_hessenberg` renvoie le polynôme caractéristique P de A calculé selon la méthode de Hessenberg (lorsque les coefficients de A sont dans un corps fini ou sont à représentation finie) et définit par :
$P(x) = (-1)^n.\det(A - x.I)$.
On tape :

`pcar_hessenberg([[4,1,-2],[1,2,-1],[2,1,0]])`

On obtient :

`[1,-6,12,-8]`

Donc le polynôme caractéristique de [[4,1,-2],[1,2,-1],[2,1,0]] est :
$x^3 - 6x^2 + 12x - 8$.

6.49.9 Polynôme minimal : `pmin`

`pmin` a un (resp deux) argument(s).
`pmin` a comme argument une matrice A d'ordre n (resp une matrice A d'ordre n et un nom de variable formelle).
`pmin` renvoie le polynôme minimal de A écrit selon la liste de ses coefficients (resp le polynôme minimal P de A écrit sous forme symbolique en utilisant le nom de variable donnée en argument).
Le polynôme minimal P de A est le polynôme de plus petit degré qui annule A ($P(A) = 0$).
On tape :

`pmin([[1,0],[0,1]])`

On obtient :

```
[1,-1]
```

On tape :

```
pmin([[1,0],[0,1]],x)
```

On obtient :

```
x-1
```

Donc le polynôme minimal de [[1,0],[0,1]] est `x-1`.
On tape :

```
pmin([[2,1,0],[0,2,0],[0,0,2]])
```

On obtient :

```
[1,-4,4]
```

On tape :

```
pmin([[2,1,0],[0,2,0],[0,0,2]],x)
```

On obtient :

```
x^2-4*x+4
```

Donc le polynôme minimal de [[2,1,0],[0,2,0],[0,0,2]] est $x^2 - 4x + 4$.

6.49.10 Comatrice : `adjoint_matrix`

La comatrice de A est une matrice B telle que $A * B = \det(A) * I$.
`adjoint_matrix` a comme argument une matrice carrée A d'ordre n.
`adjoint_matrix` renvoie une liste composée des coefficients du polynôme caractéristique P de A, d'une liste contenant les coefficients matriciels d'un polynôme Q tel que $Q(x)$ soit la comatrice de $A - x * I$ au signe près si n est pair !
On a :
$P(x) = (-1)^n * \det(A - x * I)$ et $P(A) - P(x) * I = (A - x * I) * Q(x)$
En effet le polynôme à coefficients matriciels $P(A) - P(x) * I$ est divisible par $A - x * I$ car il s'annule pour $x = A$:
$P(A) = 0$, on a donc $P(A) - P(x) * I = -P(x) * I = (A - x * I) * Q(x)$.
$Q(x)$ est donc la comatrice de $A - x * I$ au signe près si n est pair, puisque :
$(A - x * I) * Q(x) = -P(x) * I = (-1)^{n+1} * \det(A - x * I)$.
On a alors $Q(x) = I * x^{n-1} + ... + B_0$ où $B_0 =$ est la comatrice de A (au signe près si n est pair !) puisque $Q(0) = B_0$ et $A * Q(0) = (-1)^{n+1} \det(A)$.
On tape :

```
adjoint_matrix([[4,1,-2],[1,2,-1],[2,1,0]])
```

On obtient :

```
[[1,-6,12,-8],[[[1,0,0],[0,1,0],[0,0,1]],[[-2,1,-2],
[1,-4,-1],[2,1,-6]],[[1,-2,3],[-2,4,2],[-3,-2,7]]]]
```

Donc le polynôme caractéristique est :
$x^3 - 6 * x^2 + 12 * x - 8$
Le déterminant de A est égal à $-P(0)$ donc à 8
La comatrice de A est égale à $Q(0)$ donc à :
$[[1, -2, 3], [-2, 4, 2], [-3, -2, 7]]$
L'inverse de A est donc :
$1/8 * [[1, -2, 3], [-2, 4, 2], [-3, -2, 7]]$
La comatrice de $A - x * I$ est donc :
$[[x^2-2x+1, x-2, -2x+3], [x-2, x^2-4x+4, -x+2], [2x-3, x-2, x^2-6x+7]]$
On tape :

```
adjoint_matrix([[4,1],[1,2]])
```

On obtient :

```
[[1,-6,7],[[[1,0],[0,1]],[[-2,1],[1,-4]]]]
```

Donc le polynôme caractéristique est :
$x^2 - 6 * x + 7$
Le déterminant de A est égal à $+P(0)$ donc à 7
La comatrice de A est égale à $Q(0)$ donc à :
$-[[-2, 1], [1, -4]]$.
L'inverse de A est donc :
$-1/7 * [[-2, 1], [1, -4]]$.
La comatrice de $A - x * I$ est donc :
$-[[x - 2, 1], [1, x - 4]]$.

6.49.11 Matrice compagnon d'un polynôme : `companion`

`companion` a comme argument un polynôme P unitaire et le nom de sa variable.
`companion` renvoie la matrice qui a pour polynôme caractéristique le polynôme P.
Si $P(x) = x^n + a_{n-1}x^{n-1} + ... + a - 1x + a_0$, cette matrice est égale à la matrice unité d'ordre $n - 1$ bordée par $[0, 0.., 0, -a_0]$ comme première ligne, et par $[-a_0, -a_1, ..., -a_{n-1}]$ comme dernière colonne.
On tape :

```
companion(x^2+5x-7,x)
```

On obtient :

```
[[0,7],[1,-5]]
```

On tape :

```
companion(x^4+3x^3+2x^2+4x-1,x)
```

On obtient :

```
[[0,0,0,1],[1,0,0,-4],[0,1,0,-2],[0,0,1,-3]]
```

6.49.12 Réduction de Hessenberg d'une matrice : `hessenberg`

`hessenberg` a comme premier argument une matrice A et comme deuxième argument 0, -1 ou -2 ou $n > 1$ et n premier (par défaut 0).
`hessenberg` renvoie la matrice P de passage et la matrice B semblable à A dont les coefficients sous-sous-diagonaux sont nuls. On dit que B est une matrice de Hessenberg et on a $B = P^{-1}AP$ ou $B \sim P^{-1}AP$ selon le
deuxième argument.

– Avec un seul argument ou comme deuxième argument 0, les calculs sont exacts.
  On tape :

```
hessenberg([[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
            [2,-1,-1,3,1],[2,-1,1,1,2]])
```

  ou

```
hessenberg([[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
           [2,-1,-1,3,1],[2,-1,1,1,2]],0)
```

  On obtient :

$$\left[\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 3 & 8 & 5 & 5/2 & 2 \\ 2 & 1 & 1/2 & (-5)/4 & -1 \\ 0 & 2 & 1 & 2 & 0 \\ 0 & 0 & 2 & 3/2 & 2 \\ 0 & 0 & 0 & 13/8 & 7/2 \end{bmatrix}\right]$$

  On a en effet si :
  `A:=[[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],`
  `[2,-1,-1,3,1],[2,-1,1,1,2]]`
  `P,B:= hessenberg(A,0)`
  `pcar(A)=pcar(B)=[1,-10,13,71,-50,-113]`
  `inv(P)*A*P` est égal à `B`

– Avec comme deuxième argument -1, les calculs sont approchés et la matrice B est triangulaire.
  On tape :

```
hessenberg([[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
          [2,-1,-1,3,1],[2,-1,1,1,2]],-1)
```

  On obtient (on donne le résultat avec 2 digits) :

$$\left[\begin{bmatrix} 0.73 & -0.057 & -0.42 & -0.17 & -0.51 \\ 0.25 & -0.53 & 0.72 & -0.38 & -0.048 \\ 0.35 & -0.44 & -0.3 & 0.19 & 0.74 \\ 0.34 & 0.68 & 0.17 & -0.46 & 0.43 \\ 0.41 & 0.25 & 0.44 & 0.76 & -0.063 \end{bmatrix},\right.$$

$$\left.\begin{bmatrix} 6.7 & 8.7e-15 & -2e-13 & 2.7e-14 & -1.4e-13 \\ 0.0 & 4.6 & 0 & 0 & 0 \\ 0.0 & 0.0 & -1.9 & 0 & 0 \\ 0.0 & 0.0 & 0.0 & 1.7 & 0 \\ 0.0 & 0.0 & 0.0 & -0.0 & -1.2 \end{bmatrix}\right]$$

On a en effet si :
`A:=[[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],`
`[2,-1,-1,3,1],[2,-1,1,1,2]]`
`P,B:= hessenberg(A,-1)`
`inv(P)*A*P` est égal à `B` en calcul numérique.
Remarque
`hessenberg(A,-1)` est identique à `SCHUR(A)` qui est une commande `Xcas` compatible avec les calculatrices HP.

– Avec comme deuxième argument -2, les calculs sont approchés et la matrice P est orthogonale et la matrice B a ses coefficients sous-sous-diagonaux nuls.
On tape :

```
hessenberg([[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
          [2,-1,-1,3,1],[2,-1,1,1,2]],-2)
```

On obtient (on donne le résultat avec 2 digits) :

$$\left[\left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & -0.75 & 0 & -0.43 \\ 0 & 0.5 & 0.45 & -0.71 & -0.21 \\ 0 & 0.5 & -0.15 & 0 & 0.85 \\ 0 & 0.5 & 0.45 & 0.71 & -0.21 \end{array}\right],\right.$$

$$\left.\left[\begin{array}{ccccc} 3.0 & 4 & 0.0 & 1.6e-14 & 5.4e-14 \\ 4 & 2.2 & 0.83 & 0.0 & 0.0 \\ 0 & 0.83 & 0.75 & 1.7 & -1.4e-14 \\ 0 & 0 & 1.7 & 0.5 & 2 \\ 0 & 0 & 0 & 2 & 3.5 \end{array}\right]\right]$$

On a en effet si :
`A:=[[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],`
`[2,-1,-1,3,1],[2,-1,1,1,2]]`
`P,B:= hessenberg(A,-2)`
`inv(P)*A*P` est égal à `B` et `inv(P)` est égal à `tr(P)` en calcul numérique.

– Avec comme deuxième argument $n > 1$ et n premier, les calculs sont modulo n et la matrice B est triangulaire.
On tape :

```
hessenberg([[3,2,2,2,2],[2,1,2,-1,-1],[2,2,1,-1,1],
          [2,-1,-1,3,1],[2,-1,1,1,2]],13)
```

On obtient

$$\left[\left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -12 & 1 & 0 & 0 \\ 0 & 66 & -6 & 1 & 0 \\ 0 & 131 & -12 & 4 & 1 \end{array}\right], \left[\begin{array}{ccccc} 3 & -5 & -8 & 10 & 2 \\ 2 & 1 & 7 & -5 & -1 \\ 0 & 2 & 1 & -5 & -11 \\ 0 & 0 & 7 & -5 & -12 \\ 0 & 0 & 0 & 0 & 10 \end{array}\right]\right]$$

6.49.13 Forme normale de Hermite : `ihermite`

`ihermite` a comme argument une matrice à coefficient dans $\mathbb{Z}$.
`ihermite` renvoie les matrices `U` et `B` tels que `U` est inversible dans $\mathbb{Z}$, `B` est triangulaire supérieure et vérifie : `B=U*A`.
Pour faire cela on effectue la réduction sous forme échelonnée (de type Gauss) d'une matrice d'entiers en utilisant uniquement des opérations de lignes inversibles dans les entiers, en d'autres termes si `A` est la matrice originale, on calcule une matrice `U` inversible dans $\mathbb{Z}$ et une matrice `B` triangulaire superieure telles que `B = U*A`.
De plus les coefficients au-dessus de la diagonale de `A` sont en module inferieurs au pivot de la colonne divisé par 2. Puisque `det(U) = 1`, on passe aussi de `B` a `A` uniquement avec des manipulations de ligne a coefficients entiers.
On tape :

```
A:=[[9,-36,30],[-36,192,-180],[30,-180,180]];
U,B:=ihermite(A)
```

On obtient :

```
[[9,-36,30],[-36,192,-180],[30,-180,180]],
[[13,9,7],[6,4,3],[20,15,12]],[[3,0,30],[0,12,0],[0,0,60]]
```

Application : Calcul d'une Z-base d'un noyau
Soit `M` la matrice dont on cherche le noyau.
Soit `U,A:=ihermite(transpose(M))`.
On a `A=U*transpose(M)` donc `transpose(A)=M*transpose(U)`.
Les colonnes nulles de `transpose(A)` correspondent donc aux colonnes de `transpose(U)` qui sont dans `Ker(M)`. Ainsi, les lignes nulles de `A` correspondent aux lignes de `U` qui sont dans `Ker(M)`.
Exemple
Si `M:=[[1,4,7],[2,5,8],[3,6,9]]`
`U,A:=ihermite(tran(M))` renvoie :
`U:=[[-3,1,0],[4,-1,0],[-1,2,-1]]` et
`A:=[[1,-1,-3],[0,3,6],[0,0,0]]`
`A[2]=[0,0,0]` donc une base de `Ker(M)` est composée de `U[2]=[-1,2,-1]`.
On vérifie que l'a bien `M*U[2]=[0,0,0]`.

6.49.14 Forme normale de Smith : `ismith`

`ismith` a comme argument une matrice à coefficient dans $\mathbb{Z}$.
`ismith` renvoie les matrices `U,B` et `V` tels que `U` et `V` sont inversibles dans $\mathbb{Z}$, `B` est diagonale avec `B[i,i]` divise `B[i+1,i+1]` et on a `B=U*A*V`.
Les `B[i,i]` s'appellent facteurs invariants et permettent entre autre de trouver la structure des groupes abéliens de type fini.
On tape :

```
A:=[[9,-36,30],[-36,192,-180],[30,-180,180]];
U,B,V:=ismith(A)
```

On obtient :

```
[[-3,0,1],[6,4,3],[20,15,12]],
[[3,0,0],[0,12,0],[0,0,60]],
[[1,24,-30],[0,1,0],[0,0,1]]
```

Les facteurs invariants sont 3, 12 et 60.

6.50 Les isométries

6.50.1 Reconnaitre une isométrie : `isom`

`isom` a comme argument la matrice d'une application linéaire en dimension 2 ou 3.
`isom` renvoie :
- si l'application linéaire est une isométrie directe,
  la liste des éléments caractéristiques et `+1`
- si l'application linéaire est une isométrie indirecte,
  la liste des éléments caractéristiques et `-1`
- si l'application linéaire n'est pas une isométrie,
  `[0]`.

On tape :

```
isom([[0,0,1],[0,1,0],[1,0,0]])
```

On obtient :

```
[[1,0,-1],-1]
```

ce qui veut dire que cette isométrie est une symétrie par rapport au plan $x - z = 0$.
On tape :

```
isom(sqrt(2)/2*[[1,-1],[1,1]])
```

On obtient :

```
[pi/4,1]
```

cette isométrie est donc une rotation plane d'angle $\frac{\pi}{4}$.
On tape :

```
isom([[0,0,1],[0,1,0],[0,0,1]])
```

On obtient :

```
[0]
```

ce qui veut dire que ce n'est pas une isométrie.

6.50.2 Trouver la matrice d'une isométrie : `mkisom`

`mkisom` a comme argument :
- En dimension 3, la liste des éléments caractéristiques (vecteur directeur de l'axe et angle de la rotation ou vecteur de la normale au plan de symétrie) et `+1` ou `-1` (`+1` pour les isométries directes et `-1` pour les indirectes).
- En dimension 2, l'élément caractéristique (un angle ou un vecteur) et `+1` ou `-1` (`+1` pour les isométries directes et `-1` pour les indirectes).

`mkisom` renvoie la matrice de l'isométrie définie par les arguments.
On tape :

```
mkisom([[-1,2,-1],pi],1)
```

On obtient la matrice d'une rotation d'axe $[-1, 2, -1]$ et d'angle π :

```
[[-2/3,-2/3,1/3],[-2/3,1/3,-2/3],[1/3,-2/3,-2/3]]
```

On tape :

```
mkisom([pi],-1)
```

On obtient la matrice d'une symétrie par rapport à O :

```
[[-1,0,0],[0,-1,0],[0,0,-1]]
```

On tape :

```
mkisom([1,1,1],-1)
```

On obtient la matrice d'une symétrie par rapport au plan $x + y + z = 0$:

```
[[1/3,-2/3,-2/3],[-2/3,1/3,-2/3],[-2/3,-2/3,1/3]]
```

On tape :

```
mkisom([[1,1,1],pi/3],-1)
```

On obtient la matrice produit d'une rotation d'axe $[1, 1, 1]$ et d'angle $\frac{\pi}{3}$ et d'une symétrie par rapport au plan $x + y + z = 0$:

```
[[0,-1,0],[0,0,-1],[-1,0,0]]
```

On tape :

```
mkisom(pi/2,1)
```

On obtient la matrice en dimension 2, de la rotation plane d'angle $\frac{\pi}{2}$:

```
[[0,-1],[1,0]]
```

On tape :

```
mkisom([1,2],-1)
```

On obtient la matrice en dimension 2, de la symétrie plane par rapport à la droite d'équation $x + 2y = 0$:

```
[[3/5,-4/5],[-4/5,-3/5]]
```

6.51 Factorisation des matrices

La factorisation des matrices renvoie en général des matrices numériques et quelquefois des matrices symboliques.

6.51.1 Décomposition de Cholesky : `cholesky`

`cholesky` a pour argument une matrice carrée symétrique `M` definie positive.
`cholesky` renvoie une matrice symbolique (ou numérique) `P` triangulaire inférieure telle que :
`tran(P)*P=M`
On tape :

```
cholesky([[1,1],[1,5]])
```

On obtient :

```
[[1,0],[1,2]]
```

On tape :

```
cholesky([[3,1],[1,4]])
```

On obtient :

```
[[sqrt(3),0],[(sqrt(3))/3,(sqrt(33))/3]]
```

On tape :

```
cholesky([[1,1],[1,4]])
```

On obtient :

```
[[1,0],[1,sqrt(3)]]
```

Attention Si la matrice argument A n'est pas une matrice symétrique, `cholesky` ne renvoie pas d'erreur, mais `cholesky` utilise la matrice symétrique B associée à la forme quadratique q provenant de la forme bilinéaire associée à A.
On tape :

```
cholesky([[1,-1],[-1,4]])
```

ou :

```
cholesky([[1,-3],[1,4]])
```

On obtient :

```
[[1,0],[-1,sqrt(3)]]
```

6.51.2 Décomposition QR : `qr`

`qr` a comme argument une matrice carrée numérique A d'ordre n.
`qr` factorise numériquement cette matrice sous la forme $Q*R$ où Q est une matrice orthogonale (${}^tQ*Q=I$) et R est une matrice triangulaire supérieure.
`qr(A)` renvoie seulement `R` et on a `Q=A*inv(R)`.
On tape :

```
qr([[3,5],[4,5]])
```

On obtient la matrice `R` :

```
[[-5,-7],[0,-1]]
```

On tape :

```
qr([[1,2],[3,4]])
```

On obtient la matrice `R` :

```
[[-3.16227766017,-4.42718872424],[0,-0.632455532034]]
```

6.51.3 Décomposition QR (compatible TI) : `QR`

`QR` a comme argument une matrice carrée numérique A d'ordre n et deux noms de variables : `var1` et `var2`.
`QR` factorise numériquement cette matrice sous la forme $Q*R$ où Q est une matrice orthogonale (${}^tQ*Q=I$) et R est une matrice triangulaire supérieure.
`QR(A)` renvoie `R`, met `Q` dans `var1` et met `R` dans `var2`.
On a `Q=A*inv(R)`.
On tape :

```
QR([[3,5],[4,5]],Q,R)
```

On obtient la matrice `R` :

```
[[-5,-7],[0,-1]]
```

6.51.4 Décomposition LQ (compatible HP) : `LQ`

`LQ` a comme argument une matrice rectangulaire numérique A d'ordre $m\times n$.
`LQ(A)` renvoie `L, Q, P` tel que `P*A=L*Q` avec : L une matrice triangulaire inférieure d'ordre $m\times n$, Q une matrice orthogonale d'ordre $n\times n$ (${}^tQ*Q=I$) et P une matrice de permutation d'ordre $n\times n$.
On tape :

```
L,Q,P:=LQ([[4,0,0],[8,-4,3]])
```

On obtient les 3 matrices `L, Q, P` :

```
[[4.0,0,0],[8.0,5.0,0]],[[1,0,0],[0,-0.8,0.6],[0,-0.6,-0.8]],[[1,0],[0,1]]
```

On tape : `L*Q` et on obtient `P*A`.
On tape :

```
L,Q,P:=LQ([[24,18],[30,24]])
```

On obtient :

```
[[30.0,0],[38.4,1.2]],[[0.8,0.6],[-0.6,0.8]],[[1,0],[0,1]]
```

On tape : `L*Q` et on obtient `P*A`.

6.51.5 Décomposition LU : `lu`

`lu` a comme argument une matrice carrée A d'ordre n (numérique ou symbolique).
`lu(A)` renvoie une permutation p de $0..n-1$, une matrice triangulaire inférieure L avec des 1 sur sa diagonale et une matrice triangulaire supérieure U.
Ces matrices sont telles que :

- $P * A = L * U$, où P est la matrice de permutation associée à p (que l'on peut calculer avec `P:=permu2mat(p)`),
- l'équation $A * x = B$ équivaut à :

$$L{*}U{*}x = P{*}B = p(B) \text{ where } p(B) = [b_{p(0)}, b_{p(1)}..b_{p(n-1)}], \quad B = [b_0, b_1..b_{n-1}]$$

On peut aussi définir à partir de p la matrice de permutation P_n par :
$P_n[i, p(i)] = 1$ et
$P_n[i, j] = 0$ si $j \neq p(i)$.
C'est la matrice obtenue en permutant, selon la permutation p, les lignes de la matrice unité.
On peut utiliser la fonction `permu2mat` : `permu2mat(p)` renvoie la matrice `P` d'ordre n.
On tape :

```
(p,L,U):=lu([[3,5],[4,5]])
```

On obtient :

```
[1,0],[[1,0],[0.75,1]],[[4,5],[0,1.25]]
```

On a, en effet, $n = 2$ donc :

$$P[0, p(0)] = P_2[0, 1] = 1, \quad P[1, p(1)] = P_2[1, 0] = 1, \quad P = [[0, 1], [1, 0]]$$

Vérification :
On tape :

```
permu2mat(p)*A; L*U
```

On obtient :

```
[[4.0,5.0],[3.0,5.0]],[[4.0,5.0],[3.0,5.0]]
```

Il faut noter que la permutation est différente lorsque les données sont exactes (le choix du pivot est plus simple). On tape :

```
lu([[1,2],[3,4]])
```

On obtient :

```
[1,0],[[1,0],[3,1]],[[1,2],[0,-2]]
```

On tape :

```
lu([[1.0,2],[3,4]])
```

On obtient :

```
[1,0],[[1,0],[0.333333333333,1]],[[3,4],
[0,0.666666666667]]
```

6.51.6 Décomposition LU (compatible TI) : `LU`

`LU` a comme argument une matrice carrée numérique d'ordre n et trois noms de variables : `var1`, `var2` et `var3`.
`LU(A,var1,var2,var3)` renvoie P une matrice de permutation et met :

- dans `var1`, L une matrice triangulaire inférieure avec des 1 sur sa diagonale,
- dans `var2`, U une matrice triangulaire supérieure,
- dans `var3`, la matrice de permutation P renvoyée.

Ces matrices sont telles que :
l'équation $A * x = B$ équivaut à $L * U * x = P * B$.
On tape :

```
LU([[3,5],[4,5]],L,U,P)
```

On obtient :

```
[[0,1],[1,0]]
```

On tape :

```
L
```

On obtient :

```
[[1,0],[0.75,1]]
```

On tape :

```
U
```

On obtient :

```
[[4,5],[0,1.25]]
```

On tape :

```
P
```

On obtient :

```
[[0,1],[1,0]]
```

6.51.7 Valeurs singulières (compatible HP) : `SVL svl`

`SVL` ou `svl` a comme argument une matrice carrée numérique d'ordre n.
`SVL(A)` renvoie les valeurs singulières de A.
Les valeurs singulières de A sont les racines carrées positives des valeurs propres de $A * trn(A)$. Ainsi lorsque A est symétrique, les valeurs singulières de A sont les valeurs absolues de ses valeurs propres.
On tape :

```
SVL([[1,2],[3,4]])
```

ou

```
svl([[1,2],[3,4]])
```

On obtient :

```
[0.365966190626,5.46498570422]
```

On tape :

```
evalf(sqrt(eigenvals([[1,2],[3,4]]*[[1,3],[2,4]])))
```

On obtient :

```
[5.46498570422,0.365966190626]
```

On tape :

```
SVL([[1,4],[4,1]])
```

ou

```
svl([[1,4],[4,1]])
```

On obtient :

```
[5.0,3.0]
```

On tape :

```
abs(eigenvals([[1,4],[4,1]]))
```

On obtient :

```
[5,3]
```

6.51.8 Singular value decomposition : `svd`

`svd` (singular value decomposition) a comme argument une matrice carrée numérique réelle d'ordre n.
`svd` utilise la librairie lapack
`svd(A)` renvoie U une matrice orthogonale, s la diagonale d'une matrice diagonale S constituée par les valeurs singulières de A et Q une matrice orthogonale (${}^tQ * Q = I$) tel que :
$A = U.\text{diag}(s).\text{tran}(Q)$.
On tape :

```
svd([[1,2],[3,4]])
```

On obtient :

```
[[-0.404553584834,-0.914514295677],[-0.914514295677,
0.404553584834]], [5.46498570422,0.365966190626],
[[-0.576048436766,0.81741556047],[-0.81741556047,
-0.576048436766]]
```

On tape :

```
(U,s,Q):=svd([[3,5],[4,5]])
```

On obtient :

```
[[-0.672988041811,-0.739653361771],[-0.739653361771,
0.672988041811]],[8.6409011028,0.578643354497],
[[-0.576048436766,0.81741556047],[-0.81741556047,
-0.576048436766]]
```

Vérifions :
On tape :

```
U*diag(s)*tran(Q)
```

On obtient :

```
[[3.0,5.0],[4.0,5.0]]
```

6.51.9 Singular value decomposition (compatible HP) : SVD

SVD (singular value decomposition) a comme argument une matrice carrée numérique réelle d'ordre n.
SVD(A) renvoie U une matrice orthogonale, s la diagonale d'une matrice diagonale S =diag(s) constituée par les valeurs singulières de A et Q une matrice orthogonale, ,($^tQ * Q = I$) tel que :
$A = U$.diag(s).tran(Q).
On tape :

```
SVD([[1,2],[3,4]])
```

On obtient :

```
[[[0.914514295677,0.404553584834],[-0.404553584834,0.914514295677]],[0.36596
```

On tape :

```
(U,s,Q):=SVD([[3,5],[4,5]])
```

On obtient :

```
[[[0.739653361771,0.672988041811],[-0.672988041811,0.739653361771]],[0.57864
```

Vérifions :
On tape :

```
U*diag(s)*tran(Q)
```

On obtient :

```
[[3.0,5.0],[4.0,5.0]]
```

6.51.10 Recherche d'une base de vecteurs courts d'un réseau : `lll`

`lll` a comme argument une matrice inversible M à coefficients entiers.
`lll` renvoie (S, A, L, O) :

- S a comme lignes une base courte du reseau engendré par les lignes de M,
- A est la matrice de passage de la base courte à la base définie par les lignes de M ($A * M = S$),
- L est une matrice triangulaire inférieure dont les coefficients sont de module plus petits que 1/2,
- O est une matrice dont les lignes sont orthogonales et on a $L * O = S$.

Si en dimension 2, un vecteur du réseau a comme coordonnées $[a, b]$ dans la base définie par M et a comme coordonnées $[a1, b1]$ dans la base courte définie par S c'est à dire si $[a, b] * M = [a1, b1] * S$, alors :
$[a, b] = [a1, b1] * A$
$[a1, b1] * S = [a1, b1] * A * M = [a, b] * M$ et
$[a, b] * M = [a, b] * A^{-1} * S = [a1, b1] * S$
On tape :

```
(S,A,L,O):=lll([[2,1],[1,2]])
```

On obtient :

```
[[-1,1],[2,1]],[[-1,1],[1,0]],[[1,0],[1/-2,1]],[[-1,1],[3/2,3/2]]
```

Donc :
`S=[[-1,1],[2,1]]`
`A=[[-1,1],[1,0]]`
`L=[[1,0],[1/-2,1]]`
`O=[[-1,1],[3/2,3/2]]`

On a comme ancienne base : `v1=[2,1], v2=[1,2]` et
comme base courte : `w1=[-1,1], w2=[2,1].`
Puisque `w1=-v1+v2` et `w2=v1` on a
`A:=[[-1,1],[1,0]]`, `A*M==S` et `L*O==S`.
On tape :

```
(S,A,L,O):=lll([[3,2,1],[1,2,3],[2,3,1]])
```

On obtient :

```
S=[[-1,1,0],[-1,-1,2],[3,2,1]]

A= [[-1,0,1],[0,1,-1],[1,0,0]]

L= [[1,0,0],[0,1,0],[(-1)/2,(-1)/2,1]]

O= [[-1,1,0],[-1,-1,2],[2,2,2]]
```

si on tape :
`M:=[[3,2,1],[1,2,3],[2,3,1]]`
On a :
`A*M==S` et `L*O==S`

6.52 Les formes quadratiques

6.52.1 Matrice d'une forme quadratique : `q2a`

`q2a` a deux arguments : une forme quadratique q et le vecteur de composantes les variables utilisées.
`q2a` renvoie la matrice A associée à q.
On tape :

```
q2a(2*x*y,[x,y])
```

On obtient :

```
[[0,1],[1,0]]
```

6.52.2 Transformer une matrice en une forme quadratique : `a2q`

`a2q` a deux arguments : une matrice symétrique A représentant une forme quadratique q et le vecteur de composantes les variables utilisées.
`a2q` renvoie la forme quadratique q.
On tape :

```
a2q([[0,1],[1,0]],[x,y])
```

On obtient :

```
2*x*y
```

On tape :

```
a2q([[1,2],[2,4]],[x,y])
```

On obtient :

```
x^2+4*x*y+4*y^2
```

6.52.3 Méthode de Gauss : `gauss`

`gauss` a deux arguments : une forme quadratique q et le vecteur de composantes les variables utilisées.
`gauss` renvoie l'écriture de q sous forme d'une somme et différence de carrés.
On tape :

```
gauss(2*x*y,[x,y])
```

On obtient :

```
(y+x)^2/2+(-(y-x)^2)/2
```

6.52.4 Algorithme du gradient conjugué : `conjugate_gradient`

`conjugate_gradient(A,y,x0,eps)` met en œuvre l'algorithme du gradient conjugué pour résoudre $A * x = y$ à eps près, lorsque A est une matrice symétrique définie positive et x_0 une solution initiale approchée optionnelle.
On tape :

```
conjugate_gradient([[2,1],[1,5]],[1,0])
```

On obtient :

```
[5/9,-1/9]
```

On tape :

```
conjugate_gradient([[2,1],[1,5]],[1,0],[0.55,-0.11],1e-2)
```

On obtient :

```
[0.555,-0.11]
```

On tape :

```
conjugate_gradient([[2,1],[1,5]],[1,0],[0.55,-0.11],1e-10)
```

On obtient :

```
[0.555555555556,-0.111111111111]
```

6.52.5 Procédé de Gramschmidt : `gramschmidt`

`gramschmidt` a un ou deux paramètres :

- une matrice vue comme une liste de vecteurs lignes, le produit scalaireètant le produit scalaire canonique, ou
- un vecteur contenant la base d'un espace vectoriel et une fonction qui définit un produit scalaire.

`gramschmidt` donne une base orthonormale par rapport à ce produit scalaire.
On tape :

```
normal(gramschmidt([[1,1,1],[0,0,1],[0,1,0]]))
```

Ou on tape :

```
normal(gramschmidt([[1,1,1],[0,0,1],[0,1,0]],dot))
```

On obtient :

```
[[(sqrt(3))/3,(sqrt(3))/3,(sqrt(3))/3],
[(-(sqrt(6)))/6,(-(sqrt(6)))/6,(sqrt(6))/3],
[(-(sqrt(2)))/2,(sqrt(2))/2,0]]
```

Exemple
Pour les polynômes de degré $<n$, on considère le produit scalaire défini par :

$$P.Q = \int_{-1}^{1} P(x).Q(x)dx$$

On tape :

```
gramschmidt([1,1+x],(p,q)->integrate(p*q,x,-1,1))
```

Ou on écrit la fonction `p_scal`, on tape :
`p_scal(p,q):=integrate(p*q,x,-1,1)`
et on tape :

```
gramschmidt([1,1+x],p_scal)
```

On obtient :

```
[1/(sqrt(2)),(1+x-1)/sqrt(2/3)]
```

6.52.6 Tracé d'une conique : `conic conique`

`conique` a comme argument l'expression d'une conique.
`conique` trace la conique ayant pour équation l'argument égalé à zéro.
On tape :

```
conique(2*x^2+2*x*y+2*y^2+6*x)
```

On obtient :

```
le tracé de l'ellipse de centre -2+i et d'équation
           2*x^2+2*x*y+2*y^2+6*x=0
```

Remarque :
Utiliser `conique_reduite` pour avoir l'équation paramétrique de la conique.

6.52.7 Réduction d'une conique : `conique_reduite reduced_conic`

`conique_reduite` a un ou deux arguments : l'expression d'une conique et le vecteur de composantes les variables utilisées si il est différent de `[x,y]`.
`conique_reduite` renvoie une liste d'éléments :
- l'origine de la conique,
- la matrice d'un repère dans lequel la conique est réduite,
- 0 ou 1 pour savoir si la conique est dégénérée ou pas,
- l'équation réduite de la conique dans ce repère,
- un vecteur contenant son équation paramétrique ou ses équations paramétriques lorsque la conique est composée de plusieurs nappes.

On tape :

```
conique_reduite(2*x^2+2*x*y+2*y^2+5*x+3,[x,y])
```

On obtient :

```
[[-5/3,5/6],[[-1/(sqrt(2)),1/(sqrt(2))],[-1/(sqrt(2)),
-1/(sqrt(2))]],1,3*x^2+y^2+-7/6,[[(-10+5*i)/6+(1/(sqrt(2))+
(i)/(sqrt(2)))*((sqrt(14)*cos(` t`))/6+
((i)*sqrt(42)*sin(t))/6),t,0,2*pi,(2*pi)/60]]]
```

La conique n'est pas dégénérée et a pour équation réduite :

$$3x^2 + y^2 - 7/6 = 0$$

dans le repère d'origine $-5/3 + 5 * i/6$ et d'axes parallèles aux vecteurs $(-1, 1)$ et $(-1, -1)$.
Son équation paramétrique est :

$$\frac{-10 + 5 * i}{6} + \frac{(1 + i)}{\sqrt{2}} * \frac{(\sqrt{14} * cos(t) + i * \sqrt{42} * sin(t))}{6}$$

et pour le dessin, le paramètre t varie de 0 à 2π avec un pas `tstep`=$2\pi/60$.

Remarque :
Lorsque la conique est dégénérée en 1 ou 2 droite(s), chaque droite n'est pas donné par son équation paramétrique mais par la liste constituée par un vecteur normal à la droite et un point de la droite.
On tape :

```
conique_reduite(x^2-y^2+3*x+y+2)
```

On obtient :

```
[[(-3)/2,1/2],[[1,0],[0,1]],0,x^2-y^2,
[[(-1+2*i)/(1-i),(1+2*i)/(1-i)],
[(-1+2*i)/(1-i),(-1)/(1-i)]]]
```

On obtient :
`(2*sqrt(5*23297^2*126757^*21302293^2))/62906903119301897`
Soit :
`2*sqrt(5)`
On tape :
```
H1:=projection(D1,M)
longueur(M,F1)/longueur(M,H1)
```
On obtient :
```
(2^14*3*13*17*89*311*521*563*769*2609*
sqrt(2*3*49409^2*112249^2*126757^2*
21302293^2*568000439^2*6789838247809^2))/
(2^14*3^2*13*17*89*311*521*563*769*
2609*49409*112249*126757*21302293*568000439*6789838247809)
```
Soit :
`(sqrt(6))/3`

6.52.8 Tracé d'une quadrique : quadrique

quadrique a comme arguments l'expression d'une quadrique.
quadrique trace la quadrique ayant pour équation l'argument égalé à zéro.
On tape :

```
quadrique(7*x^2+4*y^2+4*z^2+4*x*y-
    4*x*z-2*y*z-4*x+5*y+4*z-18)
```

On obtient :

```
le tracé de l'ellipsoïde d'équation
7*x^2+4*y^2+4*z^2+4*x*y-4*x*z-2*y*z-4*x+5*y+4*z-18=0
```

Remarque :
Utiliser quadrique_reduite pour avoir l'équation paramétrique de la quadrique.

6.52.9 Réduction d'une quadrique : quadrique_reduite reduced_quadric

quadrique_reduite a un ou deux arguments : l'expression d'une quadrique et le vecteur de composantes les variables utilisées si il est différent de [x,y,z].
quadrique_reduite renvoie une liste d'éléments :
- l'origine,
- la matrice d'un repère dans lequel la quadrique est réduite,
- 0 ou 1 (0 si la quadrique est dégénérée),
- l'équation réduite de la quadrique dans ce repère,
- un vecteur contenant son équation paramétrique ou ses équations paramétriques si la quadrique est composée de plusieurs nappes.

Attention ! les équations paramétriques sont écrites à l'aide des variables u,v : ces variables doivent donc être libres (purge u,v avant d'appeler quadrique_reduite).
On tape :

```
quadrique_reduite(7*x^2+4*y^2+4*z^2+
  4*x*y-4*x*z-2*y*z-4*x+5*y+4*z-18)
```

On obtient une liste contenant :

- L'origine du repère (centre de symétrie de la quadrique) :
  `[11/27,(-26)/27,(-29)/54],`
- La matrice de passage :
  ```
  [[(sqrt(6))/3,(sqrt(5))/5,(-(sqrt(30)))/15],
  [(sqrt(6))/6,0,(sqrt(30))/6],
  [(-(sqrt(6)))/6,(2*sqrt(5))/5,(sqrt(30))/30]],
  ```
- 1 donc la quadrique n'est pas dégénérée
- l'équation réduite de la quadrique dans ce repère :
  `0,9*x^2+3*y^2+3*z^2+(-602)/27,`
- l'équation paramétrique de la quadrique (dans le premier repère), de paramètres [u,v] :

```
[[(sqrt(6)*sqrt(602/243)*sin(u)*cos(v))/3+
(sqrt(5)*sqrt(602/81)*sin(u)*sin(v))/5+
((-(sqrt(30)))*sqrt(602/81)*cos(u))/15+11/27,
(sqrt(6)*sqrt(602/243)*sin(u)*cos(v))/6+
(sqrt(30)*sqrt(602/81)*cos(u))/6+(-26)/27,
((-(sqrt(6)))*sqrt(602/243)*sin(u)*cos(v))/6+
(2*sqrt(5)*sqrt(602/81)*sin(u)*sin(v))/5+
(sqrt(30)*sqrt(602/81)*cos(u))/30+(-29)/54], u=(0
.. pi),v=(0 .. (2*pi)),ustep=(pi/20),
vstep=((2*pi)/20)]]
```

Donc la quadrique est un ellipsoïde eta pour équation :

$$9 * x^2 + 3 * y^2 + 3 * z^2 + (-602)/27$$

dans le repère d'origine $[11/27, (-26)/27, (-29)/54]$
de matrice de passage `P` est :

$$\left[\begin{array}{ccc} \frac{\sqrt{6}}{3} & \frac{\sqrt{5}}{5} & -\frac{\sqrt{30}}{15} \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} \\ -\frac{\sqrt{6}}{6} & \frac{2\sqrt{5}}{5} & \frac{\sqrt{30}}{30} \end{array}\right]$$

Son équation paramétrique est :

$$\left\{\begin{array}{l} x = \frac{\sqrt{6}\sqrt{\frac{602}{243}}\sin(u)\cos(v)}{3} + \frac{\sqrt{5}\sqrt{\frac{602}{81}}\sin(u)\sin(v)}{5} - \frac{\sqrt{30}\sqrt{\frac{602}{81}}\cos(u)}{15} + \frac{11}{27} \\ y = \frac{\sqrt{6}\sqrt{\frac{602}{243}}\sin(u)\cos(v)}{6} + \frac{\sqrt{30}\sqrt{\frac{602}{81}}\cos(u))}{6} - \frac{26}{27} \\ z = \frac{-\sqrt{6}\sqrt{\frac{602}{243}} * \sin(u)\cos(v)}{6} + \frac{2\sqrt{5}\sqrt{\frac{602}{81}}\sin(u)\sin(v)}{5} + \frac{\sqrt{30}\sqrt{\frac{602}{81}}\cos(u)}{30} - \frac{29}{54} \end{array}\right.$$

Remarque :
Lorsque la quadrique est dégénérée en 1 ou 2 plan(s), chaque plan n'est pas donné par son équation paramétrique mais par la liste constituée par un vecteur normal au plan et un point du plan.
On tape :

```
quadrique_reduite(x^2-y^2+3*x+y+2)
```

On obtient :

```
[[(-3)/2,1/2,0],[[1,0,0],[0,1,0],[0,0,-1]],0,x^2-y^2,
[hyperplan([1,1,0],[(-3)/2,1/2,0]),
hyperplan([1,-1,0],[(-3)/2,1/2,0])]]
```

6.53 Les expressions de plusieurs variables

6.53.1 Le gradient : `derive deriver diff grad`

`derive` (ou `diff` ou `grad`) a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
`derive` renvoie le gradient de F ($\overrightarrow{Grad}(F) = [\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}]$ si $n = 3$).
Exemple
Déterminer le gradient de $F(x, y, z) = 2x^2y - xz^3$.
On tape :

```
derive(2*x^2*y-x*z^3,[x,y,z])
```

Ou on tape :

```
diff(2*x^2*y-x*z^3,[x,y,z])
```

Ou on tape :

```
grad(2*x^2*y-x*z^3,[x,y,z])
```

On obtient :

```
[2*2*x*y-z^3,2*x^2,-(x*3*z^2)]
```

On obtient après simplification avec `normal(ans())` :

```
[4*x*y-z^3,2*x^2,-(3*x*z^2)]
```

Si on veut connaitre les points critiques de $F(x, y, z) = 2x^2y - xz^3$, il suffit de taper :

```
solve(derive(2*x^2*y-x*z^3,[x,y,z]),[x,y,z])
```

On obtient :

```
[[0,y,0]]
```

6.53.2 Le Laplacien : `laplacian`

`laplacian` a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
`laplacian` renvoie le laplacien de F ($\nabla^2(F) = \frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} + \frac{\partial^2 F}{\partial z^2}$ si $n = 3$).
Exemple
Déterminer le laplacien de $F(x, y, z) = 2x^2y - xz^3$.
On tape :

```
laplacian(2*x^2*y-x*z^3,[x,y,z])
```

On obtient :

```
4*y+-6*x*z
```

6.53.3 La matrice hessienne : hessian

hessian a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
hessian renvoie la hessienne de F qui est la matrice des dérivées d'ordre 2.
Exemple
Déterminer la hessienne de $F(x, y, z) = 2x^2y - xz^3$.
On tape :

```
hessian(2*x^2*y-x*z^3 , [x,y,z])
```

On obtient :

```
[[4*y,4*x,-(3*z^2)],[2*2*x,0,0],[-(3*z^2),0,x*3*2*z]]
```

Pour avoir la hessienne aux points critiques on tape tout d'abord :

```
solve(derive(2*x^2*y-x*z^3,[x,y,z]),[x,y,z])
```

pour avoir les points critiques.
On obtient :

```
[[0,y,0]]
```

Puis, on calcule la hessienne en ces points, on tape :

```
subst([[4*y,4*x,-(3*z^2)],[2*2*x,0,0],
[-(3*z^2),0,6*x*z]],[x,y,z],[0,y,0])
```

On obtient :

```
[[4*y,4*0,-(3*0^2)],[4*0,0,0],[-(3*0^2),0,6*0*0]]
```

et après simplification :

```
[[4*y,0,0],[0,0,0],[0,0,0]]
```

6.53.4 La divergence : divergence

divergence a deux paramètres : une expression F dependant de n variables rèelles et un vecteur de dimension n indiquant le nom de ces variables.
On a si $n = 3$:
divergence([A,B,C],[x,y,z])=$\frac{\partial A}{\partial x} + \frac{\partial B}{\partial y} + \frac{\partial C}{\partial z}$
On tape :

```
divergence([x*z,-y^2,2*x^y],[x,y,z])
```

On obtient :

```
z+-2*y
```

6.53.5 Le rotationnel : curl

curl a deux paramètres : une expression F dependant de 3 variables rèelles et un vecteur de dimension 3 indiquant le nom de ces variables.
curl désigne le rotationnel de F.
Attention, ici $n = 3$.
On a :
curl([A,B,C],[x,y,z])=$[\frac{\partial C}{\partial y} - \frac{\partial B}{\partial z}, \frac{\partial A}{\partial z} - \frac{\partial C}{\partial x}, \frac{\partial B}{\partial x} - \frac{\partial A}{\partial y}]$.
On tape :

```
curl([x*z,-y^2,2*x^y],[x,y,z])
```

On obtient :

```
[2*x^y*log(x),x-2*y*x^(y-1),0]
```

6.53.6 Le potentiel : potential

potential a deux arguments : un vecteur $\overrightarrow{V}$ de $\mathbb{R}^n$ dépendant de n variables et le vecteur constitué du nom de ces variables.
potential renvoie une fonction U telle que $\overrightarrow{\mathcal{G}rad}(U) = \overrightarrow{V}$ si bien sûr, cela est possible ! On dit alors que $\overrightarrow{V}$ dérive du potentiel U.
La solution générale est la somme d'une solution particulière et d'une constante.
On sait qu'un vecteur $\overrightarrow{V}$ est un gradient si et seulement si son rotationnel est nul : autrement dit si curl(V)=0.
potential est la fonction réciproque de derive.
On tape :

```
potential([2*x*y+3,x^2-4*z,-4*y],[x,y,z])
```

On obtient :

```
2*y*x^2/2+3*x+(x^2-4*z-2*x^2/2)*y
```

6.53.7 Champ à flux conservatif : vpotential

vpotential a deux arguments : un vecteur $\overrightarrow{V}$ de $\mathbb{R}^n$ dépendant de n variables et le vecteur constitué du nom de ces variables.
vpotential renvoie un vecteur $\overrightarrow{U}$ tel que $\overrightarrow{Rot}(\overrightarrow{U}) = \overrightarrow{V}$ si bien sûr, cela est possible ! On dit alors que $\overrightarrow{V}$ est un champ à flux conservatif ou un champ solénoïdal.
La solution générale est la somme d'une solution particulière et du gradient d'une fonction arbitraire, Xcas renvoie le vecteur solution particulière de première composante nulle.
On sait qu'un vecteur $\overrightarrow{V}$ est un rotationnel si et seulement si sa divergence est nulle : autrement dit si divergence(V)=0.
En électro-magnétisme on a :
$\overrightarrow{V}$= $\overrightarrow{B}$= le champ magnétique et
$\overrightarrow{U}$= $\overrightarrow{A}$= le potentiel vecteur.
vpotential est la fonction réciproque de curl.
On tape :

```
vpotential([2*x*y+3,x^2-4*z,-2*y*z],[x,y,z])
```

On obtient :

```
[0,(-(2*y))*z*x,-x^3/3-(-(4*z))*x+3*y]
```

6.54 Équations

6.54.1 Écrire une équation : `equal`

`equal` a comme paramètre les deux membres d'une équation.
`equal` renvoie cette équation.
On tape :

```
equal(2x-1,3)
```

On obtient :

```
(2*x-1)=3
```

On peut bien sûr écrire directement `(2*x-1)=3`.

6.54.2 Transformer une équation en diffèrence : `equal2diff`

`equal2diff` a comme paramètre une équation.
`equal2diff` renvoie la différence des deux membres de l'équation.
On tape :

```
equal2diff(2x-1=3)
```

On obtient :

```
2*x-1-3
```

6.54.3 Transformer une équation en une liste : `equal2list`

`equal2list` a comme paramètre une équation.
`equal2list` renvoie la liste formée par les deux membres de l'équation.
On tape :

```
equal2list(2x-1=3)
```

On obtient :

```
[2*x-1,3]
```

6.54.4 Pour avoir le membre de gauche d'une équation : `left` `gauche` `lhs`

`left` ou `lhs` a comme paramètre une équation ou un intervalle.
`left` ou `lhs` renvoie le membre de gauche de l'équation ou de la borne gauche de l'intervalle.
On tape :

```
left(2x-1=3)
```

Ou on tape :

```
lhs(2x-1=3)
```

On obtient :

```
2*x-1
```

On tape :

```
left(1..3)
```

Ou on tape :

```
lhs(1..3)
```

On obtient :

```
1
```

6.54.5 Pour avoir le membre de droite d'une équation : `right` `droit` `rhs`

`right` ou `rhs` a comme paramètre une équation ou un intervalle.
`right` ou `rhs` renvoie le membre de droite de l'équationou de la borne droite de l'intervalle.
On tape :

```
right(2x-1=3)
```

Ou on tape :

```
rhs(2x-1=3)
```

On obtient :

```
3
```

On tape :

```
right(1..3)
```

Ou on tape :

```
rhs(1..3)
```

On obtient :

```
3
```

6.54.6 Résolution d'équations : `solve resoudre`

`solve` permet de résoudre une équation ou un système d'équations polynômiales.
`solve` a 1 ou 2 arguments qui sont une expression xpr en x ou une expression xpr d'une variable var et le nom de cette variable var.
`solve` resout $xpr = 0$ l'inconnue étant x ou var **Attention**
La deuxième variable peut spécifier un intervalle par exemple $x = a..b$ pour n'avoir que les solutions dans l'intervalle $[a; b]$ **mais** dans ce cas les solutions seront numériques et `solve` est alors identique à `fsolve`, par exemple :
`solve(t^2-2,t=0..2)` ou `fsolve(t^2-2,t=0..2)` renvoie `[1.41421356237]` alors que `solve(t^2-2,t)` renvoie `[-(sqrt(2)),sqrt(2)]`.

- Résolution d'une équation
  `solve` a comme arguments une équation entre deux expressions (ou une expression et dans ce cas `=0` est sous-entendu), et le nom d'une variable (par défaut `x`).
  `solve` résout cette équation.
- Résolution d'un système d'équations polynômiales
  Soit `solve` a comme arguments deux vecteurs : le vecteur des équations polynômiales et le vecteur contenant le nom des variables.
  `solve` résout ce système d'équations polynômiales.

Attention
Par défaut, `solve` ne renvoie que les solutions réelles : pour avoir les solutions complexes il faut être en mode complexe c'est à dire avoir coché `Complexe` dans la configuration du cas.
De plus,

- pour les équations comportant un dénominateur, `solve` ne tient pas compte du dénominateur et renvoie les solutions du numérateur.
  Par exemple `solve(ln(abs(x-2))/ln(x))` renvoie `[1,3]` ou `solve((x^2-2x)/x)` renvoie `[0,2]`,
- pour les équations trigonométriques, `solve` ne renvoie que les solutions principales : pour avoir toutes les solutions il faut avoir coché `All_trig_sol` dans la configuration du cas.

Exemples :

- Résoudre $x^4 - 1 = 3$ On tape :
  `solve(x^4-1=3)`
  On obtient en mode réel :
  `[sqrt(2),-(sqrt(2))]`
  On obtient en mode complexe :
  `[sqrt(2),-(sqrt(2)),(i)*sqrt(2),-((i)*sqrt(2))]`
- Résoudre $\exp(x) = 2$
  On tape :
  `solve(exp(x)=2)`
  On obtient en mode réel :
  `[log(2)]`
- Résoudre en x, y le système $x + y = 1, x - y = 0$:
  On tape :
  `solve([x+y=1,x-y],[x,y])`

On obtient :

`[[1/2,1/2]]`

- Résoudre en x, y le système $x^2 + y = 2, x + y^2 = 2$
  On tape :

  `solve([x^2+y=2,x+y^2=2],[x,y])`

  On obtient :

  `[[-2,-2],[1,1],[(-sqrt(5)+1)/2,(1+sqrt(5))/2],`
  `[(sqrt(5)+1)/2,(1-sqrt(5))/2]]`
- Résoudre en x, y, z le système $x^2 - y^2 = 0, x^2 - z^2 = 0$:
  On tape :

  `solve([x^2-y^2=0,x^2-z^2=0],[x,y,z])`

  On obtient :

  `[[x,x,x],[x,-x,-x],[x,-x,x],[x,x,-x]]`
- Résoudre $\cos(2 * x) = 1/2$
  On tape :

  `solve(cos(2*x)=1/2)`

  On obtient :

  `[pi/6,(-pi)/6]`

  On obtient en ayant coché `All_trig_sol` :

  `[(6*pi*n_0+pi)/6,(6*pi*n_0-pi)/6]`

Remarque Pour pouvoir par exemple trouver l'intersection d'une droite (donnée par son équation paramétrique) et un plan on peut mettre les équations sous la forme d'une liste contenant une liste.
Exemple : Trouver l'intersection de la droite D d'équation paramétrique $[y - z = 0, z - x = 0, x - y = 0]$ et du plan P d'équation $x - 1 + y + z = 0$.
On tape

```
solve([[y-z=0,z-x=0,x-y=0],x-1+y+z=0],[x,y,z])
```

On obtient :

```
[[1/3,1/3,1/3]]
```

Remarque
Différence entre `solve` et `csolve` : En mode complexe `solve` renvoie le même résultat que `csolve` (pour `csolve` que l'on soit en mode complexe ou réel cela importe peu). Ainsi, si on ne veut pas que le résultat dépende du mode, pour avoir les solutions complexes il est préférable d'utiliser `csolve`.
On tape en mode réel :

```
solve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[1/(cos(t))]
```

car en mode réel l'inconnue `r` est considérée comme un nombre réel.
On tape en mode complexe :

```
solve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)]
```

car en mode complexe, l'expression contient `i` donc l'inconnue `r` est considérée comme un nombre complexe. La solution complexe est :
`` ` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1) `` (la partie réelle de `r` vaut `` ` x` `` qui est un nombre rèel quelconque et la partie imaginaire de `r` est fonction de `` ` x` ``).
On tape en mode complexe ou réel :

```
csolve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)]
```

car avec `csolve`, l'inconnue `r` est toujours considérée comme un nombre complexe.

6.54.7 Résoudre des équations dans $\mathbb{C}$: `resoudre_dans_C csolve cSolve`

`csolve` ou `cSolve` ou `resoudre_dans_C` résout une équation ou un système d'équations polynômiales dans $\mathbb{C}$ sans avoir besoin d'être en mode complexe.

- Résolution d'une équation
  `cSolve` a comme arguments une équation entre deux expressions ou une expression (`=0` est alors sous-entendu), et le nom d'une variable (par défaut `x`).
  `cSolve` résout cette équation dans $\mathbb{C}$ sans avoir besoin d'être en mode complexe.
- Résolution d'un système d'équations polynômiales
  `cSolve` a comme arguments deux vecteurs : le vecteur des équations du système et le vecteur du nom des variables.
  `cSolve` résout ce système d'équations polynômiales dans $\mathcal{C}$ sans avoir besoin d'être en mode complexe.

On tape :

```
cSolve(x^4-1=3)
```

On obtient :

```
[sqrt(2),-(sqrt(2)),(i)*sqrt(2),-((i)*sqrt(2))]
```

On tape :

```
cSolve([-x^2+y=2,x^2+y],[x,y])
```

On obtient :

```
[[i,1],[-i,1]]
```

On tape en mode complexe ou réel :

```
csolve(re(r*exp(-(i)*t))-1,r)
```

On obtient :

```
[` x`+(i)*1/(sin(t))*(-` x`*cos(t)+1)]
```

car avec `csolve`, l'inconnue `r` est toujours considérée comme un nombre complexe.)

6.55 Les systèmes linéaires

Dans tout ce paragraphe, on appelle "matrice augmentée" du système $A * X = B$ la matrice formée par la matrice A bordée à droite par le vecteur colonne B ("matrice augmentée" du système $A * X = B$=`border(A,tran(B))`).

6.55.1 Matrice d'un système : `syst2mat`

`syst2mat` a comme argument un vecteur contenant un système d'équations linéaires et un vecteur contenant les variables.
`syst2mat` écrit le système $AX = B$ sous la forme d'une matrice formée de A bordée à droite par $-B$.
On tape :

```
syst2mat([x+y,x-y-2],[x,y])
```

On obtient :

```
[[1,1,0],[1,-1,-2]]
```

On tape :

```
syst2mat([x+y=0,x-y=2],[x,y])
```

On obtient :

```
[[1,1,0],[1,-1,-2]]
```

Attention !!!
Il faut purger auparavant les variables (ici `x` et `y`).

6.55.2 Réduction de Gauss d'une matrice : `ref`

`ref` permet de résoudre un système d'équations linéaires que l'on écrit sous forme matricielle :

```
A*X=B
```

Le paramètre de `ref` est la "matrice augmentée" du système (celle formée par la matrice `A` du système et ayant comme dernier vecteur colonne le second membre `B`).
Le résultat est une matrice `[A1,B1]` : `A1` a des zéros au dessous de sa diagonale et les solutions de :

```
A1*X=B1
```

sont les mêmes que celles de :

```
A*X=B
```

`ref` peut travailler dans $\mathbb{Z}/p\mathbb{Z}$.
Par exemple, soit à résoudre le système dans $\mathbb{R}$ et dans $\mathbb{Z}/5\mathbb{Z}$:

$$\left\{ \begin{array}{lcr} 3x + y & = & -2 \\ 3x + 2y & = & 2 \end{array} \right.$$

On tape pour résoudre le système dans $\mathbb{R}$:

```
ref([[3,1,-2],[3,2,2]])
```

On obtient :

```
[[1,1/3,-2/3],[0,1,4]]
```

cela signifie donc que :
$y = 4$ et $x = -2$ sont solutions du système. On tape pour résoudre le système dans $\mathbb{Z}/5\mathbb{Z}$:

```
ref([[3,1,-2],[3,2,2]]%5)
```

On obtient :

```
[[1 % 5,2 % 5,1 % 5],[0 % 5,1 % 5,-1 % 5]]
```

cela signifie donc que :
$y = -1\%5$ et $x = 3\%5$ sont solutions du système.

Remarque

Lorsque le nombre de colonnes est égal au nombre de lignes +1 `ref` ne divise pas par le pivot de la derniere colonne, par exemple, on tape :

```
ref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],[1,0,0,1,-a4]])
```

On obtient :

```
[[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],[0,0,0,0,a1-a2+a3-a4]]
```

Ainsi on peut savoir que si `a1-a2+a3-a4` n'est pas nul, il n'y a pas de solution.

6.55.3 Réduction de Gauss-Jordan : `rref gaussjord`

`rref` permet de résoudre un système d'équations linéaires que l'on écrit sous forme matricielle (voir aussi 6.33.17) :

```
A*X=B
```

`rref` a un ou deux paramètres.

– Si `rref` n'a qu'un paramètre, ce paramètre est la "matrice augmentée" du système (celle formée par la matrice `A` du système et ayant comme dernier vecteur colonne le second membre `B`).
  Le résultat est une matrice `[A1,B1]` : `[A1,B1]` a des zéros de part et d'autre de sa diagonale et des 1 sur sa diagonale et les solutions de :
  `A1*X=B1`
  sont les mêmes que celles de :
  `A*X=B`
  `rref` peut travailler dans $\mathbb{Z}/p\mathbb{Z}$.
  Par exemple, soit à résoudre le système dans $\mathbb{R}$ et dans $\mathbb{Z}/5\mathbb{Z}$:

$$\left\{ \begin{array}{rcr} 3x + y & = & -2 \\ 3x + 2y & = & 2 \end{array} \right.$$

  On tape pour résoudre le système dans $\mathbb{R}$:

```
rref([[3,1,-2],[3,2,2]])
```

On obtient :

```
[[1,0,-2],[0,1,4]]
```

cela signifie donc que :
$x = -2$ et $y = 4$ sont solutions du système.
On tape pour résoudre le système dans $\mathbb{Z}/5\mathbb{Z}$:

```
rref([[3,1,-2],[3,2,2]]%5)
```

On obtient :

```
[[1 % 5,0,-2 % 5],[0,1 % 5,-1 % 5]]
```

cela signifie donc que :
$x = -2\%5$ et $y = -1\%5$ sont solutions du système.

– Si `rref` a deux paramètres, le deuxième paramètre est un entier k qui permet de ne faire Gauss-Jordan que sur les k premières colonnes ou est l'option `conserver_pivot` ou `keep_pivot`.
On tape :

```
rref([[3,1,-2,1],[3,2,2,2]],1)
```

On obtient :

```
[[3,1,-2,1],[0,1,4,1]]
```

L'option `conserver_pivot` ou `keep_pivot` permet de ne pas diviser par les pivots, cela pour éviter de diviser par des pivots nuls.
par exemple, pour résoudre le système dans $\mathbb{R}$ et dans $\mathbb{Z}/2\mathbb{Z}$:

$$\begin{cases} z_1 + z_2 & = a_1 \\ z_2 + z_3 & = a_2 \\ z_3 + z_4 & = a_3 \\ z_4 + z_1 & = a_4 \end{cases}$$

On tape pour résoudre le système dans $\mathbb{R}$:

```
A:=syst2mat([z1+z2=a1,z2+z3=a2,z3+z4=a3,z4+z1=a4],
                    [z1,z2,z3,z4])
```

puis,

```
rref(A,keep_pivot)
```

ou bien, on tape directement :

```
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
        [1,0,0,1,-a4]],keep_pivot)
```

On obtient :

```
[[a1-a2+a3-a4,0,0,a1-a2+a3-a4,0],
[0,a1-a2+a3-a4,0,-a1+a2-a3+a4,0],
[0,0,a1-a2+a3-a4,a1-a2+a3-a4,0],
[0,0,0,0,a1-a2+a3-a4]]
```

Remarque
Lorsque le nombre de colonnes est égal au nombre de lignes +1 `rref` ne divise pas par le pivot de la derniere colonne, par exemple, pour résoudre le système :

$$\begin{cases} z_1 + z_2 & = a_1 \\ z_2 + z_3 & = a_2 \\ z_3 + z_4 & = a_3 \\ z_4 + z_1 & = a_4 \end{cases}$$

On tape :

```
A:=syst2mat([z1+z2=a1,z2+z3=a2,z3+z4=a3,z4+z1=a4],
                [z1,z2,z3,z4])
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
                [1,0,0,1,-a4]])
```

On obtient :

```
[[1,0,0,1,0],[0,1,0,-1,0],[0,0,1,1,0],
        [0,0,0,0,a1-a2+a3-a4]]
```

Ainsi on peut savoir que si `a1-a2+a3-a4` n'est pas nul, il n'y a pas de solution. Puis, on tape :

```
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
               [1,0,0,1,-a4]],3)
```

On obtient :

```
[[1,0,0,1,-a1+a2-a3],[0,1,0,-1,-a2+a3],[0,0,1,1,-a3],
                [0,0,0,0,1]]
```

ou encore si on remplace `a4` par `a1-a2+a3`, on tape :

```
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
              [1,0,0,1,-a1+a2-a3]])
```

On obtient :

```
[[1,0,0,1,-a1+a2-a3],[0,1,0,-1,-a2+a3],[0,0,1,1,-a3],
                [0,0,0,0,0]]
```

On peut aussi taper :

```
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
          [1,0,0,1,-a4]],3,keep_pivot)
```

On obtient tout de suite :

```
[[1,0,0,1,-a1+a2-a3],[0,1,0,-1,-a2+a3],[0,0,1,1,-a3],
        [0,0,0,0,a1-a2+a3-a4]]
```

Donc, les solutions lorsque `a1-a2+a3-a4=0` sont :
`z1=a1-a2+a3-z4, z2=a2-a3+z4, z3=a3-z4, z4=z4.`
On tape pour résoudre le système dans $\mathbb{Z}/2\mathbb{Z}$:

```
rref([[1,1,0,0,-a1],[0,1,1,0,-a2],[0,0,1,1,-a3],
         [1,0,0,1,-a4]]%2,3,keep_pivot)
```

On obtient :

```
[[1 % 2,0,0,1 % 2,(1 % 2)*a1+(1 % 2)*a2+(1 % 2)*a3],
[0,1 % 2,0,1 % 2,(1 % 2)*a2+(1 % 2)*a3],[0,0,1 % 2,1
% 2,(1 % 2)*a3],
[0,0,0,0,(1 % 2)*a1+(1 % 2)*a2+(1 % 2)*a3+(1 % 2)*a4]]
```

Donc lorsque `(a1+a2+a3+a4=0%2)`, les solutions dans $\mathbb{Z}/2\mathbb{Z}$ sont :
`z1=a1+a2+a3+z4%2, z2=a2+a3+z4%2, z3=a3+z4%2, z4=(1 % 2)*z4.`

– `rref` permet aussi de résoudre plusieurs systèmes d'équations linéaires qui ne diffèrent que par leur second membre. On écrit alors les second membres comme les colonnes d'une matrice.
On tape :

```
rref([[3,1,-2,1],[3,2,2,2]])
```

On obtient :

```
[[1,0,-2,0],[0,1,4,1]]
```

cela signifie donc que :
$x = -2$ et $y = 4$ sont solutions du système

$$\begin{cases} 3x + y & = & -2 \\ 3x + 2y & = & 2 \end{cases}$$

et que $x = 0$ et $y = 1$ sont solutions du système

$$\begin{cases} 3x + y & = & 1 \\ 3x + 2y & = & 2 \end{cases}$$

6.55.4 Résolution de A*X=B : `simult`

`simult` permet de résoudre un système d'équations linéaires (resp plusieurs systèmes d'équations linéaires qui ne diffèrent que par leur second membre).
On écrit le (rep les) système(s) sous forme matricielle (voir aussi 6.33.17) :

```
A*X=b (resp A*X=B)
```

Les paramètres de `simult` sont la matrice `A` du système et le vecteur colonne (i.e. une matrice d'une colonne) `b` formé par le second membre du système à résoudre (resp la matrice `B` dont les colonnes sont les vecteurs `b` des second membres des systèmes à résoudre).
Le résultat est un vecteur colonne solution du système (resp une matrice dont les colonnes sont les solutions des différents systèmes).
Par exemple, soit à résoudre le système :

$$\begin{cases} 3x + y & = & -2 \\ 3x + 2y & = & 2 \end{cases}$$

On tape :

```
simult([[3,1],[3,2]],[[-2],[2]])
```

On obtient :

```
[[-2],[4]]
```

cela signifie donc que :
$x = -2$ et $y = 4$ sont solutions du système.
On tape :

```
simult([[3,1],[3,2]],[[-2,1],[2,2]])
```

On obtient :

```
[[-2,0],[4,1]]
```

cela signifie donc que :
$x = -2$ et $y = 4$ sont solutions du système

$$\begin{cases} 3x + y & = & -2 \\ 3x + 2y & = & 2 \end{cases}$$

et que $x = 0$ et $y = 1$ sont solutions du système

$$\begin{cases} 3x + y & = & 1 \\ 3x + 2y & = & 2 \end{cases}$$

6.55.5 Étape de la réduction de Gauss-Jordan d'une matrice : `pivot`

`pivot` a trois arguments : une matrice de n lignes et p colonnes et deux entiers l et c vérifiant : $0 \leq l < n$ et $0 \leq c < p$.
`pivot(A,l,c)` renvoie la matrice obtenue en créant des zéros dans la colonne `c` de `A`, avec la méthode de Gauss-Jordan, en utilisant comme pivot l'élément `A[l,c]`.
On tape :

```
pivot([[1,2],[3,4],[5,6]],1,1)
```

On obtient :

```
[[-2,0],[3,4],[2,0]]
```

On tape :

```
pivot([[1,2],[3,4],[5,6]],0,1)
```

On obtient :

```
[[1,2],[2,0],[4,0]]
```

6.55.6 Résoudre un système linéaire : `linsolve` `resoudre_systeme_lineaire`

`linsolve` permet de résoudre un système d'équations linéaires où chaque équation est de la forme $Xpr = 0$ où Xpr est une expression.
`linsolve` a comme paramètres la liste des équations et la liste des variables.
`linsolve` renvoie une liste qui est solution du système d'équations.
`linsolve` permet de résoudre aussi un système d'équations linéaires dans $\mathbb{Z}/n\mathbb{Z}$.
On tape :

```
linsolve([2*x+y+z=1,x+y+2*z=1,x+2*y+z=4],[x,y,z])
```

On obtient :

```
[1/-2,5/2,1/-2]
```

donc

$$x = -\frac{1}{2}, y = \frac{5}{2}, z = -\frac{1}{2}$$

sont solutions du système :

$$\begin{cases} 2x + y + z & = 1 \\ x + y + 2z & = 1 \\ x + 2y + z & = 4 \end{cases}$$

On tape :

```
linsolve([2*x+y+z-1,x+y+2*z-1,x+2*y+z-4]%3,[x,y,z])
```

On obtient :

```
[1 % 3,1 % 3,1 % 3]
```

donc

$$x = 1\%3, y = 1\%3, z = 1\%3$$

sont solutions du système :

$$\begin{cases} (2x + y + z - 1) & \%3 & = 0 \\ (x + y + 2z - 1) & \%3 & = 0 \\ (x + 2y + z - 4) & \%3 & = 0 \end{cases}$$

6.55.7 Norme minimale d'un système linéaire : `LSQ`

`LSQ` a pour arguments une matrice `A` et un vecteur (rep une matrice) `B` qui représentent le (resp les) système(s) linéaire(s) `A*X=B` si `B` est un vecteur (resp une matrice).
`LSQ(A,B)` calcule la norme minimale selon la méthode des moindres carrés du système linéaire `A*X=B` sur- ou sous-déterminé c'est pour estimer la solution d'un système linéaire `A*X=B` (si `B` est un vecteur) ou des systèmes linéaires `A*X=B` (si `B` est une matrice) pour :
- un système sur-déterminé (on a plus de lignes que de colonnes)
  - si `B` est un vecteur : on cherche `X` de norme euclidienne minimum qui minimise la norme euclidienne de `(AX-B)`.
  - si `B` est une matrice : on cherche `Xj` de norme euclidienne minimum parmi les solutions qui minimise la norme euclidienne de `(AXj-Bj)`
- un système sous determiné (en général on a plus de colonnes que de lignes) On cherche `X` qui minimise la norme de Frobenius de `(AX-B)` (la norme de Frobenius d'une matrice M est $\sqrt{\sum_{j,k} |M(j,k)|^2}$).
- un système exactement déterminé (le nombre de colonnes est égal au nombre de lignes et `A` est inversible).
  On utilise `inv(A)*B` pour avoir `X` qui produit des résultats faux en calcul approché si la matrice est mal conditionnée (équations indépendantes proche l'une de l'autre)

On tape :

```
LSQ([[1,2],[3,4]],[5,11])
```

On obtient :

```
[[1],[2]]
```

En effet si `X:=[[1],[2]]`, on a `[[1,2],[3,4]]*X=[[5],[11]]` On tape :

```
LSQ([[1,2],[3,4]],[7,9])
```

On obtient :

```
[[-5],[6]]
```

En effet si `X:=[[-5],[6]]`, on a `[[1,2],[3,4]]*X=[[7],[9]]` On tape :

```
LSQ([[1,2],[3,4]],[[5,7],[11,9]])
```

On obtient :

`[[1,-5],[2,6]]`

En effet si `X:=[[1,-5],[2,6]]`, on a `[[1,2],[3,4]]*X=[[5,7],[11,9]]`
On tape :

`LSQ([[1,2],[3,4],[3,6]],[5,11,13])`

On obtient :

`[[11/5],[11/10]]`

En effet puisque `linsolve([[1,2],[3,4],[3,6]]*[x,y]-[5,11,13],[x,y])` renvoie `[]`, on cherche la norme minimum de `C` définit par :
`C:= [[1,2],[3,4],[3,6]]*[x,y]-[5,11,13]`.
On a :
`norm(C)` renvoie :
`sqrt(19*x^2+64*x*y-154*x+56*y^2-264*y+315)`
`gauss(19*x^2+64*x*y-154*x*z+56*y^2-264*y*z+315*z^2,[x,y,z])`
renvoie :
`((19*x+32*y-77*z)^2)/19+((40/19*y+(-44)/19*z)^2)/(40/19)+(2*z^2)/5`
`linsolve([19*x+32*y-77=0,40/19*y+(-44)/19=0],[x,y])` renvoie :
`[11/5,11/10]`
On tape :

`LSQ([[1,2],[3,4],[3,6]],[[5,-1],[11,-1],[13,-3]]`

On obtient :

`[[11/5,1],[11/10,-1]]`

En effet `linsolve([[1,2],[3,4],[3,6]]*[x,y]-[-1,-1,-3],[x,y])` renvoie : `[1,-1]`
On tape :

`LSQ([[3,4]],[12])`

On obtient :

`[[36/25],[48/25]]`

En effet si `d:=droite(3x+4y-12)` et si `M:=projection(d,point(0))` alors `coordonnees(M)` renvoie :
`[36/25,48/25]`

6.55.8 Résolution d'une récurrence linéaire : `reverse_rsolve`

`reverse_rsolve` a comme paramètre un vecteur $v = [v_0...v_{2n-1}]$ de longueur paire égale à $2n$.
`reverse_rsolve` permet de résoudre une récurrence linéaire de degré inférieur à n

$$x_n * v_{n+k} + ... + x_0 * v_k = 0$$

où les x_j sont les $n+1$ inconnues.
`reverse_rsolve` renvoie la liste $x = [x_n, ...x_1, x_0]$ des coefficients x_j (si $x_n \neq 0$ alors $x_n = 1$).

En d'autres termes, `reverse_rsolve` résout le système d'équations linéaires de n équations à $n+1$ inconnues :

$$\begin{aligned} x_n * v_n + ... + x_0 * v_0 &= 0 \\ &... \\ x_n * v_{n+k} + ... + x_0 * v_k &= 0 \\ &... \\ x_n * v_{2*n-1} + ... + x_0 * v_{n-1} &= 0 \end{aligned}$$

La matrice A du système à résoudre a n lignes et $n+1$ colonnes :

$$A = [[v_0, v_1...v_n], [v_1, v_2, ...v_{n-1}], ..., [v_{n-1}, v_n...v_{2n-1}]]$$

`reverse_rsolve` renvoie la liste $x = [x_n, ...x_1, x_0]$ des coefficients x_j (si $x_n \neq 0$ alors $x_n = 1$) et x est la solution dy système $A * \texttt{revlist}(x)$.

Exemples

– Trouver une suite vérifiant la récurrence linéaire de degré au plus 2 et dont les premiers termes sont 1, -1, 3, 3.
  On tape :
  `reverse_rsolve([1,-1,3,3])`
  On obtient :
  `[1,-3,-6]`
  Sans `reverse_rsolve`, on aurait du écrire la matrice du système à résoudre :
  $[[1, -1, 3], [-1, 3, 3]]$
  puis utiliser la commande `rref` :
  `rref([[1,-1,3],[-1,3,3]])`
  de réponse `[[1,0,6],[0,1,3]]`
  donc puisque $x_2 = 1$, on a $x_0 = -6$ et $x_1 = -3$
  et on a bien :
  $x_0 - x_1 + 3x_2 = 0$ et $-x_0 + 3x_1 + 3x_2 = 0$
– Trouver une suite vérifiant la récurrence linéaire de degré au plus 3 et dont les premiers termes sont 1, -1, 3, 3,-1, 1.
  On tape :
  `reverse_rsolve([1,-1,3,3,-1,1])`
  On obtient :
  `[1,(-1)/2,1/2,-1]`
  La matrice du systeme à résoudre est donc :
  `[[1,-1,3,3],[-1,3,3,-1],[3,3,-1,1]]` On a si on utilise `rref` :
  `rref([[1,-1,3,3],[-1,3,3,-1],[3,3,-1,1]])`
  de réponse `[1,0,0,1],[0,1,0,1/-2],[0,0,1,1/2]]` donc puisque $x_3 = 1$, on a $x_0 = -1$, $x_1 = 1/2$ et $x_2 = -1/2$

6.56 Les équations différentielles

Pour le calcul numérique de solutions déquations différentielles on se reportera à `odesolve` et pour la représentation graphique de solutions déquations differentielles on se reportera à `plotfield`, `plotode`, `interactive_plotode`.

6.56.1 Équations différentielles : `desolve deSolve dsolve`

`desolve` (ou `deSolve`) permet de résoudre :

- les équations différentielles linéaires à coefficients constants du premier ou du deuxième ordre,
- les équations différentielles linéaires du premier ordre,
- les équations différentielles du premier ordre incomplète en y,
- les équations différentielles du premier ordre incomplète en x,
- les équations différentielles du premier ordre à variables séparées,
- les équations différentielles du premier ordre homogènes ($y' = F(y/x)$),
- les équations différentielles du premier ordre ayant un facteur intégrant,
- les équations différentielles de Bernoulli ($a(x)y' + b(x)y = c(x)y^n$),
- les équations différentielles de Clairaut ($y = x * y' + f(y')$).
- les systèmes différentiels linéaires à coefficients constants ($[y' = [[1, 2], [2, 1]]* y + [x, x + 1], y(0) = [1, 2]]$)).

Les paramètres de `desolve` :

- quand l'équation différentielle est du premier ordre, que la variable est x et que l'inconnue est y, les paramètres sont :
  l'équation différentielle, par exemple :
  `desolve(y'+x*y=0)`
  ou
  l'équation différentielle suivie de la liste $[x_0, y_0]$ qui donne comme condition initiale $y(x_0) = y_0$, par exemple
  `desolve(y'+x*y=0,[0,1])`
  ou
  la liste formée par l'équation différentielle et de la relation $y(x_0) = y_0$ qui donne la condition initiale et comme 2nd argument l'inconnue y, par exemple
  `desolve([y'+x*y=0,y(0)=1],y)`
  `desolve([y″+2*y'+y,y(0)=1,y'(0)=0],y])`
- quand la variable est x et l'inconnue n'est pas y :
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équation différentielle et les conditions initiales) et l'inconnue y (ou z ou ..), par exemple

```
desolve(z'+x*z=0,z)
desolve([y'+x*y=0,y(0)=1]);
desolve([y'+x*y=0,y(0)=1],y)
desolve([z'+x*z=0,z(0)=1],z)
desolve([y''+2*y'+y,y(0)=1,y'(0)=0]);
desolve([y''+2*y'+y,y(0)=1,y'(0)=0],y)
desolve([z''+2*z'+z,z(0)=1,z'(0)=0],z)
```

  Dans l'équation différentielle y s'écrit y et y' s'écrit y' et y" s'écrit y'' car on dérive par rapport à la variable x. Par exemple : `desolve(y″+2*y'+y,y)` et `desolve([y″+2*y'+y,y(0)=1,y'(0)=0],y)`.
- quand la variable n'est pas x (par exemple t.) et l'inconnue est ou n'est pas y :
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équa-

tion différentielle et les conditions initiales), la variable t et l'inconnue z ou l'inconnue $z(t)$ (la variable est alors t et l'inconnue est y).
Dans l'équation différentielle, y s'écrit y et y' s'écrit y' et y" s'écrit y'' car on dérive par rapport à la variable que l'on a spécifiée ; on peut aussi écrire y sous la forme $y(t)$ et $y\prime$ sous la forme `diff(y(t),t)`, y'' sous la forme `diff(y(t),t$2)` si la la variable que l'on a spécifiée est t. Mais l'écriture est moins conviviale !
Par exemple :
`desolve(diff(y(t),t$2)+2*diff(y(t),t)+y(t),y(t))` ; ou
`desolve(diff(y(t),t$2)+2*diff(y(t),t)+y(t),t,y)` ; ou
`desolve(y''+2y'+y,t,y)` ; ou
`desolve(y''+2y'+y,y(t))`
et
`desolve([y''+2*y'+y,y(0)=1,y'(0)=0],t,y)` ou
`desolve([y''+2*y'+y,y(0)=1,y'(0)=0],y(t))` ou
`desolve([z''+2*z'+z,z(0)=1,z'(0)=0],u,z)` ou
`desolve([z''+2*z'+z,z(0)=1,z'(0)=0],z(u))`.
Lorsqu'il n'y a pas de conditions initiales (ou une seule condition initiale pour une équation du second ordre), `desolve` renvoie la solution écrite avec des constantes d'intégration : `c_0, c_1` où `y(0)=c_0` et `y'(0)=c_1` ou renvoie une liste des solutions.

- quand on a un système différentiel linéaire à coefficients constants les paramètres sont :

  - si la variable est x et que l'inconnue est y, les paramètres sont le système différentiel donné sous forme matricielle, par exemple :
    `desolve(y'=[[1,2],[2,1]]*y+[x,x+1])`
    ou
    la liste formée par le système différentiel donné sous forme matricielle et de la relation qui donne la condition initiale (par $y(x_0) = [a_0, a_1]$).
    par exemple :
    `desolve([y'=[[1,2],[2,1]]*y+[x,x+1],y(0)=[1,2]])`
  - si la variable n'est pas x (par exemple t) es paramètres sont le système différentiel, la variable t et l'inconnue z ou l'inconnue $z(t)$ par exemple :
    `desolve(z'=[[1,2],[2,1]]*z+[t,t+1],t,z)` ou
    `desolve(z'=[[1,2],[2,1]]*z+[t,t+1],z(t))`
    et
    `desolve([z'=[[1,2],[2,1]]*z+[t,t+1],z(0)=[1,2]],t,z)`
    ou
    `desolve([z'=[[1,2],[2,1]]*z+[t,t+1],z(0)=[1,2]],z(t))`

Exemples

- Exemples de résolution d'une équation différentielle linéaire à coefficients constants du deuxième ordre.

  1. Résoudre :

$$y'' + y = \cos(x)$$

On tape (en tapant deux fois prime pour `y''`) :

```
desolve(y''+y=cos(x),y)
```

ou encore :

```
desolve(diff(diff(y))+y=cos(x),y)
```

On trouve :

```
c_0*cos(x)+(x+2*c_1)*sin(x)/2
```

`c_0, c_1` sont les constantes d'intégration : `y(0)=c_0` et `y'(0)=c_1`.
ou bien si la variable n'est pas `x` mais `t`, on tape :

```
desolve(derive(derive(y(t),t),t)+y(t)=cos(t),t,y)
```

ou encore

```
desolve(derive(derive(y(t),t),t)+y(t)=cos(t),[t,y])
```

On trouve alors :

```
c_0*cos(t)+(t+2*c_1)/2*sin(t)
```

`c_0, c_1` sont les constantes d'intégration : `y(0)=c_0` et `y'(0)=c_1`.

2. Résoudre :

$$y'' + y = \cos(x) \;\; y(0) = 1$$

On écrit, si veut les solutions vérifiant $y(0) = 1$:

```
desolve([y''+y=cos(x),y(0)=1],y)
```

On obtient

```
[cos(x)+(x+2*c_1)/2*sin(x)]
```

les composantes de ce vecteur sont solutions (ici on a une seule composante car on obtient une seule solution dépendant de la constante `c_1`).

3. Résoudre :

$$y'' + y = \cos(x) \;\; (y(0))^2 = 1$$

On veut les solutions vérifiant $(y(0))^2 = 1$, on tape alors :

```
desolve([y''+y=cos(x),y(0)^2=1],y)
```

On obtient

```
[-cos(x)+(x+2*c_1)/2*sin(x),
 cos(x)+(x+2*c_1)/2*sin(x)]
```

chaque composantes de cette liste est une solution, on a donc deux solutions dépendant de la constante `c_1` qui correspondent à $y(0) = 1$ et à $y(0) = -1$.

4. Résoudre :

$$y'' + y = \cos(x) \;\; (y(0))^2 = 1 \;\; y'(0) = 1$$

On veut les solutions vérifiant $(y(0))^2 = 1$ et $y'(0) = 1$, on tape alors :

```
desolve([y''+y=cos(x),y(0)^2=1,y'(0)=1],y)
```

On obtient :

```
[-cos(x)+(x+2)/2*sin(x),cos(x)+(x+2)/2*sin(x)]
```

chaque composante de cette liste est une solution.On a donc deux solutions.

5. Résoudre :
$$y'' + 2y' + y = 0$$
On tape alors :
```
desolve(y''+2*y'+y=0,y)
```
On obtient les solutions dépendant de 2 constantes d'intégration `c_0` et `c_1` :
```
(x*c_0+x*c_1+c_0)*exp(-x)
```
6. Résoudre :
$$y'' - 6y' + 9y = xe^{3x}$$
On tape alors :
```
desolve(y''-6*y'+9*y=x*exp(3*x),y)
```
On obtient :
```
(x^3+(-(18*x))*c_0+6*x*c_1+6*c_0)*1/6*exp(3*x)
```
la solution dépend de 2 constantes d'intégration `c_0` et `c_1`.

– Exemples de résolution d'une équation différentielle linéaire du premier ordre.

1. Résoudre :
$$xy' + y - 3x^2 = 0$$
On tape alors :
```
desolve(x*y'+y-3*x^2,y)
```
Ou on tape :
```
desolve(x*y'+y-3*x^2)
```
On obtient :
```
(c_0+x^3)/x
```
2. Résoudre :
$$y' + x * y = 0, y(0) = 1$$
On tape alors :
```
desolve([y'+x*y=0, 0,1])
```
Ou on tape :
```
desolve([y'+x*y=0, y(0)=1],y)
```
Ou on tape :
```
desolve((y'+x*y=0) && (y(0)=1),[x,y])
```

Ou on tape :

```
desolve((y'+x*y=0) && (y(0)=1),x,y)
```

Ou on tape :

```
desolve((y'+x*y=0) && (y(0)=1),y)
```

On obtient :

```
[exp(-x^2)/2)]
```

3. Résoudre :

$$x(x^2-1)y'+2y=0$$

On tape alors :

```
desolve(x*(x^2-1)*y'+2*y=0)
```

Ou on tape :

```
desolve(x*(x^2-1)*y'+2*y=0,y)
```

On obtient :

```
(c_0)/((x^2-1)/(x^2))
```

4. Résoudre :

$$x(x^2-1)y'+2y=x^2$$

On tape alors :

```
desolve(x*(x^2-1)*y'+2*y=x^2)
```

Ou on tape :

```
desolve(x*(x^2-1)*y'+2*y=x^2,y)
```

On obtient :

```
(c_0*x^2+x^2*ln(x))/(x^2-1)
```

Si la variable est t et non x, par exemple :

$$t(t^2-1)y'(t)+2y(t)=t^2$$

On tape :

```
desolve(t*(t^2-1)*diff(y(t),t)+2*y(t)=(t^2),y(t))
```

On obtient :

```
(c_0*t^2+t^2*ln(t))/(t^2-1)
```

5. Résoudre :

$$x(x^2-1)y'+2y=x^2, y(2)=0$$

On tape alors :

```
desolve(x*(x^2-1)*y'+2*y=x^2,[2,0])
```

Ou on tape :

```
desolve([x*(x^2-1)*y'+2*y=x^2,y(2)=0],y)
```

On obtient :

```
[(-ln(2)*x^2+x^2*ln(x))/(x^2-1)]
```

6. Résoudre :

$$\sqrt{1+x^2}y' - x - y = \sqrt{1+x^2}$$

On tape alors :

```
desolve(y'*sqrt(1+x^2)-x-y-sqrt(1+x^2))
```

Ou on tape :

```
desolve(y'*sqrt(1+x^2)-x-y-sqrt(1+x^2),y)
```

On obtient :

```
(-c_0+ln(sqrt(x^2+1)-x))/(x-sqrt(x^2+1))
```

– Exemples de résolution d'une équation différentielle à variables séparées.

1. Résoudre :

$$y' = 2\sqrt{y}$$

On tape alors :

```
desolve(y'=2*sqrt(y))
```

Ou on tape :

```
desolve(y'=2*sqrt(y),y)
```

On obtient :

```
[x^2+-2*x*c_0+c_0^2]
```

2. Résoudre :

$$xy'\ln(x) - y(3\ln(x)+1) = 0$$

On tape alors :

```
desolve(x*y'*ln(x)-(3*ln(x)+1)*y)
```

Ou on tape :

```
desolve(x*y'*ln(x)-(3*ln(x)+1)*y,y)
```

On obtient :

```
c_0*x^3*ln(x)
```

– Exemples de résolution d'une équation différentielle de Bernoulli du premier ordre, de type $a(x)y' + b(x)y = c(x)y^n$ avec n=constante réelle.
Ces équations se résolvent en divisant par y^n, car on se ramène à une équation linéaire en $u = 1/y^{n-1}$.

1. Résoudre :

$$xy' + 2y + xy^2 = 0$$

On tape alors :

```
desolve(x*y'+2*y+x*y^2)
```

Ou on tape :

```
desolve(x*y'+2*y+x*y^2,y)
```

On obtient :

```
[(-1/x+c_0))*x^2]
```

2. Résoudre :

$$xy' - 2y = xy^3$$

On tape alors :

```
desolve(x*y'-2*y-x*y^3)
```

Ou on tape :

```
desolve(x*y'-2*y-x*y^3,y)
```

On obtient :

```
[((-2*x^5/5+c_0)/x^4)^(1/-2),
-((-2*x^5/5+c_0)/x^4)(1/-2)]
```

3. Résoudre :

$$x^2y' - 2y = xe^(4/x)y^3$$

On tape alors :

```
desolve(x^2*y'-2*y-x*exp(4/x)*y^3,y)
```

On obtient :

```
[((-2*ln(x)+c_0)*exp(4*/x))^(1/-2),
-((-2*ln(x)+c_0)*exp(4/x))^(1/-2))]
```

– Exemples de résolution d'une équation différentielle homogéne du premier ordre qui se résout en posant $y = t * x$.

1. Résoudre :

$$3x^3y' = y(3x^2 - y^2)$$

On tape alors :

```
desolve(3*x^3*y'=((3*x^2-y^2)*y))
```

Ou on tape :

```
desolve(3*x^3*y'=((3*x^2-y^2)*y),y)
```

On obtient :

```
[0,pnt[c_0*exp((3/(2*` t`^2)),`
 t`*c_0*exp((3/(2*` t`^2))]]
```

donc les solutions sont $y = 0$ et la famille de courbes d'équation paramétrique $x = c_0 \exp(3/(2t^2)), y = t * c_0 \exp(3/(2t^2))$ (le paramètre est noté 't' dans la réponse).

2. Résoudre :

$$xy' = y + \sqrt{x^2 + y^2}$$

On tape alors :

```
desolve(x*y'=y+sqrt(x^2+y^2))
```

Ou on tape :

```
desolve(x*y'=y+sqrt(x^2+y^2),y)
```

On obtient :

```
[(-i)*x,(i)*x,pnt[c_0/(sqrt(` t`^2+1)-` t`),(`
      t`*c_0)/(sqrt(` t`^2+1)-` t`)]]
```

donc les solutions sont :

$$y = ix, y = -ix$$

et la famille de courbes d'équation paramétrique

$$x = c_0/(\sqrt{t^2+1} - t), y = t * c_0/(\sqrt{t^2+1} - t)$$

(le paramètre est noté 't' dans la réponse.

– Exemples de résolution d'une équation différentielle du premier ordre ayant un facteur intégrant.

1. Résoudre :

$$yy' + x$$

On tape alors :

```
desolve(y*y'+x)
```

Ou on tape :

```
desolve(y*y'+x,y)
```

On obtient :

```
[sqrt(-2*c_0-x^2),-(sqrt(-2*c_0-x^2))]
```

dans cet exemple, $xdx + ydy$ est une différentielle totale, et le facteur intégrant est 1.

2. Résoudre :

$$2xyy' + x^2 - y^2 + a^2 = 0$$

On tape alors :

```
desolve(2*x*y*y'+x^2-y^2+a^2)
```

Ou on tape :

```
desolve(2*x*y*y'+x^2-y^2+a^2,y)
```

On obtient :

```
[sqrt(a^2-x^2-c_1*x),-(sqrt(a^2-x^2-c_1*x))]
```

dans cet exemple le facteur intégrant est $1/x^2$.

– Exemple de résolution d'une équation différentielle du premier ordre incomplète en x.
Résoudre :

$$(y + y')^4 + y' + 3y = 0$$

Ce genre d'équations n'est pas résoluble directement par `Xcas`, nous allons expliquer ce qu'il faut faire pour l'aider à résoudre de telles équations. On doit trouver pour résoudre $F(y, y') = 0$, une représentation paramétrique de $F(u, v) = 0$, par exemple, $u = f(t), v = g(t)$ puis, poser $y = f(t)$ et $dy/dx = y' = g(t)$.
On a donc $dy/dt = f'(t) = y' * dx/dt = g(t) * dx/dt$.
On trouve alors, comme solution la courbe d'équation paramétrique $x(t), y(t) = f(t)$, où $x(t)$ est obtenu en résolvant l'équation différentielle $g(t)dx = f'(t)dt$.
Ici on peut poser $y + y' = t$ ce qui donne :
$y = -t - 8 * t^4$ et $y' = dy/dx = 3 * t + 8 * t^4$ et donc $dy/dt = -1 - 32 * t^3$ donc
$(3 * t + 8 * t^4) * dx = (-1 - 32 * t^3)dt$.
On tape alors :

```
desolve((3*t+8*t^4)*diff(x(t),t)=(-1-32*t^3),x(t))
```

On obtient :

```
(9*c_0 -11*ln(8*t^3+3)-ln(t^3))/9
```

on a donc comme solution la courbe d'équation :

$$x(t) = -11 * 1/9 * \ln(8 * t^3 + 3) + 1/ - 9 * \ln(t^3) + c_0, \quad y(t) = -t - 8 * t^4$$

– Exemples de résolution d'une équation différentielle de Clairaut du premier ordre, de type $y = x * y' + f(y')$ avec f continument dérivable ; c'est un cas particulier de l'équation différentielle de Lagrange, en hommage au mathḿaticien Alexis Clairaut.
On pose $dy/dx = y' = t$ et on remplace l'équation différentielle $y = x * y' + f(y')$ par le système paramétrique :
$y = x * t + f(t)$ et $dy = tdx$
ce système est équivalent au système obtenu en éliminant dy :
$y = x * t + f(t)$ et $(x + f'(t))dt = 0$
car $dy = tdx + xdt + f'(t)dt = tdx$
On a alors deux types de solutions :

  – celles qui vérifient $y = x * t + f(t)$ et $dt = 0$
  donc $t = m = cste$ et $y = mx + f(m)$
  $y = mx + f(m)$ est l'équation des droites D_m. Ces droites sont appelées les solutions générales de l'équation.
  – celles qui vérifient $y = x * t + f(t)$ et $x + f'(t) = 0$
  c'est à dire la courbe d'équation paramétrique :
  $x = -f'(t)$ et $y = -tf'(t) + f(t)$ cette solution est appelée solution singulière.
  La courbe représentative de cette solution est l'enveloppe de la famille des droites D_m.
  En effet si $y = mx + f(m)$, l'enveloppe de ces droites est l'ensemble des points de coordonnées x, y qui vérifient :
  $y = mx + f(m)$ et $0 = x + f'(m)$ lorsque $m \in \mathbb{R}$
  qui sont les 2 équations qui definissent la solution singulière/
  – Des solutions hybrides peuvent être obtenues par raccordement de ces différentes courbes solutions, d'autant plus simplement qu'il s'agit d'une

famille de droites et de sa courbe enveloppe.

Exemple

1. Résoudre :

$$xy' + y'^3 - y) = 0$$

On tape alors :

```
desolve(x*y'+y'^3-y,y)
```

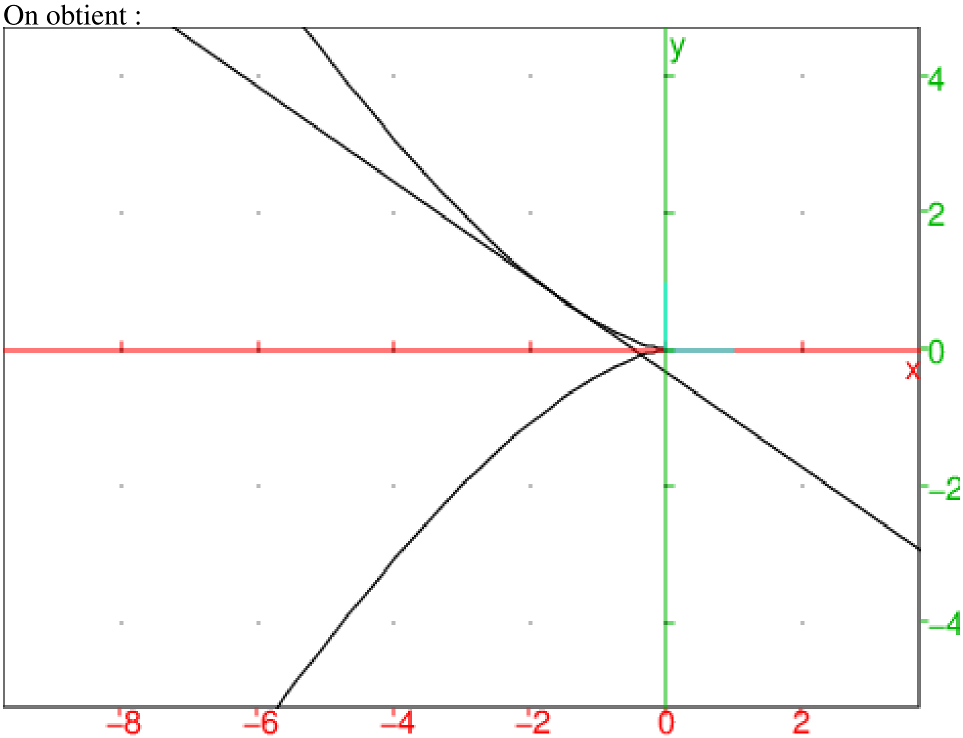
On obtient :

```
[c_0*x+c_0^3,[-3*` t`^2,-` t`*3*` t`^2+`
t`^3]]
```

On tape dans un niveau de géométrie 2d :

```
supposons(m=[-0.7,-5,5,0.1]);
droite(y=m*x+m^3);
plotparam(-3*t^2+i*(-2*t^3))
```

On obtient :



en faisant bouger le curseur m on voit que les droites enveloppe la courbe d'équation paramétrique $x = -3 * t^2, y = -2 * t^3$

2. Résoudre :

$$y - xy' = \sqrt{a^2 + b^2 * y'^2} = 0$$

On tape alors :

```
desolve(y-x*y'-sqrt(a^2+b^2*y'^2),y)
```

On obtient :

```
[c_0*x+sqrt(a^2+b^2*c_0^2), [-b^2*2*`
t`*(sqrt(a^2+b^2*` t`^2))^-1/2,-` t`*b^2*2*`
t`*(sqrt(a^2+b^2*` t`^2))^-1/2+sqrt(a^2+b^2*`
t`^2)]]
```

– Résoudre le système différentiel : $u'(t) = u(t) + 2v(t) + t$
$v'(t) = 2u(t) + v(t) + t + 1$
Puis trouver les solutions de ce sytème qui vérifie $u(0 = 1, v(0) = 2$ On tape :

```
desolve(y'=[[1,2],[2,1]]*y+[x,x+1])
```

On obtient

```
[[(9*exp(-x)+5*exp(3*x)+9*c_0*exp(-x)+9*c_0*exp(3*x)-
      9*c_1*exp(-x)+9*c_1*exp(3*x)-6*x-14)/18,
        (-9*exp(-x)+5*exp(3*x)-9*c_0*exp(-x)+
9*c_0*exp(3*x)+9*c_1*exp(-x)+9*c_1*exp(3*x)-6*x+4)/18]]
```

cela donne les expressions de $u(x)$ et $v(x)$.
Ou on tape :

```
desolve(y'=[[1,2],[2,1]]*y+[t,t+1],t,y)
```

ou on tape :

```
desolve(z'=[[1,2],[2,1]]*z+[t,t+1],t,z)
```

ou on tape :

```
desolve(z'=[[1,2],[2,1]]*z+[t,t+1],z(t))
```

On obtient

```
[[(9*exp(-t)+5*exp(3*t)+9*c_0*exp(-t)+9*c_0*exp(3*t)-
      9*c_1*exp(-t)+9*c_1*exp(3*t)-6*t-14)/18,
        (-9*exp(-t)+5*exp(3*t)-9*c_0*exp(-t)+
9*c_0*exp(3*t)+9*c_1*exp(-t)+9*c_1*exp(3*t)-6*t+4)/18]]
```

cela donne les expressions de $u(t)$ et $v(t)$.
Puis, on tape :

```
desolve([y'=[[1,2],[2,1]]*y+[x,x+1],y(0)=[1,2]])
```

On obtient

```
[[(16*exp(3*x)-3*x-7)/9,(16*exp(3*x)-3*x+2)/9]]
```

cela donne les expressions de $u(x)$ et $v(x)$ tels que $u(0) = 1$ et $v(0) = 2$.
Ou on tape :

```
desolve([z'=[[1,2],[2,1]]*z+[t,t+1],z(0)=[1,2]],t,z)
```

ou on tape :

```
desolve([z'=[[1,2],[2,1]]*z+[t,t+1],z(0)=[1,2]],z(t))
```

On obtient

```
[[(16*exp(3*t)-3*t-7)/9,(16*exp(3*t)-3*t+2)/9]]
```

cela donne les expressions de $u(t)$ et $v(t)$ tels que $u(0) = 1$ et $v(0) = 2$.

6.56.2 Transformée de Laplace et transformée de Laplace inverse : `laplace ilaplace invlaplace`

`laplace` et `ilaplace` (ou `invlaplace`) ont 1, 2 ou 3 arguments :
l'expression que l'on transforme et éventuellement le nom de 2 variables.
L'expression est une expression de la variable courante (ici x) ou l'expression que l'on transforme est une expression de la variable donnèe comme deuxième argument.

`laplace` est la transformée de Laplace de l'expression donnée comme argument et `ilaplace` (ou `invlaplace`) est la transformée de Laplace inverse de l'expression donnée comme argument. Le résultat de `laplace` et `ilaplace` (ou `invlaplace`) est une expression de variable le troisième argument ou par défaut le second argument ou par défaut x.
Attention le second argument est le nom de la variable du premier argument et est ausi le nom de la variable du résultat lorsqu'il n'y a pas de 3-ième argument, par exemple :`laplace(sin(x),t)` renvoie `sin(x)/t`

On utilise la transformée de Laplace (`laplace`) et la transformée de Laplace inverse (`ilaplace` ou `invlaplace`) pour résoudre des équations différentielles linéaires à coefficients constants, par exemple :

$$y\prime\prime + p.y\prime + q.y \ = \ f(x)$$

$$y(0) = a \ y\prime(0) = b$$

En notant $\mathcal{L}$ la transformée de Laplace, on a les relations suivantes :

$$\begin{aligned} \mathcal{L}(y)(x) &= \int_0^{+\infty} e^{-x.u} y(u) du \\ \mathcal{L}^{-1}(g)(x) &= \frac{1}{2i\pi} \int_C e^{z.x} g(z) dz \end{aligned}$$

où C est une courbe fermée contenant les pôles de `g`.
`laplace` :
On tape :

`laplace(sin(x))`

ici on ne précise pas la variable, alors l'expression que l'on transforme L'expression (ici $\sin(x)$) est une expression de la variable courante (ici x) et la transformée sera aussi une fonction de la variable x.
On obtient :

`1/(x^2+1)`

Ou on tape :

`laplace(sin(t),t)`

ici on précise le nom de la variable de la fonction que l'on transforme (ici t) et ce nom de variable sera utilisé pour la transformée de Laplace.
On obtient :

`1/(t^2+1)`

Ou on tape :

`laplace(sin(t),t,s)`

ici on précise le nom de la variable de la fonction que l'on transforme (ici t) et le nom de la variable que l'on désire avoir pour la transformée de Laplace (ici s).
On obtient :

```
1/(s^2+1)
```

`ilaplace` ou `invlaplace` :
On tape :

```
ilaplace(1/(x^2+1))
```

On obtient :

```
sin(x)
```

On tape :

```
ilaplace(1/(t^2+1),t)
```

On obtient :

```
sin(t)
```

On tape :

```
ilaplace(1/(t^2+1),t,x)
```

On obtient :

```
sin(x)
```

On utilise les propriétés suivantes :

$$\begin{aligned}
\mathcal{L}(y')(x) &= -y(0) + x.\mathcal{L}(y)(x) \\
\mathcal{L}(y'')(x) &= -y'(0) + x.\mathcal{L}(y')(x) \\
&= -y'(0) - x.y(0) + x^2.\mathcal{L}(y)(x)
\end{aligned}$$

On a donc si $y\prime\prime(x) + p.y\prime(x) + q.y(x) \ = \ f(x)$:

$$\begin{aligned}
\mathcal{L}(f)(x) &= \mathcal{L}(y'' + p.y' + q.y)(x) \\
&= -y'(0) - x.y(0) + x^2.\mathcal{L}(y)(x) - p.y(0) + p.x.\mathcal{L}(y)(x)) + q.\mathcal{L}(y)(x) \\
&= (x^2 + p.x + q).\mathcal{L}(y)(x) - y'(0) - (x + p).y(0)
\end{aligned}$$

soit, si $a = y(0)$ et $b = y'(0)$:

$$\mathcal{L}aplace(f)(x) = (x^2 + p.x + q).\mathcal{L}aplace(y)(x) - (x + p).a - b$$

La solution est alors :

$$y(x) = \mathcal{L}^{-1}((\mathcal{L}(f)(x) + (x + p).a + b)/(x^2 + p.x + q))$$

Exemple :
Résoudre :

$$y\prime\prime - 6.y\prime + 9.y \ = \ x.e^{3.x}, \quad y(0) = c\_0, \quad y\prime(0) = c\_1$$

Ici, $p = -6, \ q = 9$.
On tape :

```
laplace(x*exp(3*x))
```

On obtient :

```
1/(x^ 2-6*x+9)
```

On tape :

```
ilaplace((1/(x^2-6*x+9)+(x-6)*c_0+c_1)/(x^2-6*x+9))
```

On obtient

```
(216*x^3-3888*x*c_0+1296*x*c_1+1296*c_0)*exp(3*x)/1296
```

après simplification et factorisation (commande `factor`) la solution y s'écrit :

```
(-18*c_0*x+6*c_0+x^3+6*x*c_1)*exp(3*x)/6
```

On peut bien sûr taper directement :

```
desolve(y''-6*y'+9*y=x*exp(3*x),y)
```

On obtient bien :

```
exp(3*x)*(-18*c_0*x+6*c_0+x^3+6*x*c_1)/6
```

6.57 Transformée en z et transformée en z inverse

6.57.1 Transformée en z d'une suite, la fonction $ztrans$: `ztrans`

`ztrans` a un ou trois arguments :

- une suite donnée par son terme général a_x : la variable utilisée pour définir le terme général est x et x sera aussi le nom de la variable utilisée dans la fonction renvoyée par `ztrans`
- une suite donnée par son terme général a_n, le nom de la variable utilisée pour définir ce terme général (ici n) et le nom de la variable utilisée dans la fonction renvoyée par `ztrans` (par exemple z).

`ztrans` calcule la transformée en z de la suite donnée en argument.
On a par définition :
si $f(x) = ztrans(a_x)$ on a

$$f(x) = \sum_{n=0}^{inf} \frac{a_n}{x^n}$$

si $f(z) = ztrans(a_n, n, z)$ on a

$$f(z) = \sum_{n=0}^{inf} \frac{a_n}{z^n}$$

On tape :

```
ztrans(1)
```

On obtient :

```
x/(x-1)
```

On a en effet :
$= \sum_{n=0}^{inf} \frac{1}{x^n} = \frac{1}{1-\frac{1}{x}} = \frac{x}{x-1}$
On tape :

```
ztrans(1,n,z)
```

On obtient :

```
z/(z-1)
```

On a en effet :
$1 + \frac{1}{z} + \frac{1}{z^2} + \frac{1}{z^3} + \frac{1}{z^4} + .. = \sum_{n=0}^{inf} \frac{1}{z^n} = \frac{1}{1-\frac{1}{z}} = \frac{z}{(z-1)}$
On tape :

```
ztrans(x)
```

On obtient :

```
x/(x^2-2*x+1)
```

On tape :

```
ztrans(n,n,z)
```

On obtient :

```
z/(z^2-2*z+1)
```

On a en effet : $\frac{1}{z-1} = \sum_{n=1}^{inf} \frac{1}{z^n}$
$\frac{1}{(z-1)^2} = -(\frac{1}{(z-1)})' = \sum_{n=1}^{inf} \frac{n}{z^{n-1}}$
Donc $\frac{z}{(z-1)^2} = \sum_{n=1}^{inf} \frac{n}{z^n}$

6.57.2 Transformée en z inverse d'une fraction rationnelle, la fonction $invztrans$: `invztrans`

`invztrans` a un ou trois arguments :
- une fraction rationnelle donnée par son expression en utilisant la variable x et x sera aussi le nom de la variable utilisée dans la fonction renvoyée par, `ztrans`,
- trois arguments une fraction rationnelle donnée par son expression, le nom de la variable utilisée pour définir cette expression (paz exemple la variable z), et le nom de la variable utilisée dans la fonction renvoyée par `invztrans` (par exemple n).

`invztrans` calcule la transformée en z inverse de la fraction rationnelle donnée en argument.
On a par définition :
si $invztrans(R_x) = a_x$ on a

$$R_x = \sum_{n=0}^{inf} \frac{a_n}{x^n}$$

si $a_n = invztrans(R_z, z, n)$ on a

$$R_z = \sum_{n=0}^{inf} \frac{a_n}{z^n}$$

On tape :

```
invztrans(x/(x-1))
```

On obtient :

```
1
```

On tape :

```
invztrans(z/(z-1),z,n)
```

On obtient :

```
1
```

On a en effet : $\frac{z}{(z-1)} = \frac{1}{1-\frac{1}{z}} = 1 + \frac{1}{z} + \frac{1}{z^2} + \frac{1}{z^3} + \frac{1}{z^4} + .. = \sum_{n=0}^{inf} \frac{1}{z^n}$ On tape :

```
invztrans(x/(x-1)^2)
```

On obtient :

```
x
```

On tape :

```
invztrans(z/(z-1)^2,z,n)
```

On obtient :

```
n
```

6.58 Autres fonctions

6.58.1 Négliger les petites valeurs : `epsilon2zero`

`epsilon2zero` a comme paramètre une expression de `x`.
`epsilon2zero` renvoie l'expression où les valeurs plus petites que `epsilon` ont été remplacées par zéro dans l'expression non évaluée.
La valeur de `epsilon` peut être changé dans la configuration du `cas` (par défaut `epsilon=1e-10`).
On tape :

```
epsilon2zero(1e-13+x)
```

On obtient (avec `epsilon=1e-10`) :

```
0+x
```

On tape :

`epsilon2zero((1e-13+x)*100000)`

On obtient (avec `epsilon=1e-10`) :

`(0+x)*100000`

On tape :

`epsilon2zero(0.001+x)`

On obtient (avec `epsilon=0.0001`) :

`0.001+x`

6.58.2 Liste des variables : `lname indets`

`lname` (ou `indets`) a comme paramètre une expression.
`lname` (ou `indets`) renvoie un vecteur de composantes le nom des variables symboliques utilisées dans cette expression.
On tape :

`lname(x*y*sin(x))`

On obtient :

`[x,y]`

On tape :

`a:=2;assume(b>0);assume(c=3);`

`lname(a*x^2+b*x+c)`

On obtient :

`[x,b,c]`

6.58.3 Liste des variables et des expressions : `lvar`

`lvar` a comme paramètre une expression.
`lvar` renvoie un vecteur de composantes les noms de variables et des expressions dont cette expression dépend rationnellement.
On tape :

`lvar(x*y*sin(x)^2+ln(x)*cos(y))`

On obtient :

`[x,y,sin(x)]`

On tape :

`lvar(x*y*sin(x)^2)`

On obtient :

`[x,y,sin(x),ln(x),cos(y)]`

On tape :

`lvar(y+x*sqrt(z)+y*sin(x))`

On obtient :

`[y,x,sqrt(z),sin(x)]`

6.58.4 Liste des variables et des expressions algébriques : `algvar`

`algvar` a comme paramètre une expression.
`algvar` renvoie un vecteur de composantes le nom des variables symboliques, par ordre d'extension algebriques, utilisées dans cette expression.
On tape :

```
algvar(y+x*sqrt(z))
```

On obtient :

```
[[y,x],[z]]
```

On tape :

```
algvar(y*sqrt(x)*sqrt(z))
```

On obtient :

```
[[y],[z],[x]]
```

On tape :

```
algvar(y*sqrt(x*z))
```

On obtient :

```
[[y],[x,z]]
```

On tape :

```
algvar(y+x*sqrt(z)+y*sin(x))
```

On obtient :

```
[[x,y,sin(x)],[z]]
```

6.58.5 Test de la présence d'une variable dans une expression : `has`

`has` a comme paramètre une expression et le nom d'une variable.
`has` renvoie `1`, ou `0`, selon que la variable est présente, ou non présente, dans l'expression.
On tape :

```
has(x*y*sin(x),y)
```

On obtient :

```
1
```

On tape :

```
has(x*y*sin(x),z)
```

On obtient :

```
0
```

6.58.6 Évaluation numérique : `evalf`

`evalf` a comme paramètre une expression ou une matrice.
`evalf` renvoie la valeur numérique de l'expression ou de la matrice.
On tape :

```
evalf(sqrt(2))
```

On obtient :

```
1.41421356237
```

On tape :

```
evalf([[1,sqrt(2)],[0,1]])
```

On obtient :

```
[[1.0,1.41421356237],[0.0,1.0]]
```

6.58.7 Approximation rationnelle : `float2rational exact`

`float2rational` (ou `exact`) a comme paramètre une expression numérique réelle.
`float2rational` donne une approximation rationnelle de tous les nombres décimaux r contenus dans l'expression à moins de `epsilon` c'està dire $|r -$ `float2rational`$(r)| < \epsilon$ où ϵ est définit par `epsilon` dans la configuration du `cas` (menu `Cfg` , ou commande `cas_setup`).
On tape :

```
float2rational(1.5)
```

On obtient :

```
3/2
```

On tape :

```
float2rational(1.414)
```

On obtient :

```
707/500
```

On tape :

```
float2rational(0.156381102937*2)
```

On obtient :

```
5144/16447
```

On tape :

```
float2rational(1.41421356237)
```

On obtient :

```
114243/80782
```

On tape :

```
float2rational(1.41421356237^2)
```

On obtient :

```
2
```

Chapitre 7

Les fonctions de statistique

7.1 Les fonctions de Xcas de statistique à 1 variable

On va décrire les différentes fonctions statistiques sur un exemple :
avec la liste `A:=[0,1,2,3,4,5,6,7,8,9,10,11]`
- en prenant comme série statistique d'effectif 1 la liste `A`, ou
- en prenant comme série statistique la liste `A` avec comme effectifs encore la liste `A`.
On tape :
`A:=[0,1,2,3,4,5,6,7,8,9,10,11]`
On pourra se reporter aussi à 6.42 lorsque les arguments sont des listes et à 6.45.37 lorsque les arguments sont des matrices.

7.1.1 La moyenne : `mean moyenne`

`mean` calcule la moyenne numérique des éléments d'une liste (ou de chaque colonne d'une matrice).
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
mean(A)
```

On obtient :

```
11/2
```

En effet (0+1+...+11)=66 et 66/12=11/2
On tape :

```
mean([[1,2],[3,4]])
```

On obtient :

```
[2,3]
```

En effet (1+3)/2=2 et (2+4)/2=3.
`mean` calcule la moyenne numérique des éléments d'une liste (respectivement de chaque colonne d'une matrice) pondérée par une liste (respectivement matrice) de même taille donnée comme deuxième argument.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
mean(A,A)
```

On obtient :

```
23/3
```

En effet : 1*1+2*2+..11*11=23*12*11/6=23*2*11 et 1+2+..11=66 donc :
`mean(A,A)`=23*2*11/66=23/3
On tape :

```
mean([[1,2],[3,4]],[[1,2],[3,4]])
```

On obtient :

```
[5/2,10/3]
```

En effet : (1*1+3*3)/(1+3)=5/2 et (2*2+4*4)/(2+4)=10/3

7.1.2 L'écart-type : `stddev ecart_type`

`stddev` calcule l'écart-type numérique des éléments d'une liste (ou de chaque colonne d'une matrice).
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
stddev(A)
```

On obtient :

```
sqrt(143/12)
```

On tape :

```
stddev([[1,2],[3,4]])
```

On obtient :

```
[1,1]
```

`stddev` calcule l'écart-type numérique des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
stddev(A,A)
```

On obtient :

```
sqrt(65/9)
```

7.1.3 L'écart-type de la population : `ecart_type_population stddevp stdDev`

`stddevp` (ou `stdDev`) a comme argument une (ou deux) liste(s) :
`stddevp(l)` calcule une estimation l'écart-type numérique de la population dont est issu l'échantillon décrit par les éléments de la liste `l`, de longueur `n`, donnée en argument (`size(l)=n` et `n` doit être grand). On a :
`stddevp(l)^2=n/(n-1)* stddev(l)^2`.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
stddevp(A)
```

On obtient :

```
sqrt(13)
```

En effet : `n=size(A)=12` et `12/11*stddev(A)^2=12/11*143/12=13`.
On tape :

```
stddevp([[1,2],[3,4]])
```

On obtient :

```
[sqrt(2),sqrt(2)]
```

`stddevp(l1,l2)` calcule l'écart-type numérique de la population dont est issu l'échantillon décrit par les éléments d'une liste `l1` pondérée par une autre liste `l2` donnée comme deuxième argument.
On a :
`stddevp(l1,l2)^2=n/(n-1)* stddev(l1,l2)^2` si `n` est la taille de l'échantillon c'est à dire si `n` est la somme de la liste `l2` (`sum(l2)=n`).
On tape :

```
stddevp(A,A)
```

On obtient :

```
sqrt(22/3)
```

En effet `sum(A)=66` et $\frac{22}{3} = \frac{66}{65} * \frac{65}{9}$ **Remarque** `stddev` est l'écart type après division par `n` (taille de l'échantillon) alors que `stddevp` et son synonyme `stdDev` (nom de commande TI) est divisé par `n-1` et donne l'estimateur non biaisé de l'écart-type d'une population à partir de l'écart-type calculé avec un échantillon (la division par `n-1` permet de supprimer le biais).
Pour la variance nous ne donnons qu'une commande (division par `n`), mais il est très facile de définir une "variance d'échantillon" en prenant le carré de l'écart-type `stddevp`.

7.1.4 La variance : `variance`

`variance` calcule la variance numérique des éléments d'une liste.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
variance(A)
```

On obtient :

```
143/12
```

`variance` calcule la variance numérique des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
variance(A,A)
```

On obtient :

```
65/9
```

On tape :

```
variance([[1,2],[3,4]])
```

On obtient :

```
[1,1]
```

7.1.5 La médiane : `median`

`median` calcule la médiane des éléments d'une liste.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
median(A)
```

On obtient :

```
5.0
```

`median` calcule la médiane numérique des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
median(A,A)
```

On obtient :

```
8
```

On a en effet : $1 + 2 + 3 + ...7 = 28$ et $9 + 10 + 11 = 30$ il y a donc 28 éléments avant 8 et 30 éléments après 8.

7.1.6 Différentes valeurs statistiques : `quartiles`

`quartiles` renvoie la matrice colonne formée par : le minimum, le premier quartile, la médiane, le troisième quartile et le maximum des éléments d'une liste.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartiles(A)
```

On obtient :

```
[[0.0],[2.0],[5.0],[8.0],[11.0]]
```

On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartiles(A,A)
```

On obtient :

```
[1,6,8,10,11]
```

7.1.7 Le premier quartile : `quartile1`

`quartile1` renvoie le premier quartile des éléments d'une liste.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartile1(A)
```

On obtient le premier quartile de `A` :

```
2.0
```

`quartile1` calcule le premier quartile des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
quartile1(A,A)
```

On obtient le premier quartile de `A` ponderée par `A` :

```
6
```

7.1.8 Le troisième quartile : `quartile3`

`quartile3` renvoie le troisième quartile des éléments d'une liste.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quartile3(A)
```

On obtient le troisième quartile de `A` :

```
8.0
```

`quartile3` calcule le troisième quartile des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
quartile3(A,A)
```

On obtient le premier quartile de `A` ponderée par `A` :

```
10
```

7.1.9 Les déciles : `quantile`

`quantile(L,p)` où `L` est la série statistique et `p` un réel de [0,1[, indique la valeur du caractère à partir de laquelle la fréquence cumulée de `L` atteint ou dépasse `p`.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

```
quantile(A,0.1)
```

On obtient le premier décile :

```
1.0
```

On tape :

```
quantile(A,0.25)
```

On obtient le premier quartile :

```
2.0
```

On tape :

```
quantile(A,0.5)
```

On obtient la médiane :

```
5.0
```

On tape :

```
quantile(A,0.75)
```

On obtient le troisième quartile :

```
8.0
```

On tape :

```
quantile(A,0.9)
```

On obtient le neuvième décile :

```
10.0
```

`quantile(l1,l2,p)` calcule le quantile spécifié par le dernier argument des éléments de la liste `l1` pondérée par la liste `l2`.
On tape :

```
quantile(A,A,0.25)
```

On obtient le premier quartile de la liste `A` pondérée par `A` :

```
6
```

7.1.10 Le regroupement en classes : `classes`

`classes` permet de réaliser un regroupement en classes.
Les paramètres peuvent être :
- un vecteur ou une matrice colonne, le début de la classe et la taille des intervalles de la classe (que l'on suppose de même taille).
On tape :

```
classes([0,0.5,1,1.5,2,2.5,3,3.5,4],0,2)
```

On obtient :

```
[[0.0 .. 2.0,4],[2.0 .. 4.0,4],[4.0 .. 6.0,1]]
```

- un vecteur ou une matrice colonne, le début de la classe, la liste des centres des intervalles de la classe.
On tape :

```
classes([0,0.5,1,1.5,2,2.5,3,3.5,4],0,[1,3,5])
```

On obtient :

```
[[0.0 .. 2.0,4],[2.0 .. 4.0,4],[4.0 .. 6.0,1]]
```

- un vecteur ou une matrice colonne, la liste des intervalles de la classe.
On tape :

```
classes([0,0.5,1,1.5,2,2.5,3,3.5,4],[0..2,2..4,4..6])
```

On obtient :

```
[[0.0 .. 2.0,4],[2.0 .. 4.0,4],[4.0 .. 6.0,1]]
```

7.1.11 Regroupement de termes : `accumulate_head_tail`

`accumulate_head_tail` permet de regrouper les premiers termes et les derniers termes d'une liste en les remplacant par leur somme.
Les paramètres sont la liste, le nombre de termes que l'on regroupe au début de la liste et le nombre de termes l'on regroupe en fin de la liste.
On tape :

```
accumulate_head_tail([0,1,1,0,2,2,3,3,4,3,1,1],4,2)
```

On obtient :

```
[2,2,2,3,3,4,3,2]
```

7.1.12 La boite à moustaches : `boxwhisker moustache`

`boxwhisker` ou `moustache` permet de visualiser différentes valeurs indiquant la répartition des valeurs d'une liste.
On tape :

```
moustache(A)
```

La fenêtre graphique s'ouvre automatiquement et on obtient (à condition d'avoir défini correctement la configuration du graphique avec le menu `Cfg`), une boite rectangulaire dont la longueur est un trait allant du premier quartile Q_1 au troisième quartile Q_3, et sur laquelle un trait vertical indique la valeur de la médiane et d'où deux traits débordent : l'un va de la valeur minimum à Q_1 et l'autre de Q_3 à la valeur maximum. Sur ces deux moustaches on trouvent deux traits verticaux indiquant la valeur du premier et du neuvième décile.

7.1.13 L'histogramme : `histogram histogramme`

`histogram` trace l'histogramme des données de data, on peut préciser une liste d'effectifs, ou un nombre nc de classes ou le mimimum classmin des classes et la largeur classsize des classes.
`histogram` permet de visualiser la fonction densité des fréquences : on met en abscisse les classes et en ordonnée la densité des fréquences (si on a des valeurs discrétes elles sont considérées comme étant le centre de la classe). L'histogramme est donc un graphique en escalier dans lequel les fréquences des différentes classes sont représentées par les aires des différents rectangles situés sous les différents paliers.
On rappelle que, si l'effectif de la classe $[a_{j-1}; a_j]$ est n_j, la fréquence de la classe $[a_{j-1}; a_j]$ est $f_j = n_j/N$ (si N est l'effectif total) et la densité de fréquence de la classe $[a_{j-1}; a_j]$ est $f_j/(a_j - a_{j-1})$.
On tape :

```
histogram([[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]])
```

La fenêtre graphique s'ouvre automatiquement et on obtient l'histogramme de la série `[[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]]`, à condition d'avoir défini correctement la configuration du graphique.

L'argument de `histogram` peut aussi être une liste de valeurs discrètes, dans ce cas, les classes commencent à une valeur (`class_min`) et sont toutes de même largeur (`class_size`) soit définies par défaut (à 0 et 1, valeurs modifiables dans la configuration graphique) ou passées en second et troisième arguments.
On tape :

```
histogram([0,1,2,1,1,2,1,2,3,3])
```

alors `class_min=0` et `class_size=1` et les valeurs 0,1,2,3 ne sont donc pas centrées.
Mais si on tape :

```
histogram([0,1,2,1,1,2,1,2,3,3],-0.5,1)
```

alors `class_min=-0.5` et `class_size=1` et les valeurs 0,1,2,3 sont donc centrées.
et cela renvoie la même chose que :

```
histogram([[0,1],[1,4],[2,3],[3,2]])
```

On tape :

```
histogram(seq(rand(1000),k,0,100),0,100)
```

Ici on a choisi `class_min=0` et `class_size=100`.

7.1.14 Les fréquences : `frequencies frequences`

`frequencies` ou `frequences` a comme argument une liste.
`frequencies` ou `frequences` renvoie les fréquences des éléments de cette liste.
On tape :

```
frequences([1,2,1,1,2,1,2,4,3,3])
```

On obtient :

```
[[1,0.4],[2,0.3],[3,0.2],[4,0.1]]
```

On tape pour simuler le lancé d'une pièce de monnaie :

```
frequences([rand(2)$(k=1..100)])
```

On obtient par exemple :

```
[[0,0.6],[1,0.4]]
```

On tape :

```
frequences([rand(2)$(k=1..1000)])
```

On obtient par exemple :

```
[[0,0.506],[1,0.494]]
```

On tape :

```
frequences([rand(2)$(k=1..10000)])
```

On obtient par exemple :

```
[[0,0.4952],[1,0.5048]]
```

On tape pour simuler le lancé d'un dé cubique non pipé :

```
frequences([(rand(6)+1)$(k=1..100)])
```

On obtient par exemple :

```
[[1,0.13],[2,0.13],[3,0.18],[4,0.13],[5,0.19],[6,0.24]]
```

On tape :

```
frequences([(rand(6)+1)$(k=1..1000)])
```

On obtient par exemple :

```
[[1,0.19],[2,0.155],[3,0.167],[4,0.156],[5,0.183],[6,0.149]]
```

On tape :

```
frequences([(rand(6)+1)$(k=1..10000)])
```

On obtient par exemple :

```
[[1,0.1648],[2,0.1703],[3,0.1667],[4,0.1671],[5,0.1727],[6,0.1584]]]
```

7.1.15 Les fréquences cumulées : `cumulated_frequencies frequences_cumule`

`cumulated_frequencies` ou `frequences_cumulees` a comme argument une liste ou une matrice ayant 2 colonnes ou plus de 2 colonnes.
Lorsque `cumulated_frequencies` a comme argument une liste, `cumulated_frequencies` renvoie le diagramme des fréquences cumulées des valeurs de cette liste.
On tape :

```
cumulated_frequencies([1,2,1,1,2,1,2,4,3,3])
```

On obtient :

```
le diagramme des fréquences cumulées des valeurs de
cette liste i.e.
polygone_ouvert(0,1+i*0.4,2+i*0.7,3+i*0.9,4+i)
```

Lorsque `cumulated_frequencies` a comme argument une matrice ayant 2 colonnes :

– la première colonne représente les différentes valeurs prises, soit sous forme discrétes b_j (les b_j sont équiréparties selon `class_size` défini avec la configuration graphique) soit sous fome d'intervalles $a_{j-1}..a_j$,

– la deuxième colonne représente soit les effectifs n_j, soit les fréquences f_j des valeurs b_j ou des classes $a_{j-1}..a_j$.

Lorsque `cumulated_frequencies` a comme argument une matrice ayant plus de 2colonnes, c'est que l'on veut comparer plusieus effectis correspondant à plusieurs séries de mêmes valeurs.
Les lignes sont alors :
$[a_{j-1}..a_j, n_j]$ ou $[a_{j-1}..a_j, f_j]$ ou $[a_{j-1}..a_j, n1_j, n2_j]$ ou $[a_{j-1}..a_j, f1_j, f2_j]$ ou $[b_j, n_j]$ ou $[b_j, f_j]$ etc...
`cumulated_frequencies` permet de visualiser le diagramme des fréquences cumulées : ce sont les segments de droites qui joignent les points d'abscisse a_j et d'ordonnée $f_1 + .. + f_j$ (a_j est la borne supérieure d'une classe, f_j est la fréquence de la classe $[a_{j-1}; a_j]$ et donc $f_1 + .. + f_j$ est la fréquence cumulée de a_j)
Si on a des valeurs discrétes elles sont considérées comme étant le centre de la classe.
On tape :

```
cumulated_frequencies([[1.5..1.65,50],
[1.65..1.7,20],[1.7..1.8,30]])
```

Ou on tape :

```
cumulated_frequencies([[1.5..1.65,0.5],
[1.65..1.7,0.2],[1.7..1.8,0.3]])
```

La fenêtre graphique s'ouvre automatiquement et on obtient le diagramme cumulatif des fréquences de :
`[[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]]`
à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`).
On tape :

```
cumulated_frequencies([[1.5..1.65,50,30],
[1.65..1.7,20,50],[1.7..1.8,30,20]])
```

On tape :

```
cumulated_frequencies([[1.5..1.65,0.5,0.3],
[1.65..1.7,0.2,0.5],[1.7..1.8,0.3,0.2]])
```

La fenêtre graphique s'ouvre automatiquement et on obtient les diagrammes cumulatifs des fréquences, avec des couleurs différentes, de :
`[[1.5..1.65,50],[1.65..1.7,20],[1.7..1.8,30]]` et de
`[[1.5..1.65,30],[1.65..1.7,50],[1.7..1.8,20]]`
à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`).
Remarque
Pour avoir les fréquences cumulées d'une liste de valeurs, on peut aussi transformer cette liste en une matrice ayant 2 colonnes (valeurs,effectifs) avec la commande `classes` :

On tape :
```
L:=[(rand(6)+1)$(k=1..100)]
CL1:=classes(L)
```
On obtient :
```
[[1.0..2.0,15],[2.0..3.0,22],[3.0..4.0,23],[4.0..5.0,17],
[5.0..6.0,8],[6.0..7.0,15]]
```
Ou on tape :
```
L:=[(rand(6)+1)$(k=1..100)]
CL2:=classes(L,0.5,1)
```
On obtient :
```
[[0.5..1.5,15],[1.5..2.5,22],[2.5..3.5,23],[3.5..4.5,17],
[4.5..5.5,8],[5.5..6.5,15]]
```
On tape :
```
cumulated_frequencies(CL1)
```
On obtient :
```
le graphe des fréquences cumulées de CL1 allant de 1 à 7
```
Ou on tape :
```
cumulated_frequencies(CL2)
```
On obtient :
```
le graphe des fréquences cumulées de CL2 allant de 0.5 à
6.5 et sera donc décalé par rapport au précédent
```

7.1.16 Dessiner un diagramme en batons : `diagramme_batons`

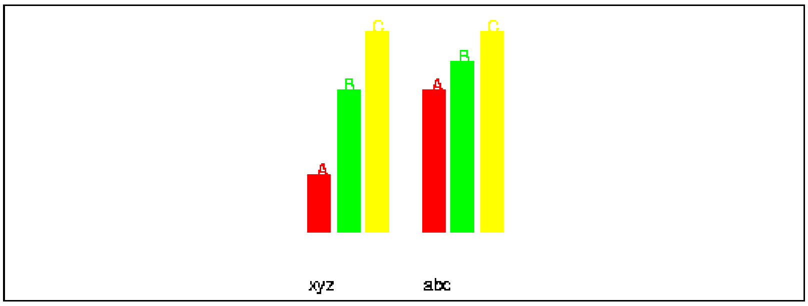
`diagramme_batons` permet de dessiner un diagramme en batons d'une série statistique à 1 variable.
`diagramme_batons` a comme argument une matrice à 2 colonnes contenant en 1ère colonne les noms et en 2ème colonne les valeurs.
On tape :

```
diagramme_batons([["France",6],["Allemagne",12],["Suisse",5]])
```

La fenêtre graphique s'ouvre automatiquement et on obtient le dessin de 3 rectangles de même largeur et de hauteur respective `6,12,5`. On peut faire plusieurs diagrammes en batons sur le même graphique.
On tape :

```
diagramme_batons([[2,"xyz","abc"],["A",2,5],["B",5,6],["C",7,7]])
```

On obtient :



7.1.17 Dessiner un diagramme en camembert : `camembert`

`camembert` permet de dessiner un diagramme en camembert d'une série statistique à 1 variable.
`camembert` a comme argument une matrice à 2 colonnes contenant en 1ère colonne les noms et en 2ème colonne les valeurs.
On tape :

```
camembert([["France",6],["Allemagne",12],["Suisse",5]])
```

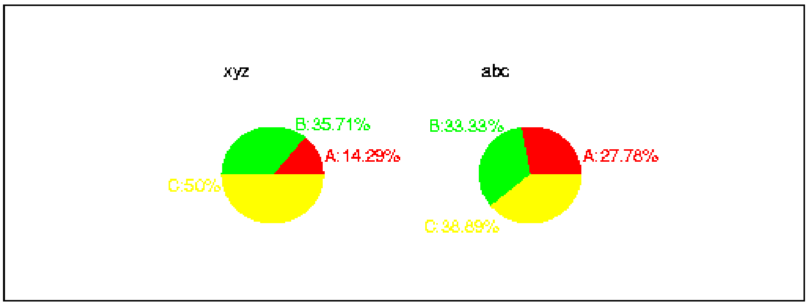
La fenêtre graphique s'ouvre automatiquement et on obtient le dessin d'un camembert coupé selon 3 parts colorées avec comme légende :

```
France 26,09% , Allemagne 52.17% , Suisse 21.74%
```

On peut faire plusieurs camemberts sur le même graphique.
On tape :

```
camembert([[2,"xyz","abc"],["A",2,5],["B",5,6],["C",7,7]])
```

On obtient :



7.2 Les fonctions statistiques à 2 variables

On continue à utiliser la liste `A` dans les exemples.
On tape :

```
A:=[0,1,2,3,4,5,6,7,8,9,10,11]
```

7.2.1 La covariance : `covariance`

La covariance de deux variables aléatoires X et Y est :
$cov(X,Y) = E((X-\bar{X})(Y-\bar{Y}))$.
`covariance` a différentes sortes d'arguments :
- quand les effectifs sont égaux à 1,
`covariance` a pour argument deux listes de même longueur ou une matrice ayant deux colonnes.
`covariance` calcule la covariance numérique des deux listes ou deux colonnes de cette matrice.
On tape :

```
covariance([1,2,3,4],[1,4,9,16])
```

On obtient :

```
25/4
```

On tape :

```
covariance([[1,1],[2,4],[3,9],[4,16]])
```

On obtient :

```
25/4
```

Car on a :
1/4*(1+8+27+64)-75/4=25/4
On tape (on a `A:=[0,1,2,3,4,5,6,7,8,9,10,11]`) :

```
covariance(A,A^2)
```

On obtient :

```
1573/12
```

- quand les effectifs sont différents de 1 :
- si les couples $a[j], b[j]$ ont pour effectif $n[j]$ ($j = 0..p-1$), `covariance` a pour argument trois listes a, b, n de même longueur p, ou une matrice de trois colonnes a, b, n et de p lignes $[a[j], b[j], n[j]]$.
`covariance` calcule la covariance numérique des deux premières listes pondérées par la liste donnée comme dernier argument ou des deux colonnes de cette matrice pondérées par la troisiéme colonne.
On tape :

```
covariance([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

Ou on tape :

```
covariance([[1,1,3],[2,4,1],[3,9,5],[4,16,2]])
```

On obtient :

```
662/121
```

- si les couples $a[j], b[k]$ ont pour effectif $N[j,k]$ ($j = 0..p-1, k = 0..q-1$), `covariance` a pour argument deux listes a, b de longueurs respectives p et q et une matrice N de p lignes et q colonnes ou encore,
afin de pouvoir écrire les données de façon plaisante dans le tableur, `covariance` peut aussi avoir deux arguments, une matrice M et -1.
M est alors un tableau à deux entrées égal à :

$$M = \begin{bmatrix} a \setminus b & b[0] & ... & b[q-1] \\ a[0] & N[0,0] & ... & N[0,q-1] \\ ... & ... & ... & ... \\ a[p-1] & N[p-1,0] & ... & N[p-1,q-1] \end{bmatrix}$$

`covariance(a,b,N)` ou `covariance(M,-1)` calcule la covariance numérique des couples $a[j], b[k]$ pondérés par $N_{j,k}$.
On tape :

```
covariance([1,2,3,4],[1,4,9,16],[[3,0,0,0],
    [0,1,0,0],[0,0,5,0],[0,0,0,2]])
```

On obtient :

```
662/121
```

On tape :

```
covariance([[b\a,1,2,3,4],[1,3,0,0,0],
  [4,0,1,0,0],[9,0,0,5,0],[16,0,0,0,2]],-1)
```

On obtient :

```
662/121
```

7.2.2 La corrélation : `correlation`

Le coefficient de corrélation linéaire de deux variables aléatoires X et Y est $\rho = \dfrac{cov(X,Y)}{\sigma(X)\sigma(Y)}$ où $\sigma(X)$ (resp $\sigma(Y)$) désigne l'écart-type de X (resp$Y$).
`correlation` a les mêmes arguments que `covariance`.

Quand les effectifs sont égaux à 1,
`correlation` a pour argument deux listes de même longueur ou une matrice ayant deux colonnes.
On tape :

```
correlation([1,2,3,4],[1,4,9,16])
```

On obtient :

```
100/(4*sqrt(645))
```

On tape :

```
correlation([[1,1],[2,4],[3,9],[4,16]])
```

On obtient :

```
100/(4*sqrt(645))
```

On tape (on a `A:=[0,1,2,3,4,5,6,7,8,9,10,11]`) :

```
correlation(A,A^2)
```

On obtient :

```
18876/(572*sqrt(1173))
```

Quand les effectifs sont différents de 1 :
- si les couples $a[j], b[j]$ ont pour effectif $n[j]$ ($j = 0..p-1$), `correlation` a pour argument trois listes a, b, n de même longueur p, ou une matrice de trois colonnes a, b, n et de p lignes $[a[j], b[j], n[j]]$.
`correlation` calcule la corrélation numérique des deux premières listes qui sont pondérées par la liste donnée comme dernier argument ou calcule la corrélation numérique des deux colonnes de cette matrice qui sont pondérées par la troisiéme colonne.
On tape :

```
correlation([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

Ou on tape :

```
correlation([[1,1,3],[2,4,1],[3,9,5],[4,16,2]])
```

On obtient :

```
662/(180*sqrt(14))
```

- si les couples $a[j], b[k]$ ont pour effectif $N[j, k]$ ($j = 0..p-1, k = 0..q-1$), `correlation` a pour argument deux listes a, b de longueurs respectives p et q et une matrice N de p lignes et q colonnes ou encore,
afin de pouvoir écrire les données de façon plaisante dans le tableur, `correlation` peut aussi avoir pour argument, une matrice M et -1.
M est alors un tableau à deux entrées égal à :

$$M = \begin{bmatrix} a \setminus b & b[0] & ... & b[q-1] \\ a[0] & N[0,0] & ... & N[0,q-1] \\ ... & ... & ... & ... \\ a[p-1] & N[p-1,0] & ... & N[p-1,q-1] \end{bmatrix}$$

`correlation(a,b,N)` ou `correlation(M,-1)` calcule la corrélation numérique des couples $a[j], b[k]$ pondérés par $N_{j,k}$.
On tape :

```
correlation([1,2,3,4],[1,4,9,16],[[3,0,0,0],[0,1,0,0],
[0,0,5,0],[0,0,0,2]])
```

On obtient :

```
662/(180*sqrt(14))
```

On tape :

```
correlation([["b\a",1,2,3,4],[1,3,0,0,0],
[4,0,1,0,0],[9,0,0,5,0],[16,0,0,0,2]],-1)
```

On obtient :

```
662/(180*sqrt(14))
```

7.2.3 Covariance et corrélation : `covariance_correlation`

`covariance_correlation` a les mêmes arguments que `covariance` : si les effectifs sont égaux à 1, `covariance_correlation` a pour argument deux listes de même longueur ou une matrice ayant deux colonnes représentant deux variables aléatoires X et Y et sinon `covariance_correlation` a pour argument trois listes de même longueur ou une matrice ayant trois colonnes représentant deux variables aléatoires X et Y et la pondération de leurs effectifs ou encore une matrice M et -1, où M donne la pondération de X (la première colonne de M sans $M[0,0]$) et de Y (la première ligne de M sans $M[0,0]$).
`covariance_correlation` renvoie la liste de la covariance $cov(X,Y)$ et du coefficient de corrélation linéaire ρ des deux variables aléatoires X et Y.
On a $\rho = \frac{cov(X,Y)}{\sigma(X)\sigma(Y)}$ où $\sigma(X)$ (resp $\sigma(Y)$) désigne l'écart-type de X (resp$Y$).
On tape :

```
covariance_correlation([[1,1],[2,4],[3,9],[4,16]])
```

On obtient :

```
[25/4,100/(4*sqrt(645))]
```

On tape (on a `A:=[0,1,2,3,4,5,6,7,8,9,10,11]`) :

```
covariance_correlation(A,A^2)
```

On obtient :

```
[1573/12,18876/(572*sqrt(1173))]
```

On tape :

```
covariance_correlation([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

Ou on tape :

```
covariance_correlation([[1,1,3],[2,4,1],[3,9,5],[4,16,2]])
```

On obtient :

```
[662/121,662/(180*sqrt(14))]
```

On tape :

```
covariance_correlation([1,2,3,4],[1,4,9,16],
[[3,0,0,0],[0,1,0,0],[0,0,5,0],[0,0,0,2]])
```

On obtient :

```
[662/121,662/(180*sqrt(14))]
```

On tape :

```
covariance_correlation([["b\a",1,2,3,4],[1,3,0,0,0],
[4,0,1,0,0],[9,0,0,5,0],[16,0,0,0,2]],-1)
```

On obtient :

```
[662/121,662/(180*sqrt(14))]
```

7.2.4 Le nuage de points : `scatterplot nuage_points`

`scatterplot` ou `nuage_points` a pour arguments deux listes ou une matrice ayant deux colonnes.
`scatterplot` ou `nuage_points` permet de visualiser le nuage de points défini par l'argument.
On tape :

```
scatterplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Ou on tape :

```
scatterplot([0,1,2,3,4],[0,1,4,9,16])
```

La fenêtre graphique s'ouvre automatiquement et on obtient le dessin des 5 points ((0,0),...(4,16)), à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`).

7.2.5 Ligne polygonale : `polygonplot ligne_polygonale`

`polygonplot` a pour arguments deux listes ou une matrice ayant deux colonnes.
`polygonplot` permet de visualiser les segments de droites reliant les différents points du nuage de points définis par l'argument et ordonnés selon les abscisses croissantes. Si vous voulez que les points soient reliés dans l'ordre donnés il faut utiliser `listplot`
On tape :

```
polygonplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Ou on tape car les points seront ordonnés selon les abscisses croissantes :

```
polygonplot([[2,4],[0,0],[3,9],[1,1],[4,16]])
```

Ou on tape :

```
polygonplot([0,1,2,3,4],[0,1,4,9,16])
```

La fenêtre graphique s'ouvre automatiquement et on obtient le dessin des 4 segments reliant les 5 points ((0,0),...(4,16)), à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`).

7.2.6 Ligne polygonale : `listplot plotlist`

`listplot` ou `plotlist` a pour argument une liste `l` ou une matrice ayant deux colonnes.
`listplot` ou `plotlist` permet de visualiser les segments reliant le nuage de points ayant pour abscisse `[0,1,2...n]` et pour ordonnée `l` ou pour coordonnées une ligne de la matrice. `listplot` ou `plotlist` relie par des segments de droites, les différents points du nuage, mais sans réordonner les points contrairement à `polygonplot` qui réordonne les points selon leur abscisse puis les relie.
On tape :

```
listplot([0,1,4,9,16])
```

Ou on tape :

```
listplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

La fenêtre graphique s'ouvre automatiquement et on obtient à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`) :

```
le dessin des 5 points ((0,0),(1,1),...(4,16)) reliés
par 4 segments
```

On tape si A est une matrice ayant 5 lignes et 2 colonnes :

```
A:=[[0,0],[1,1],[5,4],[3,9],[4,16]]
```

```
listplot(A[0..4,0..1])
```

La fenêtre graphique s'ouvre automatiquement et on obtient :

```
les 5 points reliés par 4 segments
```

Bien voir la différence entre :

```
listplot([[0,0],[1,1],[5,4],[3,9],[4,16]])
```

```
polygonplot([[0,0],[1,1],[5,4],[3,9],[4,16]])
```

Attention
`listplot([0,1,2,3,4],[0,1,4,9,16])` ou
`listplot([[0,1,2,3,4],[0,1,4,9,16]])` n'est pas valide !

7.2.7 Ligne polygonale et nuage de points : `polygonscatterplot ligne_polygonale_pointee`

`polygonscatterplot` ou `ligne_polygonale_pointee` a pour arguments deux listes ou une matrice ayant deux colonnes.
`polygonscatterplot` ou `ligne_polygonale_pointee` permet de visualiser le nuage de points défini par l'argument, en reliant par des segments de droites, les différents points du nuage en les ordonnant selon les abscisses croissantes.
On tape :

```
polygonscatterplot([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Ou on tape :

```
polygonscatterplot([0,1,2,3,4],[0,1,4,9,16])
```

La fenêtre graphique s'ouvre automatiquement et on obtient le dessin des 5 points ((0,0),...(4,16)) reliés par 4 segments, à condition d'avoir défini correctement la configuration du graphique (menu `Cfg`).

7.2.8 Interpolation linéaire : `linear_interpolate`

É tant donné une matrice à 2 lignes donnant les coordonnées de points : apres avoir ordonné les abcisses de ces points, ces points définissent une ligne polygonale. On veut avoir les coordonnées des points de cette ligne pour des points définis de maniere régulière.
`linear_interpolate` a 4 arguments, une matrice `A` à 2 lignes donnant les coordonnées des points d'une ligne polygonale, la valeur minimum des x (`xmin`), la valeur maximum des x (`xmax`) et le pas (`xstep`).
`linear_interpolate` renvoie les coordonnées des points de la ligne polygonale pour x variant de `xmin` à `xmax` avec un pas ègal à `xstep`.
Remarque on doit avoir `xmin` et `xmax` dans l'intervalle `[min(A[0]);max(A[0])]`.
On tape :

```
linear_interpolate([[1,2,6,9],[3,4,6,12]],1,9,1)
```

On obtient :

```
[[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0],[3.0,4.0,4.5,5.0,5.5,6.0,8.0,1
```

On tape :

```
linear_interpolate([[1,2,6,9],[3,4,6,12]],2,7,1)
```

On obtient :

```
[[2.0,3.0,4.0,5.0,6.0,7.0],[4.0,4.5,5.0,5.5,6.0,8.0]]
```

On tape :

```
linear_interpolate([[1,2,9,6],[3,4,6,12]],1,9,1)
```

On obtient :

```
[[1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0],[3.0,4.0,6.0,8.0,10.0,12.0,10.
```

7.2.9 Régression linéaire : `linear_regression`

Pour approcher les données par la droite des moindres carrés ayant pour équation $y = mx + b$, on utilise `linear_regression` qui renvoie le couple (m, b).
Si les données sont x_i, y_i avec $i = 1..n$, on a :
$m = \frac{cov(X,Y)}{\sigma(X)^2}$
et $b = \bar{Y} - m\bar{X}$
car la somme des carrés des distances $d_i = |y_i - mx_i - b_i|$ est minimale pour ces valeurs et ce minimum (qui est donc l'erreur quadratique moyenne verticale) vaut $(1 - \rho^2)\sigma(Y)^2$ où r est le coefficient de corrélation ($\rho = \frac{cov(X,Y}{\sigma(X)\sigma(Y)}$).
`linear_regression` a les mêmes arguments que `covariance`.
On tape :

```
linear_regression([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Ou on tape :

```
linear_regression([0,1,2,3,4],[0,1,4,9,16])
```

On obtient :

```
4,-2
```

c'est donc la fonction linéaire d'équation $y = 4x - 2$ qui approche au mieux les données.
On tape :

```
X:=[0,1,2,3,4,5,6,7,8,9,10]

Y:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22,

47.31,61.78,80.68,105]

Z:=log(Y)

linear_regression(X,Z)
```

On obtient :

```
0.266729219953,1.98904252589
```

c'est donc la fonction linéaire d'équation $z = \ln(y) = 0.267x + 1.99$ qui approche au mieux les données.

7.2.10 Graphe de la régression linéaire : `linear_regression_plot`

Pour dessiner la droite des moindres carrés $y = mx + b$, droite qui approche au mieux les données, on utilise `linear_regression_plot`.
`linear_regression_plot` a les mêmes arguments que `covariance`.
On tape :

```
linear_regression_plot([1,2,3,4],[1,4,9,16],[3,1,5,2])
```

On obtient :

`Le graphe de la droite d'équation` $y = 331 * x/70 - 22/5$

car c'est la fonction linéaire d'équation $y = 331 * x/70 - 22/5$ qui approche au mieux les données. **Remarque** On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation des données sont écrits en bleu.
Si on veut avoir l'équation et/ou le coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.

7.2.11 Régression exponentielle : `exponential_regression`

Pour approcher les données par une fonction exponnentielle d'équation $y = be^{mx} = ba^x$, on utilise `exponential_regression` qui renvoie le couple (a, b).
`exponential_regression` a les mêmes arguments que `covariance`.
On tape :

```
evalf(exponential_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Ou on tape :

```
evalf(exponential_regression([1,2,3,4],[1,4,9,16]))
```

On obtient :

```
2.49146187923,0.5
```

c'est donc la fonction exponentielle d'équation $y = 0.5 * (2.49146187923)^x$ qui approche au mieux les données.
On tape :

```
X:=[0,1,2,3,4,5,6,7,8,9,10]
Y:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22,47.31,
61.78,80.68,105]
exponential_regression(X,Y)
```

On obtient :

```
1.30568684451,7.30853268031
```

c'est donc la fonction exponentielle d'équation $y = 7.3 * (1.3)^x$ qui approche au mieux les données. On vérifie en tapant :

```
e^[linear_regression(X,ln(Y))]
```

On obtient :

```
1.30568684451,7.30853268031
```

7.2.12 Graphe de la régression exponentielle : `exponential_regression_plot`

Pour dessiner la fonction exponnentielle d'équation $y = b\exp(mx) = ba^x$ qui approche au mieux les données, on utilise `exponential_regression_plot`.
`exponential_regression_plot` a les mêmes arguments que `covariance`.
On tape :

```
exponential_regression_plot([0,1,2,3,4,5,6,7,8,9,10],[7.3,
9.53,12.47,16.3,21.24,27.73,36.22,47.31,61.78,80.68,105])
```

On obtient :

`Le graphe de la fonction expopnentielle d'équation`
$$y = 7.30853268031 * (1.30568684451)^x$$

car c'est la fonction expopnentielle d'équation :
$y = 7.30853268031 * (1.30568684451)^x$ qui approche au mieux les données. **Remarque** On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation de $X, \ln(Y)$ (si les données sont X, Y)) sont écrits en bleu.
Si on veut avoir l'équation et/ou le coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.

7.2.13 Régression logarithmique : `logarithmic_regression`

Pour approcher les données par une fonction logarithmique d'équation $y = m\ln(x)+b$, on utilise `logarithmic_regression` qui renvoie le couple (m, b).
`logarithmic_regression` a les mêmes arguments que `covariance`.
On tape :

```
evalf(logarithmic_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Ou on tape :

```
evalf(logarithmic_regression([1,2,3,4],[1,4,9,16]))
```

On obtient :

```
10.1506450002,-0.564824055818
```

c'est donc la fonction logarithmique d'équation $y = 10.15\ln(x) - 0.565$ qui approche au mieux les données.
On tape :

```
X:=[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]
```

```
Y:=[1.6,2.15,2.65,3.12,3.56,3.99,4.4,4.8,5.18,

5.58,5.92,6.27,6.62,7.06,7.3]
```

```
logarithmic_regression(X,Y)
```

On obtient :

```
2.83870854646,0.843078064152
```

c'est donc la fonction logarithmique d'équation $y = 0.84\ln(x) + 2.84$ qui approche au mieux les données .
On vérifie en tapant :

```
linear_regression(ln(X),Y)
```

On obtient :

```
2.83870854646,0.843078064152
```

et le coefficient de corrélation est :

```
correlation(ln(X),Y)
```

On obtient :

```
0.977939822434
```

On peut aussi taper pour chercher une meilleur approximation :

```
logarithmic_regression(X,log(Y))
```

On obtient :

```
0.732351031846,0.467599676658
```

c'est donc la fonction logarithmique d'équation $z = \ln(y) = 0.73\ln(x) + 0.47$ qui approche au mieux les données.
On vérifie en tapant :

```
linear_regression(ln(X),ln(Y))
```

On obtient :

```
0.732351031846,0.467599676658
```

et le coefficient de corrélation est :

```
correlation(ln(X),ln(Y))
```

On obtient :

```
0.999969474543
```

7.2.14 Graphe de la régression logarithmique : `logarithmic_regression_plot`

Pour dessiner le graphe de la fonction logarithmique d'équation $y = m \ln x + b$ qui approche au mieux les données, on utilise `logarithmic_regression_plot`.
`logarithmic_regression_plot` a les mêmes arguments que `covariance`.
On tape :

```
logarithmic_regression_plot([[1.0,1],[2,4],[3,9],[4,16]])
```

On obtient :

`Le graphe de la fonction logarithme d'équation`
$y = 10.1506450002 \ln(x) - 0.564824055818$

car c'est la fonction logarithme d'équation :
$y = 10.1506450002 \ln(x) - 0.564824055818$
qui approche au mieux les données.
Remarque On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation de $\ln(X), \ln(Y)$ (si les données sont X, Y)) sont écrits en bleu.
Si on veut avoir l'équation et/ou le coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.

7.2.15 Régression polynômiale : `polynomial_regression`

Pour approcher les données par une fonction polynômiale de degré $\leq n$ d'équation $y = a_0 x^n + .. + a_n$, on utilise, en mettant le degré n comme dernier paramètre, `polynomial_regression` qui renvoie la liste $[a_n, ..a_0]$.
`polynomial_regression` a les mêmes premiers arguments que `covariance`, le dernier argument étant le degré du polynôme renvoyé.
On tape :

```
polynomial_regression([[1,1],[2,4],[3,9],[4,16]],3)
```

Ou on tape :

```
polynomial_regression([1,2,3,4],[1,4,9,16],3)
```

On obtient :

```
[0,1,0,0]
```

c'est donc la fonction polynomiale d'équation $y = 0 * x^3 + x^2 + 0 * x + 0 = x^2$ qui approche au mieux les données. **Remarque** On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation des données sont écrits en bleu.
Si on veut avoir l'équation et/ou le coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.

7.2.16 Graphe de la régression polynomiale : `polynomial_regression_plot`

Pour approcher les données par le grahe d'une fonction polynômiale de degré $\leq n$ d'équation $y = a_0 x^n + .. + a_n$, on utilise `polynomial_regression_plot`. `polynomial_regression_plot` a les mêmes premiers arguments que `covariance`, le dernier argument étant le degré du polynôme renvoyé.
On tape :

```
polynomial_regression_plot([[1.0,1],[2,4],[3,9],[4,16]],3)
```

On obtient :

Le graphe de la fonction polynômiale de degré $\leq n$
d'équation $y = x^2$

car c'est la fonction polynômiale d'équation $y = 1 * x^2$ qui approche au mieux les données.

7.2.17 Régression puissance : `power_regression`

Pour approcher les données par une fonction puissance d'équation $y = bx^m$, on utilise `power_regression` qui renvoie le couple (m, b).
`power_regression` a les mêmes arguments que `covariance`.
On tape :

```
evalf(power_regression([[1,1],[2,4],[3,9],[4,16]]))
```

Ou on tape :

```
evalf(power_regression([1,2,3,4],[1,4,9,16]))
```

On obtient :

```
(2.0,1.0)
```

donc $y = x^2$ est la fonction puissance qui approche au mieux les données.
On tape :

```
X:=[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]

Y:=[1.6,2.15,2.65,3.12,3.56,3.99,4.4,4.8,5.18,

5.58,5.92,6.27,6.62,7.06,7.3]

power_regression(X,Y)
```

On obtient :

```
0.732351031846,1.59615829535
```

c'est donc la fonction puissance d'équation $y = 1.6 * x^{0.73}$ qui approche au mieux les données.
On vérifie en tapant :

```
linear_regression(ln(X),ln(Y))
```

On obtient :

```
0.732351031846,0.467599676658
```

On a bien :

```
e^0.467599676658=1.59615829535
```

donc
$\ln(y) = \ln(1.59615829535) + \ln(x) * 0.732351031846$
$\ln(y) = 0.467599676659+\ln(x)*0.732351031846$. et le coefficient de corrélation est :

```
correlation(ln(X),ln(Y))
```

On obtient :

```
0.999969474543
```

7.2.18 Graphe de la régression puissance : `power_regression_plot`

Pour approcher les données par le graphe d'une fonction puissance d'équation $y = bx^m$, on utilise `power_regression_plot`.
`power_regression_plot` a les mêmes arguments que `covariance`.
On tape :

```
power_regression_plot([[1.0,1],[2,4],[3,9],[4,16]])
```

On obtient :

`Le graphe de la fonction puissance d'équation` $y = 1 * x^2$

car c'est la fonction puissance d'équation $y = 1 * x^2$ qui approche au mieux les données. **Remarque** On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation des données sont écrits en bleu.
Si on veut avoir l'équation et/ou le coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.

7.2.19 Régression logistique : `logistic_regression`

Les courbes logistiques sont des courbes dont léquation $y = y(x)$ sont solutions d'une équation différentielle de la forme :
$y'/y = a * y + b$ et $y_0 = y(x_0)$ avec $a < 0$ et $b > 0$.
Les solutions sont de la forme : $y(x) = C/(1 + \exp(-\alpha(x - x_0 - k)$ avec $C = -b/a$, $\alpha = -b$ et $y_0 = (-b/a)/(1 + \exp(-b * k)$ soit
$k = -1/b * (ln(-((a * y0 + b)/(a * y0))))$ Pour vérifier, on peut taper :

```
normal(desolve(y'/y=a*y+b)
```

On obtient :

```
(-b*exp(-(b*c_0-b*x)))/(a*exp(-(b*c_0-b*x))-1)
```

Puis on peut taper :

```
normal(desolve([y'/y=a*y+b,y(x0)=y0],y)
```

On obtient :

```
[(-b*exp(b*x-b*x0+ln(y0/(a*y0+b))))/(a*exp(b*x-b*x0+ln(y0/(a*y0+b)))-1
```

On a donc : $c_0 = x_0 - \ln(y_0/(a*y_0+b))/b$ Donc, en multipliant le numérateur et dénominateur de $y(x)$ par $\exp(b*c\_0 - b*x)$ on a :
$y(x) = (-b/(\exp(b*c\_0 - b*x)*a*exp(-(b*c\_0 - b*x)) - 1)$ soit $y(x) = -b/(a - \exp(b*(x-c_0))) = (-b/(a*(1-\exp(b*(x-c_0))/a))$
On a $1/a = -\exp(-ln(-a))$ car $a < 0$ donc $y(x) = (-b/a)*(1/(1+\exp(b*(x-c_0)-\ln(-a)))$ qui est bien la forme annoncée.

Lorsque on connait les valeurs de f' en $x = x_0, x_0+1....x_0+n$, on cherche une fonction logistique $y(x)$ tel que $y'(x)$ approche au mieux les différentes valeurs de $f'(x)$.

`logistic_regression` a comme paramètres :

- une liste `L` qui contient les valeurs de y' pour $x = x_0, x_0+1....x_0+n$,
- la valeur de `x0` de x_0
- la valeur `y0` de $y(x_0)$ lorsqu'on la connait sinon `Xcas` arrive à l'estimer...

`logistic_regression(L,x0,y0` renvoie les fonctions `y(x)` et `y'(x)`, la constante `C`, `y1M` et `xM` avec `y1M` est la valeur `y'(xM)` qui est le maximum de y' obtenu en $x = xM$, et enfin le coefficient de correlation linéaire `R` de $Y = y'/y$ en fonction de y avec la droite $Y = a*y+b$.
À partir de la liste `L`, `Xcas` calcule la liste `Ly` en utilisant la formule $y(t+1) - y(t) = y'(t)$, donc, on a `Ly=[y0,y0+y0',y0+y0'+y1',....]`. Puis `Xcas` fait une régression linéaire de `L/Ly` en fonction de `Ly` pour avoir les valeurs de a et b ($y'/y = a*y+b$ et $y_0 = y(x_0)$) puis touve la solution de cette équation différentielle On tape :

```
logistic_regression([0.0,1.0,2.0,3.0,4.0],0,1)
```

On obtient avec écrit en bleu la signification des valeurs renvoyées :

```
[(-17.77)/(1+exp(-0.496893925384*x+2.82232341488+3.14159265359*i)),
(-2.48542227469)/(1+cosh(-0.496893925384*x+2.82232341488+3.14159265359
 -17.77,-1.24271113735,5.67993141131+6.32246138079*i,
                    0.307024935856]
```

On tape :

```
evalf(logistic_regression([1,2,4,6,8,7,5],0,2))
```

Ou on tape :

```
logistic_regression(evalf([1,2,4,6,8,7,5]),0,2.0))
```

On obtient :

```
[64.8358166583/(1.0+exp(-0.551746244591*x+2.95837880348)),
14.4915280084/(1.0+cosh(-0.551746244591*x+2.95837880348)),
64.8358166583,7.24576400418,5.36184674112,-0.81176431297]
```

Pour retouver la valeur -0.81176431297 du coefficient de corrélation, on tape :
`L:=[1,2,4,6,8,7,5];`
`y0:=2.0;`
`Ly:=makelist(y0,1,size(L))+cumSum(L)`
On obtient : `[3,5,9,15,23,30,35]`
puis
`correlation(L/Ly,Ly)` qui renvoie
`-0.81176431297`

7.2.20 Graphe de la régression logistique : `logistic_regression_plot`

Lorsque on connait les valeurs de $f'(x)$ en $x = x_0, x_0+1....x_0+n$, pour tracer le graphe de la derivée $y'(x)$ d'une fonction logistique $y(x)$, solution de l'équation $y'/y = a*y+b$ ($a < 0$ et $b > 0$) vérifiant $y_0 = y(x_0)$ et tel que $y'(x)$ approche au mieux les différentes valeurs de $f'(x)$, on utilise `logistic_regression_plot`.
`logistic_regression_plot` a les mêmes arguments que `logistic_regression`.
On tape :

```
logistic_regression_plot([1,2,4,6,8,7,5],0,2.0)
```

On obtient avec écrit en bleu les valeurs renvoyées par :

Le graphe de la fonction solution de $y'/y = a*y+b$ avec
$y'(0) = 1, y'(1) = 2, y'(3) = 4...y'(6) = 5, \quad y(0) = 2$

car c'est la fonction logistique qui approche au mieux les données. **Remarque** On remarquera que l'équation de la courbe representée ainsi que la valeur du coefficient de corrélation des données sont écrits en bleu.
Si on veut avoir l'équation et/ou le carré du coefficient de corrélation sur le dessin il faut rajouter comme dernier argument, l'option `equation` et/ou `correlation`.
Par exemple, `logistic_regression_plot([1,2,4,6,8,7,5],0,2.0,correlation)` et `R2=0.658961299812` s'inscrit sur le graphe.

7.3 Les fonctions aléatoires de `Xcas`

7.3.1 Pour initialiser les nombres aléatoires : `srand randseed RandSeed`

`srand` (ou `randseed` ou `RandSeed`) sert à initialiser la suite des nommbres aléatoires que l'on obtient avec `rand()` ou avec `randnorm()`.
`RandSeed` a toujours un argument entier, alors que `randseed` ou `srand` peut ne pas avoir d'arguments (dans ce cas le générateur aléatoire est intialisé avec l'horloge du système).
Ainsi, `srand(n)` (ou `randseed(n)` ou `RandSeed(n)`) avec `n` entier sert à initialiser la suite des nommbres aléatoires pour que l'on puisse obtenir la même

suite aléatoire avec `rand()` ou avec `randnorm()`.
On tape :

```
srand
```

On obtient par exemple :

```
1054990506
```

On tape :

```
srand(121054990506)
```

On obtient :

```
121054990506
```

Ou on tape :

```
RandSeed(10549905061234)
```

On obtient par exemple :

```
10549905061234
```

7.3.2 Tirage équiréparti `rand alea hasard`

Tirage équiréparti sur $[0,1[$ **:** `rand() alea() hasard`

`rand()` renvoie au hasard, de façon équiprobable, un nombre réel de $[0,1[$.
On tape :

```
rand()
```

ou on tape

```
alea()
```

ou on tape (attention `hasard` n'utilise pas de `()` et est utilisé pour le langage de la tortue)

```
hasard
```

On obtient par exemple :

```
0.912569261115
```

Pour avoir, au hasard, de façon équiprobable, un nombre de $[0;1[$, on peut aussi utiliser (voir le paragraphe suivant) :

```
rand(0,1)
```

On obtient :

```
0.391549611697
```

Tirage aléatoire équiréparti sur l'intervalle [a ;b[: `rand(a,b)` `hasard(a,b)` `rand(a..b)()` `hasard(a..b)()`

Si `a` et `b` sont des réels `rand(a,b)` désigne un nombre décimal aléatoire compris dans l'intervalle $[a;b[$.
Donc, `rand(a,b)` ou (`hasard(a,b)`) renvoie au hasard, et de façon équiprobable, un nombre décimal de $[a;b[$.
Pour avoir, au hasard et de façon équiprobable, un nombre décimal de $[0;1[$, on tape :

```
rand(0,1)
```

On obtient :

```
0.391549611697
```

ou on tape (attention `hasard` utilise des `()` qui englobent hasard et sesparamètres et `hasard` est utilisé pour le langage de la tortue)

```
(hasard 0,1)
```

On obtient par exemple :

```
0.912569261115
```

Pour avoir, au hasard et de façon équiprobable, un nombre décimal de $[0;0.5[$, on tape :

```
rand(0,0.5)
```

On obtient :

```
0.303484437987
```

Pour avoir, au hasard et de façon équiprobable, un nombre décimal de $]-0.5;0]$, on tape :

```
rand(0,-0.5)
```

ou on tape :

```
rand(-0.5,0)
```

On obtient par exemple :

```
-0.20047219703
```

Si `a` et `b` sont des réels `rand(a..b)` ou `alea(a..b)` ou `hasard(a..b)` désigne une fonction qui est un générateur de nombres aléatoires compris dans l'intervalle $[a;b[$.
Donc, `rand(a..b)()` renvoie au hasard, et de façon équiprobable, un nombre décimal de $[a;b[$.
Pour avoir, au hasard et de façon équiprobable, un nombre décimal de $[0;1[$, on tape :

```
rand(0..1)()
```

On obtient :

```
0.391549611697
```

Pour avoir, au hasard et de façon équiprobable, plusieurs nombres aléatoires décimaux compris dans l'intervalle $[1; 2[$, on tape :

```
r:=rand(1..2)
```

puis il suffit de taper `r()`.
On tape :

```
r()
```

On obtient :

```
1.14160255529
```

Tirage aléatoire d'entiers équirépartis sur $[0, .., n[$ **:** `rand(n) alea(n) hasard(n)`

Si `n` est un entier relatif `rand(n)` ou `hasard(n)` renvoie au hasard, et de façon équiprobable, un entier de $[0, 1, .., n[$ (ou de $]n, ..1, 0]$ si n est negatif).
On tape :

```
rand(2)
```

Ou on tape :

```
alea(2)
```

Ou on tape :

```
hasard(2)
```

ou

```
hasard 2
```

On obtient :

```
1
```

ou on obtient :

```
0
```

On tape :

```
rand(-2)
```

Ou on tape :

```
hasard(-2)
```

On obtient :

```
-1
```

ou on obtient :

```
0
```

On tape pour avoir un entier aléatoire entre 6 et 10, bornes comprises :

```
6+rand(11-6)
```

Ou on tape :

```
6+hasard(11-6)
```

On obtient par exemple :

```
8
```

7.3.3 Tirage aléatoire sans remise de p objets parmi n : `rand alea hasard`

`rand` a dans ce cas, soit 2, soit 3 arguments.
Si `rand` a 2 arguments : les arguments sont un entier p et une liste L alors `rand(p,L)` renvoie, au hasard, `p` éléments de la liste `L`.
Si `rand` a 3 arguments : les arguments sont trois entiers `p,min,max` alors `rand(p,min,max)` renvoie, au hasard, `p` entiers de `[min,..,max]` On tape :

```
rand(3,["r","r","r","r","v","v","v"])
```

On obtient :

```
["r","r","v"]
```

On tape :

```
rand(2,1,10)
```

On obtient :

```
[3,7]
```

On tape :

```
rand(2,4,10)
```

On obtient :

```
[5,7]
```

7.3.4 Tirage selon une loi binomiale : `randbinomial`

`randbinomial(n,p)` renvoie au hasard des nombres entiers répartis selon la loi binomiale $B(n,p)$ de moyenne `n*p` et d'écart type `sqrt(n*p*(1-p))`.
`randbinomial(n,p)` a pour valeur le nombre de succès dans une suite de `n` tirages indépendants lorsque pour chaque tirage, un succès est de probabilité `p`.
On tape :

```
randbinomial(10,0.6)
```

On obtient par exemple :

```
4
```

ou on obtient par exemple :

```
8
```

On tape :

```
randbinomial(100,0.4)
```

On obtient par exemple :

```
43
```

7.3.5 Tirage selon une loi multinomiale : `randmultinomial`

`randmultinomial(P)` renvoie un index ou `randmultinomial(P,K)` renvoie un élément de la liste `K`, aléatoirement distribué selon la loi multinomiale de probabilités donnée par la liste `P`.
On tape :

```
randmultinomial([1/2,1/3,1/6]
```

On obtient par exemple :

```
1
```

Ce qui correspond à l'indice de l'objet obtenu avec la probabilité `1/3`.
On tape :

```
randmultinomial([1/2,1/3,1/6],["R","V","B"])
```

On obtient par exemple :

```
"R"
```

7.3.6 Tirage selon une loi de Poisson : randpoisson

randpoisson(λ) renvoie au hasard des nombres entiers répartisselon la loi de Poisson $P(\lambda)$ de moyenne λ et d'écart type $\sqrt{\lambda}$.
On a :
Proba($X \leq k$)=exp$(-\lambda)\frac{\lambda^k}{k!}$.
On tape :

```
randpoisson(10.6)
```

On obtient par exemple :

```
8
```

ou on obtient par exemple :

```
17
```

On tape :

```
randpoisson(2.4)
```

On obtient par exemple :

```
1
```

7.3.7 Tirage selon une loi normale : randnorm randNorm

randnorm(m,sigma) ou randNorm(m,sigma) renvoie au hasard des nombres répartis selon la loi normale de moyenne m et d'écart type sigma.
On tape :

```
randnorm(0,1)
```

On obtient par exemple :

```
0.549605372982
```

ou on obtient par exemple :

```
-0.58946494465
```

On tape :

```
randnorm(2,1)
```

On obtient par exemple :

```
2.54178274488
```

7.3.8 Tirage selon une loi exponentielle : `randexp`

`randexp(a)` renvoie au hasard des nombres répartis selon la loi exponentielle de paramètre `a` positif.
La densité de probabilité est proportionnelle à $\exp(-a*t)$ et on a :
Proba($X \leq t$)=$a \int_0^t \exp(-a*u)du$.
On tape :

```
randexp(1)
```

On obtient par exemple :

```
0.310153677284
```

ou on obtient par exemple :

```
0.776007926195
```

7.3.9 Matrice aléatoire : `ranm randmatrix randMat`

`ranm` (ou `randmatrix` ou `randMat`) (voir aussi 6.26.32 et 6.44.3) peut avoir comme argument :

- un entier `s`, dans ce cas `ranm` renvoie une liste de longueur `s` dont les éléments sont des entiers pris au hasard de façon équiprobable dans :
  `[-99,-98,...,98,99]`.
  On tape :
  ```
  ranm(5)
  ```
  On obtient par exemple :
  ```
  [-40,27,4,-1,94]
  ```
- deux entiers `n,p`, dans ce cas `ranm` renvoie une matrice de `n` lignes et `p` colonnes dont les éléments sont des entiers pris au hasard de façon équiprobable dans `[-99,-98,...,98,99]`.
  On tape :
  ```
  ranm(2,3)
  ```
  On obtient par exemple :
  ```
  [[-32,53,-44],[10,-4,25]]
  ```
- deux entiers `n,p` et un entier relatif `a`, dans ce cas `ranm` renvoie une matrice de `n` lignes et `p` colonnes dont les éléments sont des entiers pris au hasard de façon équiprobable dans `[0;a[` (ou `]a;0]` si `a` est négatif)
  On tape :
  ```
  ranm(2,3,10)
  ```
  On obtient par exemple :
  ```
  [[8,3,7],[7,9,1]]
  ```
- deux entiers `n,p` et un intervalle `a..b`, dans ce cas `ranm` renvoie une matrice de `n` lignes et `p` colonnes dont les éléments sont des réels pris au hasard de façon équiprobable dans `[a;b[`.
  On tape :
  ```
  ranm(2,3,0..1)
  ```
  On obtient par exemple :
  ```
  [[0.840187716763,0.394382926635,0.783099223394],
  ```

`[0.798440033104,0.911647357512,0.197551369201]]`

– deux entiers `n,p` et une fonction aléatoire de `Xcas` qu'il faut quoter, dans ce cas `ranm` renvoie une matrice de `n` lignes et `p` colonnes dont les éléments sont pris au hasard selon la fonction donnée en troisième argument.
  Les fonctions données en troisième argument qui doivent être quoter, peuvent être :
  `'rand(n)'`
  `'binomial(n,p)'` ou `binomial,n,p` ou `'randbinomial(n,p)'`
  `'multinomial(P,K)'` ou `multinomial,P,K` ou `'randmultinomial(P,K)'`
  `'poisson(`λ`)'` ou `poisson,` λ ou `'randpoisson(`λ`)'`
  `'normald(`μ`,`σ`)'` ou `normald,`μ`,`σ ou `'randnorm(`μ`,`σ`)'`
  `'exp(a)'` ou `exp,a` ou `'randexp(a)'`
  `'fisher(n,m)'` ou `fisher,n,m` ou `'randfisher(n,m)'`
  On tape :

```
ranm(3,2,'rand(3)')
```

  ou

```
ranm(3,2,3)
```

  On obtient par exemple :

```
[[2,1],[0,0],[1,0]]
```

  On tape :

```
ranm(1,2,'randnorm(0,1)')
```

  On obtient par exemple :

```
[[1.37439065645,-1.33195982697]]
```

7.4 Densité, fonction de répartition et leur inverse

7.4.1 Probabilité que X égale k lorsque $X \in \mathcal{B}(n,p)$: `binomial`

La probabilité pour que X égale k lorsque $X \in \mathcal{B}(n,p)$ (k ($0 \leq k \leq n$) est nombre de succés lors de n tirages indépendants (i.e.avec remise) de probabilité p) est égale à :
`binomial(n,k,p)=comb(n,k)*p^k*(1-p)^(n-k).`
Le troisième paramètre est égal par défaut à 1, on a donc dans ce cas :
`binomial(n,k)=comb(n,k).`
On tape :

```
binomial(10,2,0.4)
```

On obtient :

```
0.120932352
```

On tape :

```
binomial(4,2,0.5)
```

On obtient :

```
0.375
```

7.4.2 Fonction de répartition de la loi binomiale : `binomial_cdf`

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n, p)$, on a :
`binomial_cdf(n,p,x)`=$Proba(X \leq x)$=
`binomial(n,0,p)+...+binomial(n,floor(x),p)`.
`binomial_cdf(n,p,x,y)`=$Proba(x \leq X \leq y)$=
`binomial(n,ceil(x),p)+...+binomial(n,floor(y),p)`.
On tape :

```
binomial_cdf(4,0.5,2)
```

On obtient :

```
0.6875
```

On peut vérifier que :

```
binomial(4,0,0.5)+binomial(4,1,0.5)+binomial(4,2,0.5)
```

```
=0.6875
```

On tape :

```
binomial_cdf(2,0.3,1)
```

On obtient :

```
0.91
```

On tape :

```
binomial_cdf(2,0.3,1,2)
```

On obtient :

```
0.51
```

7.4.3 Fonction de répartition inverse de la loi binomiale : `binomial_icdf`

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n, p)$, si on a :
`binomial_icdf(n,p,x)=h` c'est que
$Proba(X \leq h) = x$=`binomial_cdf(n,p,h)`.
On tape :

```
binomial_icdf(4,0.5,0.9)
```

On obtient :

```
3
```

On tape :

```
binomial_icdf(2,0.3,0.95)
```

On obtient :

```
2
```

7.4.4 Probabilité que X égale k lorsque $X \in \mathcal{N}eg\mathcal{B}in(n,p)$: `negbinomial`

La loi binomiale négative est une distribution de probabilité discrète. Elle dépend de 2 paramètres : un entier n (le nombre de succès attendus) et un réel p de $]0,1[$ (la probabilité d'un succés).
On la note $\mathcal{N}eg\mathcal{B}in(n,p)$.
Elle permet de décrire la situation suivante : on fait une suite de tirages indépendants (avec pour chaque tirage, la probabilité p d'avoir un succès) jusqu'à obtenir n succès. La variable aléatoire représentant le nombre d'échecs qu'il a fallut avant d'avoir n succès, suit alors une loi binomiale négative.
Cette loi est aussi connue sous le nom de loi de Pascal en l'honneur de Blaise Pascal et de loi de Polya, en l'honneur de George Pólya.
`negbinomial(n,p,k)`=comb$(n+k-1,k)*p^n*(1-p)^k$ pour $k=0,1,2..$ La loi se généralise pour deux paramètres r et p, où r peut prendre des valeurs réelles strictement positives. On a alors :
`negbinomial(r,p)`=$\dfrac{\Gamma(r+k)}{k!\,\Gamma(r)}\,p^r\,q^k$

Remarque
Si on définit `comb(n,k)` pour $n<0$ par `comb(n,k)=n*(n-1)*..*(n-k-1)/k!`,
alors Si $X \in \mathcal{N}eg\mathcal{B}in(n,p)$ ($n \in \mathbb{N}$ et $p \in [0;1]$) alors $\mathtt{Proba(X=k)} = \mathtt{p^n}*(\mathtt{p}-1)^{\mathtt{k}} * \mathtt{comb}(-\mathtt{n},\mathtt{k})$ ce qui justifie le nom de loi binomiale négative et qui facilite le calcul de l'espérance (égale à $n(1-p)/p$) et de la variance (égale à $n(1-p)/p^2$).
On tape :

```
negbinomial(10,12,0.4)
```

On obtient :

```
0.0670901607617
```

On tape :

```
negbinomial(4,2,0.5)
```

On obtient :

```
0.15625
```

On tape :

```
negbinomial(1,2,0.4)
```

On obtient $0.6^2 * 0.4$ soit :

```
0.144
```

7.4.5 Fonction de répartition de la loi binomiale négative : `negbinomial_cdf`

Lorsqu'une variable aléatoire X suit une loi binomiale négative $\mathcal{N}eg\mathcal{B}in(n,p)$, on a :
`negbinomial_cdf(n,p,x)`=$Proba(X \leq x)$=
`negbinomial(n,0,p)+...+negbinomial(n,floor(x),p)`.
`negbinomial_cdf(n,p,x,y)`=$Proba(x \leq X \leq y)$=
`negbinomial(n,ceil(x),p)+...+negbinomial(n,floor(y),p)`.
On tape :

`negbinomial_cdf(4,0.5,2)`

On obtient :

`0.34375`

On peut vérifier que :

`negbinomial(4,0,0.5)+negbinomial(4,1,0.5)+negbinomial(4,2,0.5)`

`0.34375`

On tape :

`negbinomial_cdf(2,0.3,1)`

On obtient :

`0.216`

On tape :

`negbinomial_cdf(2,0.3,3)`

On obtient :

`0.47178`

7.4.6 Fonction de répartition inverse de la loi binomiale négative : `negbinomial_icdf`

Lorsqu'une variable aléatoire X suit une loi binomiale négative $\mathcal{N}eg\mathcal{B}in(n,p)$, lorsqu'on a `negbinomial_icdf(n,p,x)=h` c'est que
$Proba(X \leq h) = x$=`negbinomial_cdf(n,p,h)`.
On tape :

`negbinomial_icdf(4,0.5,0.9)`

On obtient :

`8`

On tape :

`negbinomial_icdf(2,0.3,0.95)`

On obtient :

`12`

Les programmes naifs pour calculer les fonctiond précédentes

```
negbino(n,p,k):=comb(-n,k)*p^n*(p-1)^k;
negbin(n,p,k):=comb(n+k-1,k)*p^n*(1-p)^k;
negbi(n,p,k):=(n+k-1)!/k!/(n-1)!*p^n*(1-p)^k;
negbino_cdf(n,p,x):=sum(negbi(n,p,k),k=0..floor(x));
negbino_icdf(n,p,x):={
 local k:=0;
 tantque negbino_cd(n,p,k)<x faire
   k:=k+1;
   ftantque;
   return k;
   }:;
```

7.4.7 Probabilité que X égale $[k_0, k_1..k_j] + K$ lorsque X suit une loi multinomiale de probabilité $[p_0, p_1, ..p_j] = P$: `multinomial`

La probabilité pour que X égale $K = [k_0, k_1..k_j]$ (avec $k_0 + .. + k_j = n$) lorsque X suit une loi multinomiale de probabilité $P = [p_0, p_1, ..p_j]$ (avec $p_0 + .. + p_j = 1$) est :

$$\texttt{multinomial(n,P,K)} = \frac{n!}{(k_0!k_1!..k_j!)} * (p_0^{k_0} p_1^{k_1} ..p_j^{k_j})$$

Remarque `multinomial(n,P,K)` avec $P = [p_0, p_1, ..p_j]$ et $K = [k_0, k_1..k_j]$ renvoie $\frac{n!}{(k_0!k_1!..k_j!)} * (p_0^{k_0} p_1^{k_1} ..p_j^{k_j})$ même si $p_0 + .. + p_j \neq 1$ mais renvoie une erreur si $k_0 + .. + k_j \neq n$.

On tape :

`multinomial(10,[0.2,0.3,0.5],[3,2,5])`

On obtient :

`0.0567`

On vérifie :
$10!/(3! * 2! * 5!) * 0.2^3 * 0.3^2 * 0.5^5 = 0.0567$.
Donc la probabilité d'obtenir [3,2,5] (3 fois l'objet ayant la probabilité 0.2 d'être tiré, 2 fois l'objet ayant la probabilité 0.3 d'être tiré et 5 fois l'objet ayant la probabilité 0.5 d'être tiré) lors de 10 tirages d'un objet parmi ces 3 objets est 0.0567.
On tape :

`multinomial(4,[0.5,0.5],[2,2])`

Ou on tape :

`binomial(4,2,0.5)`

On obtient :

`0.375`

7.4.8 Probabilité pour que X égale k lorsque $X \in \mathcal{P}(\mu)$: poisson

La densité de probabilité de la loi de Poisson de paramètre mu, c'est à dire de moyenne mu et d'écart-type mu est :
poisson(mu,k)= exp(-mu)*mu^k/k!=$Proba(X = k)$ avec $X \in \mathcal{P}(mu)$.
On tape :

```
poisson(10.0,9)
```

On obtient :

```
0.125110035721
```

7.4.9 Fonction de répartition de Poisson : poisson_cdf

Lorsqu'une variable aléatoire X suit une loi de Poisson de paramètre mu, c'est à dire de moyenne mu et d'écart-type mu, on a :
$Proba(X \leq x)$=poisson_cdf(mu,x) avec $X \in \mathcal{P}(mu)$. et
$Proba(x1 \leq X \leq x2)$=poisson_cdf(mu,x1,x2)=poisson_cdf(mu,x1)+...+poisson_
c'est à dire :
poisson_cdf(mu,x1,x2)=poisson_cdf(mu,x2)-poisson_cdf(mu,x1-1)
poisson_cdf(mu,x) est la fonction de répartition de la loi de Poisson de paramètre mu, On tape :

```
poisson_cdf(10.0,3)
```

On obtient :

```
0.0103360506759
```

On tape :

```
poisson_cdf(10.0,3,10)
```

On obtient :

```
0.572703699517
```

7.4.10 Fonction de répartition inverse de Poisson : poisson_icdf

Lorsqu'une variable aléatoire X suit une loi de Poisson de paramètre mu, c'est à dire de moyenne mu et d'écart-type mu, on a :
poisson_icdf(mu,t) = h équivaut à
$Proba(X \leq h)$ =t= poisson_cdf(mu,h) avec $X \in \mathcal{P}(mu)$.
poisson_icdf(mu,t) est l'inverse de la fonction de répartition de la loi de Poisson de paramètre mu, On tape :

```
poisson_icdf(10.0,0.975)
```

On obtient :

```
0.125110035721
```

7.4.11 Densité de probabilité de la loi normale : `loi_normale` `normald`

- `normald(x)` ou `loi_ normale(x)` est la densité de probabilité de la loi normale centrée réduite (de moyenne 0 et d'écart-type 1).
  `normald(x)` ou `loi_ normale(x)` est égale à :

$$\frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}}$$

- `normald(`μ`,`σ`,x)` est la densité de probabilité de la loi normale de moyenne μ et d'écart-type σ,
  `normald(`μ`,`σ`,x)`, est égale à $\dfrac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{(x-\mu)}{\sigma})^2}$

On tape :

```
normald(1)
```

On obtient :

```
exp(-1/2)/sqrt(2*pi)
```

On tape :

```
normald(2,1,3)
```

On obtient :

```
exp(-1/2)/sqrt(2*pi)
```

7.4.12 Fonction de répartition de la loi normale : `normal_cdf` `normald_cdf`

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, on a : $Proba(X \leq x)$= `normal_cdf(x)` et
$Proba(x \leq X \leq y)$= `normal_cdf(x,y)` et
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, on a :
$Proba(X \leq x)$= `normal_cdf(`μ`,`σ`,x)`.
$Proba(x \leq X \leq y)$= `normal_cdf(`μ`,`σ`,x,y)`.
On tape :

```
normal_cdf(1.96)
```

On obtient :

```
0.975002104852
```

On tape :

```
normal_cdf(0,1.96)
```

On obtient :

```
0.475002104852
```

car `normal_cdf(0)=1/2` et 0.975002104852-0.5=0.475002104852
On tape :

```
normal_cdf(1,2,1.96)
```

On obtient :

```
0.684386303484
```

On tape :

```
normal_cdf(1,2,1.1,2.9)
```

On obtient :

```
0.309005067853
```

7.4.13 Fonction de répartition inverse normale : `normal_icdf normald_icdf`

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, si on a `normal_icdf(x)=h` c'est que l'on a :
$Proba(X \leq h)$=`x=normal_cdf(h)`.
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, si on a `normal_icdf(`μ,σ`,x)=h` c'est que l'on a :
$Proba(X \leq h)$=`x=normal_cdf(`μ,σ`,h)`.
On tape :

```
normal_icdf(0.975)
```

On obtient :

```
1.95996398454
```

On tape :

```
normal_icdf(1,2,0.495)
```

On obtient :

```
0.974933060984
```

On tape :

```
normal_icdf(1,2,normal_cdf(1,2,0.975))
```

On obtient :

```
0.975
```

On tape :

```
normal_cdf(1,2,normal_icdf(1,2,0.495))
```

On obtient :

```
0.495
```

7.4.14 Complément à 1 de la fonction de répartition de la loi normale : UTPN

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, on a : $Proba(X > x)$= `UTPN(x)`.
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et de variance v, on a : $Proba(X > x) =$ `UTPN(`μ`,v,x)`.
On tape :

```
UTPN(1.96)
```

On obtient :

```
0.0249978951482
```

On tape :

```
UTPN(1,4,1.96)
```

On obtient :

```
0.315613696516
```

Attention
Les paramètres de `UTPN` sont la moyenne de X, la variance de X, et la valeur x, alors que, pour `normal_cdf` les paramètres sont la moyenne de X, l'écart-type de X et la valeur x.
On tape :

```
UTPN(1.96)+normal_cdf(1.96)
```

On obtient :

```
1
```

On tape :

```
UTPN(1,4,1.96)+normal_cdf(1,2,1.96)
```

On obtient :

```
1
```

7.4.15 Densité de probabilité de la loi de Student : student studentd

`student(n,x)` est la densité de probabilité de la loi de Student ayant n degrés de liberté.
`student(n,x)`, est égale à : $\frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})\sqrt{n\pi}}\left(1+\frac{x^2}{n}\right)^{\frac{-n-1}{2}}$
où Γ est définie pour $x > 0$ par $\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$
On tape :

```
student(2,3)
```

On obtient :

```
Gamma(3/2)*sqrt(11/2)^-1*2/(sqrt(2*pi)*11)
```

On tape :

```
evalf(student(2,3))
```

On obtient :

```
0.0274101222343
```

7.4.16 Fonction de répartition de la loi de Student : `student_cdf`

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, on a :
$Proba(X \leq x)$= `student_cdf(n,x)`.
$Proba(x \leq X \leq y)$= `student_cdf(n,x,y)`.
On tape :

```
student_cdf(5,2)
```

On obtient :

```
0.949030260585
```

On tape :

```
student_cdf(5,-2)
```

On obtient :

```
0.0509697394149
```

On tape :

```
student_cdf(5,-2,2)
```

On obtient :

```
0.89806052117
```

car $0.949030260585 - 0.0509697394149 = 0.89806052117$

7.4.17 Fonction de répartition inverse de Student : `student_icdf`

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, si on a `student_icdf(n,x)=h` c'est que :
$Proba(X \leq h)$=x=`student_cdf(n,h)`.
On tape :

```
student_icdf(5,0.95)
```

On obtient :

```
2.01504837333
```

7.4.18 Complément à 1 de la fonction de répartition de la loi de Student : `UTPT`

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, on a : $Proba(X > x)$= `UTPT(n,x)`.
On tape :

```
UTPT(5,2)
```

On obtient :

```
0.0509697394149
```

7.4.19 Densité de probabilité de la loi du χ^2 : `chisquare chisquared`

`chisquare(n,x)` est la densité de probabilité de la loi du χ^2 ayant n degrés de liberté.
`chisquare(n,x)`, est égale à $\frac{1}{2^{\frac{n}{2}}\Gamma(\frac{n}{2})}x^{\frac{n}{2}-1}e^{-\frac{x}{2}}$ où Γ est définie pour $x > 0$ par $\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$
On tape :

```
chisquare(5,2)
```

On obtient :

```
sqrt(2)*2*exp(-1)/(Gamma(5/2)*sqrt(2)*2^2)
```

On tape :

```
evalf(chisquare(5,2))
```

On obtient :

```
0.138369165807
```

7.4.20 Fonction de répartition de la loi du χ^2 : `chisquare_cdf`

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, on a :
$Proba(X \leq x)$= `chisquare_cdf(n,x)`.
$Proba(x \leq X \leq y)$= `chisquare_cdf(n,x,y)`.
On tape :

```
chisquare_cdf(5,11)
```

On obtient :

```
0.948620016517
```

On tape :

```
chisquare_cdf(3,11)
```

On obtient :

```
0.988274124422
```

7.4.21 Fonction inverse de la fonction de répartition de la loi du χ^2 : `chisquare_icdf`

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, si on a `chisquare_icdf(n,x)=h` c'est que :
$Proba(X \leq h)$=`x= chisquare_cdf(n,h).`
On tape :

```
chisquare_icdf(5,0.95)
```

On obtient :

```
11.0704976935
```

7.4.22 Complément à 1 de la fonction de répartition de la loi du χ^2 : `UTPC`

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, on a :
$Proba(X > x)$= `UTPC(n,x).`
On tape :

```
UTPC(5,11)
```

On obtient :

```
0.0513799834831
```

7.4.23 Densité de probabilité de la loi de Fisher-Snédécor : `fisher fisherd snedecor snedecord`

`fisher(n1,n2,x)` ou `snedecor(n1,n2,x)` est a densité de probabilité de la loi de Fisher-Snédécor ayant `n1,n2` degrés de liberté.
`fisher(n1,n2,x)`, vaut pour $x \geq 0$:
$\dfrac{n1}{n2}^{\frac{n1}{2}} \dfrac{\Gamma(\frac{n1+n2}{2})}{\Gamma(\frac{n1}{2})\Gamma(\frac{n2}{2})} x^{\frac{n1-2}{2}} \left(1+(\dfrac{n1}{n2})x\right)^{\frac{-n1-n2}{2}}$
où Γ est définie pour $x > 0$ par $\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$
On tape :

```
fisher(5,3,2.5)
```

Oo on tape :

```
snedecor(5,3,2.5)
```

On obtient :

```
0.10131184472
```

On tape :

```
fisher(4,2,1)
```

On obtient :

```
8/27
```

On tape :

```
fisher(4,2,1)
```

On obtient :

```
8/27
```

7.4.24 La fonction de répartition de la loi de Fisher-Snédécor : `fisher_cdf` `snedecor_cdf`

Lorsqu'une variable aléatoire X suit une loi de Fisher-Snédécor ayant degrés de liberté $n1, n2$, on a :
$Proba(X \leq x)$= `fisher_cdf(n1,n2,x)=snedecor_cdf(n1,n2,x)`.
$Proba(x \leq X \leq y)$= `fisher_cdf(n1,n2,x,y)=snedecor_cdf(n1,n2,x,y)`.
On tape :

```
fisher_cdf(5,3,9)
```

Ou on tape :

```
snedecor_cdf(5,3,9)
```

On obtient :

```
0.949898927032
```

On tape :

```
fisher_cdf(3,5,9)
```

On obtient :

```
0.981472898262
```

7.4.25 Inverse de la fonction de répartition de la loi de Fisher-Snédécor : `fisher_icdf` `snedecor_icdf`

Lorsqu'une variable aléatoire X suit une loi de Fisher-Snédécor ayant comme degrés de liberté $n1, n2$, si on a :
`fisher_icdf(n1,n2,x)=h` c'est que :
$Proba(X \leq h)$=x=`fisher_cdf(n1,n2,h)`.
On tape :

```
fisher_icdf(5,3,0.95)
```

Ou on tape :

```
snedecor_icdf(5,3,0.95)
```

On obtient :

```
9.01345516752
```

On tape :

```
1/fisher_icdf(3,5,0.05)
```

On obtient :

```
9.01345516752
```

Remarque :
`fisher_icdf(n1,n2,p)=1/fisher_icdf(n2,n1,1-p)`

7.4.26 Complément à 1 de la fonction de répartition de la loi de Fisher-Snédécor : `UTPF`

Lorsqu'une variable aléatoire X suit une loi de Fisher-Snédécor ayant comme degrés de liberté $n1, n2$, on a : $Proba(X > x)$= `UTPF(n1,n2,x)`.
On tape :

```
UTPF(5,3,9)
```

On obtient :

```
0.050101072968
```

7.4.27 Densité de probabilité de la loi gamma : `gammad`

`gammad` a 3 paramètres : deux réels a, b strictement positifs et un réel x positif ou nul.
`gammad(a,b,x)` renvoie la densité de probabilité de la loi gamma à savoir :
$x^(a-1) * \exp(-bx) * b^a/\Gamma(a)$
On tape :

```
gammad(2,1,3)
```

On obtient :

```
3/exp(3)
```

On tape :

```
gammad(2.2,1.5,0.8)
```

On obtient :

```
0.510330619114
```

7.4.28 Fonction de répartition de la loi gamma : `gammad_cdf`

Lorsqu'une variable aléatoire X suit une loi gamma de paramètre a et b, on a :
$Proba(X \leq x)$= `gammad_cdf(a,b,x)`.
$Proba(x \leq X \leq y)$= `gammad_cdf(a,b,x,y)`.
On a :
`gammad_cdf(a,b,x)` est égale à `igamma(a,b*x,1)` et
`gammad_cdf(a,b,x,y)` est égale à `igamma(a,b*y,1)-igamma(a,b*x,1)`
On rappelle que :
`igamma(a,x)` est égale à `int(e^-t*t^(a-1),t=0..x)` ($\int_0^x e^{-t} * t^{a-1} dt$)
et que
`igamma(a,x,1)` est égale à `igamma(a,x)/Gamma(a)` On tape :

`gammad_cdf(2,1,0.53181160839)`

On obtient :

`0.1`

On tape :

`gammad_cdf(2,1,1.67834699002)`

On obtient :

`0.500000000001`

7.4.29 Fonction de répartition inverse de la loi gamma : `gammad_icdf`

Lorsqu'une variable aléatoire X suit une loi gamma de paramètre a et b, si on a `gammad_icdf(a,b,x)=h` c'est que l'on a :
$Proba(X \leq h)$=x=`gammad_cdf(a,b,h)`.
On tape :

`gammad_icdf(2,1,0.5)`

On obtient :

`1.67834699002`

On tape :

`gammad_icdf(2,1,0.1)`

On obtient :

`0.53181160839`

7.4.30 Densité de probabilité de la loi beta : `betad`

`betad` a 3 paramètres : deux réels a, b strictement positifs et un réel x de $[0, 1]$.
`betad(a,b,x)` renvoie la densité de probabilité de la loi beta à savoir :
$\Gamma(a + b) * x^(a - 1) * (1 - x)^(b - 1)/(\Gamma(a) * \Gamma(b))$
On tape :

`betad(2,1,0.3)`

On obtient :

`0.6`

On tape :

`betad(2.2,1.5,0.8)`

On obtient :

`1.46143068876`

7.4.31 Fonction de répartition de la loi beta : `betad_cdf`

Lorsqu'une variable aléatoire X à valeur dans [0,1] suit une loi beta de paramètre a et b, on a :
$Proba(X \leq x)$= `betad_cdf(a,b,x)` avec $x \in [0,1]$.
$Proba(x \leq X \leq y)$=`betad_cdf(a,b,x,y)` avec $x \in [0,1]$ et $y \in [0,1]$.
On a :
`betad_cdf(a,b,x)` est égale à $\frac{\beta(a,b,x) * \Gamma(a+b)}{\Gamma(a) * \Gamma(b)}$ et
`betad_cdf(a,b,x,y)` est égale à :
$\frac{\Gamma(a+b) * (\beta(a,b,y) - \beta(a,b,x))}{\Gamma(b) * \Gamma(a)}$
On rappelle que l'on a :
`Beta(a,b)` est égale à `int(t^(a-1)*(1-t)^(b-1),t=0..1)` ,
`Beta(a,b,p)` est égale à `int(t^(a-1)*(1-t)^(b-1),t=0..p)` ,
`Beta(a,b,p,1)` est égale à `=Beta(a,b,p)/Beta(a,b)`
On tape :

```
betad_cdf(2,3,0.2)
```

On obtient :

```
0.1808
```

On tape :

```
betad_cdf(2,3,0.9)
```

On obtient :

```
0.9963
```

7.4.32 Fonction de répartition inverse de la loi beta : `betad_icdf`

Lorsqu'une variable aléatoire X suit une loi beta de paramètre a et b, si on a `betad_icdf(a,b,x)=h` c'est que l'on a :
$Proba(X \leq h)$=`x=betad_cdf(a,b,h)`.
On tape :

```
betad_icdf(2,3,0.1808)
```

On obtient :

```
0.2
```

On tape :

```
betad_icdf(2,3,0.9963)
```

On obtient :

```
0.9
```

7.4.33 Densité de probabilité de la loi géométrique : `geometric`

`geometric` a 2 paramètres : un réel p de $]0,1[$ et un entier $n \geq 1$.
`geometric(p,n)` renvoie la densité de probabilité de la loi géométrique à savoir : $(1-p)^{n-1} * p$
On tape :

```
geometric(0.2,3)
```

On obtient :

```
0.128
```

On tape :

```
geometric(0.5,5)
```

On obtient :

```
0.03125
```

7.4.34 Fonction de répartition de la loi géométrique : `geometric_cdf`

Lorsqu'une variable aléatoire X à valeur dans $\mathbb{N}^*$ suit une loi géométrique de paramètre p ($0 < p < 1$), on a :
$Proba(X \leq x)$= `geometric_cdf(p,x)` avec $x \in \mathbb{N}^*$.
$Proba(x \leq X \leq y)$=`geometric_cdf(r,x,y)` avec $x \in \mathbb{N}^*$ et $y \in \mathbb{N}^*$.
On a :
`geometric_cdf(p,x)` est égale à $1-(1-p)^x$ et
`geometric_cdf(p,x,y)` est égale à :
$(1-p)^{x-1}-(1-p)^y$
On tape :

```
geometric_cdf(0.2,3)
```

On obtient :

```
0.488
```

On tape :

```
geometric_cdf(0.2,5)
```

On obtient :

```
0.67232
```

On tape :

```
geometric_cdf(0.2,3,5)
```

On obtient :

```
0.18432
```

On tape :

`geometric_cdf(0.5,1)`

On obtient :

`0.5`

On tape :

`geometric_cdf(0.5,5)`

On obtient :

`0.96875`

On tape :

`geometric_cdf(0.5,1,5)`

On obtient :

`0.46875`

7.4.35 Fonction de répartition inverse de la loi géométrique : `geometric_icdf`

Lorsqu'une variable aléatoire X suit une loi géométrique de paramètre p ($0 < p < 1$), si on a `geometric_icdf(p,x)=h` avec $x \in [0,1]$ c'est que l'on a :
$Proba(X \leq h)$=`x=geometric_cdf(p,h)`.
On tape :

`geometric_icdf(0.2,0.488)`

On obtient :

`3`

On tape :

`geometric_icdf(0.5,0.5)`

On obtient :

`1`

7.4.36 Densité de probabilité de la loi de Cauchy : `cauchy cauchyd`

`cauchyd` a 1 ou 3 paramètres : deux réels a, b strictement positifs (par défaut $a = 0$ et $b = 1$) et un réel x.
`cauchyd(a,b,x)` renvoie la densité de probabilité de la loi de Cauchy à savoir :
`cauchy(x)`=$\dfrac{1}{(x^2+1)*pi}$
`cauchy(a,b,x)`=$\dfrac{b}{((x-a)^2+b^2)*pi}$
On tape :

`cauchyd(0.3)`

Ou on tape :

```
cauchyd(0,1,0.3)
```

On obtient :

```
0.292027418517
```

On tape :

```
cauchyd(2,1,3)
```

On obtient :

```
1/(2*pi)
```

On tape :

```
cauchy(2.2,1.5,0.8)
```

On obtient :

```
0.113412073462
```

7.4.37 Fonction de répartition de la loi de Cauchy : `cauchy_cdf` `cauchyd_cdf`

Lorsqu'une variable aléatoire X à valeur dans $\mathbb{R}$ suit une loi de Cauchy de paramètre a et b, on a :
$Proba(X \leq x)$= `cauchy_cdf(a,b,x)` avec $x \in \mathbb{R}$.
$Proba(x \leq X \leq y)$=`cauchy_cdf(a,b,x,y)` avec $x \in \mathbb{R}$ et $y \in \mathbb{R}$.
On a :
`cauchy_cdf(a,b,x)` est égale à $\frac{1}{2} + \frac{1}{\pi} atan(\frac{x-a}{b})$ et
`cauchy_cdf(a,b,x,y)` est égale à :
$\frac{1}{\pi}(atan(\frac{y-a}{b}) - atan(\frac{x-a}{b}))$
On tape :

```
cauchy_cdf(2,3,1.4)
```

On obtient :

```
0.437167041811
```

On tape :

```
cauchy_cdf(2,3,-1.9)
```

On obtient :

```
0.20871440016
```

On tape :

```
cauchy_cdf(2,3,-1.9,1.4)
```

On obtient :

```
0.228452641651
```

7.4.38 Fonction de répartition inverse de la loi de Cauchy : `cauchy_icdf` `cauchyd_icdf`

Lorsqu'une variable aléatoire X suit une loi de Cauchy de paramètre a et b, si on a `cauchy_icdf(a,b,p)=h` avec $p \in [0, 1]$ c'est que l'on a :
$Proba(X \leq h)$=p=`cauchy_cdf(a,b,h)`.
On tape :

```
cauchy_icdf(2,3,0.23)
```

On obtient :

```
-1.40283204777
```

On tape :

```
cauchy_icdf(2,3,0.44)
```

On obtient :

```
1.42771939334
```

7.4.39 Densité de probabilité de la loi uniforme : `uniform` `uniformd`

`uniformd` a 3 paramètres : deux réels a, b strictement positifs et un réel x de $[a, b]$.
`uniformd(a,b,x)` renvoie la densité de probabilité de la loi uniforme à savoir :
`uniform(a,b,x)`=$\frac{1}{b-a}$ On tape :

```
uniform(2,5,4)
```

On obtient :

```
1/3
```

On tape :

```
uniform(2.2,3.5,2.8)
```

On obtient :

```
0.769230769231
```

7.4.40 Fonction de répartition de la loi uniforme : `uniform_cdf` `uniformd_cdf`

Lorsqu'une variable aléatoire X à valeur dans $[a, b]$ suit une loi uniforme de paramètre a et b, on a :
$Proba(X \leq x)$=`uniform_cdf(a,b,x)` avec $x \in [a, b]$.
$Proba(x \leq X \leq y)$=`uniform_cdf(a,b,x,y)` avec $x \in [0, 1]$ et $y \in [0, 1]$.
On a :
`uniform_cdf(a,b,x)` est égale à : $\dfrac{x-a}{b-a}$ et
`uniform_cdf(a,b,x,y)` est égale à : $\dfrac{y-x}{b-a}$
On tape :

```
uniform_cdf(2,4,3.2)
```

On obtient :

```
0.6
```

On tape :

```
uniform_cdf(2,3,2.5)
```

On obtient :

```
0.5
```

7.4.41 Fonction de répartition inverse de la loi uniforme : `uniform_icdf` `uniformd_icdf`

Lorsqu'une variable aléatoire X suit une loi uniforme de paramètre a et b, si on a `uniform_icdf(a,b,x)=h` avec $x \in [0,1]$ c'est que l'on a :
$Proba(X \leq h)$=x=`uniform_cdf(a,b,h)`.
On tape :

```
uniform_icdf(2,3,0.5)
```

On obtient :

```
2.5
```

On tape :

```
uniform_icdf(2,3,0.9963)
```

On obtient :

```
2.9963
```

7.4.42 Densité de probabilité de la loi exponentielle : `exponential` `exponentiald`

`exponential` ou `exponentiald` a 2 paramètres : un réel λ strictement positifs et un réel x positif ou nul.
`exponentiald(lambda,x)` renvoie la densité de probabilité de la loi exponentielle à savoir :
$\lambda \exp(-\lambda x)$ On tape :

```
exponential(2.1,3.5)
```

On obtient :

```
0.00134944395675
```

On tape :

```
exponential(2.1,0.5)
```

On obtient :

```
0.734869273133
```

7.4.43 Fonction de répartition de la loi de exponentielle : `exponential_cdf` `exponentiald_cdf`

Lorsqu'une variable aléatoire X à valeur dans $\mathbb{R}^+$ suit une loi exponentielle de paramètre $\lambda > 0$ on a :
$Proba(X \leq x)$= `exponential_cdf(lambda,x)` avec $x \geq 0$.
$Proba(x \leq X \leq y)$=`exponential_cdf(lambda,x,y)` avec $x \geq 0$ et $y \geq 0$.
On a :
`exponential_cdf(lambda,x)` est égale à $1 - \exp(-\lambda * x)$ et
`exponential_cdf(lambda,x,y)` est égale à :
$\exp(-\lambda * x) - \exp(-\lambda * y)$
On tape :

```
exponential_cdf(2.3,3.2)
```

On obtient :

```
0.99936380154
```

On tape :

```
exponential_cdf(2.3,0.9)
```

On obtient :

```
0.873814218295
```

On tape :

```
exponential_cdf(2.3,0.9,3.2)
```

On obtient :

```
0.125549583246
```

7.4.44 Fonction de répartition inverse de la loi exponentielle : `exponential_icdf` `exponentiald_icdf`

Lorsqu'une variable aléatoire X suit une loi exponentielle de paramètre $\lambda > 0$, si on a `exponential_icdf(lambda,p)=h` avec $p \in [0, 1]$ c'est que l'on a :
$Proba(X \leq h)$=`p=exponential_cdf(lambda,h)`.
On tape :

```
exponential_icdf(2.3,0.87)
```

On obtient :

```
0.887052534142
```

On tape :

```
exponential_icdf(2.3,0.5)
```

On obtient :

```
0.301368339374
```

7.4.45 Densité de probabilité de la loi de Weibull : `weibull weibulld`

`weibulld` a 3 ou 4 paramètres : deux réels k, λ strictement positifs et 1 ou 2 réels θ, x positif ou nul (par défaut $\theta = 0$).
`weibull(k,lambda,theta,x)` renvoie la densité de probabilité de la loi de Weibull à savoir :
$\frac{k}{\lambda}(\frac{x-\theta}{\lambda})^{k-1} * \exp(-(\frac{x-\theta}{\lambda})^k)$
On tape :

```
weibull(2,1,3)
```

Ou on tape :

```
weibull(2,1,0,3)
```

On obtient :

```
6/exp(9)
```

On tape :

```
weibull(2.2,1.5,0.8)
```

On tape :

```
weibull(2.2,1.5,0.0,0.8)
```

On tape :

```
weibull(2.2,1.5,0.4,1.2)
```

On obtient :

```
0.536773868051
```

7.4.46 Fonction de répartition de la loi de Weibull : `weibull_cdf weibulld_cdf`

Lorsqu'une variable aléatoire X à valeur dans $\mathbb{R}^+$ suit une loi de Weibull de paramètres réels k, λ strictement positifs et θ positif ou nul (par défaut $\theta = 0$), on a :
$Proba(X \leq x)$= `weibull_cdf(k,lambda,theta,x)` avec $x \in \mathbb{R}^+$.
$Proba(x \leq X \leq y)$=`weibull_cdf(k,lambda,theta,x,y)` avec $x \in \mathbb{R}^+$ et $y \in \mathbb{R}^+$.
On a :
`weibull_cdf(k,lambda,theta,x)` est égale à $1 - \exp(-(\frac{x-\theta}{\lambda})^k)$ et
`weibull_cdf(k,lambda,theta,x,y)` est égale à :
$\exp(-(\frac{x-\theta}{\lambda})^k) - \exp(-(\frac{y-\theta}{\lambda})^k)$
On tape :

```
weibull_cdf(2,3,5)
```

Ou on tape :

```
weibull_cdf(2,3,0,5)
```

On obtient :

`1-exp(-25/9)`

On tape :

`weibull_cdf(2,3,1.9)`

On obtient :

`0.330424340391`

On tape :

`weibull_cdf(2.2,1.5,0.4,1.9)`

On obtient :

`0.632120558829`

On tape :

`weibull_cdf(2.2,1.5,0.4,1.2)`

On obtient :

`0.221853318885`

On tape :

`weibull_cdf(2.2,1.5,0.4,1.2,1.9)`

On obtient :

`0.410267239944`

7.4.47 Fonction de répartition inverse de la loi de Weibull : `weibull_icdf` `weibulld_icdf`

Lorsqu'une variable aléatoire X suit une loi de Weibull de paramètre réels k, λ strictement positifs et θ positif ou nul (par défaut $\theta = 0$), si on a `weibull_icdf(k,lambda,theta,` avec $p \in [0, 1]$ c'est que l'on a :
$Proba(X \leq h)$=p=`weibull_cdf(k,lambda,theta,h)`.
On tape :

`weibull_icdf(2,3.5,0.5)`

On obtient :

`2.91394113905`

On tape :

`weibull_icdf(2.2,1.5,0.4,0.632)`

On obtient :

`1.89977657604`

7.4.48 Distribution de Kolmogorov-Smirnov : `kolmogorovd`

`kolmogorovd(x)` renvoie la distribution de Kolmogorov-Smirnov à savoir :

$$1 - 2 * \sum_{k=1}^{+\infty} (-1)^{k-1} \exp(-k^2 x^2)$$

On tape :

```
kolmogorovd(1.36)
```

On obtient :

```
0.950514123245
```

7.4.49 Test de Kolmogorov-Smirnov : `kolmogorovt`

`kolmogorovt(l1,l2)` renvoie le test de Kolmogorov-Smirnov d'adéquation à une loi de distribution continue, entre 2 échantillons `l1` et `l2` lorsque la loi est inconnue, ou
`kolmogorovt(l1,s)` renvoie le test de Kolmogorov-Smirnov d'adéquation à une loi de distribution continue, entre un échantillon `l1` et une loi `s`.
On tape :

```
kolmogorovt(randvector(100,normald,0,1),
      randvector(100,normald,0,1))
```

On obtient :

```
[0.21,1.48492424049]
```

On tape :

```
kolmogorovt(randvector(100,normald,0,1),normald(0,1))
```

On obtient :

```
[0.0797807811891,0.797807811891]
```

7.4.50 Fonction génératrice des moments d'une loi de probabilité : `mgf`

`mgf` a comme paramètres le nom d'une loi de probabilité (parmi les lois : normale, binomiale, Poisson, beta, gamma) et les paramètres de cette loi.
`mgf` renvoie la fonction génératrice des moments de cette loi de probabilité.
On tape :

```
mgf(normald,1,0)
```

On obtient :

```
exp(t)
```

On tape :

```
mgf(poisson,5)
```

On obtient :

```
exp(5*(exp(t)-1))
```

On tape :

```
mgf(binomial,n,p)
```

On obtient :

```
(1-p+p*exp(t))^n
```

7.4.51 Distribution cumulée pour une loi de probabilité : `cdf`

`cdf` a comme paramètre le nom d'une loi de probabilité suivi des paramètres de cette loi.
`cdf` renvoie la distribution cumulée pour cette loi de probabilité. On tape :

```
cdf(binomial,10,0.5,4)
```

On obtient :

```
0.376953125
```

On tape :

```
cdf(normald,0.0,1.0,2.0)
```

On obtient :

```
0.977249868052
```

On tape :

```
cdf(betad,2,3,0.9963)
```

On obtient :

```
0.99999979795
```

7.4.52 Distribution cumulée inverse pour une loi de probabilité : `icdf`

`cdf` a comme paramètre le nom d'une loi de probabilité suivi des paramètres de cette loi.
`icdf` renvoie la distribution cumulée inverse pour cette loi de probabilité.
On tape :

```
icdf(binomial,10,0.5,0.6)
```

On obtient :

```
5
```

On tape :

```
icdf(normald,0.0,0.5,0.975)
```

On obtient :

```
0.97998199227
```

On tape :

```
icdf(betad,2,3,0.9963)
```

On obtient :

```
0.9
```

7.4.53 Chaine de Markov : `markov`

`markov(M)` calcule des éléments caractéristiques d'une matrice de transition d'une chaine de Markov `M`.
`markov(M)` renvoie la liste des suites d'états récurrents positifs, la liste des probabilités stationnaires correspondantes, la liste des autres composantes fortement connexes, la liste des probabilités de finir sur les états récurrents positifs.
On tape :

```
markov([[0,0,1/2,0,1/2],[0,0,1,0,0],[1/4,1/4,0,1/4,1/4],
          [0,0,1/2,0,1/2],[0,0,0,0,1]])
```

On obtient :

```
[[4]],[[0,0,0,0,1]],[[3,1,2,0]],[[1],[1],[1],[1],[1]]
```

7.4.54 Génération d'une marche aléatoire sur un graphe d'état : `randmarkov`

`randmarkov(M,i0,n)` génère une suite de `n` états (chaine de Markov) partant de `i0` dont les probabilites de transitions sont données par `M` (matrice stochastique) ou
`randmarkov(v,i0)` génère une matrice stochastique ayant p boucles récurrentes `v=[n1,...,np]` et `i0` états transients
`randmarkov(M,i0,n)` renvoie la liste des états qui a `n+1` éléments.
On tape :

```
randmarkov([[0,0,1/2,0,1/2],[0,0,1,0,0],[1/4,1/4,0,1/4,1/4],
          [0,0,1/2,0,1/2],[0,0,0,0,1]],2,20)
```

On obtient par exemple :

```
[2,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4]
```

On tape :

```
randmarkov([1,2,1,3],4)
```

On obtient :

```
une matrice carrée de dimension [11,11]
```

en effet $1+2+1+3+4=11$

7.5 Les tests d'hypothèses

7.5.1 Généralités

Concernant une variable aléatoire X, on veut comparer la valeur effective d'un paramètre p à une valeur attendue p_0. Il s'agit de savoir si la valeur observée sur un échantillon est vraisemblable avec $p = p_0$
Test statistique : procédure conduisant au vu de léchantillon à rejeter, avec un certain risque d'erreur une hypothèse que l'on cherche à tester appelée H_0. La procédure de test est fondée sur une opposition d'hypothèses et on note H_1 l'hypothèse alternative.
Test bilatéral : test pour lequel l'hypothèse H_0 est rejetée, si la statistique utilisée prend une valeur en dehors d'un intervalle. **Test unilatéral à droite** : test pour lequel l'hypothèse H_0 est rejetée, si la statistique utilisée prend une valeur inférieure à une valeur.
Test unilatéral à gauche : test pour lequel l'hypothèse H_0 est rejetée, si la statistique utilisée prend une valeur supérieure à une valeur.
Construction d'un test :
- choix des hypothèses H_0 et H_1,
- choix d'une variable statistique S servant de variable de décision
- détermination de la région critique au seuil α,
- énonce de la règle de décision.
Utilisation du test :
- prélèvement d'un échantillon,
- au vu de la valeur observée de S, on rejete ou on accepte H_0.

7.5.2 `normalt`

`normalt` a pour arguments :

1. la liste $[ns, ne]$ où ns est le nombre de succès lorsqu'on fait ne essais, ou la liste $[m, t]$ où m est la moyenne et t la taille de l'échantillon, ou la liste des données de l'échantillon
2. la proportion de succés ou la moyenne de la population ou une liste de données d'un échantillon témoin
3. l'écart-type (cet argument est optionnel si des données sont fournies, il est déduit des données)
4. l'hypothèse alternative H1 `'!='` ou < ou >
5. le niveau de confiance, cet argument est optionnel (valeur par défaut 0.05)

`normalt` réalise le test Z d'hypothèses pour une loi normale.
`normalt` renvoie `0` ou `1` et affiche un résumé du test.

Exemple 1 : on tape :

```
normalt([10,30],.5,.02,'!=',0.1)
```

On obtient :

```
0
```

avec en vert le résumé du test :
*** TEST RESULT 0 ***
Summary Z-Test null hypothesis mu1=mu2, alt. hyp. mu1 !=mu2.
Test returns 0 if probability to observe data is less than 0.1
(null hyp. mu1=mu2 rejected with less than alpha probability error)
Test returns 1 otherwise (can not reject null hypothesis)
Data mean mu1=10, population mean mu2=0.5
alpha level 0.1, multiplier*stddev/sqrt(sample size)= 1.64485*0.02/5.47723
Le test a échoué, il y a donc moins d'une chance sur 10 que le nombre de succès soit de 10 pour 30 essais avec une proportion attendue de 0.5 et un écart-type de 0.02, on rejette l'hypothèse $H0$.

Exemple 2 : on tape :

```
normalt([0.48,50],0.5,0.1,'<')
```

On obtient :

```
1
```

avec en vert le résumé du test :
*** TEST RESULT 1 ***
Summary Z-Test null hypothesis mu1=mu2, alt. hyp. mu1<mu2.
Test returns 0 if probability to observe data is less than 0.05
(null hyp. mu1=mu2 rejected with less than alpha probability error)
Test returns 1 otherwise (ca`class_min`n not reject null hypothesis)
Data mean mu1=0.48, population mean mu2=0.5
alpha level 0.05, multiplier*stddev/sqrt(sample size)= 1.64485*0.1/7.07107
Ici le test réussit, on ne peut pas exclure (au seuil de confiance 0.05) l'observation d'une proportion de 0.48 avec 50 essais pour une proportion théorique de 0.5 et un écart-type de 0.1

7.5.3 `studentt`

`studentt` a pour arguments :

1. la liste $[ns, ne]$ où ns est le nombre de succès lorsqu'on fait ne essais, ou la liste $[m, t]$ où m est la moyenne et t la taille de l'échantillon, ou la liste des données de l'échantillon,
2. la proportion de succés ou la moyenne de la population ou une liste de données d'un échantillon témoin,
3. l'écart-type (cet argument est optionnel si des données sont fournies, il est déduit des données),
4. l'hypothèse alternative H1 `'!='` ou < ou >,
5. le niveau de confiance, cet argument est optionnel (valeur par défaut 0.05)

`studentt` réalise le test Z d'hypothèses pour une loi de student. On utilise de préférence le test `studentt` plutôt que `normalt` lorsque la taille de l'échantillon est petite.
`studentt` renvoie `0` ou `1` et affiche un résumé du test.
On tape :

```
studentt([10,20],.5,.02,'!=',0.1)
```

On obtient :

```
0
```

avec en vert le résumé du test :
*** TEST RESULT 0 ***
Summary T-Test null hypothesis mu1=mu2, alt. hyp. mu1 !=mu2.
Test returns 0 if probability to observe data is less than 0.1
(null hyp. mu1=mu2 rejected with less than alpha probability error)
Test returns 1 otherwise (can not reject null hypothesis)
Data mean mu1=10, population mean mu2=0.5, degrees of freedom 20
alpha level 0.1, multiplier*stddev/sqrt(sample size)= 1.32534*0.02/4.47214
Le test a échoué, il y a donc moins d'une chance sur 10 que le nombre de succès soit de 10 pour 20 essais avec une proportion attendue de 0.5 et un écart-type de 0.02, on rejette l'hypothèse $H0$.
On tape :

```
studentt([0.48,20],0.5,0.1,'<')
```

On obtient :

```
1
```

avec en vert le résumé du test :
*** TEST RESULT 1 ***
Summary T-Test null hypothesis mu1=mu2, alt. hyp. mu1<mu2.
Test returns 0 if probability to observe data is less than 0.05
(null hyp. mu1=mu2 rejected with less than alpha probability error)
Test returns 1 otherwise (can not reject null hypothesis)
Data mean mu1=0.48, population mean mu2=0.5, degrees of freedom 20
alpha level 0.05, multiplier*stddev/sqrt(sample size)= 1.72472*0.1/4.47214. Ici le test réussit, on ne peut pas exclure (au seuil de confiance 0.05) l'observation d'une proportion de 0.48 avec 20 essais pour une proportion théorique de 0.5 et un écart-type de 0.1

7.5.4 `chisquaret`

`chisquaret` a pour arguments :

– la liste des données d'un échantillon,
– une loi de distribution ou la liste des données d'un autre échantillon,
– les paramètres de cette loi de distribution ou les paramètres : `classes` suivi de `class_min` et de `class_dim`.

`chisquaret` est le test du Chi2 d'adéquation entre 2 (or n) échantillons ou entre un échantillon et une loi de distribution (multinomiale ou donnée par une fonction).
`chisquaret` renvoie la valeur d^2 de la statistique D^2 où :
$D^2 = \sum_{j=1}^{k} \frac{(n_j - e_j)^2}{e_j}$ avec
k est le nombre de classes de l'échantillon, $n_1...n_k$ les effectifs de chaque classe de l'échantillon et e_1,..,e_k sont les effectifs théoriques (si chaque valeur x_j est obtenue avec la probabilité théorique p_j on a $e_j = np_j$) On tape :

```
chisquaret([57,54])
```

On obtient :

```
0.0810810810811
```

avec en vert le résumé du test :
On suppose que les données sont des effectifs de classes, adéquation à la distribution uniforme.
Adéquation d'un échantillon à une distribution discrète Résultat du test du Chi2 0.0810810810811.
On rejette l'adéquation si supérieur à `chisquare_icdf(1,0.95)` qui vaut `3.84145882069` or `chisquare_icdf(1,1-alpha)` si alpha !=5%.
On tape :

```
chisquaret([1,1,1,1,1,0,0,1,0,0,1,1,1,0,1,1,0,1,1,0,0,0,0],[.4,.6])
```

On obtient :

```
0.115942028986
```

avec en vert le résumé du test :
Adéquation d'un échantillon à une distribution discrète
Résultat du test du Chi2 0.115942028986,
On rejette l'adéquation si supérieur à `chisquare_icdf(1,0.95)` qui vaut `3.84145882069` or `chisquare_icdf(1,1-alpha)` si alpha !=5%.
On tape :

```
chisquaret(ranv(1000,binomial,10,.5),binomial)
```

On obtient :

```
?
```

On tape :

```
chisquaret(ranv(1000,binomial,10,.5),binomial,11,.5)
```

On obtient :

```
?
```

On régle dans la configuration graphique les valeurs de `class_min` et de `class_dim` à -2 et 0.25 et on tape :

```
L:=ranv(1000,normald,0,.2)
```

```
chisquaret(L,normald)
```

Ou on tape :

```
chisquaret(L,normald,classes,-2,.25)
```

On obtient par exemple :

```
3.7961112714
```

avec en vert le résumé du test :
Densité normale, estimation de la moyenne et de l'ecart-type par les données0.00109620704977 0.201224413347
Adéquation d'un échantillon à `normald_cdf(.00109620704977,0.201224413347)`, résultat du test du Chi2 3.7961112714,
On rejette l'adéquation si supérieur à `chisquare_icdf(4,0.95)` qui vaut `9.48772903678` ou `chisquare_icdf(4,1-alpha)` si alpha !=5%.
Vérifions :
On tape :

```
size(classes(L))
```

Ou on tape :

```
size(classes(L,-2,.25))
```

On obtient par exemple :

```
6
```

On tape :

```
C:=classes(L)
```

Ou on tape :

```
C:=classes(L,-2,.25)
```

On obtient par exemple :

```
[[(-0.75) .. -0.5,3],[(-0.5) .. -0.25,113],[(-0.25) ..
0,382],[0 .. 0.25,393],[0.25 .. 0.5,102],[0.5 ..
0.75,7]]
```

On a bien : $n = 3 + 113 + 382 + 393 + 102 + 7 = 1000$
On compare cette distribution empirique L à une distribution théorique d'effectifs e_1,..,e_k (si chaque valeur x_j est obtenue avec la probabilité théorique p_j on a $e_j = np_j$).
Calculons la valeur d^2 de la statistique :
$D^2 = \sum_{j=1}^{k} \frac{(n_j - e_j)^2}{e_j}$
On estimer les paramètres μ et σ à partir de l'échantillon (μ par la moyenne m de l'échantillon et σ par $s\sqrt{\frac{n}{n-1}}$ où s est l'écart-type de l'échantillon et n vaut ici 1000) :
On tape :

```
mu:=mean(L);s:=stddev(L)
```

On obtient par exemple :

```
0.00109620704977,0.201123775974
```

On tape :

```
sigma:=s*sqrt(1000/999)
```

On obtient par exemple :

```
0.201224413347
```

On tape :

```
p1:=normal_cdf(mu,sigma,-0.75,-0.50)
```

On obtient :

```
0.00628817561212
```

On tape :

```
p2:=normal_cdf(mu,sigma,-0.5,-0.25)
```

On obtient :

```
0.0996616023075
```

On tape :

```
p3:=normal_cdf(mu,sigma,-0.25,0)
```

On obtient :

```
0.391782175726
```

On tape :

```
p4:=normal_cdf(mu,sigma,0,0.25)
```

On obtient :

```
0.394119790487
```

On tape :

```
p5:=normal_cdf(mu,sigma,0.25,0.5)
```

On obtient :

```
0.101472226962
```

On tape :

```
p6:=normal_cdf(mu,sigma,0.5,0.75)
```

On obtient :

```
0.00648235374944
```

On calcule la valeur d^2 de la statistique D^2, on tape :

```
P:=[p1,p2,p3,p4,p5,p6]:;
sum((C[k,1]-P[k]*1000)^2/(P[k]*1000),k=0..5)
```

On obtient bien :

```
3.79611127141
```

7.5.5 `kolmogorovt`

`kolmogorovt` a pour arguments :
- la liste des données d'un échantillon,
- une loi de distribution ou la liste des données d'un autre échantillon,
- les paramètres de cette loi de distribution.

`kolmogorovt` est le test de Kolmogorov-Smirnov d'adéquation à une loi de distribution **continue**, entre 2 échantillons l1 l2 (loi inconnue supposée continue) ou entre 1 échantillon l1 et une loi continue.
`kolmogorovt` renvoie 3 valeurs préfixées, D est la valeur brute de la statistique, K vaut $D\sqrt{n}$ où n est la taille de l'échantillon (ou $D\sqrt{n_1 n_2/(n_1+n_2)}$ si on teste l'adéquation de 2 échantillons de tailles n_1 et n_2), K tend vers la distribution de Kolmogorov-Smirnov `kolmogorovd` lorsque n tend vers l'infini, la 3ième valeur est 1-`kolmogorovd`(K) qui est d'autant plus proche de 1 que l'adéquation est bonne (on rejettera donc l'hypothèse nulle - qui est l'adéquation - lorsque cette dernière valeur est proche de 0). Le menu Aide, Exemples, probas, kolmogorov donne une illustration de ce test : on teste l'adéquation d'un vecteur aléatoire tiré selon une loi avec cette même loi : on fait le calcul de la statistique K pour 200 tirages et on représente graphiquement le cumul de fréquences avec la distribution de KS sur le même graphique (cela marche bien pour les deux premières lois qui sont continues, mais pas pour la troisième loi qui est discrète)
On tape :

```
kolmogorovt(randvector(100,normald,0,1),normald(0,1))
```

On obtient :

```
["D=",0.104642990054,"K=",1.04642990054,
  "1-kolmogorovd(K)=",0.223512817343]
```

On ne peut pas rejeter l'adéquation car `1-kolmogorovd(K)` est n'est pas proche de zéro.

On tape :

```
kolmogorovt(randvector(100,normald,0,1),student(2))
```

On obtient :

```
["D=",0.158026574028,"K=",1.58026574028,
  "1-kolmogorovd(K)=",0.0135504875564]
```

On rejette l'adéquation car `1-kolmogorovd(K)` est proche de zéro.

7.5.6 Des exemples

Voici quelques exercices :

1. **Exercice**
   Pour tester l'efficacité d'un vaccin antigrippal on soumet 300 personnes à une expérience :
   - sur 100 personnes non vaccinées, 32 sont atteintes par la grippe,
   - sur 200 personnes vaccinées, 50 sont atteintes par la grippe,

Ce résultat permet-il d'apprécier l'efficacité du vaccin ?
On calcule les valeurs $f1$ et $f2$ qui sont les proportions des grippés des deux échantillons on tape :
`f1:=32/100`
`f2:=50/200`
On tape :
`f1-f2`
On obtient :
`7/100`
Donc $|f_1 - f_2| = 0.07$
On calcule la valeur p proportion des grippés lorsqu'on reunit les deux échantillons on tape :
`p:=82/300`
On obtient :
`41/150`
Donc $p \simeq 0.273333333333$
On calcule s_{12}, on tape :
`s12:=sqrt(p*(1-p)*(1/100+1/200))`
On obtient :
`sqrt(4469/1500000)`
Donc $s_{12} \simeq 0.0545832697201$
On tape :
`normalt([32,100],[50,200],0.0545832697201,'!=',0.05)`
On obtient :
`0`
et en vert le resumé du test :
Moyenne estimee en utilisant le(s) echantillon(s) 125
*** TEST RESULT 0 ***
Summary Z-Test null hypothesis mu1=mu2, alt. hyp. mu1 !=mu2. Test returns 0 if probability to observe data is less than 0.05 (null hyp. mu1=mu2 rejected with less than alpha probability error) Test returns 1 otherwise (can not reject null hypothesis) Data mean mu1=32, population mean mu2=125 alpha level 0.05, multiplier*stddev/sqrt(sample size)= 1.95996*0.0545833/10
`1.95996*0.0545833/10` renvoie `0.0106981084668`
On tape :
`a:=normal_icdf(0,sqrt(4469/1500000),0.975)`
On obtient :
`0.10698124281`
Puisque `|f1-f2|=0.07<a=0.10698124281`, on en déduit que les deux échantillons ne sont pas significativement différents au seuil de 5% : on peut donc dire que le vaccin n'est pas efficace mais ce n'est pas une certitude...

La statistique $D^2 = \sum_{j=1}^{k} \frac{(n_j - e_j)^2}{e_j}$ est une bonne mesure de l'écart entre les effectifs observés et les effectifs théoriques : plus D^2 est proche de zéro, plus la distribution de l'échantillon est conforme à la distribution théorique.
On calcule la valeur $d2$ de D^2 on tape :

`d2:=300*(150*32-68*50)^2/(100*200*82*218)`
On obtient :
`7350/4469`
donc $d^2 \simeq 1.645$
On cherche la valeur h qui vérifie :
$Proba(\chi_1^2 > h) = 0.05$ ou encore $Proba(\chi_1^2 \leq h) = 0.95$
pour cela on tape :
`chisquare_icdf(1,0.95)`
On obtient :
`3.84145882069`
donc $h \simeq 3.84$
Puisque d2 $\simeq$ 1.645 $<$ 3.84 on en déduit que les deux échantillons ne sont pas significativement différents au seuil de 5% : on peut donc mettre en doute l'efficacité du vaccin.
On tape :
`?`
On obtient :
`?`

2. **Exercice**
   Sur un registre d'état civil, on a relevé 552 naissances dont 289 garçons.
   a/ Estimer la fréquence `p` de naissance d'un garçon.
   b/ Donner un intervalle de confiance pour cette estimation.
   On tape :
   `?`
   On obtient :
   `?`
3. **Exercice**
   Dans un hôpital sur un échantillon de 458 malades admis pendant un trimestre il y a eu 141 décès. Estimer le pourcentage de décès par un intervalle de confiance au seuil de 0.01.
   On tape :
   `?`
   On obtient :
   `?`
4. **Exercice : comparaison de deux moyennes**
   Pour une même épreuve, voici les notes obtenues dans une classe de terminale du lycée A.
   6,10,14,17,9,6,4,12,9,10,10,11,12,18,10,9,11,8,7,10.
   et les notes obtenues dans une classe de terminale du lycée B.
   2,10,14,13,9,6,1,12,9,10,10,10,12,15,19,9,11,8,9,10
   1/ Analyser les résultats de chaque groupe.
   2/ Peut-on considérer que les 2 groupes sont issus d'une même population ?
5. **Exercice : comparaison de deux moyennes**
   Deux entreprises A et B livrent des pièces dans des paquets de 100 pièces. On note X_1 (resp X_2) la variable aléatoire égale au nombre de pièces défectueuses par paquet provenant de A (resp B).
   On note $\overline{X}_1$ (resp $\overline{X}_2$) la variable aléatoire égale au nombre moyen de pièces

défectueuses par paquet pour des échantillons aléatoires de 49 paquets (resp 64 paquets) provenant de A (resp B).
I) Sur un échantillon de 49 paquets provenant de A on compte le nombre de pièces défectueuses dans chaque paquet et on trouve :
7, 5, 5, 4, 4, 4, 9, 7, 9, 2, 7, 8, 7, 8, 4, 4, 9, 10,
5, 10, 6, 4, 5, 6, 1, 2, 5, 7, 8, 0, 6, 0, 1, 5, 2, 0,
5, 2, 3, 3, 4, 1, 3, 10, 1, 0, 10, 2, 7
1/ Calculer la moyenne m_1 et l'écart-type s_1 de cet échantillon.
2/ Donner une estimation de la moyenne μ_1 et de l'écart-type σ_1 de X_1.
3/ Donner une estimation de la moyenne et de l'écart-type de $\overline{X}_1$.
On tape :
`?`
On obtient :
`?`
On tape :
`?`
On obtient :

On tape :
`?`
On obtient :
`?`

6. **Exercice**
   On a administré un somnifère A à 50 personnes choisies au hasard et on a observé une moyenne de sommeil de 8h22 avec un écart-type de 0h24.
   On a administré un somnifère B à 100 personnes choisies au hasard et on a observé une moyenne de sommeil de 7h15 avec un écart-type de 0h30.
   Ces deux somnifères ont-ils une efficacité signicativement différente ? de combien ?

7. **Exercice : Échantillons appariés**
   On a fait faire une double correction de 30 copies par deux examinateurs A et B afin de comparer leur notation. Les copies sont numerotées de 0 à 29.
   On a obtenu pour A :
   `13,15,12,15,8,7,11,10,9,13,3,18,17,5,9,10,`
   `11,14,12,10,9,8,13,6,8,16,14,11,12,10`
   On a obtenu pour B :
   `12,13,12,15,7,5,12,10,8,13,4,17,16,4,9,11,10,`
   `13,13,9,10,7,14,8,7,15,13,10,13,10`
   On tape :
   `?`
   On obtient :
   `?`

8. **Exercice : Jet d'un dé et test du χ^2**
   On jette un dé 90 fois et on a obtenu :
   1 a été obtenu 11 fois,
   2 a été obtenu 16 fois,

3 a été obtenu 17 fois,
4 a été obtenu 22 fois,
5 a été obtenu 14 fois,
6 a été obtenu 10 fois.
Peut-on admettre au vu de cette expérience que le dé est régulier ?
On tape :
?
On obtient :
?

9. **Exercice : Jet d'un dé et test du** χ^2
On jette un dé 180 fois et on a obtenu :
1 a été obtenu 22 fois,
2 a été obtenu 32 fois,
3 a été obtenu 34 fois,
4 a été obtenu 44 fois,
5 a été obtenu 28 fois,
6 a été obtenu 20 fois.
Peut-on admettre au vu de cette expérience que le dé est régulier ?
On tape :
?
On obtient :
?

Chapitre 8

Les fonctions de programmation

8.1 La forme d'une fonction, d'un programme et d'un script

8.1.1 Le choix de l'éditeur

On écrit une fonction ou un programme dans l'éditeur de son choix : on peut utiliser les éditeurs de `Xcas` (que l'on ouvre avec `Alt+p`) pour écrire des petits programmes mais il est préférable d'utiliser par exemple `emacs` pour écrire des programmes plus conséquents. Dans ce cas, on sauve le programme dans un fichier, par exemple `toto.cas`, puis dans la ligne de commande de `Xcas` on tape :
`read("toto.cas")` ou encore
avec le menu `Fich` sous-menu `Charger` on sélectionne `toto.cas`.

8.1.2 La forme d'une fonction

Une fonction renvoie une valeur : l'instruction `return valeur;` ou `retourne valeur;` fait sortir du programme et renvoie `valeur`.
Une fonction a un nom (par exemple `toto`), puis on met entre des parenthèses les arguments de la fonction (par exemple si `toto` a besion de deux paramètres `a` et `b` on met `toto(a,b)`). On met ensuite := puis on met entre des accolades (ou entre `begin` et `end`) le bloc qui définit la fonction c'est à dire la suite des instructions (chaque instuction se termine par `;`) et si l'algorithme définissant la fonction a besoin de variables locales, ces variables devront être déclarées en mettant au début du bloc `local` puis, les noms des variables locales séparés par des virgules puis, `;` pour terminer cette déclaration. Ces variables locales peuvent être initialisées lors de leur déclaration.
On écrit par exemple :

```
toto(a,b):={
local q,r,s;
<instrucion-1>;
.......... ;
<instrucion-n>;
return r;
}
```

Attention Les variables locales ne sont pas des variables formelles et doivent toujours être affectèes dans le corps du programme : on ne définira donc pas, les variables formelles, avec `local`.
Si on veut qu'une variable déclarèe avec `local` soit formelle (par ex `a`), il faudra mettre dans le corps du programme soit `purge(a)` soit `assume(a,symbol)`.
Exemple

```
kk(a):={
local x,c;
c:=4*a;
return solve((x-c)^2-a=0,x);
}:;
f(a):={
local x,c;
c:=4*a;
assume(x,symbol);
return solve((x-c)^2-a=0,x);
}:;
g(a):={
local c;
c:=4*a;
return solve((x-c)^2-a=0,x);
}:;
```

On tape :
`kk(1),f(1),g(1)`
On obtient :
`list[],list[3,5],list[3,5]` Il ne faut pas tenir compte du message renvoyé pas `Xcas` lors de la compilation :
"Attention : x, declarée(s) comme variable(s) globale(s) en compilant g" car dans `solve` la variable `x` est toujours symbolique.

Remarque on peut initialiser les variables locales en même temps que leur déclaration à condition de mettre des parenthèses, par exemple :
`local (q:=1),(r:=0),s;`

Attention Il est important de déclarer les variables locales car une variable globale est évaluée avant l'execution de la fonction qui s'en sert lorsque cette fonction est appelée par une autre fonction...on risque donc d'avoir des ennuis si une fonction qui utilise une variable globale est appelée par une autre fonction, par exemple :

```
truc(a,b):={
  if (b!=0) {
    r:=irem(a,b);
  } else {
    r:=b;
  }
  return r;
};
```

```
machin(a,b):={
local rr;
rr:=truc(a,b);
return rr;
}
```

L'exécution de `truc(45,6)` ne pose pas de problème et renvoie 3, mais l'exécution de `machin(45,6)` renvoie le message d'erreurs :
`sto 3 not allowed! Error: Bad Argument Type`
car lorsque `truc` est appelé par `machin` `r` qui est une variable globale est evaluée et alors `r:=irem(a,b)` n'est pas possible car `r` vaut 3....
Il est donc important de tenir compte du message donné par `Xcas` lors de la compilation de `truc` :
`//Parsing truc//Warning, check that the following variables are global: r compiling truc` à condition que les variables signalées ne soient pas des variables formelles. Voici comme exemple le programme qui donne la valeur de la suite de Fibonnacci définie par $u_0 = u0, u_1 = u1, u_{n+2} = u_{n+1} + u_n$. Dans ce programme on utilise les variables formelles `x,A,B` qui doivent être purgées.
On tape :

```
u(n,uo,u1):={
local L,a,b;
//verifier que A,B,x ne sont pas affectées
[a,b]:=solve(x^2-x-1,x);
L:=linsolve([A+B=uo,A*a+B*b=u1],[A,B]);
return normal(L[0]*a^n+L[1]*b^n);
};
```

On tape :

`u(3,0,1)`

On obtient :

`2`

Dans ce programme, les variables `x,A,B` ne doivent pas être déclarées comme variables locales car ce sont des variables formelles : il ne faut donc pas tenir compte lors de la compilation du warning : `// Warning: x A B declared as global variable(s) compiling u`

8.1.3 La forme d'un programme

Un programme a la même forme qu'une fonction mais ne renvoie pas de valeurs, mais un programme peut afficher des résultats ou faire des dessins : dans ce cas toutes les instructions d'affichage (`afficher` ou `Disp;`) se feront en bleu dans l'écran intermédiaire situé avant la réponse et les instructions de dessins se feront dans l'écran `DispG`. En particulier les dessins récursifs ne seront visibles que dans l'écran `DispG`.

– Pour ouvrir l'écran `DispG`, on utilise le menu `Cfg`▶`Montrer`▶ `DispG` ou l'instruction `DispG;` (ne pas mettre de parenthèses).
– Pour effacer l'écran `DispG`, on utilise la commande `ClrGraph()` ou `erase`.
– Pour cacher l'écran `DispG`, on utilise le menu `Cfg`▶`Cacher`▶ `DispG` ou l'instruction `DispHome;` (ne pas mettre de parenthèses).

Lors de l'exécution du programme, on aura comme réponse, la valeur (ou le dessin) de la dernière instruction du programme.
Il est préférable de mettre `return 0` à la fin d'un programme et de le transformer ainsi en fonction : l'exécution de ce programme renverra 0 comme réponse ce qui voudra dire que son exécution s'est bien passée.

8.1.4 La forme d'un script

Un script est une suite d'instructions que l'on met dans un fichier (chaque instuction se termine par `;`).

8.2 Exécuter une fonction pas à pas

On peut exécuter une fonction pas à pas, en utilisant le debuggeur :
on se sert de l'instruction `debug` avec comme argument une fonction et ses arguments et cela permet d'exécuter la fonction pas à pas. On a alors la possibilité de voir l'évolution des variables de son choix (on se reportera pour plus de détails à la section 8.7).

8.3 La séquence d'instructions

Un bloc d'instructions ou une séquence d'instructions doit être parenthésé soit par `{ }`, soit par `begin` et `end`.
Entre ces accolades (ou entre `begin` et `end`) on met les instructions en les terminant par un point-virgule `;`

8.4 Les instructions de base

8.4.1 Les commentaires : `comment //`

`comment` a comme argument une chaine de caractère (il faut mettre les `"`) alors que `//` n'a pas besion des `"` mais doit se terminer par un retour à la ligne : cela veut veut dire que l'argument de `comment` ou ce qui se trouve entre `//` et le retour à la ligne n'est pas pris en compte par le programme et que c'est un commentaire.
On tape le programme dans `un éditeur de programmes` puis on le valide avec le bouton `OK` :

```
f(x):={comment("f:R->R^2")
    return [x+1,x-1];}
```

ou :

```
f(x):={//f:R->R^2
return [x+1,x-1];}
```

On obtient :

```
la définition commentée de la fonction f(x)=[x+1,x-1]
```

8.4.2 Les entrées : `input, saisir, Input, InputStr, saisir_chaine, textinput`

`input, saisir, Input` permettent de saisir des expressions et `InputStr, saisir_chaine, textinput` permettent de saisir des chaines de caractères.
`input, saisir, Input` et `InputStr, saisir_chaine, textinput` ont comme argument le nom d'une variable (resp une séquence de noms de variables) ou une chaîne de caractères (chaîne donnant à l'utilisateur des indications sur la valeur à entrer) et le nom d'une variable (resp une séquence de chaînes de caractères et une séquence de noms de variables).
`input, saisir, Input` et InputStr, saisir_chaine, textinput ouvrent une fenêtre où on peut entrer la valeur de la variable donnée comme argument et où on retrouve, comme légende, la chaîne de caractères mise dans l'argument (si on la mise ! ! !).
Avec `input, Input, saisir` on peut entrer des nombres ou des chaînes de caractères (il faut alors mettre "...") ou des noms de variables (sans mettre "...").
Avec `InputStr, textinput, saisir_chaine` on ne peut entrer que des chaînes de caractères, mais on n'a donc pas besoin de mettre "...".
On tape :

```
input(a)
```

ou :

```
input("a=?",a)
```

On obtient :

```
une fenêtre où on peut entrer la valeur de a
```

On tape :

```
12 dans cette fenêtre puis OK
```

puis :

```
a+3
```

On obtient :

```
15
```

On tape :

```
input("polynome",p,"valeur",a)
```

On obtient :

```
une fenêtre où on peut entrer les valeurs de p avec la
  légende "polynome" et de a avec la légende "valeur"
```

On tape :

```
InputStr(a)
```

ou :

```
InputStr("reponse=",a)
```

On obtient :

```
une fenêtre où on peut entrer la valeur de a
```

On tape :

```
12 dans cette fenêtre puis OK
```

puis :

```
a+3
```

On obtient :

```
123
```

car `a` est la chaîne de catractères `"12"` et le `"+"` concaténe les deux chaînes `"12"` et `"3"`.

8.4.3 Fonction testant si une touche est pressée : `getKey`

`getKey` est une fonction qui n'a pas d'argument et qui renvoie le code `ASCII` de la touche pressée dans l'écran `xterm` de la fenêtre du terminal ou `0` si aucune touche n'est pressée dans cet écran .
On tape :

```
testkey():={
local a;
while (1==1) {
a:=getKey();
if (a!=0) break;
}
return a;
}
```

puis :

```
testkey()
```

puis :

```
A
```

On obtient :

```
65
```

8.4.4 Fonction testant le type de son argument : `type`

`type` est une fonction qui a un argument et qui renvoie le type de cet argument par exemple :
`DOM_FLOAT,DOM_INT,DOM_COMPLEX,..,DOM_IDENT,DOM_LIST,`
`DOM_SYMBOLIC,DOM_RAT,..DOM_SYMBOLIC,DOM_STRING`.
Il y a 20 types différents qui sont représentés par un entier entre 1 et 20.
`type` permet de tester une erreur d'entrée.
On tape :

```
type(3.14)
```

On obtient :

```
DOM_FLOAT
```

On tape :

```
type(3.14)+0
```

On obtient :

```
1
```

On tape :

```
type(3)
```

On obtient :

```
DOM_INT
```

On tape :

```
type(3)+0
```

On obtient :

```
2
```

On tape :

```
type(3% 5)
```

On obtient :

```
15
```

Remarque : utilisation de type pour les graphes
voir aussi 3
Si le graphe dépend d'une fonction utilisateur, il faut que la fonction soit définie même lorsque le(s) paramètre(s) a (ont) une valeur formelle, ce qui peut se faire en testant le type du paramètre, comme dans l'exemple suivant, f est définie en testant du type du paramètre par :

```
f(x):= {
  if (type(x)==DOM_IDENT) return 'f'(x);
  while(x>0){ x--; }
  return x;
}:;
```

Ainsi `f(x)` renvoie `f(x)` et `f(3.1)` renvoie `-0.9`.
ou bien

```
g(x):= {
  if (type(x)!=DOM_LIST) return 'g'(x);
  if (type(x)==DOM_IDENT) return 'g'(x);
  while(x[0]>0){x[0]:=x[0]-1; }
  return x;
}:;
```

Ainsi `g(x)` renvoie `g(x)`, `g([x,y,z])` renvoie `g([x,y,z])` et `g([1.1,2,3])` renvoie `[-0.9,2,3]`.
Par exemple si `G:=plotfunc(g(x)`, le test permet d'utiliser le graphe `G` de `g` dans des commandes comme `translation(1+i,G)`...

8.4.5 Fonction testant si le type de son argument est une séquence : `subtype`

`subtype` est une fonction qui a un argument et qui renvoie 1 si cet argument est une séquence et 0 sinon.
On tape :

`subtype(3,1,4)`

On obtient :

`1`

On tape :

`subtype[3,1,4]`

On obtient :

`0`

8.4.6 Fonction testant le type de son argument : `getType`

`getType` est une fonction qui a un argument et qui renvoie le type de cet argument par exemple :
`STR,EXPR,VAR,NONE,PIC,LIST,MAT,FUNC,NUM....`
`getType` ne renvoie pas la même chose que `type`, c'est pour des raisons de compatibilité.
On tape :

`getType(3.14)`

On obtient :

```
NUM
```

On tape :

```
getType(x)
```

On obtient :

```
VAR
```

On tape :

```
getType(x+2)
```

On obtient :

```
EXPR
```

8.4.7 Fonction testant le type de son argument : `compare`

`compare` est une fonction qui a deux arguments et qui renvoie `1` si ses arguments ont des types différents ou ou si ses arguments sont de même type et sont mis dans l'ordre croissant, et qui renvoie `0` sinon.
On tape :

```
compare(1,2)
```

On obtient :

```
1
```

On tape :

```
compare(2,1)
```

On obtient :

```
0
```

On tape :

```
compare("3","a")
```

On obtient :

```
1
```

On tape :

```
compare("a",3)
```

On obtient :

```
0
```

On tape :

`compare(3,"a")`

On obtient :

`1`

On tape :

`compare("a",3)`

On obtient :

`0`

En effet on a : `type(3)=DOM_INT=2` et `type("a")=DOM_STRING=12`

8.4.8 Les sorties : `print, Disp, afficher`

`print` (ou `Disp` ou `afficher`) a comme argument une séquence de chaîne de caractères ou de noms de variables.
`print` (ou `Disp` ou `afficher`) écrit en bleu les chaînes de caractères et le contenu des variables dans l'écran situé avant la réponse.
On tape :

`a:=12`

puis :

`print("a=",a)`

On obtient :

`"a=",12 écrit en bleu et la réponse est 1`

8.4.9 Pour effacer les sorties : `ClrIO`

`ClrIO` n'a pas d'argument, il efface toutes les sorties faites dans le niveau où il a été tapé.
On tape :

`a:=12`

puis :

`print("a=",a)`

On obtient :

`"a=",12 s'écrit en bleu et la réponse est 1`

On tape :

`print("a=",a);ClrIO()`

On obtient :

`(1,1) comme réponse`

8.4.10 Les sorties de a^b : `printpow`

L'instruction `printpow` modifie la façon dont les puissances sont affichées en mode texte ou après une commande `print`.
`printpow` a comme argument soit `0`, soit `1`, soit `-1`.
`printpow(0)` écrit a^b selon l'écriture mathématique `a^b`.
`printpow(1)` écrit a^b selon l'écriture en langage C `pow(a,b)`.
`printpow(-1)` écrit a^b selon l'écriture d'autres langages `a**b`.
De plus ces 3 syntaxes sont admises en input.
On tape :

```
printpow(0)
```

puis :

```
print(2^3)
```

On obtient :

```
2^3 écrit en bleu et 8 comme réponse
```

On tape :

```
printpow(1)
```

puis :

```
print(2^ 3)
```

On obtient :

```
pow(2,3) écrit en bleu et 8 comme réponse
```

On tape :

```
printpow(-1)
```

puis :

```
print(2^3)
```

On obtient :

```
2**3 écrit en bleu et 8 comme réponse
```

8.4.11 Sortie dans une petite fenêtre : `output Output`

`output` ou `Output` a comme argument une séquence de chaîne de caractères ou de noms de variables.
`output` ou `Output` écrit les chaînes de caractères et le contenu des variables dans une petite fenêtre (la même que celle qui s'ouvre lors d'un `input`). Cela permet donc d'inclure un `output` dans un `input`.
On tape :

```
input(output("Calcul de
p(a)"),"polynome",p,"valeur",a)
```

On obtient :

```
une fenêtre ayant pour titre Calcul de p(a), où on
peut entrer la valeur de p (par exemple x->x+1) et a
(par exemple 2)
```

8.4.12 Les affectations infixées : `=>` `:=` `=<`

`:=` `=>` `=<` sont des opérateurs infixés qui permettent de stocker une valeur dans une variable, mais leurs arguments de `=>` et de `:=` `=<` ne sont pas dans le même ordre !
De plus `:=` et `=<` n'ont pas le même effet lorsque le premier argument est un élément d'une liste (ou matrice) contenu dans une variable. En effet, `=<` modifie lélément d'une liste (ou matrice) par référence (voir 8.4.14)

- `=>` est la version infixée de `sto`.
  `=>` stocke le premier argument dans la variable donnée comme deuxième argument.
  On tape :
  `4=>a`
  Ou on tape :
  `sto(4,a)`
  On obtient :
  `4 et la variable a contient 4`
- `:=` et `=<` ont comme arguments, un nom de variable et la valeur à stocker.
  On tape :
  `a:=4`
  Ou on tape :
  `a=<4)`
  On obtient :
  `4 et la variable a contient 4`
  On tape :
  `A:=[0,1,2,3,4]`
  `B:=A`
  puis par exemple pour modifier `A[3]`, on tape :
  `A[3]=<33`
  puis
  `A,B`
  On obtient :
  `[0,1,2,33,4],[0,1,2,33,4]`
  Les variables `A` et `B` contiennent toutes les deux la liste `[0,1,2,33,4]`, en effet :
  `A:=[0,1,2,3,4]` fait pointer `A` sur la liste `[0,1,2,3,4]`, puis
  `B:=A` fait pointer `B` sur `A` donc sur la liste `[0,1,2,3,4]`,
  `A[3]=<33` modifie la liste pointée par `A` donc modifie la liste `[0,1,2,3,4]` en la liste `[0,1,2,33,4]` et donc `B` est modifié. Il en serait de même si on avait mis `B[3]=<33` au lieu de `A[3]=<33`, les variables `A` et `B` contiendraient toutes les deux la liste `[0,1,2,33,4]`.

Remarque
On peut faire plusieurs affectations à la fois en utilisant les séquences ou les listes.
On tape :

`[a,b,c]:=[1,2,3]`

On obtient :

`[1,2,3]`

`a contient 1, b contient 2 et c contient 3`

Ou on tape :

`(a,b,c):=(1,2,3)`

On obtient :

`(1,2,3)`

`a contient 1, b contient 2 et c contient 3`

Attention
Si `a` contient `5` et que l'on tape :
`(a,b):=(2,a)`
alors `a` contient `2` et `b` contient `5`
Si `a` contient `1` et que l'on tape :
`(a,b):=(2,a)`
alors `a` contient `2` et `b` contient `1`
`(a,b):=(b,a+b)`
alors `a` contient `1` et `b` contient `3`

8.4.13 L'affectation par copie : `copy`

`copy` permet de cloner une liste, un vecteur ou une matrice, en général pour le stocker dans une variable.
`copy` a un argument, la liste (vecteur ou matrice) à cloner.
`A:= copy(B)` recopie la liste (vecteur, matrice) B dans A.
On tape :

`B:=copy([[4,5],[2,6]])`

ou :

`A:=[[4,5],[2,6]];B:=copy(A)`

Puis on tape :

`A,B`

On obtient :

`[[4,5],[2,6]],[[4,5],[2,6]]`

Quand doit-on utiliser `copy` ?
`copy` est surtout utile pour des listes ou des matrices qui seront par la suite modifiées par référence avec =<.
On tape :

`A:=[0,1,2,3,4]`

`B:=copy(A)`

Puis, si on tape :

```
A[3]=<33
A,B
```

On obtient :

```
[0,1,2,33,4],[0,1,2,3,4]
```

Puis si on tape :

```
B[3]=<55
A,B
```

On obtient :

```
[0,1,2,33,4],[0,1,2,55,4]
```

8.4.14 Les différences entre := , =< et `copy`

Attention, := et =< ne sont pas équivalents car =< modifie lélément d'une liste ou matrice par référence.

L'affectation =< ne diffère de := que si on modifie un élément d'une liste (ou matrice) contenue dans une variable, par exemple si `A` contient la liste `[0,1,2,3,4]` i.e. si on a tapé `A:=[0,1,2,3,4]` et que l'on veut modifier la valeur de `A[3]` et changer `3` en `33`, on peut écrire dans ce cas, `A[3]:=33` ou `A[3] =< 33` mais ces deux instructions sont différentes. En effet `A` pointe vers une liste car `A:=[0,1,2,3,4]`, et `A[3] =< 33` modifie cette liste en la liste `[0,1,2,33,4]`, ainsi toutes les variables de `Xcas` qui pointent vers cette liste seront modifiées. Alors qu'avec `A[3]:=33` la liste est dupliquée, la copie est modifiée et `A` pointe vers la copie. Il faut donc utiliser =< avec précautions car tous les objets pointant sur cette liste (ou matrice) seront modifiés.

- On tape :
  ```
  A:=[0,1,2,3,4]
  B:=A
  A[3]=<33
  A,B
  ```
  ou :
  ```
  A:=[0,1,2,3,4]
  B:=A
  B[3]=<33
  A,B
  ```
  On obtient :
  ```
  [0,1,2,33,4],[0,1,2,33,4]
  ```
  En effet la liste `[0,1,2,3,4]` a été modifiée par l'instruction `A[3]=<33` (ou par `B[3]=<33`) en la liste `[0,1,2,33,4]`. Les listes `A` et `B` pointent sur cette liste donc `A` et `B` sont modifiées toutes les deux.
- On tape :
  ```
  A:=[0,1,2,3,4]
  ```

```
B:=A
A[3]:=33
A,B
```

On obtient :

```
[0,1,2,33,4],[0,1,2,3,4]
```

car `A[3]:=33` fait une copie de la liste `[0,1,2,3,4]` et modifie cette copie en `[0,1,2,33,4]` puis `A` pointe sur cette copie et cela ne modifie pas `B`. On tape :

```
A:=[0,1,2,3,4]
B:=A
B[3]:=33
A,B
```

On obtient :

```
[0,1,2,3,4],[0,1,2,33,4]
```

car `B[3]:=33` fait une copie de la liste `[0,1,2,3,4]` et modifie cette copie en `[0,1,2,33,4]` puis `B` pointe sur cette copie et cela ne modifie pas `A`.

– On tape :

```
A:=[0,1,2,3,4]
B:=copy(A)
```

Puis, si on tape :

```
A[3]=<33
A,B
```

On obtient :

```
[0,1,2,33,4],[0,1,2,3,4]
```

On tape :

```
A:=[0,1,2,3,4]
B:=copy(A)
B[3]=<33
A,B
```

On obtient :

```
[0,1,2,3,4],[0,1,2,33,4]
```

En effet `B` pointe sur la copie de `A`, donc une modifiucation par référence de `A[3]` (resp`B[3]`) ne modifie pas `B` (resp `A`).

8.4.15 Les instructions `copy` et =< dans un programme

=< est surtout utile quand on fait beaucoup de modifications dans une liste ou matrice de grande taille, par exemple dans un programme. Mais attention, il faudra utiliser `copy` lors de l'initialisation des variables listes ou matrices qui seront modifiées avec =< pour que les modifications se fassent sur la copie.
Voir aussi 2.5.3 et 8.4.14.
Attention
Dans un programme si on met :

```
local a;
a:=[0,1,2,3,4];
...
a[3] =< 33;
```

Cela modifie le programme lui-même et la ligne `a:=[0,1,2,3,4];` est remplacée par `a:=[0,1,2,33,4];` dans le programme compilé.
Donc dans un programme il faut écrire :

```
local a;
a:=copy([0,1,2,3,4]);
...
a[3] =< 33;
```

ou

```
local a,c;
c:=[0,1,2,3,4];
a:=copy(c);
...
a[3] =< 33;
```

pour que ce soit la copie de `[0,1,2,3,4]` qui soit modifiée et non la liste `[0,1,2,3,4]` elle-même.

Remarque

On peut aussi utiliser les commandes `makelist`, `seq` ou `$`, par exemple :
`a:=makelist(n,n,0,4);` ou
`a:=[n$(n=0..4)];` puis
`a[3] =< 33;` car `makelist(n,n,0,4)` ou `[n$(n=0..4)]` est une liste qui va se créer à chaque exécution du programme.

Exemple

On veut écrire un programme qui renvoie les nombres entiers inférieurs ou égaux à n et dont la somme des chiffres (en base 10) vaut 5. On tape :

```
somme5(n):={
local L,j,k;
L:=seq(0,0);
k:=0;
for(j:=0;j<=n;j++){
if (sum(convert(j,base,10))==5){
L[k]=<j;
k:=k+1;
}
}
return L;
}:;
```

On tape :

```
somme5(50)
```

On obtient :

```
5,14,23,32,41,50
```

Dans cet exemple, il faut bien comprendre et faire l'initialisation de `L` correctement.
Dans ce programme, on a choisit d'avoir le résultat sous la forme d'une séquence, et on a initialisé `L` avec `L:=seq(0,0)`. Si on avait mis `L:=NULL`, le programme n'aurait pas fonctionné correctement ! En effet, si on tape :

```
fauxsom5(n):={
local L,j,k;
L:=NULL;
k:=0;
for(j:=0;j<=n;j++){
if (sum(convert(j,base,10))==5){
L[k]=<j;
k:=k+1;
}
}
return L;
}:;
```

puis :

`fauxsom5(30)`

on obtient :

`5,14,23,32,41,50`

Cela semble juste..... Mais, lors de l'exécution de `fauxsom5(50)`, au départ `L` pointe sur la séquence `NULL`, puis cette séquence `NULL` est modifiée et devient à la fin du programme `5,14,23,32,41,50` et le programme, si on ne le recompile pas, a lui aussi été modifié est est devenu

```
fauxsom5(n):={
local L,j,k;
L:=5,14,23,32,41,50;
k:=0;
for(j:=0;j<=n;j++){
if (sum(convert(j,base,10))==5){
L[k]=<j;
k:=k+1;
}
}
return L;
}:;
```

Si on tape maintenant :

`fauxsom5(30)`

On obtient une séquence trop longue :

`5,14,23,32,41,50`

alors que si on tape :

`somme5(30)`

On obtient :

`5,14,23`

Régle
Dans un programme, si on veut utiliser `=<` pour modifier une liste `L` ou une séquence `LL`, il faut faire attention à l'initialisation.
Par exemple, pour initialiser

- `L` à la liste vide, on tape :
  `L:=[seq(0,0)]` ou `L:=[0$0]` ou `L:=copy([])` car `[seq(0,0)]` ou `[0$0]` désigne une liste qui va se créer à chaque exécution du programme.
- `LL` à la séquence `NULL`, on tape :
  `LL:=seq(0,0)]` ou `LL:=0$0` ou `LL:=copy(NULL)` car `seq(0,0)` ou `0$0` désigne une séquence qui va se créer à chaque exécution du programme.

Il faut donc utiliser `=<` avec précautions car tous les objets pointant sur cette matrice seront modifiés.

8.4.16 L'affectation prefixée : `sto Store`

`sto` (ou `Store`) permet de stocker une valeur dans une variable.
`sto` (ou `Store`) a deux arguments, la valeur à stocker et un nom de variable.
Attention
La variable doit être purgée avant l'utilisation de `sto` (ou `Store`).
On tape :

`sto(3,a)`

ou :

`Store(3,a)`

On obtient :

`3 et la variable a contient 3`

Remarque
On peut faire plusieurs affectations à la fois en utilisant les séquences ou les listes.
On tape :

`sto([1,2,3],[a1,a2,a3])`

ou :

`Store([1,2,3],[a1,a2,a3])`

On obtient :

`[1,2,3] et a1 contient 1, a2 contient 2, a3 contient 3`

8.4.17 L'affectation d'une égalité : `assign`

`assign` permet de stocker une valeur dans une variable.
`assign` a un ou deux arguments :

- un argument : une égalité entre un nom de variable et la valeur à stocker, ou une listed'égalité entre des noms de variable et les valeurs à stocker,
- deux arguments : un nom de variable et la valeur à stocker.

Attention
La variable doit être purgée avant l'utilisation de `assign`.
On tape :

```
assign(a,3)
```

ou :

```
assign(a=3)
```

On obtient :

```
3 et la variable a contient 3
```

On tape :

```
assign([a1=1,a2=2,a3=3)
```

On obtient :

```
[1,2,3] et a1 contient 1, a2 contient 2 et a3 contient 3
```

Remarque `assign` sert surtout en mode Maple, voici un exemple : On tape en mode Maple :

```
purge(a,b)
```

```
sol:=solve([a+b=1,a-b=3],[a,b])
```

On obtient :

```
[a=2,b=(-1)]
```

On tape :

```
assign(sol)
```

On obtient :

```
[2,-1] et ainsi a contient 2 et b contient -1
```

en mode `Xcas` la même suite d'instructions donnent :

```
sol:=solve([a+b=1,a-b=3],[a,b])
```

On obtient la liste des solution donc ici une matrice 1×1 :

```
[[2,-1]]
```

On tape :

```
[a,b]:=sol[0])
```

On obtient :

```
[2,-1] et ainsi a contient 2 et b contient -1
```

8.4.18 L'instruction conditionnelle : `if then else end, si alors sinon fsi`

`if` est suivi d'un booléen entre des parenthèses.
`if` permet de faire, ou de ne pas faire, un bloc d'instructions, selon la valeur de ce booléen.
`if` utilisé avec `else` permet de choisir de faire, soit un bloc d'instructions, soit un autre bloc d'instructions , selon la valeur du booléen.
`fi` est suivi d'un booléen et de `then`.
Selon la valeur de ce booléen `if` permet de faire, ou de ne pas faire, les instructions situées entre `else` et `end`.
Selon la valeur de ce booléen `if` utilisé avec `else` permet de choisir de faire, soit les instructions situées entre `then` et `else`, soit les instructions situées entre `else` et `end`.
`si` est suivi d'un booléen et de `alors`.
Selon la valeur de ce booléen `si` permet de faire, ou de ne pas faire, les instructions situées entre `alors` et `fsi`.
Selon la valeur de ce booléen `si` utilisé avec `sinon` permet de choisir de faire, soit les instructions situées entre `alors` et `sinon`, soit les instructions situées entre `sinon` et `fsi`.
On tape :

```
a:=3;b:=2;
```

puis :

```
if (a>b) {a:=a+5;b:=a-b;}
```

On obtient :

```
6
```

On tape :

```
a:=3;b:=2;
```

puis :

```
if a>b then a:=a+5;b:=a-b;end
```

On obtient :

```
6
```

On tape :

```
a:=3;b:=2;
```

puis :

```
si a>b alors a:=a+5;b:=a-b;fsi
```

On obtient :

```
6
```

On tape :

```
maxi(a,b):={if (a>b) {return a;} else {return b;}}
```

puis :

```
maxi(2,3)
```

On obtient :

```
3
```

On tape :

```
maxi(a,b):={if a>b then return a; else return b;end}
```

puis :

```
maxi(2,3)
```

On obtient :

```
3
```

On tape :

```
maxi(a,b):={si a>b alors retourne a; sinon retourne
b;fsi}
```

puis :

```
maxi(2,3)
```

On obtient :

```
3
```

8.4.19 L'instruction conditionnelle : `if then elif else end`

Lorsqu'il y a plusieurs `if...else if...` à la suite on peut aussi utiliser un `elif` qui est une écriture condensée de `else if`. L'instruction `elif` permet une écriture plus lisible car un seul `end` suffit pour clore le `if`. Le dernier `elif` peut comporter ou ne pas comporter un `else`. Voici la syntaxe :

```
if (condition1) then
action1;
elif (condition2) then
action2;
elif (condition3) then
action3;
end
```

ou bien `if (condition1) then`

```
action1;
elif (condition2) then
action2;
```

```
elif (condition3) then
action3;
else action4;
end
```

On tape par exemple pour définir la fonction f définit par :

- $f(x) = 8$ si $x > 8$
- $f(x) = 4$ si $4 < x \geq 8$
- $f(x) = 2$ si $2 < x \geq 4$
- $f(x) = 1$ si $0 < x \geq 2$
- $f(x) = 0$ si $x \leq 0$

```
f(a):= {
  if a>8 then
    return 8;
  elif a>4 then
    return 4;
  elif a>2 then
    return 2;
  elif a>0 then
    return 1;
  elif a<=0 then
    return 0;
  end;
}:;
```

ou plutôt en utilisant un `else` à la place du dernier `elif`, on tape :

```
f(a):= {
  if a>8 then
    return 8;
  elif a>4 then
    return 4;
  elif a>2 then
    return 2;
  elif a>0 then
    return 1;
  else
    return 0;
  end;
}:;
```

8.4.20 L'instruction conditionnelle : `switch`

`switch` a comme paramètre une expression.
Selon la valeur de cette expression, marquée par `case` (ou par `default`), `switch` permet de faire le bloc d'instructions correspondant à la valeur d'une expression qui doit être une valeur entière.
On tape :

```
opere(a,b,c):={
switch(c) {
  case 1 : {a:=a+b;break;}
  case 2 : {a:=a-b;break;}
  case 3 : {a:=a*b;break;}
  default : {a:=a^b;}
}
return a;
}
```

puis :

`opere(2,3,1)`

On obtient :

`5`

On tape :

`opere(2,3,2)`

On obtient :

`-1`

8.4.21 La boucle : `for pour fpour`

`for` et `pour` se sert d'une variable d'incrémentation par exemple `j` (doit être déclarée comme variable locale).

Première syntaxe

On précise entre des parenthèses et en les séparant par un point virgule, la valeur de départ, la condition d'arrêt et la façon dont on incrémente la variable d'incrémentation, puis on met un bloc d'instructions :

`for` permet de faire plusieurs fois un bloc d'instructions selon la valeur de la variable d'incrémentation.

`for (j:=j1;j<=j2;j:=j+3) {<instructions> }`

Par exemple :

`S:=0;for (j:=3;j<20;j:=j+3) {S:=S+j;}`

Deuxième syntaxe

On commence par `for`, puis, on précise avec les mots `from`, `to` et `step`, la valeur de départ, la condition d'arrêt et la faç on dont on incrémente cette variable d'incrémentation, puis on met les instructions à effectuer entre `do` et `end_for` :

`for j from j1 to j2 step p do <instructions> end_for;`

Ou on utilise sa traduction. On commence par `pour`, puis, on précise avec les mots `de`, `jusque` et `pas`, la valeur de départ, la condition d'arrêt et la façon dont on incrémente cette variable d'incrémentation, puis on met les instructions à effectuer entre `faire` et `fpour` :

`pour j de j1 jusque j2 pas p faire <instructions> fpour;`

Par exemple :

`S:=0;for j from 3 to 20 step 3 do S:=S+j; end_for;`

`S:=0;pour j de 3 jusque 20 pas 3 faire S:=S+j; fpour;`

Troisième syntaxe

On peut à l'aide d'un ensemble, d'une liste ou d'une plage de valeurs `A` donner les différentes valeurs que doit prendre la variable d'incrémentation `j` en mettant :
`for j in A do ....end_for;`.
ou sa traduction :
`pour j in A faire <instructions> fpour;` Par exemple lorsque `j` s'incrémente de 1 :
`S:=0;for j in 1..5 do S:=S+j; end_for;`
ou sa traduction
`S:=0;pour j in 1..5 faire S:=S+j; fpour;`
lorsque `j` décrit un ensemble de valeurs :
`A:=[3,6,9,15,30,45];S:=0;for j in A do S:=S+j; end_for;`
ou sa traduction
`A:=[3,6,9,15,30,45];S:=0;pour j in A faire S:=S+j; fpour;`
ou
`A:=%3,6,9,15,30,45%;S:=0;for j in A do S:=S+j; end_for;`
ou sa traduction
`A:=%3,6,9,15,30,45%;S:=0;pour j in A faire S:=S+j; fpour;`

Attention

Ne pas choisir `i` comme variable d'incrémentation car `i` représente un nombre complexe ! ! !
On tape :

```
somfor(n):={
local j,s:=0;
for (j:=1;j<=n;j++){
   s:=s+1/j;
}
return s;
}
```

```
sompour(n):={
local j,s:=0;
pour j de 1 jusque n faire
   s:=s+1/j;
fpour
retourne s;
}
```

puis :

`somfor(5)`

ou

`sompour(5)`

On obtient :

`137/60`

8.4.22 La fonction : `seq`

`seq` n'est pas une instruction mais une fonction qui permet de renvoyer la liste constituée par les différentes valeurs du premier argument lorsque le deuxième argument varie selon les valeurs des arguments suivants : valeur de départ, valeur d'arrivée, pas (pas=1 par défaut).
`seq(f(k),k,1,3)=[f(1),f(2),f(3)]`
`seq(f(k),k,1,5,2)=[f(1),f(3),f(5)]`
La fonction `seq` est utile pour tracer une suite de points dans les écrans de géométrie.
Exemple
On veut représenter les 10 premiers termes de la suite :
$u_n = (1 + \frac{1}{n})^n = f(n)$ par les points $n + i * f(n)$.
On ouvre un écran de géomérie et on tape :

```
f(n):=(1+1/n)^n
```

```
seq(point(k+i*f(k)),k,1,10)
```

On obtient :

```
On voit les points dans cet écran de géomérie
```

Si on tape :

```
for (k:=1;k<11;k++) {point(k+i*f(k));}
```

On obtient :

```
les points sont seulement dans l'écran DispG qu'on
ouvre avec Cfg->Montrer->DispG ou avec DispG()
```

Mais si on ouvre un écran de géomérie et si on tape :

```
L:=[];for (k:=1;k<11;k++)
{L:=append(L,point(k+i*f(k)));}:;L;
```

On obtient :

```
On voit les points dans cet écran de géomérie
```

On peut aussi utiliser la syntaxe comme avec Maple, `seq(2^k,k=0..8)`, en ajoutant éventuellement un pas `seq(2^k,k=0..8,1)`

8.4.23 La boucle : `repeat until` et `repeter jusqua`

`repeat` et `repeter` permettent de faire plusieurs fois des instructions.
`repeat` est suivi d'instructions, de `until` et d'un booléen.
`repeter` est suivi d'instructions, de `jusqua` et d'un booléen.
`repeat until` et `repeter jusqua` permettent de faire plusieurs fois ces instructions jusquà ce que la valeur de ce booléen soit vraie.
On tape :

```
repeat
saisir("Entrez a>1",a);
until a>4;
```

Ou on tape :

```
repeter
saisir("Entrez a>4",a);
jusqua a>1;
```

Cela aura pour effet de redemander la valeur de a jusqu'à ce qu'elle soit strictement supérieur à 4.

8.4.24 La boucle : `while` et `tantque`

`while` et `tantque` permettent de faire plusieurs fois des instructions.
`while` est suivi d'un booléen mis entre des parenthèses et d'un bloc d'instructions.
`while` permet de faire plusieurs fois ce bloc d'instructions tant que la valeur de ce booléen est vrai.
`tantque` est suivi d'un booléen, de `faire` d'instructions puis de `ftantque`.
`tantque` permet de faire plusieurs fois les instructions situèes entre `faire` et `ftantque` tant que la valeur de ce booléen est vrai.
On tape :

```
somwhile(eps):={
local s,j;
s:=0;
j:=1;
while (1/j>=eps){
 s:=s+1/j;
 j:=j+1;
}
return s;
}
```

Ou on tape :

```
somtantque(eps):={
local s,j;
s:=0;
j:=1;
tantque 1/j>=eps faire
 s:=s+1/j;
 j:=j+1;
ftantque
retourne s;
}
```

puis :

```
somwhile(0.2)
```

ou

```
somantque(0.2)
```

On obtient :

```
137/60
```

Remarque
On peut aussi écrire un while avec un for :
while (<condition>)<instructions> est équivalent à :
for (;<condition>;)<instructions>
On aurait donc pu écrire :

```
somwhilefor(eps):={
local s,j;
s:=0;
j:=1;
for (;1/j>=eps;){
 s:=s+1/j;
 j:=j+1;
}
return s;
}
```

ou encore

```
somforwhile(eps):={
local s,j;
s:=0;
for (j:=1;1/j>=eps;j++){
 s:=s+1/j;
}
return s;
}
```

On tape :

```
somwhile(0.1);somforwhile(0.1);somwhilefor(0.1)
```

On obtient :

```
7129/2520,7129/2520,7129/2520
```

8.4.25 Modifier l'ordre d'exécution des instructions : label goto

label permet de repérer une instruction dans un programme par un nom de variable.
goto permet de modifier l'ordre d'exécution des instructions, en exécutant les instructions du programme à partir d'une instruction repérée par un label.
label a comme arguments, une suite de caractères : par exemple label(truc) ou label(1).
goto a comme arguments un nom de label : par exemple goto(truc) ou goto(1).
On tape :

```
essai(eps):={
local s,j;
s:=0;
j:=0;
label(truc);
j:=j+1;
s:=s+1/j;
if (1/j>=eps) goto(truc);
print(s);
return 0;
}
```

puis :

```
essai(0.2)
```

On obtient :

```
s:137/60 écrit en bleu, puis 0 comme réponse
```

8.5 Les autres instructions

8.5.1 Pour lire les entrées à partir d'un fichier : `read`

`read` a comme argument le nom d'un fichier (fichier crée par `write` ou par la sauvegarde d'un éditeur et dans lequel on a écrit, par exemple, plusieurs fonctions ou instructions).
`read` renvoie la liste des valeurs stockées dans le fichier.
On tape :

```
a:=3.14; b:=456;write("titi",a,b);
```

puis :

```
read("titi")
```

On obtient :

```
[3.14,456] et les variables a et b sont affectées par
3.14 et 456
```

Attention
Si `a` et `b` existent leurs valeurs seront remplacées sans préavis par les valeurs stockées dans le fichier :
le principe de `read` est que l'utilisateur met dans un fichier par exemple des programmes ou des grosses variables, donc il sait ce qu'il y a dedans, et lors d'un `read` les instructions sont exécutées et seules les reponses sont affichées.

8.5.2 Pour écrire des variables et leur contenu dans un fichier : `write`

`write` a comme argument un nom de fichier et des noms de variables.
`write` crée un fichier du nom spécifié (et donc efface le fichier précédent, si celui-ciexistait déjà) et écrit les affectations de ces variables dans ce fichier .
On tape :

```
a:=3.14; b:=456
```

puis :

```
write("titi",a,b)
```

On obtient :

```
le fichier "titi" dans lequel il y a a:=3.14; b:=456;
```

On tape :

```
testwrite():={
local a;
a:=[];
for (k:=1;k<5;k++){a:=concat(a,k*6.55);}
write("titi",a);
return a;
}
```

puis :

```
testwrite()
```

On obtient :

```
le fichier "titi" dans lequel il y a,
      a:=[6.55,13.1,19.65,26.2]
```

8.5.3 Pour écrire des sorties dans un fichier : `fopen fprint fclose`

Voici les commandes pour ecrire dans un fichier :
`f:=fopen("essai")` crèe un fichier du nom spécifié ici `essai` (et donc l'efface si celui-ci existait déjà) et lui associer la variable `f`.
`fprint(f,12)` écrit 12 dans le fichier `essai`.
`fclose(f)` ferme le fichier `essai`.
On tape :

```
testfprint(nom):={
 local a,f;
 a:=[];
 f:=fopen(nom);
 for (k:=1;k<11;k++){
 fprint(f,k);
 }
 fclose(f);
 return a;
 }
```

puis :

```
testfprint("toto")
```

On obtient :

```
la création d'un fichier "toto" dans lequel il y a
12345678910
```

8.5.4 Pour utiliser une chaîne comme nom de variable ou comme nom de fonction : `#`

`#` permet d'utiliser une chaîne de caractères comme nom de variable ou comme nom de fonction ou comme nom d'instruction ou comme nom de répertoire.
`#` est surtout utile dans un programme.
`#` a comme argument une chaîne ou le nom d'une variable qui contient une chaîne ou une expression renvoyant une chaîne.
`#` transforme la chaîne en une expression mais n'évalue pas cette expression. Donc, pour faire une affectation, on ne doit pas écrire `#"a:=2"`, mais on peut écrire `#"a":=2` (voir aussi `expr` 8.5.6).
On tape :

```
a:="abc";#a:=3
```

ou :

```
#"abc":=3
```

On obtient :

```
La variable abc contient 3
```

On tape :

```
b:="ifactor";(#b)(54)
```

ou :

```
(#"ifactor")(54)
```

On obtient :

```
2*3^3
```

On tape :

```
#"ifactor(54)"
```

On obtient :

```
ifactor(54)
```

8.5.5 Pour utiliser une chaîne comme un nombre : `expr`

`expr` permet d'utiliser une chaîne de chiffres ne commençant pas par un zéro comme un nombre entier écrit en base 10 ou une chaîne de chiffres comportant un point comme un nombre décimal écrit en base 10.
On tape :

```
expr("123")+1
```

On obtient :

```
124
```

On tape :

```
expr("45.67")+2.12
```

On obtient :

```
47.79
```

`expr` permet d'utiliser une chaîne de chiffres ne comportant pas de `8`, ni de `9`, et commençant par un zéro comme un nombre entier écrit en base `8`.
On tape :

```
expr("0123")
```

On obtient :

```
83
```

En effet $1 * 8^2 + 2 * 8 + 3 = 83$
Remarque :
Si on tape `expr("018")` on obtient le nombre décimale `18.0`.
`expr` permet d'utiliser une chaîne contenant des chiffres et les lettres `a,b,c,d,e,f`, et commençant par `0x` comme un nombre entier écrit en base 16.
On tape :

```
expr("0x12f")
```

On obtient :

```
303
```

En effet $1 * 16^2 + 2 * 16 + 15 = 303$ Voir aussi 6.5.16

8.5.6 Pour utiliser une chaîne comme nom de commande : `expr`

`expr` permet d'utiliser une chaîne de caractères comme une commande.
`expr` est surtout utile dans un programme.
`expr` a comme argument une chaîne de caractères pouvant être interprétée comme une commande (ou le nom d'une variable qui contient une chaîne ou une expression renvoyant une chaîne).
`expr` transforme la chaîne en une expression, puis évalue cette expression :
pour faire une affectation, on ne doit pas écrire `expr("a"):=2`, mais on doit écrire `expr("a:=2")` (voir aussi `expr` 8.5.4)
On tape :

```
expr("c:=1")
```

On obtient :

```
La variable c contient 1
```

On tape :

```
a:="ifactor(54)";expr(a)
```

ou :

```
expr("ifactor(54)")
```

On obtient :

```
2*3^3
```

8.5.7 Évaluer une expression sous la forme d'une chaîne : `string`

`string` évalue une expression et renvoie sa valeur sous la forme d'une chaîne de caractères.
On peut aussi utiliser la concaténation de l'expression avec une chaîne vide.
On tape :

```
string(ifactor(6))
```

Ou on tape :

```
ifactor(6)+""
```

Ou on tape :

```
""+ifactor(6)
```

On obtient :

```
"2*3"
```

On tape :

```
string(quote(ifactor(6)))
```

On obtient :

```
"ifactor(6)"
```

8.6 D'autres instructions utiles

8.6.1 Définir une fonction ayant un nombre variable d'arguments : `args`

`args` ou `args(NULL)` renvoie la liste des arguments d'une fonction sous forme d'une liste : l'élément d'indice `0` est la fonction, les suivants sont les arguments passés à la fonction. Cela permet de travailler avec des fonctions ayant un nombre d'arguments non fixé a l'avance.
Remarque on ne peut pas mettre `args()` mais `args` ou `args(NULL)` (`NULL` est obligatoire) on peut aussi utiliser `(args)[0]` pour désigner la fonction, `(args)[1]` pour désigner le nom du premier argument... mais il ne faut pas oublier les parenthèses autour de `args` !
On tape :

```
testargs():={ local y; y:=args; return y[1];}
```

```
testargs(12,5)
```

On obtient :

```
12
```

On tape :

```
somme():={
 local s,a;
 a:=args;
 s:=0;
 for (k:=1;k<size(a);k++){
 s:=s+a[k];
 }
 return s;
 }
```

puis :

```
somme(1,2,3,4)
```

On obtient :

```
10
```

8.6.2 Pour sortir d'une boucle : `break`

`break` n'a pas d'argument ni parenthèse.
`break` permet de sortir d'une boucle.
On tape :

```
testbreak(a,b):={
local r;
while (1==1){
  if (b==0){break;}
```

```
  r:=irem(a,b);
  a:=b;
  b:=r;
}
return a;
}
```

puis on tape :

`testbreak(4,0)`

On obtient :

```
4 car il y a eu l'interruption de la boucle while
quand b==0
```

8.6.3 Pour ne pas faire la fin d'une boucle : `continue`

`continue` n'a pas d'argument ni parenthèse.
`continue` dans une boucle, permet d'ignorer la fin de l'itération et ainsi de passer directement à l'itération suivante.
On tape :

```
testcont(b):={
local a;
a:=10;
while (a>0) {
a:=a-3;
if (a>=b) {continue;}
print(a);
}
}
```

puis on tape :

`testcont(4)`

On obtient :

`a:1 et a:-2 écrit en bleu et 1 comme réponse`

car `a` est imprimé que si `a<b=4`

8.6.4 Ouvrir l'ècran `DispG` depuis un programme : `DispG`

`DispG` n'a pas d'arguments.
`DispG;` permet d'ouvrir l'écran `DispG` depuis un programme.
On tape (ne pas mettre de parenthèses) :

`DispG;`

On obtient :

`l'ouverture de l'écran DispG`

8.6.5 Effacer l'ècran `DispG` depuis un programme : `ClrGraph`

`ClrGraph` n'a pas d'arguments.
`ClrGraph` permet d'ouvrir l'écran `DispG` depuis un programme.
On tape :

```
ClrGraph()
```

Ou on tape :

```
ClrGraph
```

On obtient :

```
l'effacement de l'écran DispG
```

8.6.6 Fermer l'ècran `DispG` depuis un programme : `DispHome`

`DispHome` n'a pas d'arguments.
`DispHome` permet de fermer l'écran `DispG` depuis un programme.
On tape (ne pas mettre de parenthèses) :

```
DispHome;
```

On obtient :

```
l'écran DispG se ferme
```

8.7 Le debuggeur

8.7.1 Ouvrir le débuggeur : `debug`

`debug` a comme argument une fonction et ses arguments.
`debug` ouvre l'écran de la fenêtre de mise au point (ou débugger) avec sa barre de boutons. On a la possibilité :

- d'exécuter le programme au pas à pas avec le bouton `sst` qui inscrit la commande `sst()` dans la ligne de commande,
- de mettre des points d'arrêts (breakpoint) avec `break` (ou la commande `breakpoint`),
- d'aller directement avec`cont` à une ligne précise marquée par un point d'arrêt (ou la commande `cont`),
- de voir avec `watch` les variables que l'on désire surveiller (ou la commande `watch`),
- d'exécuter au pas à pas les instructions d'une fonction utilisateur avec `dans` qui inscrit la commande `sst_in()` dans la ligne de commande, ou encore
- de sortir brutalement du débuggeur avec `tuer` (ou la commande `kill`).

On tape par exemple :
`debug(pgcd(15,25))`.
Il faut bien sûr que le programme `pgcd` existe.
Par exemple on a tapé :

```
pgcd(a,b):={
local r;
while(b!=0) {
r:=irem(a,b);
a:=b;
b:=r;
}
return a;
}
```

Puis si on veut observer les variables `a` et `b`, on clique sur `watch` : `watch` s'inscrit dans la ligne `eval` et on compléte cette ligne en `watch(a)` puis `enter`, puis on clique sur `watch` et on compléte `watch` en `watch(b)` puis `enter`.
Ensuite on clique sur `sst`, et on voit à chaque ètape (la ligne qui est exécutée est en surbrillance) les valeurs de `a` et `b` dans la zone située en dessous de la barre des boutons.

8.7.2 Instruction du debuggeur : `watch`

`watch` a comme argument le nom des variables que l'on désire surveiller.
On tape par exemple :

`watch(a,b)`

On obtient :

`l'évolution de a et de b lors du déroulement du programme`

8.7.3 Instruction du debuggeur : `rmwatch`

`rmwatch` a comme argument le nom des variables que l'on ne désire plus surveiller.
On tape par exemple :

`rmwatch(a,b)`

On obtient :

`l'évolution de a et de b lors du déroulement du programmen'est plus visible`

8.7.4 Instruction du debuggeur : `breakpoint`

`breakpoint` a comme argument le numéro de la ligne où l'on veut un point d'arrêt.
On tape par exemple :

`breakpoint(4)`

On obtient :

`cont() provoquera le déroulement du programme et un arrêt à la ligne 4`

8.7.5 Instruction du debuggeur : `rmbreakpoint`

`rmbreakpoint` a comme argument le numéro de la ligne où l'on ne veut plus un point d'arrêt.
On tape par exemple :

```
rmbreakpoint(4)
```

On obtient :

```
le point d'arrêt à la ligne 4 a été effacé
```

8.7.6 Instruction du debuggeur : `cont`

`cont` n'a pas d'argument.
`cont()` est une instruction du debuggeur et permet de continuer un programme arrêté, par un point d'arrêt, dans le debuggeur.
On tape :

```
cont()
```

On obtient :

```
le déroulement du programme jusqu'au prochain point
d'arrêt
```

8.7.7 Instruction du debuggeur : `kill`

`kill` n'a pas d'argument.
`kill()` est une instruction du debuggeur et permet de sortir du debuggeur.
On tape :

```
kill()
```

On obtient :

```
on sort du débuggeur
```

8.7.8 Instruction en vue d'un debugage : `halt`

`halt` n'a pas d'argument.
`halt()` est une instruction dans un programme pour programmer un point d'arrêt dans le debuggeur.
On tape dans un programme :

```
halt();
```

On obtient lors du debogage de ce programme :

```
un arrêt du déroulement du programme à cet endroit
```

8.7.9 Utilisation des instructions du debuggeur : `cont halt kill`

Les instructions `cont halt kill` n'ont pas d'arguments.
`cont halt kill` sont des instructions du debuggeur (voir ci-dessus).
`cont()` et `kill()` s'utilisent uniquement dans une session de debogage, quand le programme est arrêté. Typiquement, `kill()` s'utilise quand on a vu où se trouve l'erreur, et que on va savoir corriger le programme, et qu'il est donc inutile de continuer l'exécution du programme buggué.
`cont()` s'utilise pour atteindre le point d'arrêt suivant.
`halt()` peut être mis comme instruction dans le programme pour programmer un point d'arrêt (cela évite de faire une commande `breakpoint` avec un numéro de ligne) Par exemple on tape :

```
testhalt(x):={
  local y;
  y:=x;
  halt();
  return(y);
}
```

On tape :

`debug(testhalt(5))`

On obtient :

`l'ouverture du debuggeur et l'arrêt du programme`

On tape :

`cont() pour continuer le debugage ou kill() pour l'arrêter`

Attention
Si on tape juste `testhalt(5)` sans mettre `debug`, le debuggeur s'ouvre mais, la liste des instructions formant le programme n'est pas affichée donc il vaut mieux faire `debug(testhalt(5))` puis enlever les `halt()` quand le programme est au point.

8.7.10 Avoir un arrêt momentané : `Pause`

`Pause` est une instruction qui permet de suspendre l'exécution d'un programme.
`Pause` n'a pas d'argument ou a comme argument le temps de pause en secondes : on ne met pas de parenthèses, on écrit :
`Pause;` ou `Pause n;`
Lorsque `Pause` n'a pas d'argument (ne pas mettre de parenthèses), une fenêtre $\boxed{\textbf{i}\ \texttt{Paused}}$ s'ouvre et l'exécution reprend lorsque l'utilisateur appuie sur `Enter` ou sur `Close` de cette fenêtre.
On tape : `a:=21;b:=5` puis,

`print(a);Pause;print(b);`

On obtient :

`21 s'écrit en bleu, un arrêt puis la fenêtre` **i** `Paused` `s'ouvre, on tape sur Close et 5 s'écrit en bleu et on a (1,0,1) comme réponse`

On tape : `a:=21;b:=5` puis,

```
print(a);Pause 3;print(b);
```

On obtient :

`21 s'écrit en bleu, un arrêt de 3 secondes, puis 5 s'écrit en bleu et on a (1,0,1) comme réponse`

8.7.11 Avoir un arrêt momentané : `WAIT`

`WAIT` est une instruction qui permet de suspendre l'exécution d'un programme.
`WAIT` a comme argument le temps de pause en secondes.
On tape : `a:=21;b:=5` puis,

```
print(a);WAIT(3);print(b);
```

On obtient :

`21 s'écrit en bleu, un arrêt de 3 secondes, puis 5 s'écrit en bleu et on a (1,0,1) comme réponse`

8.7.12 Intercepter une erreur : `try..catch`

On fait le bloc d'instructions après `try` : en cas d'erreur le message d'erreur est stocké dans la variable `err` sous forme d'une chaine, et `catch` exécute le bloc d'instructions après `catch (err)`.
On tape :

```
try{A:=idn(2)*idn(3)}
catch(erreur)
{print("l'erreur est "+erreur)}
```

On obtient :

```
"l'erreur est * Invalid dimension"
```

On tape :

```
essai(x):={
local y,err;
try {y:=[[1,2]]+x;}
catch (err){y:="l'erreur est "+err;}
return y;}
```

On tape :

```
essai([1])
```

On obtient :

`"l'erreur est + Bad Argument Value"`

On tape :

`essai([[3,4]])`

On obtient :

`[[4,6]]`

8.7.13 Générer une erreur : `throw error ERROR`

`throw` ou `error` ou `ERROR` permet de générer une erreur en provoquant l'affichage d'une erreur.
On tape :

```
f(x):={
if (type(x)!=DOM_INT)
  throw(erreur);
else
  return x;
}
```

puis

`f(1.2)`

On obtient :

`"erreur Error: Bad Argument Value"`

puis

`f(12)`

On obtient :

`12`

On peut capter l'erreur grace à cette fonction `f` que l'on utilise dans la fonction `g` suivante :

```
g(x):={
try { f(x); } catch (err){ x:=0; }
return x;
}
```

puis

`g(1.2)`

On obtient :

`0`

puis

`g(12)`

On obtient :

`12`

La fonction `g(x)` renvoie, `x` si `x` est un entier, et `0` sinon.

Chapitre 9

Les fonctions de géométrie 2-d

9.1 Généralités

Toutes les commandes graphiques faites dans une ligne d'entrée auront en réponse l'ouverture d'un écran graphique.
Les dessins de la géométrie 2-d se font en général dans un écran de `géométrie 2-d` qui est un écran graphique interactif muni d'un éditeur de commandes et d'une barre de menus (on ouvre un écran de `géométrie 2-d` avec `Alt+g`).
Les dessins faits dans un écran de `géométrie 2-d` sont interactifs : on peut définir des points et des segments avec la souris et modifier une figure en faisant bouger un de ses points avec la souris.

9.2 Les fonctions de base

9.2.1 Effacer l'écran `DispG` : `erase`

`erase` n'a pas d'argument.
On rappelle que l'écran `DispG` est l'écran sur lequel est enregistré toutes les commandes graphiques depuis le début de la session, sans distinction de niveau. Il permet en particulier de visualiser les affichages graphiques intermédiaires d'un programme (en effet seuls les objets graphiques renvoyés par `return` ou `retourne` peuvent être affichés en réponse dans un niveau où on exécute un programme).
`erase` efface l'écran `DispG` et influe sur la commande `graph2tex` car seules les sorties graphiques faites postérieurement à la commande `erase()` sont prises en compte par la commande `graph2tex`.
On tape :

```
erase
```

ou :

```
erase()
```

On obtient :

```
L'effacement de l'écran DispG et efface l'historique
interne des graphiques
```

Remarque
`erase` n'efface pas un écran de géométrie, en effet on n'a pas besoin d'effacer un écran de géométrie car soit on change ses commandes d'entrée, soit on ouvre un autre écran de géométrie pour faire une nouvelle figure.

9.2.2 Effacer les axes : `switch_axes`

`switch_axes` met ou enlève les axes de l'écran géométrique.
Vous pouvez aussi effacer les axes (resp les faire réapparaitre) en cochant (resp décochant) `Montrer les axes` avec le bouton `cfg` de l'écran de géométrie.
`switch_axes` a comme paramètres `1` pour mettre les axes, `0` pour les enlever, ou aucun paramètre, dans ce cas, `Xcas` met les axes, si ils n'y sont pas, et les enlève si ils y sont.
On tape :

```
switch_axes(0)
```

On obtient :

```
L'effacement des axes de l'écran graphique
```

On tape ensuite :

```
switch_axes()
```

Ou

```
switch_axes(1)
```

On obtient :

```
Le tracé des axes de l'écran graphique
```

9.2.3 Tracer les vecteurs unitaires : `vecteur_unitaire_Ox_2d` `Ox_2d_unit_vector`, `vecteur_unitaire_Oy_2d`, `Oy_2d_unit_vector`

`vecteur_unitaire_Ox_2d(), Ox_2d_unit_vector()` trace le vecteur unitaire de l'axe des x de écran de géométrie 2-d.
`vecteur_unitaire_Oy_2d(), Oy_2d_unit_vector()` trace le vecteur unitaire de l'axe des y de écran de géométrie 2-d.
Vous pouvez effacer les axes (resp les faire réapparaitre) en cochant (resp décochant) `Montrer les axes` avec le bouton `cfg` de l'écran de géométrie.
Ces commandes n'ont pas de paramètre. On peut mettre une légende avec la commande `legende`.
On tape :

```
vecteur_unitaire_Ox_2d(),legende(1,"u",vert,quadrant4)
```

On obtient :

Le vecteur unitaire de l'axe des x de écran de géométrie 2-davec u écrit en vert

On tape :

`vecteur_unitaire_Oy_2d(),legende(i,"v",vert,quadrant2)`

On obtient :

Le vecteur unitaire de l'axe des y de écran de géométrie 2-d avec v écrit en vert

9.2.4 Tracer un repère : `repere_2d frame_2d`

`repere_2d() frame_2d()` trace le repère de écran de géométrie 2-d.
Vous pouvez effacer les axes (resp les faire réapparaitre) en cochant (resp décochant) `Montrer les axes` avec le bouton `cfg` de l'écran de géométrie.
Ces commandes n'ont pas de paramètre.
On tape :

`repere_2d()`

On obtient :

Le repère de écran de géométrie 2-d

9.2.5 Effacer les points définis par `TX` et `TY`

Lorsque les axes sont dessinés, les points ayant pour coordonnées des multiples de `TX` et de `TY` sont dessinés.
Pour effacer ces points, en gardant les axes il suffit de mettre 0 dans `TX` et/ou dans `TY` : on définit `TX` et `TY` avec la configuration graphique.

9.2.6 Avoir du papier pointé : `papier_pointe dot_paper`

`papier_pointe` permet d'avoir du papier pointé c'est à dire de tracer un reseau de points sur des droites horizontales et sur des droites de pente $\tan(t)$.
`papier_pointe` a 3,4 ou 5 arguments : l'unité `ux` sur l'axe des x, l'angle `t` et l'unité `uy` sur l'axe des y et éventuellement `x=xmin..xmax,y=ymin..ymax` pour n'avoir que des points dans le rectangle `[xmin..xmax]*[ymin..ymax]`.
Par défaut `xmin,xmax,ymin,ymax` sont les valeurs définies dans `Cfg->Configuration graphique`.
Vous avez la possibilité d'enlever les axes orthonormés en utilisant le bouton `cfg` et en décochant `Montrer les axes`.
On tape pour avoir les sommets d'un réseau :

`papier_pointe(0.6,pi/2,0.6)`

On obtient :

Le pointage de tous les sommets du reseau formé par des carrés dans le rectangle $[xmin, xmax] \times [ymin, ymax]$

On tape :

`papier_pointe(1,pi/3,sqrt(3)/2,x=-2..3,y=-1..2)`

On obtient :

Le pointage de tous les sommets du reseau formé par des triangles équilatéraux dans le rectangle $[-2, 3] \times [-1, 2]$

9.2.7 Avoir du papier avec des lignes : `papier_ligne` `line_paper`

`papier_ligne` permet d'avoir du papier avec des lignes espacées de ux sur l'axe des x et faisant un angle t avec l'horizontale.
`papier_ligne` a 2, 3 ou 4 arguments : l'unité `ux` sur l'axe des x, l'angle `t` formé par l'axe des y et des x et éventuellement `x=xmin..xmax,y=ymin..ymax` pour avoir ces lignes dans le rectangle `[xmin..xmax]*[ymin..ymax]`. Par défaut `xmin,xmax,ymin,ymax` sont les valeurs définies dans `Cfg->Configuration graphique`.
Vous avez la possibilité d'enlever les axes orthonormés en utilisant le bouton `cfg` et en décochant `Montrer les axes`.
On tape :

```
papier_ligne(0.6,pi/3)
```

On obtient :

des lignes espacées de 0.6 sur l'axe des x et faisant un angle de $\pi/3$ avec Ox dans le rectangle $[xmin, xmax] \times [ymin, ymax]$

On tape

```
papier_ligne(0.6,pi/3,x=-2..3,y=-1..2)
```

On obtient :

des lignes espacées de 0.6 sur l'axe des x et faisant un angle de $\pi/3$ avec Ox dans le rectangle $[-2, 3] \times [-1, 2]$

9.2.8 Avoir du papier quadrillé : `papier_quadrille` `grid_paper`

`papier_quadrille` permet d'avoir du papier quadrillé c'est à dire de tracer un reseau formé de droites horizontales et de droites de pente $\tan(t)$.
`papier_quadrille` a 3,4,5 arguments : l'unité `ux` sur l'axe des x, l'angle `t` et l'unité `uy` sur l'axe des y et éventuellement `x=xmin..xmax,y=ymin..ymax` pour avoir ces lignes dans le rectangle `[xmin..xmax]*[ymin..ymax]`. Par défaut `xmin,xmax,ymin,ymax` sont les valeurs définies dans `Cfg->Configuration graphique`.
Vous avez la possibilité d'enlever les axes orthonormés en utilisant le bouton `cfg` et en décochant `Montrer les axes`.
On tape pour avoir un quadrillage :

```
papier_quadrille(1,pi/2,1)
```

On obtient :

Le reseau formé par des carrés dans le rectangle $[xmin, xmax] \times [ymin, ymax]$

On tape pour avoir un quadrillage dans un rectangle :

```
papier_quadrille(1,pi/3,sqrt(3)/2,x=-2..3,y=-1..2)
```

On obtient :

Le reseau formé par des losanges d'angle $\pi/3$ dans le rectangle $[-2,3] \times [-1,2]$

9.2.9 Avoir du papier triangulé : papier_triangule triangle_paper

papier_triangule permet d'avoir du papier triangulé c'est à dire de tracer un reseau triangulaire d'angle t.
papier_pointe a 3,4 ou 5 arguments : l'unité ux sur l'axe des x, l'angle t et l'unité uy sur l'axe des y et éventuellement x=xmin..xmax,y=ymin..ymax pour avoir ces lignes dans le rectangle [xmin..xmax]*[ymin..ymax]. Par défaut xmin,xmax,ymin,ymax sont les valeurs définies dans Cfg->Configuration graphique.
Vous avez la possibilité d'enlever les axes orthonormés en utilisant le bouton cfg et en décochant Montrer les axes.
On tape pour avoir un réseau triangulaire :

```
papier_triangule(1,pi/2,1)
```

On obtient :

Le reseau formé par des moitiés de carrés et dans le rectangle $[xmin,xmax] \times [ymin,ymax]$

On tape pour avoir un reseau triangulaire :

```
papier_triangule(1,pi/3,sqrt(3)/2,x=-2..3,y=-1..2)
```

On obtient :

Le reseau formé par des triangles équilatéraux dans le rectangle $[xmin,xmax] \times [ymin,ymax]$

9.2.10 Changer les paramètres de la fenêtre graphique : xyztrange

xyztrange met ou enlève les axes de l'écran géométrique. xyztrange a 15 paramètres ce sont des paramètres que l'on peut définir dans l'écran de la configuration du graphique. On ouvre cet écran avec la configuration graphique (voir 1.6.2).
On tape ou plutôt on utilise le menu Cfg :

```
xyztrange(-5,5,-5,2,-10,10,-1,6,-5,5,-1.2384,2,1,0,1)
```

On obtient :

La fenêtre visible est [-5;5]*[-1.2834;2]

Attention à la différence entre "fenêtre de calcul" et "fenêtre visible" : si la fenêtre de calcul est plus grande, on pourra aller voir ce qui se passe "ailleurs" avec les flèches faisant parties des boutons de l'écran géométrique.
De plus, si on a demandé un repère orthonormé en cliquant sur $\perp$, la fenêtre visible demandée sera contenue dans ce qui est réellement visible.

9.3 Les attributs des objets graphiques

9.3.1 Généralités sur les attributs

Les attributs des objets graphiques peuvent être définis par des commandes ou avec des paramètres optionnels qui sont soit globaux soit locaux.
Certains arguments optionnels, comme `titre, labels`, sont globaux, ils sont mis au début d'une ligne de commandes ayant comme dernière instruction une instruction graphique et sont applicables à toutes les instructions graphiques de cette ligne de commandes. Par exemple :
`titre="A et son conjugué B";A:=point(1+i);B:=point(1-i).`
Ils sont déscrits ci-après.
Certains noms comme `affichage, couleur, legende` peuvent être des noms de commandes ou des noms de paramètres optionnels :

- Les commandes s'appliquent à l'objet graphique mis comme argument ou, s'il n'y a pas d'arguments, modifient les attributs par défaut qui seront utilisés pour les instructions graphiques qui suivent.
  On tape :
  `couleur(rouge)`
  On obtient :
  `1 et toutes les commandes graphiques sans paramètre optionnel de couleur seront dessinées en rouge`
  On tape :
  `couleur(triangle(-1-i,1,1+i),bleu)`
  On obtient :
  `le triangle(-1-i,1,1+i) est dessiné en bleu`
- Il y a deux sortes d'attributs : les attributs globaux qui sont partagés par toutes les commandes graphiques affichées dans un même écran graphique et les attributs individuels.
  On tape (attribut individuel) :
  `triangle(0,1,1+i,couleur=bleu)`
  On obtient :
  `le triangle(0,1,1+i) dessiné en bleu`
  On tape (attribut global) :
  `titre="medianes";triangle(-1-i,1,1+i); mediane(-1-i,1,1+i); mediane(1,-1-i,1+i); mediane(1+i,1,-1-i)`
  On obtient :
  `le titre "medianes" pour le triangle(0,1,1+i) et ses médianes`

9.3.2 Les commandes d'attributs

Les attributs des objets graphiques peuvent être définis par des commandes qui sont :

- `legend` ou `legende`,
- `display` ou `affichage` ou `color` ou `couleur`

Les paramètres de ces commandes sont :
`objet_graphique,nom_de_l'attribut=valeur1+valeur2+...`
et les valeurs de cet attribut sont valables uniquement pour l'objet graphique spécifié.
Par exemple :
```
legende(point(1+i),"A",rouge,quadrant3,epaisseur_point_2)
affichage(point(1+i),rouge+epaisseur_point_6+point_carre)
affichage(point(1+i,legend="A"),epaisseur_point_2+point_carre)
```
Remarque
Si la commande `display(..)` ou `affichage(...)` est tapée dans une ligne de commandes, on peut utiliser ces commandes avec comme paramètre
`valeur1+valeur2+...` et
alors la commande est globale pour tous les objets définis ensuite dans cette ligne de commandes. Il faut éviter d'utiliser ces commandes de cette façon dans un écran de géométrie, car les attributs sont déjà définis globalement dans `cfg` et le bouton des attributs.

9.3.3 Les paramètres optionnels d'attributs

Les attributs des objets graphiques peuvent aussi se définir à partir de paramètres optionnels. Ils sont de la forme `nom_de_l'attribut=valeur1+valeur2+...`.
Il y a deux sortes de paramètres optionnels :
- les paramètres optionnels globaux sont mis au début d'une ligne de commandes ayant comme dernière instruction une instruction graphique et sont applicables à toutes les instructions graphiques de cette ligne de commandes,
- les paramètres optionnels locaux sont passés en paramètre d'une commande graphique.

La liste des attributs et leurs valeurs possibles sont :
- `display=filled` ou `affichage=rempli` pour dessiner des figures pleines (est utilisé seulement comme attribut local),
- `axes=1` ou `axes=true` et `axes=0` ou `axes= false` montre ou cache les axes. Cela est utilisé seulement comme attribut global et suivi d'une commande graphique,
- `couleur=0`...`couleur=255` ou `color=0`...`color=255` pour dessiner avec la couleur indiquée (est utilisé seulement comme attribut local),
- `couleur=` une valeur entière entr 256 et 256+7*16+14 pour dessiner avec une couleur de l'arc en ciel,
- `epaisseur=1`... `epaisseur=7` ou `thickness=1`... pour faire des traits plus ou moins épais. `epaisseur` est un attribut pour compatibilité `Maple` (est utilisé seulement comme attribut local),
- `nstep=400` permet de spécifier le nombre de points d'échantillonage de la fonction à représenter en 3-d (est utilisé seulement comme attribut local),
- `tstep=0.3` permet de donner le saut d'échantillonnage d'un paramètre de tracé paramétrique en général `t`, par exemple
  `plotparam(sin(3*t)+i*cos(3*t),t=0..2*pi/3,tstep=0.01)`
  (est utilisé seulement comme attribut local),
- `ustep=0.3` permet de donner le saut d'échantillonnage de la première variable d'un tracé paramétrique 2-d, en général `u`, (est utilisé seulement

comme attribut local),
- `vstep=0.3` permet de donner le saut d'échantillonnage de la deuxième variable d'un tracé paramétrique 2-d, en général `v`, (est utilisé seulement comme attribut local),
- `xstep=0.01` permet de donner le saut d'échantillonnage de la variable `x` (est utilisé seulement comme attribut local),
- `ystep=0.01` permet de donner le saut d'échantillonnage de la variable `y` (est utilisé seulement comme attribut local),
- `zstep=0.01` permet de donner le saut d'échantillonnage de la variable `z` (est utilisé seulement comme attribut local),
- `frames=10` or `trames=10` permet de spécifier le nombre de graphes calculés dans une animation de graphe par les commandes `animate` ou `animate3d` (est utilisé seulement comme attribut local),
- `labels=["u","v"]` permet de renommer les axes (est utilisé seulement comme attribut global),
- `legende` ou `legend` (est utilisé comme attribut global ou local ou comme commande) :
  - `legende=["mn","kg"]` (utilisé comme attribut global) permet de mettre comme légende le nom des unités sur les axes, à condition qu'en dehors d'un écran de géométrie, cette commande soit suivie sur la même ligne par une commande graphique. Par exemple :
    `legende=["mn","kg"];point(1+i)`, ou
    `legend=["xunit","yunit","zunit"]; point([1,1,1])`
  - `legende(1+i*sin(1),"sin(1)")` ou
    `legende(point(1+i*sin(1)),"sin(1)")` (utilisé comme commande) ou
    `point(1+i*sin(1),legende="sin(1)")` (utilisé comme attribut local) permet de mettre une légende en un point de l'écran, donné par son affixe. On peut spécifier la couleur ou la position de la légende par rapport au point en mettant d'autres paramètres par exemple :
    `legende(1+i*sin(1),"sin(1)",quadrant4,rouge)`
    (utilisé comme commande) ou
    `point(1+i*sin(1),`
    `legende="sin(1)",affichage=quadrant4+rouge)`
    (utilisé comme attribut local)
  - `legende([20,60],"graphe")` (utilisé comme commande) permet de mettre une légende en un point de l'écran, donnè par ses coordonnées en pixels ([0,0] se trouve en haut et à gauche de l'écran),
  - `polygone(-1,-i,1,2*i,legende="p")` permet de mettre une légende au milieu du côté reliant le dernier au premier sommet du polygone ici -1,2*i (utilisé comme attribut local),
  - `point([i,1,2,2*i],`
    `legende="1+i",display=quadrant2+rouge)`
    permet de mettre une légende au milieu du segment 2,2+i,
- `style` n'a qu'une valeur reconnue : `point` (c'est pour la compatibilité `Maple`) et `style=point` (utilisé comme attribut local) permet de dessiner une droite selon des pointillés. Ces pointillés correspondent à `dashdot_line` ou à `ligne_tiret_point`. On a :

`droite(y=x,style=point,affichage=vert+line_width_2)`
est identique à :
`droite(y=x,affichage=vert+ligne_tiret_point+ line_width_2)`,

- `titre="..."` ou `title="..."` permet de mettre un titre à la fen'être graphique (est utilisé seulement comme attribut global).
- `gl_texture=` peut être un attribut local ou global.
`gl_texture="nom_fichier_image"` permet de mettre une image sur un objet graphique 3-d ou sur un rectangle 2-d si il est utilisé comme attribut local. Par exemple `carre(i,1+i,gl_texture="image.jpg")` dessine l'image contenue dans le fichier `"image.jpg"` dans le carré, et `gl_texture=` permet de mettre un fond sur un graphique 2-d si il est utilisé comme attribut global. Par exemple `gl_texture="image.jpg"; carre(i,1+i)` dessine un carré avec comme fond d'écran l'image contenu dans le fichier `"image.jpg"`
- `gl_x_axis_name="xname"`, `gl_y_axis_name="yname"`, `gl_z_axis_name=""` : utilisé comme attribut global, définit individuellement les noms des axes x, y, z,
- `gl_x_axis_unit="xunit"`, `gl_y_axis_unit="yunit"`, `gl_z_axis_unit="zunit"` : utilisé comme attribut global, définit individuellement les unités des axes x, y, z,
- `gl_x_axis_color=n`, `gl_y_axis_color=n`, `gl_z_axis_color=n` : utilisé comme attribut global, définit individuellement les couleurs des axes x, y, z avec la couleur `n`,
- `gl_x=xmin..xmax`, `gl_y=ymin..ymax`, `gl_z=zmin..zmax` : utilisé comme attribut global, definit la configuration du graphique (pas compatible avec l'interactivité),
- `gl_xtick=`, `gl_ytick=`, `gl_ztick=` : utilisé comme attribut global met des marques sur les axes selon la valeur donnée,
- `gl_shownames=true or false` : utilisé comme attribut global, montre ou cache les noms des objets,
- `gl_rotation=[x,y,z]` : utilisé comme attribut global, definit l'axe de rotation pour les animations des scénes 3-d,
- `gl_quaternion=[x,y,z,t]` : utilisé comme attribut global definit le quaternion pour la visualisation des scénes 3-d (pas compatible avec l'interactivité),
- d'autres options d' OpenGL pour configurer la lumière existent mais ne sont pas décrites ici.

Remarque
Les options d'opengl sont à utiliser avec une grande prudence car elles ne sont pas compatibles avec l'interactivité.

9.3.4 Mettre une légende : `legende`

`legende` a deux ou trois paramètres :

- le 1er paramètre est un point (ou son affixe) ou une liste de 2 entiers. Cet argument permet de savoir où l'on doit mettre la légende : soit à partir du point soit à partir de la position en pixels donnée par la liste de 2 coordonnées

entières (en partant du coin en haut à gauche et les y dirigés vers le bas).
- le 2ième paramètre est la légende (soit une chaîne de caractères, soit le nom d'une variable : dans ce cas `legende` affiche le contenu de la variable),
- éventuellement un 3ième paramètre pour indiquer le quadrant dans lequel il faut mettre la légende et dont la valeur est :
  `quadrant1,quadrant2,quadrant3,quadrant4`
  (par défaut c'est `quadrant1`).

`legende` met la chaîne de caractères à cet endroit donné. situé en haut à gauche et les y sont dirigés vers le bas).
On tape :

```
legende(1+i,"bonjour")
```

On obtient :

```
Le point 1+i et "bonjour" écrit à partir du point 1+i
dans le premier quadrant
```

On tape :

```
legende(1+i,"bonjour",quadrant4)
```

Ou on tape :

```
legende(1+i,quadrant4,"bonjour")
```

On obtient :

```
Le point 1+i et "bonjour" écrit dans le quatrième
quadrant du point 1+i
```

On tape :

```
legende([30,20],"bonjour")
```

On obtient :

```
"bonjour" écrit à partir du point en pixels x=30 et
y=20
```

On tape :

```
legende([30,20],"bonjour",quadrant4)
```

On obtient :

```
"bonjour" écrit à partir du point en pixels x=30 et
y=20 dans le quatrième quadrant
```

Pour mettre une légende à un angle voir 9.17.5
Attention
Le nom `legend` (ou `legende`) peut servir de paramètre global pour mettre en légende le nom des unités sur les axes. Dans ce cas, `legende`) est suivi de = puis d'un vecteur composé de 2 chaines qui sont le nom des unités (on met une chaine vide si on ne veut plus avoir l'unité). Le nom `legend` (ou `legende`) peut aussi

servir de paramètre local qui sera le paramètre optionnel d'une commande `point` ou d'une commande traçant un polygone (`triangle...polygone`). Dans ce cas, la légende s'inscrit au niveau du point ou au niveau du milieu du côté reliant le dernier au premier sommet du polygone. Le nom du paramètre optionnel (ici unité). Le nom `legend` ou `legende`) est suivi de = puis de la légende (soit une chaine de caractères, soit un nom de variable contenant une chaine de caractères) est le paramètre que l'on passe comme argument supplémentaire dans les commandes graphiques.
On tape :

```
legende=["mn","kg"]
```

On obtient :

$-6mn$ $6mn$ selon l'axe des x et $-5kg$ $5kg$ selon l'axe des y

On tape :

```
point(1+i,legende="bonjour")
```

Ou on tape :

```
a:="bonjour";point(1+i,legende=a)
```

On obtient :

```
Le point 1+i et "bonjour" écrit au point 1+i
```

On tape :

```
polygone(-1,-i,1,2*i,legende="P")
```

Ou on tape :

```
a:="P";polygone(-1,-i,1,2*i,legende=P)
```

On obtient :

```
Le polygone -1,-i,1,2*i et "P" écrit au point -1/2+i
```

Remarque
Pour mettre une legende a un angle on utilise `angle` avec comme dernier paramàtre une chaine vide (dans ce cas un petit arc désignera l'angle) ou une chaine que sera la legende (dans ce cas un petit arc et la chaine désignera l'angle) par exemple :
`angle(0,1,1+i,"")` ou `angle(0,1,1+i,"a")` trace l'angle et
`angle(0,1,1+i,"")[0]` ou `angle(0,1,1+i,"a")[0]` renvoie $\pi/4$. Mais **Attention** `angle(0,1,1+i,"")` ou `angle(0,1,1+i,"a")` renvoie une figure géométrique i.e. le trac̀e de l'angle. Donc si vous voulez que `a` désigne la valeur de l'angle et avoir aussi le dessin il faut taper par exemple :
`a:=angle(0,1,1+i);` et `angle(0,1,1+i,"a")`

9.3.5 Les commandes d'affichage : `display color affichage couleur`

`affichage couleur` peut avoir comme paramètre différentes valeurs, reliées par +, permettant de gérer les attributs des graphiques à venir d'une ligne de commandes normale (la commande est alors globale) ou de gérer les attributs d'un objet graphique ou d'une légende (la commande est alors locale).
Attention
`affichage(0)` remet l'affichage par défaut : couleur noire, figure non pleine, les points ont des légendes...

Pour ne pas afficher le nom de l'objet

`hidden_name` est une valeur de paramètre de la commande `affichage` qui permet de ne pas afficher le nom de l'objet. On tape (ici la commande `affichage` est locale au point `A`) :

```
affichage(A:=point(1+i),hidden_name);B:=point(-1);
D:=droite(A,B)
```

On obtient :

```
A est dessiné sans son nom et B et la droite A,B sont
dessinés avec leur nom
```

On tape (ici la commande `affichage` est locale) :

```
A:=point(1+i);B:=point(-1);affichage(D:=droite(A,B),hidden_name)
```

On obtient :

```
Les points A et B et leurs noms et la droite A,B
dessinée sans son nom
```

Attention NE PAS mettre `affichage(hidden_name)` comme commande globale car après, aucun nom sera visible, et seule la commande `affichage(0)` remettra l'affichage par défaut.

Mettre de la couleur

Un entier entre 0 et 255 (ou un nom de couleur) représente une couleur de la palette des couleurs que l'on peut voir avec le bouton des couleurs de l'écran de géométrie et, un entier entre 256 et 381 représente les couleurs de l'arc en ciel. Pour voir les couleurs de l'arc en ciel il suffit d'exécuter le programme suivant :

```
arcenciel():={
local j,C;
C:=[];
for (j:=256;j<382;j++){
C:=append(C,carre(j,j+1,couleur=j+rempli));
}
C;
}
```

Pour voir l'arc en ciel, on tape `a:=arcenciel():;`, puis `a` pour voir toutes les couleurs (le numéro d'une couleur est alors égal à son abscisse) ou par exemple `a[300-256]` pour voir la couleur de numéro 300.

Autre méthode, on tape :

```
ciel():={ local j;
for (j:=0;j<126;j++)
carre(j,(j+1),affichage=rempli+256+j);}
```
puis,
`ClrGraph;ciel()` et on ouvre la fenêtre `DispG` avec le menu `Cfg` sous menu `Montrer` puis `DispG`. On regle `cfg` de cette fenêtre en prenant `WX-=-1`, `WX+=127`, `WY-=0` et `WY+=1` et on voit ainsi les 126 couleurs de l'arc en ciel !

La commande `couleur` ou `affichage` avec comme paramètre un entier entre 0 et 381 (ou un nom de couleur) permet de tracer un objet géométrique avec la couleur spécifiée.
`couleur` ou `affichage` a un argument (resp deux arguments) : le nom de la couleur ou l'entier désignant cette couleur (noir=0, rouge=1, vert=2, jaune=3, bleu=4) (resp l'objet géométrique ou une légende et le nom de la couleur ou le numéro de la couleur).

Attention

Le nom `couleur` ou `affichage` ou display ou `color` peut aussi servir de paramètre optionnel d'une commande graphique. Dans ce cas, le nom du paramètre optionnel (ici `couleur` ou `affichage` ou `color`) est suivi de = puis de la valeur de l'entier désignant la couleur, est le paramètre que l'on passe comme argument supplémentaire dans les commandes graphiques.
On tape :

```
couleur(segment(0,1+i),rouge)
```

Ou on tape :

```
segment(0,1+i,couleur=rouge)
```

On obtient :

```
Le segment(0,1+i) est dessiné en rouge
```

On tape :

```
couleur(point(1+i),2)
```

Ou on tape :

```
point(1+i, couleur=2)
```

On obtient :

```
Le point 1+i apparait avec une croix (x) verte
```

On tape :

```
affichage(rouge)
```

On obtient :

```
après cette instruction, les dessins resultant de
     commandes graphiques se feront en rouge
```

On tape :

```
couleur(legende(1+i,"AA"),rouge)
```

Ou on tape :

```
legende(1+i,"AA",couleur=rouge)
```

On obtient :

```
Le point(1+i) dessiné en rouge a comme légende rouge
                          AA
```

Pour dessiner une figure pleine

`rempli` ou `filled` est une valeur de paramètre de la commande `affichage` qui permet d e dessiner une figure pleine. On tape (ici la commande `affichage` est globale) :

```
affichage(rempli);carre(0,1+i);triangle(-1,0,-i);
```

On obtient :

```
le carre(0,1+i) plein et le triangle(-1,0,-i) plein
```

Les différents affichages de points

`couleur` ou `affichage` peut avoir comme argument la forme de la représentation des points 2-d à l'affichage. Ces formes peuvent avoir des traits plus ou moins épais selon la valeur de `n` de `point_width_n`.
La représentation des points 3-d a comme forme un carré (c'est `point_carre`) plus ou moins gros selon la valeur de `n` de `point_width_n`.
Les différentes options sont :

- `point_losange` ou `rhombus_point` pour représenter un point par un petit losange ($\diamond$),
- `point_carre` ou `square_point` pour représenter un point par un petit carré ($\square$),
- `point_croix` ou `cross_point` pour représenter un point par une croix ($\times$) c'est la représentation par défaut,
- `point_etoile` ou `star_point` pour représenter un point par une étoile (*),
- `point_plus` ou `plus_point` pour représenter un point par le signe plus (+),
- `point_point` pour représenter un point par un point ($\cdot$).
- `point_triangle` ou `triangle_point` pour représenter un point par un triangle ($\triangle$),
- `point_invisible` ou `invisible_point` pour ne pas représenter un point.

Les différentes épaisseurs sont :
`point_width_1` ou `epaisseur_point_1`,`point_width_2` ou `epaisseur_point_2`...`point_width_8` ou `epaisseur_point_8`

Attention

Le nom `couleur` ou `color` ou `affichage` est aussi le nom d'un paramètre que l'on passe comme argument supplémentaire dans les commandes géométriques lorsqu'il est suivi de =, puis de la valeur d'un attribut puis de +, puis de la valeur d'un autre attribut etc...

On tape :

```
couleur(point(1+i),rouge+point_carre+point_width_3)
```

Ou on tape :

```
point(1+i,affichage=rouge+point_carre+point_width_3)
```

On obtient :

```
Le point (1+i) est dessiné en rouge à l'aide d'un
petit carré
```

On tape :

```
affichage(point_carre)
```

On obtient :

```
après cette instruction, les points seront dessinés à
l'aide d'un petit carré
```

Les différents formes d'affichage des lignes

`couleur` ou `affichage` peut avoir comme argument la forme de la représentation des lignes à l'affichage.

Les différentes options sont :

- `solid_line` ou `ligne_trait_plein` est un argument de `couleur` et permet de dessiner une ligne en trait plein (c'est la représentation par défaut),
- `dash_line` ou `ligne_tiret` est un argument de `couleur` et permet de dessiner une ligne en pointillé.
- `dashdot_line` ou `ligne_tiret_point` est un argument de `couleur` et permet de dessiner une ligne avec un tiret et un point,
- `dashdotdot_line` ou `ligne_tiret_pointpoint` est un argument de `couleur` et permet de dessiner une ligne avec un tiret et deux points,
- `cap_flat_line` ou `ligne_chapeau_plat` est un argument de `couleur` et permet de dessiner une ligne plate,
- `cap_round_line` ou `ligne_chapeau_rond` est un argument de `couleur` et permet de dessiner une ligne tubulaire,
- `cap_square_line` ou `ligne_chapeau_carre` est un argument de `couleur` et permet de dessiner une ligne à section carr'ee.

On tape :

```
couleur(segment(0,1+i),rouge+dash_line)
```

On obtient :

```
Le segment (0,1+i) est dessiné en rouge en pointillé
```

Les différentes épaisseurs d'affichage des lignes

`couleur` ou `affichage` peut avoir comme argument l'épaisseur de la représentation des lignes à l'affichage.
Les différentes options d'épaisseurs sont :
`line_width_1`, `line_wdith_2`,...`line_width_8` ou
`epaisseur_ligne_1`, `epaisseur_ligne_2`,...`epaisseur_ligne_8`.
`line_width_n` est un argument de `couleur` ou `affichage` et permet de dessiner une ligne plus ou moins epaisse.
On tape :

```
couleur(droite(y=1),rouge+line_width_1)
```

On obtient :

```
La droite (y=1) est dessinée en rouge et en trait fin
```

On tape :

```
couleur(droite(y=2),rouge+line_width_7)
```

On obtient :

```
La droite (y=2) est dessinée en rouge et en trait très
épais
```

9.4 Comment définir un objet géométrique sans le tracer : `nodisp`

`nodisp` appliqué à une commande permet de ne pas afficher la réponse, même quand il s'agit d'une commande graphique.
On peut aussi terminer la commande par `:;` pour ne pas générer de réponse.
On tape :

```
C:=point(1+i)
```

```
cercle(C,sqrt(2))
```

On obtient :

```
Le point C d'affixe 1+i et le cercle de centre C et de
rayon sqrt(2) sont tracés
```

On tape :

```
nodisp(C:=point(1+i))
```

Ou on tape :

```
C:=point(1+i):;
```

puis

```
cercle(C,sqrt(2))
```

On obtient :

```
Seul le cercle de centre 1+i et de rayon sqrt(2) est
tracé
```

9.5 Comment définir et tracer sans nom, un objet géométrique

`nodisp` permet de définir un objet géométrique sans le tracer. On peut ensuite tracer l'objet en mettant comme commande son nom, mais alors, son nom n'apparaitra pas sur la figure.
On peut aussi définir un objet géométrique et utiliser `eval`. On tape :

```
nodisp(C:=point(1+i))
```

Ou on tape :

```
C:=point(1+i):;
```

puis on tape :

```
C
```

Ou on tape :

```
eval(C:=point(1+i))
```

On obtient :

```
Le point 1+i est marqué d'une croix mais il n'a pas de
nom
```

9.6 Comment définir et tracer un objet géométrique avec son nom

L'affectation d'un objet géométrique dans une variable permet de définir cet objet géométrique et de le tracer avec une légende ayant comme nom, le nom de la variable.
Si on veut donner à cet objet géométrique un nom différent de celui de la variable, on peut définir l'objet géométrique avec une affectation qui se termine par `:;` et utiliser `legende`. Voici des exemples : On tape :

```
nodisp(B:=point(1+i))
```

Ou on tape :

```
B:=point(1+i):;
```

Le point `B` est défini mais n'est pas tracé.
puis on tape :

```
legende(B,"Bien")
```

Ou on tape :

```
point(affixe(B),legende="Bien")
```

On obtient :

`Le point 1+i est marqué d'une croix et a comme nom "Bien"`

Remarque
Si on veut définir l'objet géométrique sans le tracer, puis le faire apparaitre avec son nom, on peut aussi utiliser `eval` (voir la commande `eval`6.13.2).
On tape :

```
nodisp(B:=point(1+i))
```

Ou on tape :

```
B:=point(1+i):;
```

Le point `B` est défini mais n'est pas tracé.
puis on tape :

```
B:=eval(B,1))
```

Ou on tape :

```
B:=B
```

Ou on tape :

```
legende(B,"B")
```

Ou on tape :

```
point(affixe(B),legende="B")
```

On obtient :

`Le point 1+i est marqué d'une croix et a comme nom "B"`

Si on tape :

```
B:=eval(B,2))
```

On obtient :

`Le point 1+i est marqué d'une croix et n'a pas de nom`

9.7 Comment lire et créer une image

9.7.1 Qu'est-ce qu'une image ?

Pour `Xcas` une image est une liste `a` de 5 éléments :
- `a[0]` est une liste de 3 éléments : le nombre de canaux (3 ou 4) et le nombre de lignes n et de colonnes p utilisés pour la dimension de l'image,
- `a[1]` correspond au canal rouge : c'est une matrice n, p de nombres entiers entre 0 et 255,
- `a[2]` correspond au canal vert : c'est une matrice n, p de nombres entiers entre 0 et 255,
- `a[3]` correspond au canal pour la transparence : c'est une matrice n, p de nombres entiers entre 0 et 255,
- `a[4]` correspond au canal bleu : c'est une matrice n, p de nombres entiers entre 0 et 255,

9.7.2 Pour lire une image : `readrgb`

`readrgb` peut lire un fichier contenant une image. Ce fichier peut être un fichier `.jpg` ou `.png` ou `.gif`.
`readrgb` renvoie [[nombre_canaux,largeur,hauteur],rouge,vert,transparence,bleu] où rouge,vert,transparence,bleu sont des matrices.
On tape :

```
a:=readrgb("image.jpg")
```

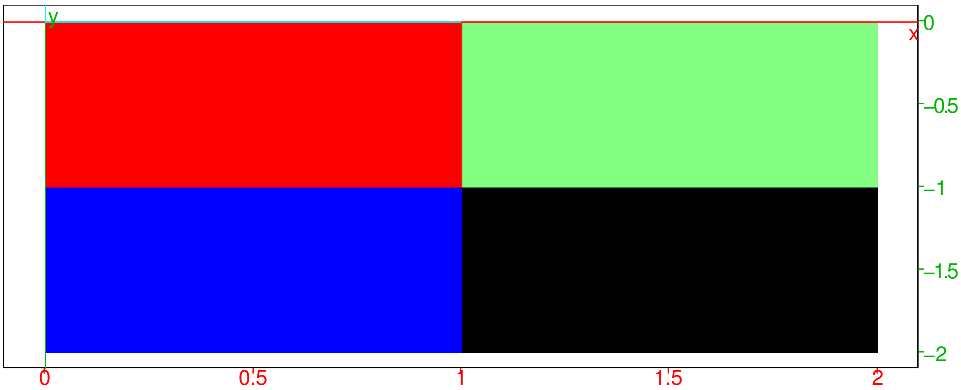
On obtient :

```
une liste de 5 éléments qui sont [4,20,10] suivi de 4
matrices de dimension 20,10 qui indiquent où sont
situé les couleurs rouge,vert,transparence,bleu]
```

9.7.3 Pour recréer ou créer une image : `writergb`

`writergb` permet de stocker une image lue avec `readrgb` dans un fichier de suffixe `.png`.
ou
de créer une image et de la stocker dans un fichier de suffixe `.png`. On peut créer cette image avec des niveaux de gris ou des niveaux des différentes couleurs.

– On posséde une variable (par exemple `a`) contenant une image lue avec `readrgb`.
  `writergb` a deux arguments : le nom du fichier dans lequel on veut stocker la nouvelle image et une liste contenant `a[0]` (liste contenant le nombre de canaux et les dimensions de la matrice des pixels de cette image), puis les couleurs de cette image qui sont les matrices `a[1]` pour la couleur rouge, `a[2]` pour la couleur verte, `a[3]` pour la transparence et `a[4]` pour la couleur bleu.
  La transparence permet de superposer plusieurs images : sa valeur va de 0 à 255 (si la transparence vaut 0 c'est un cache !).
  On tape :
  ```
  a:=readrgb("image.jpg")
  ```
  Puis on tape :
  ```
  writergb("imagevrb.png",[a[0],a[2],a[1],a[3],a[4]])
  ```
  Puis on tape :
  ```
  gimp "imagevrb.png"
  ```
  On obtient :
  ```
  l'image de départ dans laquelle le rouge est
  devenu vert et le vert est devenu rouge
  ```
  Essayer de crèer une image :
  ```
  writergb("essai.png",[[4,2,2],[[255,0],[0,0]],
  [[0,255],[0,0]],[[255,125],[255,255]],
  [[0,0],[255,0]]])
  ```
  vous obtenez (avec un vert attenué du au 125 de la 3ième matrice) :



ou encore essayer :

```
writergb("image1.png",[[4,2,3],[[255,0,255],[0,255,0]],
 [[0,0,255],[0,0,0]],[[255,125,125],[255,255,255]],
                  [[0,0,0],[255,255,0]]])
```

– On peut aussi créer et stocker des images, au format PNG avec une version simplifiée de la syntaxe : pas d'argument correspondant au nombre de canaux et aux dimensions de la matrice des pixels de cette image ni de matrice correspondant à la transparence.
  `writergb` a alors deux ou quatre arguments : le nom du fichier dans lequel on veut stocker la nouvelle image et la matrice des niveaux de gris des pixels, ou 3 matrices (du rouge, du vert et du bleu) donnant la couleur RGB des pixels.
  Par exemple :

  ```
  writergb("image.png",[[255,0],[0,255]])
  ```
  crée et affiche une image 2x2 pixels au format PNG en 256 niveaux de gris (0 noir, 255 blanc)

  ```
  writergb("image.png",[[255,0],[0,0]],[[0,255],[0,0]],
                       [[0,0],[255,0]])
  ```
  crée et affiche une image 2x2 pixels au format PNG en RGBA avec 256 niveaux pour chaque couleur (rouge, bleu, vert). Ici la première ligne rouge,vert et la deuxième ligne est bleu,noir. Essayez :

  ```
  writergb("essai.png",[[4,300,300],makemat(0,300,300),
        makemat(0,300,300), makemat(255,300,300),
             makemat(0,300,300)+idn(300)*255])
  ```

9.8 Comment faire une démonstration : `assume`

Pour pouvoir faire une démonstration en géométrie, il suffit de demander au calcul formel de faire les calculs, en choisissant bien les paramètres du problème et ces paramètres doivent être formels....En géométrie les points peuvent avoir des coordonnèes exactes ou numériques ou formelles. Mais pour faire une figure, il faut affecter les paramètres formels. On peut le faire de différentes façons :

- on utilise a commande `assume` avant la création du point à coordonnèes formelles. Par exemple `assume(a=2.1);A:=point(a+i);` fera la figure en donnant à `a` la valeur `2.1`, mais `a` ne sera pas affecté dans les calculs qui seront faits ultérieurement avec le paramètre formel `a` : si on tape `longueur(0,A)`, on obtient `sqrt((-a)^2+1)`
- on utilise l'écran de configuration des attributs d'un point déjà crée avec des coordonnèes exactes ou numériques et on coche la rubrique `symb` figurant sur cet écran.
  Pour faire apparaitre l'écran de configuration des attributs d'un point où figure la rubrique `symb`, il faut être en mode `point` ou en mode `Pointeur` et uniquement dans ces 2 modes. On suppose que l'on a déjà défini un point, par exemple `A:=point(2.1,1)`. On clique sur `A` avec le bouton droit de la souris, on fait apparaitre l'écran de configuration des attributs du point `A` où figure la rubrique `symb`. On coche `symb` avec la souris et cela a pour effet de rajouter les deux lignes `assume(Ax=[2.1,-5.0,5.0])`, `assume(Ay=[1,-5.0,5.0])` et de définir automatiquement le point `A` avec les coordonnées symboliques `(Ax,Ay)`. **Attention** la rubrique `symb` n'apparait que dans l'écran de configuration des attributs d'un point non symbolique : autrement dit si on a défini un point avec des coordonnées symboliques c'est définitif !

9.9 Les points en géométrie plane

9.9.1 Les points et les nombres complexes

L'affixe d'un point en géométrie plane est un nombre complexe.
La commande `point` a comme argument un nombre complexe c (ou un couple de 2 nombres a, b) et renvoie le point d'affixe c (ou d'affixe $a + ib$) tracé dans un écran graphique.
Certaines fonctions peuvent admettre comme argument soit un point, soit un nombre complexe, mais quelquefois le résultat n'est pas le même.
Exemples
`re(1+2*i)` renvoie `1`
Si `A:=point(1+2*i)`, `re(A)` renvoie un point d'affixe `1`, et `im(A)` renvoie un point d'affixe `2*i`
`cercle(1,1+i)` renvoie le cercle de centre le point d'affixe 1 et de rayon égal à $\sqrt{2}$=`abs(1+i)`,
`cercle(1,point(1+i))` renvoie le cercle de diamètre le point d'affixe 1 et le point d'affixe (1+i)

9.9.2 Le point en géométrie plane : `point`

Voir aussi : 10.4.1 `point` en géométrie 3-d.
Pour obtenir un point il suffit d'être en mode `point` et de cliquer avec la souris (bouton gauche) pour qu'un point s'affiche avec un nom.
Ce nom est crée automatiquement : `A` puis `B` etc...
On peut aussi utiliser la commande `point` :
`point` a comme argument un nombre complexe ou un couple de 2 nombres réels

représentant ses coordonnées rectangulaires (pour définir un point par ses coordonnées polaires il faut utiliser `polar_point` ou `point_polaire`.
Attention
Si `a,b` est un couple de 2 nombres complexes dont 1 n'est pas réel, `K:=point(a,b)` renvoie 2 points de même nom (ici `K`) l'un d'affixe `a`, l'autre d'affixe `b`.
Lorsque `a,b` est un couple de 2 nombres réels, `A:=point(a,b)` renvoie et dessine le point ayant pour affixe `a+ib`.
`point` renvoie et dessine le point ayant pour affixe son argument. Ce point aura la forme choisie par la commande `affichage` : par défaut il aura la forme d'une croix et il aura la forme d'un point si on a tapé `affichage(point_point)`.
On tape :

```
point(1+i)
```

On obtient :

```
Le point d'affixe 1+i est tracé avec une croix
```

On tape :

```
A:=point(-2,1)
```

On obtient :

```
Le point d'affixe -2+i est tracé avec une croix et est
noté A
```

On tape :

```
point(-2,i)
```

On obtient :

```
Les 2 points d'affixe -2 et i sont tracés avec une
croix
```

Remarque Lorqu'on fait une affectation par exemple `A:=point(-2+i)` cela a pour effet de stocker le `point(-2+i)` dans la variable `A`, d'ouvrir un écran graphique si on n'est pas dans un niveau de géométrie et de dessiner le point avec une croix et de lui mettre comme légende le nom qui est situé à gauche de := ici `A`.
Si on fait plusieurs affectations avec un seul signe := par exemple :
`A,B:=point(-2+i),point(2+i)` la variable `A` contient le `point(-2+i)`, la variable `B` contient `point(-2+i)` mais les deux points auront le même nom : `A,B` et on ne pourra pas deplacer ces points en mode `pointeur`. Pour éviter cela on doit taper sur des niveaux séparés :
`L:=point(-2+i),point(2+i):;A:=L[0];B:=L[1]`
Autre exemple
`LL:=point(-2,i):;C:=LL[0];D:=LL[1]` définit le point `C` d'affixe -2 et le point `D` d'affixe i (car l'affixe de `D` n'est pas réel !).

9.9.3 Définir au hasard un point 2-d : `point2d`

`point2d` a comme argument une séquence de noms de points.
`point2d` définit au hasard, les coordonnées entières (entre -5 et +5) des points 2d donnés en argument.
On tape :

```
point2d(A,B,C)
```

Puis on tape :

```
triangle(A,B,C)
```

On obtient :

```
Le tracé d'un triangle ABC
```

Attention
Les points définit par la commande `point2d` sont fixés une fois pour toute et donc ne pourront pas être bougé avec la souris.

9.9.4 Le point en polaire en géométrie plane : `polar_point point_polaire`

`polar_point(r,t)` ou `point_polaire(r,t)` renvoie le point (en 2D) de coordonnées polaires les arguments `r` et `t` c'est à dire le point d'affixe `r*exp(i*t)`.
On tape :

```
polar_point(2,pi/4)
```

Ou on tape :

```
point_polaire(2,pi/4)
```

On obtient :

```
Le tracé du point d'affixe 2*exp(i*pi/4)
```

9.9.5 Un des points d'intersection de deux objets géométriques : `single_inter inter_unique inter_droite line_inter`

Voir aussi : 10.4.3 pour la géométrie 3-d.
`inter_unique` ou `inter_droite` a 2 ou 3 arguments qui sont deux objets géométriques et éventuellement un 3ième argument qui est soit un point soit une liste de points.
`inter_unique` renvoie l'un des points d'intersection de ces deux objets géométriques. Si on a mis un point `A` (ou son affixe) comme troisième argument `inter_unique` renvoie le point d'intersection le plus proche de `A` et si on a mis une liste de points `L` (ou une liste d'affixe) comme troisième argument `inter_unique` renvoie le point d'intersection qui ne se trouve pas dans la liste `L`.
On tape :

```
A:=inter_unique(droite(0,1+i),droite(1,i))
```

On obtient :

```
Le point d'affixe 1/2+i/2 est tracé avec une croix et
est noté A
```

On tape :

```
B:=inter_unique(cercle(0,1),droite(-1,i))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté B
```

On tape :

```
B1:=inter_unique(cercle(0,1),droite(-1,i),[i])
```

On obtient :

```
Le point d'affixe -1 est tracé avec une croix et est
noté B1
```

On tape :

```
B2:=inter_unique(cercle(0,1),droite(-1,1+2*i),1+2*i)
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté B2
```

On tape :

```
C:=inter_unique(cercle(1,sqrt(2)),cercle(0,1))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C
```

On tape :

```
C1:=inter_unique(cercle(1,sqrt(2)),cercle(0,1),[i])
```

On obtient :

```
Le point d'affixe -i est tracé avec une croix et est
noté C1
```

On tape :

```
C2:=inter_unique(cercle(1,sqrt(2)),cercle(0,1),i/2)
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C2
```

9.9.6 Les points d'intersection de deux objets géométriques : `inter`

Voir aussi : 10.4.4 pour la géométrie 3-d.
`inter` a soit 2 arguments soit 3 arguments.

- si `inter` a 2 arguments qui sont deux objets géométriques.
  `inter` renvoie la liste des points d'intersection de ces deux objets géométriques.

- si `inter` a 2 arguments qui sont deux objets géométriques et un point.
  `inter` renvoie le point d'intersection de ces deux objets géométriques le plus proche du point donné comme troisième argument.

On tape en géométrie plane :

```
A:=inter(droite(0,1+i),droite(1,i))[0]
```

On obtient :

```
Le point d'affixe 1/2+i/2 est tracé avec une croix et
est noté A
```

On tape en géométrie plane :

```
B:=inter(cercle(0,1),droite(1,i))[0]
```

```
C:=inter(cercle(0,1),droite(1,i))[1]
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté B
```

```
Le point d'affixe 1 est tracé avec une croix et est
noté C
```

On tape en géométrie plane :

```
L:=inter(cercle(0,1),droite(1,i))
```

On obtient :

```
Les points d'affixe i et 1 sont tracés avec une croix
et sont notés L
```

On tape en géométrie plane :

```
D:=inter(cercle(0,1),droite(1,i),point(1/2))
```

On obtient :

```
Le point d'affixe 1 est tracé avec une croix et est
noté D
```

9.9.7 Orthocentre d'un triangle : `orthocenter` `orthocentre`

`orthocenter` ou `orthocentre` a comme argument un triangle ou trois points ou trois nombres complexes désignant l'affixe de trois points.
`orthocenter` ou `orthocentre` trace et renvoie le point qui est l'orthocentre du triangle ou du triangle formé par ces trois points.
On tape :

```
orthocentre(0,1+i,-1+i)
```

Ou on tape :

```
orthocentre(triangle(0,1+i,-1+i))
```

On obtient :

```
Le point d'affixe 0 est tracé avec une croix
```

On tape :

```
T:=triangle(-i,2+i,-1+i);H:=orthocentre(T)
```

On obtient :

```
Le triangle T et le point H d'affixe 0 sont tracés
```

9.9.8 Le milieu d'un segment : `midpoint` `milieu`

Voir aussi : 10.4.5 pour la géométrie 3-d.
`milieu` ou `midpoint`, en géométrie plane, a comme argument 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
`milieu` ou `midpoint` renvoie et dessine le point milieu du segment défini par ces deux points.
On tape :

```
milieu(-1,1+i)
```

On obtient :

```
Le point d'affixe i/2 est tracé avec une croix
```

9.9.9 L'isobarycentre de n points : `isobarycenter` `isobarycentre`

Voir aussi : 10.4.6 pour la géométrie 3-d.
`isobarycentre` ou `isobarycenter` a comme argument la liste (ou la séquence) de n points ou de n nombres complexes représentant l'affixe de ces points.
`isobarycentre` ou `isobarycenter` renvoie et trace un point qui est l'isobarycentre de ces n points.
On tape :

```
isobarycentre(0,2,2*i)
```

On obtient :

```
Le point d'affixe 2/3+2*i/3 est tracé avec une croix
```

9.9.10 Point défini comme barycentre de n points : `barycenter barycentre`

Voir aussi : 10.4.7 pour la géométrie 3-d et 6.12.10.
`barycentre` ou `barycenter`, en géométrie plane, a comme argument 2 listes de longueur 2 ou une matrice ayant 2 colonnes ou ayant 2 lignes) :

- 2 listes de longueur 2
  Pour chaque liste, le premier élément contient le point A_j (ou le nombre complexe a_j représentant l'affixe de ce point) et le deuxième élément contient le coefficient réel α_j affecté à A_j ($j = 1..2$)
- une matrice ayant 2 colonnes et n lignes ($n \geq 2$)
  Le $j$ième élément de la première colonne de la matrice contient le point A_j (ou le nombre complexe a_j représentant l'affixe de ce point), le $j$ième élément de la deuxième colonne contient le coefficient réel α_j affecté à A_j ($j = 1..n$).
- une matrice ayant 2 lignes et n colonnes ($n \geq 3$)
  Le $j$ième élément de la première ligne de la matrice contient le point A_j (ou le nombre complexe a_j représentant l'affixe de ce point), le $j$ième élément de la deuxième ligne contient le coefficient réel α_j affecté à A_j ($j = 1, 2, ..n$).

`barycentre` ou `barycenter` renvoie et trace le point qui est le barycentre des points A_j d'affixes a_j affectés des coefficients réels α_j lorsque $\sum \alpha_j \neq 0$.
Si $\sum \alpha_j = 0$, `barycentre` ou `barycenter` renvoie une erreur.
On tape :

```
barycentre([1+i,1],[1-i,1])
```

Ou on tape :

```
barycentre([point(1,1),1],[point(1,-1),1])
```

Ou on tape :

```
barycentre([[1+i,1],[1-i,1]])
```

Ou on tape :

```
barycentre([[point(1,1),1],[point(1,-1),1]])
```

On obtient :

```
Le point d'affixe 1 est tracé avec une croix
```

On tape :

```
barycentre([[1+i,1],[1-i,1],[1+4*i,2]])
```

Ou on tape :

```
barycentre([[point(1,1),1],[point(1,-1),1],[point(1+4*i),2]])
```

Ou on tape :

```
barycentre([[1+i,1-i,1+4i],[1,1,2]])
```

Ou on tape :

```
barycentre([[point(1,1),point(1,-1),point(1,4)],[1,1,2]])
```

On obtient :

```
Le point d'affixe 1+2i est tracé avec une croix
```

9.9.11 Le centre d'un cercle : `centre center`

`centre` ou `center` a comme argument le nom d'un cercle (voir la définition du cercle 9.14.1).
`centre` ou `center` renvoie et trace le centre de ce cercle.
On tape :

```
C:=centre(cercle(0,point(2*i)))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C
```

On tape :

```
M:=centre(cercle(point(1+i),1))
```

On obtient :

```
Le point d'affixe 1+i est tracé avec une croix et est
noté M
```

9.9.12 Les sommets d'un polygone : `vertices vertices_abc sommets sommets_abc`

`sommets` ou `sommets_abc` ou `vertices` ou `vertices_abc` a comme argument un polygone.
`sommets` ou `sommets_abc` ou `vertices` ou `vertices_abc` renvoie la liste des sommets de ce polygone et les trace.
Attention Si le polygone a n sommets la liste sera de longueur n.
On tape :

```
sommets(triangle_equilateral(0,2))
```

On obtient :

```
les points
[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*(i)+1))/2,0)] sont
tracés avec une croix
```

On tape :

```
C:=sommets(triangle_equilateral(0,2))[2]
```

On obtient :

Le point d'affixe $1+i*\sqrt{3}$ est tracé avec une croix et est noté C

Attention
Si on tape :

```
T:=triangle_equilateral(0,2,C):;sommets(T[0])
```

On obtient :

```
les points
[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*(i)+1))/2,0)] sont
tracés avec une croix
```

9.9.13 Les sommets d'un polygone : `vertices_abca sommets_abca`

`sommets_abca` a comme argument le nom d'un polygone.
`sommets_abca` renvoie la liste "fermée" des sommets de ce polygone et les trace.
Attention Si le polygone a n sommets la liste sera de longueur $n + 1$ car elle commence et se termine par le premier sommet (liste "fermée").
On tape :

```
sommets_abca(triangle_equilateral(0,2))
```

On obtient :

```
[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*(i)+1))/2,0),pnt(0,0)]
```

9.9.14 Point sur un objet géométrique : `element`

`element` peut avoir différents types d'arguments :

- un intervalle $a..b$ et deux réels la valeur et le pas (par défaut la valeur vaut $(a + b)/2$ et le pas $(b - a)/100$), par exemple, `t:=element(0..pi)` ou `t:=element(0..pi,pi/2)` ou `t:=element(0..pi,pi/2,pi/100.0)` signifie que `t` peut prendre une valeur quelconque de l'intervalle $[0; \pi]$ et le deuxième argument $\pi/2$ donne la valeur qui définit `t` : cela a pour effet d'avoir en haut et à droite un curseur noté `t` que l'on peut faire bouger à la souris de 0 à π, avec à gauche de ce curseur un nombre égal à la valeur de `t` du curseur.
- un objet géométrique et un réel (par défaut ce réel vaut $1/2$), par exemple, `A:=element(cercle(0,2),1)` signifie que `A` se trouve sur le cercle de centre 0 et de rayon 2 et a comme affixe $2 * \exp(i)$ (car $2 * \exp(i * t)$ est l'équation paramétrique du cercle(0,2) et le deuxième argument 1 donne la valeur du paramètre t pour définir `A`). Par exemple, `A:=element(cercle(0,1))` signifie que `A` se trouve sur le cercle de centre 0 et de rayon 1, le point `A` sera tracé en prenant $t = 1/2$ comme valeur du paramètre de l'équation paramétrique de l'objet géométrique (ici `affixe(A)` = $2 * \exp(i/2)$). Lorsque ensuite on déplacera `A` avec la souris, `A` se déplacera sur l'objet géométrique.
- un objet géométrique et un nom de variable (par exemple `t`) défini auparavant par la commande `element` : par exemple `t:=element(0..pi)`. Si on tape `A:=element(cercle(0,2),t)`, alors t est la variable de paramétrage de l'obget géométrique défini par le premier argument, c'est à dire que `A` se trouve sur le cercle de centre 0 et de rayon 2 et que `A` a comme affixe $2 * \exp(i * t)$, car $2 * \exp(i * t)$ est l'équation paramétrique du cercle(0,2). Il est donc obligatoire dans ce cas de définir auparavant le deuxième argument (ici `t`) comme étant l'élément d'un intervalle.
  On tape par exemple :

  ```
  t:=element(0..pi)
  ```

  puis

  ```
  A:=element(cercle(0,2),t)
  ```

  cela a pour effet d'avoir en haut et à droite un curseur noté `t` que l'on peut faire bouger à la souris de 0 à π, avec à gauche de ce curseur un nombre égal à la valeur de `t` du curseur. Ce curseur permet de faire bouger le point `A` sur le demi-cercle supérieur du cercle de centre 0 et de rayon 1 (car $0 \leq t \leq \pi$) et cela sans tracer ce demi-cercle. On tape par exemple :

```
A:=point(1);B:=point(2+i)
t:=element(0..2)
```

puis

```
M:=element(droite(A,B),t)
```

`M` est un point de la droite `AB` et on a `M=A+t*(B-A)` c'est à dire `M=(1-t)*A+t*B` pout parcourir le segment `AB` , il faut mettre `t:=element(0..1)` ou encore `M:=element(segment(A,B),t)` qui aura pour effet de laisser `M` en `A` si `t<0` et de laisser `M` en `B` si `t>1`.

– une ligne polygonale `LP` et `[floor(t),frac(t)]` avec `t` défini auparavant par la commande `element` : par exemple `t:=element(0..5)` si `LP` a 5 côtés.
Les côtés de la ligne polygonale `LP` ont comme numèro : 0,1....
Si par exemple `LP` a 5 côtés et a pour sommets `A(0),...A(4),A(5)=A(0)`, on tapera :

```
t:=element(0..5)
M:=element(LP,[floor(t),frac(t)])
```

Ainsi selon les valeurs de `t`, `M` va parcourir les 5 côtés de `LP` : `M` sera situé sur le côté de numéro `n=floor(t)` et on aura `M=frac(t)*A(n)+(1-frac(t))*A(n+1)`.
Par exemple :

```
A:=point(0);
B:=point(4);
C:=point(4*i);
t:=element(0..3);
T:=triangle(A,B,C);
M:=element(T,[floor(t),frac(t)]);
```

Attention Si à un point M d'affixe m, défini comme élément d'une courbe C, on ajoute un complexe a, cela définit un point N de la courbe C qui est le projeté du point d'affixe $m + a$ sur C.
Par contre si un point M d'affixe m, défini comme élément d'une courbe C, on ajoute un point A d'affixe a, cela définit un point P d'affixe $m + a$ Par exemple, étant donné 3 points M, A, B, si on veut définir le point `N` vérifiant par exemple : $\overrightarrow{MN} = \overrightarrow{AB}$, on peut taper : `N:=M+(B-A)` à condition que M ne soit pas défini comme élément d'une courbe C. En effet si on a tapé `M:=element(C)` il faut définir N en tapant : `N:=affixe(M)+B-A` ou `N:=M+B-A` (sans parenthèses) car `N:=M+B-A` est interprété en `N:=(M+B)-A` car il n'y a pas de règle de priorité entre + et − alors que
`M+(B-A)` renvoie un élément de la courbe C qui est le projeté de N sur C.
On a donc, si on tape :
`A:=point(-2,2);B:=point(1,3);C:=cercle(0,1);`
`M:=element(C);N:=affixe(M)+B-A` ;(ou `N:=M+B-A` ;) `N` n'est pas sur la courbe `C`
si on tape :
`P:=M+(B-A)` (ou P :=projection(C,N) ;) `P` est sur la courbe `C`

9.10 Les droites en géométrie plane

9.10.1 La droite et la droite orientée en géométrie plane : `line droite`

Voir aussi : 3.10.1 et10.5.1 pour la géométrie 3-d et 9.17.12 pour avoir la valeur de la pente d'une droite.
`droite`, en géométrie plane, a comme argument deux points (ou deux nombres complexes représentant l'affixe de ces points) ou une liste de deux points (ou de deux complexes) ou a comme argument un point et `pente=m` ou a comme argument un point et son vecteur directeur `[u1,u2]` ou encore une équation de droite de la forme a*x+by+c=0.
`droite` renvoie et trace la droite définie par les deux arguments.
On tape :

```
droite(0,1+i)
```

On obtient :

```
La droite d'équation y=x est tracée
```

On tape :

```
droite(1+i,pente=1)
```

On obtient :

```
La droite d'équation y=x est tracée
```

On tape :

```
droite(1+i,[3,3])
```

On obtient :

```
La droite d'équation y=x est tracée
```

On tape :

```
droite(y-x=0)
```

On obtient :

```
La droite d'équation y=x est tracée
```

Remarque : l'orientation de la droite
Voir aussi : 10.5.2 pour la géométrie 3-d.
`droite` définit une droite orientée :

– Lorsque la droite est donnée par deux points, son orientation est définie par l'ordre des points donnés en argument. Par exemple `droite(A,B)` définit une droite orientée par le vecteur $\overrightarrow{AB}$.

– Lorsque c'est une équation qui définit la droite on écrit l'équation sous la forme :"membre_de_gauche-membre_de_droite=0" pour avoir une équation de la droite de la forme `a*x+by+c=0` et alors le vecteur orientant la droite est `[b,-a]` ou encore son orientation est définie par le produit vectoriel 3-d de son vecteur normal (de cote 0) et de [0,0,1]. Par exemple `droite(y=2*x)` est orientée par `[1,2]` car son équation est `-2*x+y=0` et `cross([-2,1,0],[0,0,1])=[1,2,0]`.
– Lorsque la droite est donnée par un point A et sa pente m, son orientation est définie par le vecteur $\overrightarrow{AB}$ avec $B = A + 1 + i * m$.
– Lorsque la droite est donnée par un point A et son vecteur directeur, son orientation est définie par le vecteur directeur.

Attention
Si on veut tracer la droite en couleur en utilisant un argument supplémentaire par exemple, `couleur=1`, il faut impérativement que cet argument soit le troisième argument : donc si la droite est définie par une liste il faut transformer cette liste en une séquence avec `op`, par exemple, on tape :

```
A:=point(0,0);
B:=point(2,1);
C1:=cercle(A,B-A);
C2:=cercle(B,A-B);
droite(op(inter(C1,C2)),couleur=1)
```

alors que `droite(inter(C1,C2),'couleur'=1)` vous répondra `"Invalid dimension"`

9.10.2 La demi-droite en géométrie plane : `half_line demi_droite`

Voir aussi : 10.5.3 pour la géométrie 3-d.
`demi_droite`, en géométrie plane, a comme argument 2 points (ou 2 nombres complexes représentant l'affixe de ces points ou encore une liste de 2 points ou de 2 complexes).
`demi_droite` renvoie et trace la demi-droite d'origine le premier argument est passant par le deuxième argument.
On tape :

```
demi_droite(0,-1+i)
```

On obtient :

```
La demi-droite d'origine 0, passant par -1+i
```

9.10.3 Le segment en géométrie plane : `segment`

Voir aussi : 10.5.4 pour la géométrie 3-d.
`segment`, en géométrie plane, a comme argument 2 points (ou 2 nombres complexes représentant l'affixe de ces points ou encore une liste de 2 points ou de 2 complexes).
`segment` renvoie et trace le segment défini par les deux arguments.
On tape :

```
segment(-1,i)
```

On obtient :

```
Le segment -1, i
```

9.10.4 Le segment : `Line`

`Line` a comme argument 4 nombres réels représentant les coordonnées de ces points.
`Line(a,b,c,d)` renvoie et trace le segment défini par les deux points `a+i*b` et `c+i*d`.
On tape :

```
Line(-1,1,2,-2)
```

On obtient :

```
Le segment -1+i,2-2*i
```

9.10.5 Le vecteur en géométrie plane : `vector vecteur`

Voir aussi : 10.5.5 pour la géométrie 3-d.
`vecteur`, en géométrie plane, a comme arguments soit :

- deux points A, B ou deux nombres complexes représentant l'affixe de ces points ou deux listes représentant les coordonnées de ces points.
  `vecteur` définit et dessine le vecteur $\overrightarrow{AB}$
- un point A (ou un nombre complexe représentant l'affixe de ce point ou une liste représentant les coordonnées de ce point) et un vecteur V (définition récursive).
  `vecteur` définit et dessine le vecteur $\overrightarrow{AB}$ tel que $\overrightarrow{AB} = \overrightarrow{V}$. Si `W:=vecteur(A,V)`, alors le point `B` tel que $\overrightarrow{AB} = \overrightarrow{V}$ est `point(W[1,1])` ou `point(coordonnees(V)+coordonnees(`
  ou `A+(affixe(V)[1]-affixe(V)[0])`.

On tape :

```
vecteur(point(-1),point(i))
```

Ou on tape :

```
vecteur(-1,i)
```

Ou on tape :

```
vecteur([-1,0],[0,1])
```

On obtient :

```
Le tracé du vecteur d'origine -1 et d'extrémité i
```

On tape :

```
V:=vecteur(point(-1),point(i))
```

On tape :

```
vecteur(point(-1+i),V)
```

Ou on tape :

```
vecteur(-1+i,V)
```

Ou on tape :

```
vecteur([-1,1],V)
```

On tape :

```
point([-1,1],coordonnees(V))
```

On obtient :

```
Le tracé du vecteur d'origine -1+i et d'extrémité 2*i
```

On tape :

```
D:=point([-1,1]+coordonnees(V))
```

On obtient :

```
D le point(2*i)
```

Remarque
En calcul formel, on travaille sur la liste des coordonnées des vecteurs que l'on obtient avec la commande `coordonnees` (cf 9.16.4).

9.10.6 Les droites parallèles : `parallel parallele`

Voir aussi : 10.5.6 pour la géométrie 3-d.
`parallele`, en géométrie plane, a comme arguments un point A (ou un nombre complexe représentant l'affixe de ce point) et une droite d.
`parallele` renvoie et dessine la droite parallèle à la droite d passant par le point A.
On tape :

```
parallele(0,droite(1,i))
```

On obtient :

```
La droite d'équation y=-x est tracée
```

9.10.7 Les droites perpendiculaires en 2-d : `perpendicular perpendiculaire`

Voir aussi : 10.5.7 et 10.5.8 pour la géométrie 3-d.
`perpendiculaire`, en géométrie plane, a comme arguments un point A (ou un nombre complexe représentant l'affixe de ce point) et une droite d (ou 2 points).
`perpendiculaire` renvoie et dessine la droite perpendiculaire à la droite d passant par le point A .
On tape :

```
perpendiculaire(0,droite(1,i))
```

Ou on tape :

```
perpendiculaire(0,1,i)
```

On obtient :

```
La droite d'équation y=x
```

9.10.8 Les tangentes à un objet géométrique plan : `tangent`

Voir aussi : 10.6.3 pour la géométrie 3-d et 3.10.5 pour les tangentes à un graphe.
`tangent` peut avoir comme arguments :
- un objet géométrique G et un point A ou,
- un point A défini par `element` dont les paramètres sont, un objet géométrique G et un réel représentant la valeur du paramètre de l'équation paramétrique de G.
`tangent` renvoie une liste de droites et dessine ces droites qui sont les tangentes à cet objet géométrique G et qui passent par le point A.
On tape :

```
tangent(cercle(0,1),point(1+i))
```

On obtient :

```
La droite d'équation x=1 et la droite d'équation y=1
```

On tape :

```
tangent(element(cercle(0,1),1))
```

On obtient :

```
La tangente au cercle de centre 0 et de rayon 1, au
                point d'affixe exp(i)
```

On tape :

```
tangent(circle(i,1+i),point((1+i*sqrt(3))*2))
```

On obtient :

2 tangentes au cercle de centre i et de rayon $\sqrt{2}$
issues du point((1+i*sqrt(3))*2)

9.10.9 La médiane d'un triangle issue d'un sommet : `median_line` `mediane`

`mediane` a comme argument 3 points (ou 3 nombres complexes représentant l'affixe de ces points ou encore une liste de 3 points ou de 3 complexes).
`mediane` renvoie et dessine la droite passant par le premier argument et par le milieu du segment défini par les deux autres arguments.
On tape :

```
mediane(0,1,i)
```

On obtient :

```
La droite d'équation y=x est tracée
```

9.10.10 La hauteur d'un triangle issue d'un sommet : `altitude hauteur`

`hauteur` a comme argument 3 points (ou 3 nombres complexes représentant l'affixe de ces points ou encore une liste de 3 points ou de 3 complexes).
`hauteur` renvoie et dessine la droite qui passe par le premier argument et qui est perpendiculaire à la droite définie par les deux autres arguments.
On tape :

```
hauteur(0,1,i)
```

On obtient :

```
La droite d'équation y=x
```

9.10.11 La médiatrice d'un segment : `perpen_bisector mediatrice`

Voir aussi : 10.6.2 pour la géométrie 3-d.
`mediatrice`, en géométrie plane, a comme argument un segment ou 2 points (ou 2 nombres complexes représentant l'affixe de ces points ou encore une liste de 2 points ou de 2 complexes).
`mediatrice` renvoie et dessine une droite qui est la médiatrice du segment défini par les deux arguments.
Remarques
`mediatrice(A,B)` est pour `Xcas` une droite définie par les points : `M:=milieu(A,B)` et `N:=rotation(M,pi/2,M+(A-B))`.
Si vous donner comme argument à `mediatrice` 2 droites au lieu de 2 points, `Xcas` tracera la médiatrice du premier point définissant la première droite et du deuxième point définissant la deuxième droite, (i.e `mediatrice(droite(A,B), droite(C,D))` sera identique à `mediatrice(A,D)` et si `D1:=droite(A,B); D2:=mediatrice(droite(A,B))` alors `mediatrice(D1,D2)` sera identique à `mediatrice(A,D2)` qui sera identique à `mediatrice(A,N)` avec `M:=milieu(A,B)` et `N:=rotation(M,pi/2,M+(A-B))`).
On tape :

```
mediatrice(1,i)
```

Ou on tape :

```
mediatrice(segment(1,i))
```

On obtient :

```
La droite d'équation y=x
```

9.10.12 La bissectrice intérieure d'un angle : `bisector bissectrice`

`bissectrice` a comme argument 3 points (ou 3 nombres complexes représentant l'affixe de ces points ou encore une liste de 3 points ou de 3 complexes).
`bissectrice` renvoie et dessine une droite qui est la bissectrice de l'angle de sommet le premier argument et défini par le deux autres arguments.
On tape :

```
bissectrice(0,1,i)
```

On obtient :

```
La droite d'équation y=x
```

9.10.13 La bissectrice extérieur d'un angle : `exbisector exbissectrice`

`exbissectrice` a comme argument 3 points (ou 3 nombres complexes représentant l'affixe de ces points ou encore une liste de 3 points ou de 3 complexes).
`exbissectrice` renvoie et dessine une droite qui est la bissectrice extérieure de l'angle de sommet le premier argument et défini par le deux autres arguments.
On tape :

```
exbissectrice(0,1,i)
```

On obtient :

```
La droite d'équation y=-x
```

9.11 Les triangles

Pour les dessins dans l'espace voir la section 10.7

9.11.1 Généralités

`Xcas` permet de tracer des triangles quelconques, isocèles, rectangles, équilatéraux avec les commandes : `triangle, triangle_isocele, triangle_rectangle, triangle_equilateral`.
Ces commandes ont comme arguments ce qu'il faut pour définir le triangle (au moins 2 sommets) et éventuellement comme dernier argument le nom d'une variable. Ce nom de variable servira à définir et à tracer le troisième sommet avec sa légende.
Attention
Ces commandes soit renvoient et tracent le triangle demandé, soit renvoient une liste composée du triangle demandé et du sommet demandé et tracent ce triangle et son troisième sommet avec sa légende.

9.11.2 Le triangle quelconque : `triangle`

Voir aussi : 10.7.1 pour la géométrie 3-d.
`triangle`, en géométrie plane, a comme arguments : 3 points (ou 3 nombres complexes représentant l'affixe de ces points ou encore une liste de 3 points ou de 3 complexes).
`triangle` renvoie et trace le triangle ayant pour sommets ces 3 points.
On tape :

```
triangle(-1,i,1+i)
```

On obtient :

```
Le triangle de sommets -1, i, 1+i
```

9.11.3 Le triangle isocèle : `isosceles_triangle triangle_isocele`

Voir aussi : 10.7.2 pour la géométrie 3-d.

`triangle_isocele`, en géométrie plane, a trois ou quatre arguments.
Description des arguments :

- si il a trois arguments, ce sont : 2 points A et B (ou 2 nombres complexes représentant l'affixe de ces points) et un réel qui désigne la mesure en radians (ou en degrés) de l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$.
  `triangle_isocele(A,B,c)` renvoie et trace le triangle ABC isocèle de sommet A ($AB = AC$) et tel que l'angle $(\overrightarrow{AB}, \overrightarrow{AC}) = c$ radians (ou degrés), sans définir le point C).
  On tape :
  `triangle_isocele(i,1,-3*pi/4)`
  On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :
  `Le triangle isocéle de sommets -1, i,` $-\sqrt{2}$`+i`
- si il a quatre arguments, le dernier argument est le nom d'une variable qui servira à définir le troisième sommet.
  On tape :
  `triangle_isocele(i,1,-3*pi/4,C)`
  On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :
  `Le triangle isocéle de sommets -1, i,` $-\sqrt{2}$`+i`
  On tape :
  `normal(affixe(C))`
  On obtient :
  `-sqrt(2)+i`

9.11.4 Le triangle rectangle : `right_triangle triangle_rectangle`

Voir aussi : 10.7.3 pour la géométrie 3-d.
`triangle_rectangle`, en géométrie plane, a trois ou quatre arguments.
Description des arguments :

- si il a trois arguments, ce sont : 2 points A et B (ou 2 nombres complexes représentant l'affixe de ces points) et un réel k non nul.
  `triangle_rectangle(A,B,k)` renvoie et trace le triangle ABC rectangle en A : ce triangle est direct si $k > 0$, indirect si $k < 0$ et est tel que $AC = |k| * AB$.
  Ainsi si l'angle $(\overrightarrow{BC}, \overrightarrow{BA}) = \beta$ radians (ou degrés), on a $\tan(\beta) = k$.
  On remarquera que si C est le transformé de B dans la similitude de centre A de rapport $|k|$ et d'angle $(k/|k|) * \pi/2$.
  On tape :
  `triangle_rectangle(i,-i,2)`
  On obtient :
  `Le triangle rectangle de sommets i, -i, 4+i`
  On tape :
  `triangle_rectangle(i,-i,-2)`

On obtient :

`Le triangle rectangle de sommets i, -i, -4+i`

– si il a quatre arguments, le dernier argument est le nom d'une variable qui servira à définir le troisième sommet.
  On tape :

  `triangle_rectangle(i,-i,2,D)`

  On obtient :

  `Le triangle rectangle de sommets i, -i, 4+i`

  On tape :

  `normal(affixe(D))`

  On obtient :

  `4+i`

9.11.5 Le triangle équilatèral : `equilateral_triangle` `triangle_equilateral`

Voir aussi : 10.7.4 pour la géométrie 3-d.
`triangle_equilateral`, en géométrie plane, a deux ou trois arguments.
Description des arguments :

– si il a deux arguments, ce sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
  `triangle_equilateral(A,B)` renvoie et trace le triangle équilatéral direct ABC mais sans définir le point C. On tape :

  `equilateral_triangle(0,2)`

  On obtient :

  `le triangle équilatèral de sommets les points d'affixe 0,2,1+i*sqrt(3)`

  Pour définir le troisième sommet C, on peut donner un nom au triangle (par exemple T) et utiliser la commande `sommets(T)` qui renvoie la liste des sommets de T. On définira alors `C:= sommets(T)[2]` mais il est plus simple de rajouter C le nom du dernier sommet comme troisième argument.

– si il a trois arguments, le dernier argument est le nom d'une variable qui servira à définir et à tracer le troisième sommet avec sa légende. On tape :

  `equilateral_triangle(0,2,C)`

  On obtient :

  `le triangle équilatèral de sommets les points d'affixe 0,2,1+i*sqrt(3)`

  On tape :

  `normal(affixe(C))`

  On obtient :

  `1+i*sqrt(3)`

9.12 Les quadrilatères

Voir aussi : 10.7.1 pour la géométrie 3-d.
Pour les dessins dans l'espace voir la section 10.8

9.12.1 Généralités

Xcas permet de tracer des quadrilatères quelconques, des carrés, des losanges, des rectangles, des parallélogrammes avec les commandes : `quadrilatere, carre, losange, rectangle, parallelogramme`.
Ces commandes ont comme arguments ce qu'il faut pour définir le quadrilatère (au moins 2 sommets) et éventuellement comme dernier argument le nom d'une ou deux variables. Ces noms de variables serviront à définir et à tracer les derniers sommets avec leur légende.
Attention
Ces commandes soit renvoient et tracent le quadrilatère demandé, soit renvoient une liste composée du quadrilatère demandé et du ou des sommets demandés et tracent ce quadrilatère avec une légende pour son troisième ou pour son troisième et son quatrième sommet.

9.12.2 Le carré : `square carre`

Voir aussi : 10.8.1 pour la géométrie 3-d.
`carre`, en géométrie plane, peut avoir de deux à quatre arguments.
Description des arguments :

- si il a deux arguments ce sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
  `carre(A,B)` renvoie et trace le carrè $ABCD$ de sens direct, mais sans définir les points D et C.
  On tape :
  `carre(0,1+i)`
  On obtient :
  `Le carré de sommets 0, 1+i, 2*i, -1+i`
- si il a trois (resp quatre) arguments, les 2 derniers paramètres sont le nom d'une (resp deux) variables qui serviront à définir l'avant-dernier sommet (resp les deux autres sommets). On tape :
  `carre(0,1+i,C,D)`
  On obtient :
  `Le carré de sommets 0, 1+i, 2*i, -1+i`
  On tape :
  `affixe(C)`
  On obtient :
  `2*i`
  On tape :
  `affixe(D)`
  On obtient :
  `-1+i`

9.12.3 Le losange : `rhombus losange`

Voir aussi : 10.8.2 pour la géométrie 3-d.
`losange`, en géométrie plane, peut avoir de trois à cinq arguments.
Description des arguments :

– si il a trois arguments ce sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points et un nombre réel a.
`losange(A,B,a)` renvoie et trace le losange $ABCD$ tel que :
$\overrightarrow{AB}, \overrightarrow{AD}) = a$ radians (ou degrés), mais sans définir les points C et D.
On tape :

`losange(-2*i,sqrt(3)-i,pi/3)`

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le losange de sommets -2*i,` $\sqrt{3}$`-i,` $\sqrt{3}$`+i, 0`

– si il a quatre (resp cinq) arguments, le dernier paramètre (resp les 2 derniers paramètres) est (resp sont) le (resp les) nom(s) d'une (resp deux) variable(s) qui servent à définir l'avant-dernier sommet (resp les 2 derniers sommets).
On tape :

`losange(-2*i,sqrt(3)-i,pi/3,E,F)`

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le losange de sommets -2*i,` $\sqrt{3}$`-i,` $\sqrt{3}$`+i, 0`

On tape :

`normal(affixe(E))`

On obtient :

`sqrt(3)+i`

On tape :

`normal(affixe(F))`

On obtient :

`0`

9.12.4 Le rectangle : `rectangle`

Voir aussi : 10.8.3 pour la géométrie 3-d.
`rectangle`, en géométrie plane, peut avoir de trois à cinq arguments.
Description des arguments :

– si il a trois arguments ce sont : deux points (ou deux nombres complexes représentant l'affixe de ces points) et un nombre réel k non nul.
`rectangle(A,B,k)` renvoie et trace le rectangle $ABCD$ tel que :
$AD = |k| * AB$ et $(\overrightarrow{AB}, \overrightarrow{AD}) = (k/|k|) * \pi/2$,
c'est à dire tel que :
$affixe(D) = affixe(A) + k * \exp(i * \pi/2) * (affixe(B) - affixe(A))$
mais sans définir les points C et D.
Remarque Si k est complexe, on a :
$affixe(D) = affixe(A) + k * \exp(i * \pi/2) * (affixe(B) - affixe(A))$
et on peut ainsi se retrouver avec le tracé d'un parallélogramme.
On tape :

`rectangle(0,1+i,1/2)`

On obtient :

`Le rectangle de sommets 0,1+i,1/2+3*i/2,-1/2+i/2`

On tape :

`rectangle(0,1+i,-1/2)`

On obtient :

`Le rectangle de sommets 0,1+i,3/2+i/2,1/2-i/2`

On tape :

`rectangle(0,1,1+i)`

On obtient :

`Le parallélogramme de sommets 0,1,i,-1+i car`
$-1+i=(1+i)*\exp(i*\pi/2)$

– si il a cinq arguments les 2 derniers paramètres sont les noms de deux variables qui serviront à définir les 2 derniers sommets.
  On tape :

  `rectangle(0,1+i,-1/2,G,H)`

  On obtient :

  `Le rectangle de sommets 0,1+i,3/2+i/2,1/2-i/2`

  On tape :

  `normal(affixe(G))`

  On obtient :

  `(3+i)/2`

  On tape :

  `normal(affixe(H))`

  On obtient :

  `(1-i)/2`

9.12.5 Le parallélogramme : `parallelogram parallelogramme`

Voir aussi : 10.8.4 pour la géométrie 3-d.
`parallelogramme`, en géométrie plane, a trois arguments ou quatre arguments.
Description des arguments :

– si il a trois arguments ce sont : trois points (ou trois nombres complexes représentant l'affixe de ces points).
  `parallelogramme(A,B,C)` renvoie et trace le parallélogramme $ABCD$ tel que : $\overrightarrow{AD}=\overrightarrow{BC}$ mais sans définir le point D.
  On tape :

  `parallelogramme(0,1,2+i)`

  On obtient :

  `Le parallélogramme de sommets 0,1,2+i,1+i`

  On tape :

  `parallelogramme(1,0,-1+i)`

  On obtient :

  `Le parallélogramme de sommets 1,0,-1+i,i`

– si il a quatre arguments le dernier paramètre est le nom d'une variable qui servira à définir le sommet manquant.
  On tape :

  `parallelogramme(0,1,2+i,F)`

  On obtient :

  `Le parallélogramme de sommets 0,1,2+i,1+i et le point F d'affixe 1+i`

  On tape :

  `normal(affixe(F))`

  On obtient :

`1+i`

9.12.6 Le quadrilatère : `quadrilateral quadrilatere`

Voir aussi : 10.8.5 pour la géométrie 3-d.
`quadrilatere(A,B,C,D)`, en géométrie plane, renvoie et tracele quadrilatère $ABCD$.
On tape :

```
quadrilatere(0,1,1+i,-1+2*i)
```

On obtient :

```
Le "cerf-volant" de sommets 0, 1, 1+i, 1+2*i
```

9.13 Les polygones

Voir aussi : 10.9 pour la géométrie 3-d.

9.13.1 L'hexagone : `hexagon hexagone`

Voir aussi : 10.9.1 pour la géométrie 3-d.
Voir aussi : 9.13.2 pour la géométrie 2-d et 10.9.2 pour la géométrie 3-d.
`hexagone`, en géométrie plane, peut avoir de deux à 6 arguments.
Description des arguments :
- si il a deux arguments ce sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
`hexagone(A,B)` renvoie et trace le hexagone $ABCDEF$ de sens direct, mais sans définir les points D,C,E et F.
On tape :

```
hexagone(0,1)
```

On obtient :

```
L'hexagone de sommets
0,1,3/2+i*sqrt(3)/2,1+i*sqrt(3),i*sqrt(3),-1/2+i*sqrt(3)/2
```

- si il a six arguments, les 4 derniers paramètres sont le nom de deux variables qui serviront à définir les deux autres sommets. On tape :

```
hexagone(0,1,C,D,E,F)
```

On obtient :

```
L'hexagone de sommets
0,1,3/2+i*sqrt(3)/2,1+i*sqrt(3),i*sqrt(3),-1/2+i*sqrt(3)/2
```

On tape :

```
affixe(C)
```

On obtient :

`3/2+i*sqrt(3)/2`

On tape :

`affixe(D)`

On obtient :

`1+i*sqrt(3)`

On tape :

`affixe(E)`

On obtient :

`i*sqrt(3)`

On tape :

`affixe(F)`

On obtient :

`-1/2+i*sqrt(3)/2`

9.13.2 Les polygones réguliers : `isopolygon isopolygone`

Voir aussi : 10.9.2 pour la géométrie 3-d.
`isopolygone`, en géométrie plane, a trois arguments.
Les argument sont :
- soit deux points ou deux nombres complexes et un nombre entier positif k
- soit deux points ou deux nombres complexes et un nombre entier négatif k.
Lorsque $k > 0$, `isopolygone` trace le polygone régulier direct ayant k cotés et comme sommets consécutifs les deux premiers arguments.
On tape :

`isopolygone(0,1,4)`

On obtient :

`Le carré de sommets 0,1,1+i,i`

Lorsque $k < 0$, `isopolygone` trace le polygone régulier direct ayant $-k$ cotés, comme centre le premier argument, et comme sommet le deuxième argument.
On tape :

`isopolygone(0,1,-4)`

On obtient :

`carré de sommets 1,i,-1,-i`

9.13.3 Le polygone : `polygon polygone`

Voir aussi : 10.9.3 pour la géométrie 3-d.
`polygone`, en géométrie plane, a comme argument la liste (ou la séquence) de n points ou de n nombres complexes représentant l'affixe de ces points.
`polygone` renvoie et trace le polygone ayant pour sommets ces n points.
On tape :

```
polygone(-1,-1+i/2,i,1+i,-i)
```

On obtient :

`Le polygone de sommets -1,-1+i/2,i,1+i,-i`

On tape :

```
polygone(makelist(x->exp(i*pi*x/3),0,5,1))
```

On obtient :

`L'hexagone de sommets 1,`$e^{\frac{i\pi}{3}}$, $e^{\frac{2i\pi}{3}}$, .., $e^{\frac{5i\pi}{3}}$

9.13.4 La ligne polygonale : `open_polygon polygone_ouvert`

Voir aussi : 10.9.4 pour la géométrie 3-d.
`polygone_ouvert`, en géométrie plane, a comme argument la liste (ou la séquence) de n points ou de n nombres complexes représentant l'affixe de ces points.
`polygone_ouvert` renvoie et trace la ligne polygonale ayant pour sommets ces n points.
On tape :

```
polygone_ouvert(-1,-1+i/2,i,1+i,-i)
```

On obtient :

`La ligne polygonale de sommets -1,-1+i/2,i,1+i,-i`

On tape :

```
polygone_ouvert(makelist(x->exp(i*pi*x/3),0,5,1))
```

On obtient :

`La ligne polygonale de sommets 1,`$e^{\frac{i\pi}{3}}$, $e^{\frac{2i\pi}{3}}$, .., $e^{\frac{5i\pi}{3}}$

9.13.5 L'enveloppe convexe de points du plan : `convexhull`

L'instruction `convexhull` calcule l'enveloppe convexe d'un ensemble de points du plan donné par des points ou des affixes de points, elle renvoie une liste de complexes affixes des sommets de l'enveloppe convexe. L'algorithme utilisé est le scan de Graham. On peut utiliser `polygone` sur le résultat de `convexhull` pour obtenir le tracé de l'enveloppe convexe.

Par exemple, on tape

```
polygone(convexhull(0,1,1+i,1+2i,-1-i,1-3i,-2+i))
```

pour obtenir l'enveloppe convexe des points d'affixe (0,0), (1,0), (1,1), (1,2), (-1,-1), (1,-3), (-2,1).

9.14 Les cercles

9.14.1 Le cercle et ses arcs : `circle cercle`

Voir aussi : 9.14.2 pour les arcs de cercle.
Voir aussi : 10.10 pour les cercles de la géométrie 3-d.
`cercle` a un ou deux arguments pour dessiner un cercle et de quatre à six arguments pour dessiner un arc de cercle.
Description des arguments :

– Avec un argument : l'argument de `cercle` est alors l'équation du cercle ayant comme variables x et y,
– Avec deux arguments `cercle` désigne un cercle :
  Le premier argument est un point ou un nombre complexe considéré comme l'affixe d'un point.
  C'est le deuxième argument qui détermine si on trace le cercle avec la donnée de son centre et de son rayon (le deuxième argument est alors un nombre complexe de module le rayon) ou avec la donnée de son diamètre (le deuxième argument est un point).
  - `cercle(C,r)` où `C` est un point (ou un nombre complexe) et `r` un nombre complexe, trace le cercle de centre `C` et de rayon le module de `r`. Cela est utile, par exemple, pour avoir le cercle de centre `A` qui passe par `B` on tape `cercle(A,B-A)`.
  - `cercle(A,B)` où `A` est un point ou un nombre complexe et `B` un point, trace le cercle de diamètre `AB`.
  On tape :
  ```
  cercle(x^2+y^2-2*x-2*y)
  ```
  On obtient :
  ```
  Le cercle de centre 1+i et de rayon sqrt(2) est
  tracé
  ```
  On tape :
  ```
  cercle(-1,i)
  ```
  On obtient :
  ```
  Le cercle de centre -1 et de rayon 1 est tracé
  ```
  On tape :
  ```
  cercle(-1, point(i))
  ```
  On obtient :
  ```
  Le cercle de diamétre -1,i
  ```
  Attention Il faut bien voir la différence entre :
  `cercle(A,B-A)`, `cercle(A,B)` et `cercle(A,affixe(B)-A)`.
  `A:=point(1);B:=point(i/2);cercle(A,B-A)` ou `cercle(A,affixe(B-A))` ou `cercle(A,abs(B-A))` dessine le cercle de centre `A` passant par `B`,
  `cercle(A,B))` dessine le cercle de diamètre `A,B`,
  `cercle(A,affixe(B)-A)` dessine le cercle de diamètre `A,C` où `C` est le point d'affixe `(-2+i)/2`. En effet `affixe(B)-A)` désigne un point `C` tel que :
  `affixe(C)=affixe(B)-affixe(A)=i/2-1=(-2+i)/2`.
– Avec de quatre à six arguments, `cercle` désigne un arc de cercle. Dans ce cas les deux premiers arguments déterminent le cercle qui porte l'arc (voir

ci-dessus) et les deux arguments suivants sont les angles au centre des points qui délimitent l'arc et les deux derniers arguments sont des noms de variables qui contiendront les points qui délimitent l'arc. Le troisième et le quatrième argument sont les mesures des angles au centre des points qui délimitent l'arc, ces angles sont mesurés en radians (ou en degrés) à partir de l'axe défini par les deux premiers arguments si le deuxième argument est un point (définition du cercle par son diamètre) ou de l'axe défini par son centre C et le point $A = C + r$ si le deuxième argument est un complexe égal à r (définition du cercle par son centre et un complexe dont le module est égal au rayon).
Le cinquième et le sixième argument ne sont pas obligatoires et servent à définir les extrémités de l'arc.
On tape :

```
cercle(-1,1,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`L'arc AB (A=point(0) et B=point(`$\frac{-1+\sqrt{2}+i*\sqrt{2}}{2}$`)) du cercle de centre -1 et de rayon 1 est tracé`

En effet l'angle est compté à partir de l'axe (-1,0) et donc l'angle 0 est le point(0).
On tape :

```
cercle(-1,i,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`L'arc AB (A=point(-1+i) et B=point(`$\frac{-1-\sqrt{2}+i*\sqrt{2}}{2}$`)) du cercle de centre -1 et de rayon 1 est tracé`

En effet, l'angle est compté à partir de l'axe (-1,i-1) et donc l'angle 0 est le point d'affixe i-1.
On tape :

```
cercle(-1, point(i),0,pi/4,A,B)
```

On obtient :

`L'arc AB (A=point(i) et B=point(`$\frac{-1+i*(1+\sqrt{2})}{2}$`)) du cercle de diamétre -1,i`

En effet, l'angle est compté à partir de l'axe (-1,i) et donc l'angle 0 est le point d'affixe i.

9.14.2 Les arcs de cercle : `arc`

Voir aussi : 9.14.1 pour cercles et arcs de cercle.
`arc` a de trois à cinq arguments : deux points `A,B` (ou deux nombres complexes `a,b`) et un nombre réel α représentant la mesure de l'arc `AB` en radians ($-2*\pi \leq \alpha \leq 2*\pi$). le quatrième et le cinquième ne sont pas obligatoires et sont des noms de variables qui contiendront le centre et le rayon du cercle qui porte l'arc.
L'arc `AB` est donc porté par le cercle de centre : $(a+b)/2+i*(b-a)/(2*\tan(\alpha/2))$.
`arc(A,B,`α`)` est l'arc d'où l'on voit le segment `AB` sous l'angle $-\pi + \alpha/2$ si $2\pi > \alpha$>0, ou sous l'angle $\pi + \alpha/2$ si $-2\pi < \alpha$<0.
Pour avoir l'arc capable `AB` de mesure β c'est à dire l'arc d'où l'on voit le segment `AB` sous l'angle $-\pi < \beta < \pi$ il faut taper :

`arc(A,B,2*(-pi+`β`))` si $\pi > \beta$>0 ou
`arc(A,B,2*(pi+`β`))` si $-\pi < \beta$<0 .
Attention Le signe de α donne le sens de parcours de l'arc `AB` par exemple, `arc(A,B,3*pi/2)` et `arc(A,B,-pi/2)` forment un cercle complet.
On tape :

```
arc(1,i,pi/2)
```

On obtient :

```
L'arc (1,i) du cercle de centre 0 et de rayon 1
```

On tape :

```
arc(1,i,pi/2,C,r)
```

On obtient :

```
L'arc (1,i) du cercle de centre C=point(0) et de rayon
r=1
```

On tape :

```
arc(2,2*i,pi,C,r)
```

On obtient :

```
Le demi-cercle de centre C=point(1+i), de rayon
r=sqrt(2) et allant du point(2) au point(2*i) dans le
sens +
```

Remarque
Lorsque `cercle` a quatre arguments, `cercle` dessine aussi un arc de cercle (cf 9.14.1).

9.14.3 Le cercle (compatibilité TI) : `Circle`

`Circle` a quatre arguments : les coordonnées du centre (xc, yc), la valeur du rayon r et 0 ou 1 (par défaut 1).
Si le quatrième paramètre est 1 ou est différent de zéro, `Circle` dessine le cercle de centre (xc, yc) et de rayon r,
Si le quatrième paramètre est 0, `Circle` efface le cercle de centre (xc, yc) et de rayon r.
On tape :

```
Circle(-1,0,2,1)
```

On obtient :

```
Le cercle de centre -1, de rayon 2 est tracé
```

On tape :

```
Circle(-1,0,2,0)
```

On obtient :

```
Le cercle de centre -1, de rayon 2 est effacé
```

9.14.4 Le cercle inscrit : `incircle inscrit`

`inscrit` a trois paramètres qui définissent les sommets d'un triangle.
`inscrit` dessine et renvoie le cercle inscrit de ce triangle.
On tape :

```
inscrit(-1,i,1+i)
```

On obtient :

```
Le cercle inscrit du triangle(-1,i,1+i)
```

9.14.5 Le cercle circonscrit : `circumcircle circonscrit`

`circonscrit` a trois paramètres qui définissent les sommets d'un triangle.
`circonscrit` dessine et renvoie le cercle circonscrit de ce triangle.
On tape :

```
circonscrit(-1,i,1+i)
```

On obtient :

```
Le cercle circonscrit du triangle(-1,i,1+i)
```

9.14.6 Le cercle exinscrit : `excircle exinscrit`

`exinscrit` a trois paramètres qui définissent les sommets d'un triangle.
`exinscrit` dessine et renvoie le cercle exinscrit dans l'angle intérieur du premier sommet de ce triangle.
On tape :

```
exinscrit(-1,i,1+i)
```

On obtient :

```
Le cercle exinscrit dans l'angle de sommet -1 du
          triangle(-1,i,1+i) est tracé
```

9.14.7 Puissance d'un point par rapport à un cercle : `powerpc puissance`

Si un point A est à une distance d du centre d'un cercle C de rayon r, la puissance de A par rapport au cercle C est égale à $d^2 - r^2$.
On tape :

```
puissance(cercle(0,1+i),3+i)
```

On obtient :

```
8
```

En effet : $r = \sqrt{2}$ et $d = \sqrt{10}$ donc $d^2 - r^2 = 8$

9.14.8 Axe radical de deux cercles : `radical_axis axe_radical`

L'axe radical de deux cercles C_1 et C_2 est le lieu des points qui ont même puissance par rapport à C_1 et à C_2.
On tape :

```
axe_radical(cercle(0,1+i),cercle(1,1+i)))
```

On obtient :

```
Le tracé de la droite x=1/2
```

En effet : la droite x=1/2 est la médiatrice du segment [0 ;1]

9.15 Les coniques

9.15.1 L'ellipse : `ellipse`

Voir aussi : 10.11.1 pour la géométrie 3-d.
`ellipse`, en géométrie plane, a 1 ou 3 paramètres :

- un paramètre : son équation de variables x et y. `ellipse(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- trois paramètres : ses deux foyers et un de ces points (ou son affixe si cette affixe n'est pas réelle) ou ses deux foyers et un réel (son demi-grand axe).
  `ellipse(F1,F2,A)` trace l'ellipse passant par `A` et de foyers `F1` et `F2` ou,
  `ellipse(F1,F2,a)` où `a` est un nombre réel, trace l'ellipse de foyers `F1` et `F2` et de demi-grand axe `|a|`.

On tape :

```
ellipse(-i,i,1+i)
```

On obtient :

```
L'ellipse de foyers -i, i et passant par 1+i
```

On tape :

```
ellipse(-i,i,sqrt(5)-1)
```

On obtient :

```
L'ellipse de foyers -i, i et de demi-grand axe
sqrt(5)-1
```

On tape :

```
ellipse(x^2+2*y^2-1)
```

ou on tape :

```
ellipse(sqrt(2)/2,-sqrt(2)/2,1)
```

On obtient :

```
L'ellipse de centre 0 et de demi-grand axe 1 et de
foyers sqrt(2)/2 et -sqrt(2)/2
```

9.15.2 L'hyperbole : `hyperbola hyperbole`

Voir aussi : 10.11.2 pour la géométrie 3-d.
`hyperbole`, en géométrie plane, a 1 ou 3 paramètres :

- un paramètre : son équation de variables x et y. `hyperbole(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- trois paramètres : ses deux foyers et un de ces points (ou son affixe si cette affixe n'est pas réelle) ou ses deux foyers et un réel (son demi-grand axe). `hyperbole(F1,F2,A)` trace l'hyperbole passant par `A` et de foyers `F1` et `F2` ou,
  `hyperbole(F1,F2,a)` où `a` est un nombre réel, trace l'hyperbole de foyers `F1` et `F2` et de demi-grand axe `|a|`.

On tape :

```
hyperbole(-i,i,1+i)
```

On obtient :

```
L'hyperbole de foyers -i, i et passant par 1+i
```

On tape :

```
hyperbole(-i,i,1/2)
```

On obtient :

```
L'hyperbole de foyers -i, i et de demi-grand axe 1/2
```

On tape :

```
hyperbole(x^2+2*y^2-1)
```

ou on tape :

```
hyperbole(sqrt(6)/2,-sqrt(6)/2,1)
```

On obtient :

```
L'hyperbole de centre 0 et de demi-grand axe 1 et de
foyers sqrt(6)/2 et -sqrt(6)/2
```

9.15.3 La parabole : `parabola parabole`

Voir aussi : 10.11.3 pour la géométrie 3-d.
`parabole`, en géométrie plane, a 1 ou 2 paramètres :

- un paramètre : son équation de variables x et y. `parabole(p(x,y))` trace la conique d'équation `p(x,y)=0` si `p(x,y)` est un polynôme de degré 2.
- deux paramètres : deux points (ou leurs affixes si la deuxième affixe n'est pas réelle), représentant son foyer et son sommet ou encore un point (le sommet) ou l'affixe de son sommet et un nombre réel `c`.
  `parabole(F,S)` renvoie et dessine la parabole de foyer `F` et de sommet `S`.

`parabole(S,c)` renvoie et dessine la parabole de sommet `S(xS+i*yS)` et d'équation $y = yS + c*(x - xS)^2$. Il faut savoir que si p est le paramètre de la parabole, on a $FS = p/2$ et $c = 1/(2*p)$.

On tape :

```
parabole(0,i)
```

On obtient :

```
La parabole de foyer 0 et de sommet i
```

On tape :

```
parabole(0,1)
```

On obtient :

La parabole de sommet 0 et déquation $y = x^2$

On tape :

```
parabole(x^2-y-1)
```

ou on tape :

```
parabole(-i,1)
```

ou on tape :

```
parabole(i,-i)
```

On obtient :

```
La parabole de sommet -i et de foyers i
```

9.16 Les mesures

9.16.1 L'affixe d'un point ou d'un vecteur : `affix affixe`

`affixe` est une fonction ayant comme argument un point ou un vecteur ou les coordonnées d'un point ou d'un vecteur 2-d.
`affixe` renvoie l'affixe du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `affixe(A)` renvoie $x_A + i * y_A$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `affixe(A-B)` ou `affixe(vecteur(B,A))` renvoie $x_A - x_B + i * (y_A - y_B)$ (car `A-B` désigne le vecteur $\overrightarrow{BA}$) et `cooedonnees(vecteur(B,A))` renvoie $[x_A + iy_A, x_B + i * y_B]$.
On tape :

```
affixe(point(i))
```

On obtient :

```
i
```

On tape :

```
affixe(point(i)-point(1+2*i))
```

On obtient :

```
-1-i
```

9.16.2 L'abscisse d'un point ou d'un vecteur : `abscissa abscisse`

Voir aussi : 10.12.1 pour la géométrie 3-d.
`abscisse`, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
`abscisse` renvoie l'abscisse du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `abscisse(A)` renvoie x_A,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `abscisse(A-B)` renvoie $x_A - x_B$ (car `A-B` désigne le vecteur $\overrightarrow{BA}$).
On tape :

```
abscisse(point(1+2*i))
```

On obtient :

```
1
```

On tape :

```
abscisse(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
abscisse(1+2*i)
```

On obtient :

```
1
```

On tape :

```
abscisse([1,2])
```

On obtient :

```
1
```

9.16.3 L'ordonnée d'un point ou d'un vecteur : `ordinate ordonnee`

Voir aussi : 10.12.2 pour la géométrie 3-d.
`ordonnee`, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
`ordonnee` renvoie l'ordonnée du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `ordonnee(A)` renvoie y_A,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `ordonnee(A-B)` renvoie $y_A - y_B$ (`A-B` désigne le vecteur $\overrightarrow{BA}$).
On tape :

```
ordonnee(point(1+2*i))
```

On obtient :

```
2
```

On tape :

```
ordonnee(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
ordonnee(1+2*i)
```

On obtient :

```
2
```

On tape :

```
ordonnee([1,2])
```

On obtient :

```
2
```

9.16.4 Les coordonnées d'un point, d'un vecteur ou d'une droite : `coordinates coordonnees`

Voir aussi : 10.12.4 pour la géométrie 3-d.
`coordonnees`, en géométrie plane, est une fonction ayant comme argument un point ou un nombre complexe ou un vecteur ou une droite.
`coordonnees` renvoie la liste de l'abscisse et de l'ordonnée du point ou du vecteur ou la liste des affixes de deux points de la droite orientée.

- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `coordonnees(A)` renvoie $[x_A, y_A]$,

– si le point B a pour coordonnées cartésiennes (x_B, y_B), `coordonnees(vecteur(A,B))` ou `coordonnees(B-A)` renvoie $[x_B - x_A, y_B - y_A]$ (alors que B-A renvoie $(x_B - x_A) + i * (y_B - y_A)$ car B-A désigne l'affixe du vecteur $\overrightarrow{AB}$ en géométrie plane),
– si le vecteur V a pour coordonnées cartésiennes (x_V, y_V), `coordonnees(V)` ou `coordonnees(vecteur(A,V))` renvoie $[x_V, y_V]$,
– si une droite d est définie par deux points A et B, `coordonnees(d)` renvoie `[affixe(A),affixe(B)]`, si d est définie par son équation, `coordonnees(d)` renvoie `[affixe(A),affixe(B)]` où A et B sont deux points de la droite d, le vecteur AB ayant même orientation que d.

On tape :

```
coordonnees(point(1+2*i))
```

Ou on tape :

```
coordonnees(1+2*i)
```

On obtient :

```
[1,2]
```

On tape :

```
coordonnees(point(1+2*i)-point(i))
```

Ou on tape :

```
coordonnees(point(1+2*i)-point(i))
```

On obtient :

```
[1,1]
```

On tape :

```
coordonnees(vecteur(point(i),point(1+2*i)))
```

Ou on tape :

```
coordonnees(vecteur(i,1+2*i))
```

Ou on tape :

```
coordonnees(vecteur([0,1],[1,2]))
```

On obtient :

```
[1,1]
```

On tape :

```
coordonnees(1+2*i)
```

Ou on tape :

```
coordonnees(vecteur(1+2*i))
```

Ou on tape :

```
coordonnees(vecteur(point(i),vecteur(1+2*i)))
```

On obtient :

```
[1,2]
```

On tape :

```
coordonnees(point(i),vecteur(1+2*i))
```

On obtient :

```
[1,2]
```

On tape :

```
d:=droite(-1+i,1+2*i)
```

Ou on tape

```
d:=droite(point(-1,1),point(1,2))
```

Puis,

```
coordonnees(d)
```

On obtient :

```
[-1+i,1+2*i]
```

On tape :

```
d:=droite(y=(1/2*x+3/2))
```

On obtient :

```
[(3*i)/2,1+2*i]
```

On tape :

```
d:=droite(x-2*y+3=0)
```

On obtient :

```
[(3*i)/2,(-4+i)/2]
```

Attention
`coordonnees` peut aussi avoir comme argument une séquence ou une liste de points. `coordonnees` renvoie alors la séquence ou la liste des listes des coordonnées de ces points, par exemple `coordonnees(i,1+2*i)` ou `coordonnees(point(i),point`
renvoie la séquence :
`[0,1],[1,2]`
et
`coordonnees([i,1+2*i])` ou `coordonnees([point(i),point(1+2*i)])`
renvoie la matrice :
`[[0,1],[1,2]]` donc
`coordonnees([1,2])` renvoie la matrice :
`[[1,0],[2,0]]` car `[1,2]` est considéré comme la liste de 2 points d'affixe `1` et `2`.

9.16.5 Les coordonnées rectangulaire d'un point : rectangular_coordinates coordonnees_rectangulaires

rectangular_coordinates ou coordonnees_rectangulaires renvoie la liste de l'abscisse et de l'ordonnée d'un point donné par la liste de ses coordonnées polaires.
On tape :

```
coordonnees_rectangulaires(2,pi/4)
```

Ou on tape :

```
coordonnees_rectangulaires(polar_point(2,pi/4))
```

On obtient :

```
[2/(sqrt(2)),2/(sqrt(2))]
```

9.16.6 Les coordonnées polaire d'un point : polar_coordinates coordonnees_polaires

polar_coordinates ou coordonnees_polaires renvoie la liste du module et de l'argument de l'affixe d'un point (en 2D) ou d'un nombre complexe ou de la liste de coordonnées rectangulaires.
On tape :

```
coordonnees_polaires(1+i)
```

Ou on tape :

```
coordonnees_polaires(point(1+i))
```

Ou on tape :

```
coordonnees_polaires([1,1])
```

On obtient :

```
[sqrt(2),pi/4]
```

9.16.7 L'équation cartésienne d'un objet géométrique : equation

Voir aussi : 10.12.5 pour la géométrie 3-d.
equation permet d'avoir l'équation cartésienne d'un objet géométrique.
Attention ! ! ! il faut auparavant purger les variables x et y en tapant purge(x) et purge(y).
On tape :

```
equation(droite(-1,i))
```

On obtient :

```
(x-y)=-1
```

On tape :

```
equation(cercle(-1,i))
```

On obtient :

```
((x+1)^2+y^2)=1
```

qui est l'équation du cercle de centre -1 et de rayon le module de i.

9.16.8 L'équation paramétrique d'un objet géométrique : `parameq`

Voir aussi : 10.12.6 pour la géométrie 3-d.
`parameq`, en géométrie plane, permet d'avoir l'équation paramétrique d'un objet géométrique sous la forme du nombre complexe $x(t) + i * y(t)$.
Attention ! ! ! il faut auparavant purger la variable `t` en tapant : `purge(t)` ou `t:='t'`.
On tape :

```
parameq(droite(-1,i))
```

On obtient :

```
-t+(1-t)*(i)
```

On tape :

```
parameq(cercle(-1,i))
```

On obtient :

```
-1+exp(i*t)
```

On tape :

```
normal(parameq(ellipse(-1,1,i)))
```

On obtient :

```
sqrt(2)*cos(t)+(i)*sin(t)
```

9.17 Les mesures

9.17.1 Remarques générales sur l'affichage des mesures

Les commandes concernant les mesures et se terminant pas `en` ou `at` (resp `enbrut` ou `atraw`) gérent l'affichage de ces mesures en un point (qui est le dernier argument de ces commandes) avec une légende succinte (resp sans légende). Si on préfère une légende plus sophistiquée il faut utiliser la commande `legende` ou `affichage` avec `legende` comme argument. On tape par exemple :

```
C:=carre(0,1);a:=aire(C);p:=perimetre(C)
```

```
legend(1+i,"aire(C)="+string(a))
```

On obtient :

```
aire(C)=1 s'écrit au d'affixe 1+i
```

puis, on tape :

```
affichage(legend(1,"perimetre(C)="+string(p)),vert)
```

On obtient :

`perimetre(C)=4 s'écrit en vert au point 1`

alors que si on tape :

`aireen(C,1+i)`

On obtient :

`aC=1 s'écrit au d'affixe 1+i`

puis, si on tape :

`perimetreen(C,1)`

On obtient :

`pC=4 s'écrit au d'affixe 1`

ou bien si on tape :

`aireenbrut(C,1+i)`

On obtient :

`1 s'écrit au d'affixe 1+i`

puis, si on tape :

`perimetreenbrut(C,1)`

On obtient :

`4 s'écrit au d'affixe 1`

9.17.2 La longueur d'un segment et distance entre les deux objets géométriques : `distance longueur`

Voir aussi : 10.12.7 pour la géométrie 3-d.
`distance` ou `longueur` a comme argument deux points (ou deux nombres complexes qui sont l'affixe de ces points) ou deux objets géométriques.
`distance` ou `longueur` renvoie la longueur du segment défini par ces deux points ou la distance entre les deux objets géométriques.
On tape :

`longueur(-1,1+i)`

On obtient :

`sqrt(5)`

On tape :

`longueur(0,droite(-1,1+i))`

On obtient :

`sqrt(5)/5`

On tape :

`longueur(cercle(0,1),droite(-2,1+3i))`

On obtient :

`sqrt(2)-1`

Attention Lorsque le calcul de longueur utilise des paramètres, il faut être en mode réel. Par exemple :

```
assume(a=[4,0,5,0.1])};
A:=point(0);
B:=point(a);
simplify(longueur(B,A))
simplify(longueur(A,B))
```

en mode complexe on a :
`simplify(longueur(B,A))` renvoie `a` mais
`simplify(longueur(A,B))` renvoie `-a`.
en mode réel on a :
`simplify(longueur(B,A))` et `simplify(longueur(A,B))` renvoie `c`.

9.17.3 La longueur d'un segment et son affichage : `distanceat distanceen` et `distanceatraw distanceenbrut`

`distanceat` ou `distanceen` et `distanceatraw` ou `distanceenbrut` sont des commandes obtenues lorsqu'on utilse, dans un niveau de géométrie $2d$ ou $3d$, le bouton `Mode->Mesures->ditanceen` ou `Mode->Mesures->distanceenbrut`.
Dans ce cas il faut que tous les objets, sauf les points, désignés par la souris soient déjà nommés.
On peut néammoins taper ces commandes dans une ligne de commande.
`distanceat` ou `distanceen` a 3 arguments : le nom de 2 points et 1 point (ou l'affixe de ce point) ou encore le nom de 2 objets géométriques et un point (ou l'affixe de ce point).
Attention Il faut que les 2 premiers arguments soient des noms.
`distanceat` ou `distanceen` renvoie le point donné en 3-ième argument, calcule la longueur du segment défini par les deux premiers points ou la distance entre les 2 objets géométriques et affiche à l'endroit du 3-ième point, cette longueur précédée d'une légende.
On tape (on doit donnerle nom des objets) :

`A:=point(-1);B:=point(1+i);segment(A,B);`

`distanceat(A,B,0.4i)`

ou

`distanceen(A,B,0.4i)`

On obtient :

```
AB=sqrt(5) s'écrit au point(0.4i)
```

On tape (on doit donnerle nom des objets) :

```
C:=point(0);d:=droite(-1,1+i)
distanceen(C,d,i/2)
```

On obtient :

```
Cd=sqrt(5)/5 s'écrit au point(i/2)
```

On tape (on doit donner le nom des objets) :

```
c:=cercle(0,1); d:=droite(-2,1+3i)
distanceen(c,d,0)
```

On obtient :

```
cd=sqrt(2)-1 s'écrit au point(0)
```

`distanceatraw` ou `distanceenbrut` a comme argument trois points (ou 2 points et 1 nombre complexe qui est l'affixe d'un point) ou encore 2 objets géométriques et un point (ou l'affixe de ce point).
`distanceatraw` ou `distanceenbrut` renvoie le point donné en 3-ième argument, calcule la longueur du segment défini par les deux premiers points ou la distance entre les 2 ojets géométriques et affiche cette longueur à l'endroit du 3-ième point.
On tape :

```
A:=point(-1);B:=point(1+i);segment(A,B);
distanceenbrut(A,B,0.4i)
```

On obtient :

```
sqrt(5) s'écrit au point(0.4i)
```

On tape :

```
C:=point(0);d:=droite(-1,1+i)
distanceenbrut(C,d,i/2)
```

ou on tape (on n'est pas obligé de donner un nom aux objets) :

```
distanceenbrut(0,droite(-1,1+i),i/2)
```

On obtient :

```
sqrt(5)/5 s'écrit au point(i/2)
```

On tape :

```
c:=cercle(0,1); d:=droite(-2,1+3i)
distanceenbrut(c,d(-2,1+3i),0)
```

ou on tape (on n'est pas obligé de donner un nom aux objets) :

```
distanceenbrut(cercle(0,1),droite(-2,1+3i),0)
```

On obtient :

```
sqrt(2)-1 s'écrit au point(0)
```

9.17.4 Le carré de la longueur d'un segment : `distance2 longueur2`

Voir aussi : 10.12.8 pour la géométrie 3-d.
`longueur2` a comme argument deux points (ou 2 points et 1 nombre complexe qui est l'affixe d'un point).
`longueur2` renvoie le carré de la longueur du segment défini par ces deux points.
On tape :

```
longueur2(-1,1+i)
```

On obtient :

```
5
```

9.17.5 La mesure d'un angle : `angle`

Voir aussi : 10.12.9 pour la géométrie 3-d.
`angle` a comme argument trois points (ou trois nombres complexes qui sont l'affixe de ces points) ou 2 points et une droite et éventuellement une chaine de caractères pour afficher un petit arc de cercle avec ce symbole en légende afin de représenter l'angle sur la figure (l'arc est remplacé par le symbole quart de carre si l'angle vaut pi/2 ou -pi/2).
`angle` renvoie la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième argument se trouve sur le premier coté de l'angle et le troisième argument se trouve sur le deuxiéme coté.
Remarque
Si le troisième argument est une droite, cette droite donne la direction du 2ième côté de l'angle. Ainsi :
`angle(A,B,C)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
`angle(A,B,C,"")` trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté.
`angle(A,B,C,"a")` trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté noté a.
`angle(A,B,C,"")[0]` ou `angle(A,B,C,"a")[0]` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
On tape :

```
angle(0,1,1+i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
pi/4
```

On tape :

```
angle(0,1,1+i,"")
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
[pi/4,cercle(point(0,0),1/5)]
```

et l'angle est repéré par un arc de cercle sans légende.
On tape :

```
angle(0,1,1+i,"a")
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
[pi/4,cercle(point(0,0),1/5)]
```

et l'angle est repéré par un arc de cercle avec `a` comme légende.
On tape :

```
angle(0,1,i,"a")
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
[pi/2,polygone(point(1/5,0),point(1/5,1/5),point(0,1/5),point(0,1/5))]
```

et l'angle droit est repéré par un quart de carré avec `a` comme légende.

9.17.6 La mesure d'un angle et son affichage : `angleat angleen` et `angleatraw angleenbrut`

`angleat` ou `angleen` (resp `angleatraw` ou `angleenbrut`) sont des commandes qui permettent l'affichage avec légende (resp sans légende) de la valeur d'un angle désigné en cliquant (on cliqueson sommet puis ses 2 côtés et enfin la position de la l'affichage). Ces commandes sont obtenues lorsqu'on utilse, dans un niveau de géométrie $2d$ ou $3d$, le bouton `Mode->Mesures->angleen` ou `Mode->Mesures->angleenbrut`. Dans ce cas il faut que tous les objets désignés par la souris soient déjà nommés, sauf les points qui seront alors définis par les différents clicks de la souris.

On peut néammoins taper ces commandes dans une ligne de commande.
`angleat` ou `angleen` a comme argument quatre points (ou 3 points et 1 nombre complexe qui est l'affixe d'un point).
Attention Il faut que les 3 premiers points aient un nom.
`angleat` ou `angleen` renvoie le 4-ième point, calcule la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième argument se trouve sur le premier coté de l'angle et le troisième argument se trouve sur le deuxiéme coté et cette mesure est affichée, précédée d'une légende, à l'endroit du 4-ième point.
Ainsi `anglen(A,B,C,D)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB},\overrightarrow{AC})$ et cette messue sera affichée, pécédée de αA=, à l'endroit du point `D`.
On tape :

```
A:=point(-1);B:=point(1+i);C:=point(i);
segment(A,B);segment(A,C);
```

`angleen(A,B,C,0.2i)`

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

α`A=atan(1/3) s'écrit au point(0.4i)`

`angleatraw` ou `angleenbrut` a comme argument quatre points (ou 3 points et 1 nombre complexe qui est l'affixe d'un point).
`angleatraw` ou `angleenbrut` renvoie le 4-ième point, calcule la mesure (en radians ou en degrés) de l'angle orienté de sommet le premier argument, le deuxième point se trouve sur le premier coté de l'angle et le troisième point se trouve sur le deuxiéme coté et cette mesure est affichée à l'endroit du 4-ième point. Ainsi `angleenbrut(A,B,C,D)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB},\overrightarrow{AC})$ et cette mesure sera affichée à l'endroit du point `D`.
On tape :

```
A:=point(-1);B:=point(1+i);C:=point(i);
segment(A,B);segment(A,C);
angleenbrut(A,B,C,0.2i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`atan(1/3) s'écrit au point(0.4i)`

9.17.7 Représentation graphique de l'aire d'un polygone : `tracer_aire graphe_aire aire_graphe plotarea areaplot`

Avec un polygone comme argument, `aire_graphe` ou `graphe_aire` ou `tracer_aireplotarea` permet de représenter et d'afficher l'aire d'un polygone. L'aire est positive si le le polygone est direct et est négative sinon On tape :

```
plotarea(polygone(point(1),point((1+i)*1/2),point(1+i)))
```

On obtient :

`le polygone rempli avec -1/4 affiché au point(1)`

On tape :

```
plotarea(polygone(point(1),point(1+i),point((1+i)*1/2)))
```

On obtient :

`le polygone rempli avec 1/4 affiché au point(1)`

Remarque Si on ne voit pas l'affichage, on tape par exemple :

```
plotarea(polygone(point(0),point(1),point(1+i),point((1+i)/2),point(i)))
```

On obtient :

```
le polygone rempli, mais ne l'affichage au point(0)
                     est caché
```

On tape alors :

```
plotarea(polygone(point(0),point(1),point(1+i),point((1+i)/2),point(i)),affic
```

On obtient :

```
le polygone rempli, avec 3/4 affiché au au point(0)
```

On peut aussi taper, pour n'avoir que la valeur :

```
plotarea(polygone(point(0),point(1),point(1+i),point((1+i)/2),point(i)))[3]
```

On obtient :

```
3/4
```

9.17.8 Aire d'un polygone : `area aire`

`aire` calcule l'aire d'un cercle ou d'un polygone étoilé par rapport à son premier sommet.
On tape :

```
aire(triangle(0,1,i))
```

On obtient :

```
1/2
```

On tape :

```
aire(carre(0,2))
```

On obtient :

```
4
```

9.17.9 L'aire d'un polygone et son affichage : `areaat aireen` et `areaatraw aireenbrut`

`areaat` ou `aireen` et `areaatraw` ou `aireenbrut` sont des commandes obtenues lorsqu'on utilse, dans un niveau de géométrie $2d$ ou $3d$, le bouton `Mode->Mesures->aireen` ou `Mode->Mesures->airenbrut`. Dans ce cas il faut que tous les objets désignés par la souris soient déjà nommés.
On peut néammoins taper ces commandes dans une ligne de commande.

`areaat` ou `aireen` et `areaatraw` ou `aireenbrut` ont comme arguments un cercle ou un polygone étoilé par rapport à son premier sommet et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`areaat` ou `aireen` renvoie le point, calcule l'aire du cercle ou polygone étoilé par rapport à son premier sommet et affiche cette aire à l'endroit du point avec une légende. **Attention** Il faut que le cercle ou le polygone ait un nom.
`areaatraw` ou `aireenbrut` renvoie le point, calcule l'aire du cercle ou du polygone étoilé par rapport à son premier sommet et affiche cette aire à l'endroit du point.
On tape :

```
t:=triangle(0,1,i)
```

```
aireen(t,(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2 avec la légende
```

On tape :

```
aireenbrut(triangle(0,1,i),(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2
```

On tape :

```
cc:=cercle(0,2)
```

```
aireen(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
aireenbrut(cercle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
c:=carre(0,2)
```

```
aireen(c,2.2)
```

On obtient :

```
4 s'écrit au point(2.2) avec une légende
```

On tape :

`aireenbrut(carre(0,2),2.2)`

On obtient :

`4 s’écrit au point(2.2)`

On tape :

`h:=hexagone(0,1)`

`aireen(h,1.2)`

On obtient :

`3*sqrt(3)/2 s’écrit au point(1.2) avec une légende`

On tape :

`aireenbrut(hexagone(0,1),1.2)`

On obtient :

`3*sqrt(3)/2 s’écrit au point(1.2)`

9.17.10 Périmètre d’un polygone : perimeter perimetre

Voir aussi `arcLen` **??**.
`perimetre` calcule le périmètre d’un cercle, d’un arc de cercle ou d’un polygone.
On tape :

`perimetre(cercle(0,1))`

On obtient :

`2*pi`

On tape :

`perimetre(cercle(0,1,pi/4,pi))`

On obtient :

`3*pi/4`

On tape :

`perimetre(arc(0,pi/4,pi))`

On obtient :

`sqrt(2)*pi/4`

On tape :

`perimetre(triangle(0,1,i))`

On obtient :

`2+sqrt(2)`

On tape :

`perimetre(carre(0,2))`

On obtient :

`8`

9.17.11 Périmètre d'un polygone et son affichage :`perimeterat perimetreen` et `perimeteratraw perimetreenbrut`

`perimeterat` ou `perimetreen` et `perimeteratraw` ou `perimetreenbrut` sont des commandes obtenues lorsqu'on utilse, dans un niveau de géométrie $2d$ ou $3d$, le bouton `Mode->Mesures->perimetreen` ou `Mode->Mesures->perimetreenbrut`.
Dans ce cas il faut que tous les objets désignés par la souris soient déjà nommés.
On peut néammoins taper ces commandes dans une ligne de commande.

`perimeterat` ou `perimetreen` et `perimeteratraw` ou `perimetreenbrut` ont comme argument un cercle ou un polygone et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`perimeterat` ou `perimetreen` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point avec une légende. **Attention** Il faut que le polygone ait un nom.
`perimeteratraw` ou `perimetreenbrut` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point.
On tape :

```
t:=triangle(0,1,i)
```

```
perimetreen(t,(1+i)/2)
```

On obtient :

`2+sqrt(2) s'écrit au point((1+i)/2) avec une légende`

On tape :

```
perimetreenbrut(triangle(0,1,i),(1+i)/2)
```

On obtient :

`2+sqrt(2) s'écrit au point((1+i)/2)`

On tape :

```
c:=carre(0,2)
```

```
perimetreen(c,2.2)
```

On obtient :

`8 s'écrit au point(2.2) avec une légende`

On tape :

```
perimetreenbrut(carre(0,2),2.2)
```

On obtient :

`8 s'écrit au point(2.2)`

On tape :

```
cc:=cercle(0,2)
```

```
perimetreen(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
perimetreenbrut(cercle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
h:=hexagone(0,1)
```

```
perimetreen(h,1.2)
```

On obtient :

```
6 s'écrit au point(1.2) avec une légende
```

On tape :

```
perimetreenbrut(hexagone(0,1),1.2)
```

On obtient :

```
6 s'écrit au point(1.2)
```

9.17.12 Pente d'une droite : `slope pente`

`slope pente` est soit une commande soit un attribut de la commande `droite` pour cela voir 9.10.1
Lorsque `slope pente` est une commande son argument est une droite ou un segment ou deux points ou deux nombres complexes.
`slope pente` renvoie la pente de la droite définie par le segment ou par les 2 points ou leurs affixes.
On tape :

```
pente(droite(1,2i))
```

Ou on tape :

```
pente(segment(1,2i))
```

Ou on tape :

```
pente(point(1),point(2i))
```

Ou on tape :

`pente(1,2i)`

On obtient :

`-2`

On tape :

`pente(droite(2y-x=3))`

On obtient :

`1/2`

On tape :

`pente(tangente(plotfunc(sin(x)),pi/4))`

Ou on tape :

`pente(droite_tangente(sin(x),pi/4))`

On obtient :

`(sqrt(2))/2`

9.17.13 Pente d'une droite et son affichage : `slopeat, penteen` et `slopeatraw, penteenbrut`

`slopeat` ou `penteen` et `slopeatraw` ou `penteenbrut` est une commande obtenue lorsqu'on utilse, dans un niveau de géométrie $2d$ ou $3d$, le bouton `Mode->Mesures->pentee` ou `Mode->Mesures->penteenbrut`. Dans ce cas il faut que tous les objets désignés par la souris soient déjà nommés.
On peut néammoins taper ces commandes dans une ligne de commande.
`slopeat` ou `penteen` ont 2 arguments le nom d'une droite (ou d'un segment) et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
et `slopeatraw` ou `penteenbrut` ont comme arguments une droite (ou un segment) et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`slopeat` ou `penteen` renvoie le point, calcule la pente de la droite (ou du segment) et affiche cette pente à l'endroit du point avec une légende. **Attention** `slopeat` ou `penteen`, il faut mettre le nom de la droite (ou du segment).
`slopeatraw` ou `penteenbrut` renvoie le point, calcule la pente de la droite (ou du segment) et affiche cette pente à l'endroit du point.
On tape :

`d:=droite(1,2i)`

`penteen(d,i)`

On obtient :

`"sd=-2" s'écrit au point(i)`

On tape :

```
penteenbrut(droite(1,2i),i)
```

Ou on tape :

```
penteenbrut(segment(1,2i),i)
```

On obtient :

```
-2 s'écrit au point(i)
```

On tape :

```
d1:=droite(2y-x=3),2*i)
```

```
penteen(d1,2*i)
```

On obtient :

```
"sd1=1/2" s'écrit au point(2*i) avec une légende
```

On tape :

```
penteenbrut(droite(2y-x=3),2*i)
```

On obtient :

```
1/2 s'écrit au point(2*i)
```

On tape :

```
T:=tangente(plotfunc(sin(x)),pi/4)
```

```
slopeat(T,i)
```

On obtient :

```
"sT=(sqrt(2))/2" s'écrit au point(i)
```

On tape :

```
penteenbrut(tangente(plotfunc(sin(x)),pi/4),i)
```

Ou on tape :

```
penteenbrut(droite_tangente(sin(x),pi/4),i)
```

On obtient :

```
(sqrt(2))/2 s'écrit au point(i)
```

9.17.14 Avoir comme réponse la valeur d'une mesure affichée : extract_measure, extraire_mesure

extract_measure ou extraire_mesure permet d'obtenir la valeur d'une mesure qui a été affichée.
extract_measure ou extraire_mesure a pour argument la commande qui a affiché cette mesure.
On tape :

```
A:=point(-1);B:=point(1+i);C:=point(i);
```

```
extraire_mesure(angleen(A,B,C,0.2i))
```

ou

```
extract_measure(angleat(A,B,C,0.2i))
```

On obtient :

```
atan(1/3)
```

On tape :

```
extraire_mesure(distanceenbrut(A,B,C,0.2i))
```

ou

```
extract_measure(distanceatraw(A,B,0.4i))
```

On obtient :

```
sqrt(5)
```

9.17.15 Le rayon d'un cercle : radius rayon

rayon a comme argument un cercle.
rayon renvoie la longueur du rayon de ce cercle.
On tape :

```
rayon(cercle(-1,i))
```

On obtient :

```
1
```

On tape :

```
rayon(cercle(-1,point(i)))
```

On obtient :

```
sqrt(2)/2
```

9.17.16 La longueur d'un vecteur : `abs`

`abs` a comme argument un nombre complexe ou un vecteur défini par deux points (c'est à dire la différence de ces deux points).
`abs` renvoie le module du nombre complexe, ou la longueur du vecteur donné en argument.
On tape :

```
abs(1+i)
```

Ou on tape :

```
abs(point(1+2*i)-point(i))
```

On obtient :

```
sqrt(2)
```

9.17.17 L'angle d'un vecteur avec Ox : `arg`

`arg` a comme argument un nombre complexe ou un vecteur défini par deux points (c'est à dire la différence de ces deux points).
`arg` renvoie l'argument du nombre complexe ou l'angle polaire du vecteur donné en argument.
On tape :

```
arg(1+i)
```

On obtient :

```
pi/4
```

9.17.18 Pour normaliser un nombre complexe : `normalize`

`normalize` divise le nombre complexe par son module pour avoir un nombre complexe de module 1.
On tape :

```
normalize(3+4*i)
```

On obtient :

```
(3+4*i)/5
```

9.18 Les transformations

9.18.1 Généralités

Les transformations ci-dessous :
`translation, rotation, homothetie, similitude, symetrie, projection, inversion`
peuvent toujours être considérées, soit comme des fonctions (les arguments sont les paramètres servant à définir la transformation), soit comme agissant sur le dernier

argument (les premiers arguments sont les paramètres servant à définir la transformation et l'objet géométrique à transformer est mis comme dernier paramètre).
L'objet géométrique à transformer peut être de tout type comme point, droite, cercle, polygone, courbe paramètrée, graphe de fonctions etc... Par exemple si `P` est l'objet géométrique à transformer on peut avoir :
`P:=point(1+i)` ou `P:=droite(x=1)` ou `P:=plan(z=0)` ou `P:=cercle(point(1,1),1)` ou `P:=plotfunc(x2-2)` etc...

9.18.2 La translation : `translation`

Voir aussi : 10.14.2 pour la géométrie 3-d.
`translation`, en géométrie plane, a un ou deux arguments : le vecteur de translation donné par un vecteur géométrique ou par la liste de ses coordonnées ou par son affixe (la différence de deux points ou un nombre complexe) et éventuellement l'objet géométrique à transformer.
Lorsque `translation` a un argument, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
t:=translation(1+i)
```

Puis :

```
t(-2)
```

On obtient :

```
Le point -1+i tracé avec une croix (x) noire
```

Lorsque `translation` a deux arguments, `translation` dessine et renvoie le transformé du deuxième argument dans la translation de vecteur le premier argument.
On tape :

```
translation([1,1],-2)
```

Ou on tape :

```
A:=point(1);B:=point(2+i);translation(vecteur(A,B),-2)
```

Ou on tape :

```
translation(1+i,-2)
```

Ou on tape :

```
A:=point(1);B:=point(2+i);translation(B-A,-2)
```

On obtient :

```
Le point -1+i tracé avec une croix (x) noire
```

On tape :

```
translation(1+i,droite(-2,-i))
```

On obtient :

```
La droite passant par -1+i et 1
```

9.18.3 La symétrie droite et la symétrie point : `reflection symetrie`

Voir aussi : 10.14.3 pour la géométrie 3-d.
`symetrie`, en géométrie plane, a un ou deux arguments : un point ou une droite et éventuellement l'objet géométrique à transformer.
Lorsque `symetrie` a un argument, c'est une fonction qui agit sur un objet géométrique : quand le premier argument est un point (ou un nombre complexe) il s'agit de la symétrie par rapport à ce point (ou par rapport au point d'affixe ce nombre complexe) et quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite.
On tape :

```
sp:=symetrie(-1)
```

Puis :

```
sp(1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
sd:=symetrie(droite(-1,i))
```

Puis :

```
sd(1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

Lorsque `symetrie` a deux arguments, `symetrie` dessine et renvoie le transformé du deuxième argument dans la symétrie définie par le premier argument : quand le premier argument est un point (ou un nombre complexe) il s'agit de la symétrie par rapport à ce point (ou par rapport au point d'affixe ce nombre complexe) et quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite.
On tape :

```
symetrie(-1,1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
symetrie(droite(-1,i),1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

9.18.4 La rotation : `rotation`

Voir aussi : 10.14.4 pour la géométrie 3-d.
`rotation`, en géométrie plane, a deux ou trois arguments.
Lorsque `rotation` a deux arguments ce sont : un point (le centre de rotation) et un réel (la mesure de l'angle de rotation) ; `rotation` est alors une fonction qui agit sur un objet géométrique (point, droite etc...)
On tape :

```
r:=rotation(i,-pi/2)
```

Puis :

```
r(1+i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point 0 tracé avec une croix (x) noire
```

Lorsque `rotation` a trois arguments, ce sont : un point (le centre de rotation), un réel (la mesure de l'angle de rotation) et l'objet géométrique à transformer ; `rotation` dessine et renvoie alors le transformé du troisième argument dans la rotation de centre le premier argument et d'angle de mesure le deuxième argument.
On tape :

```
rotation(i,-pi/2,1+i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point 0 tracé avec une croix (x) noire
```

On tape :

```
rotation(i,-pi/2,droite(1+i,-1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
La droite passant par 0 et -1+2*i
```

9.18.5 L'homothétie : `homothety homothetie`

Voir aussi : 10.14.5 pour la géométrie 3-d.
`homothetie`, en géométrie plane, a deux ou trois arguments : un point (le centre de l'homothétie), un réel (la valeur du rapport de l'homothétie) et éventuellement l'objet géométrique à transformer.
Lorsque `homothetie` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
h:=homothetie(i,2)
```

Puis :

```
h(1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

Lorsque `homothetie` a trois arguments, `homothetie` dessine et renvoie le transformé du troisième argument dans l'homothétie de centre le premier argument et de rapport le deuxième argument.
On tape :

```
homothetie(i,2,1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

On tape :

```
homothetie(i,2,cercle(1+i,1))
```

On obtient :

```
Le cercle de centre 2+i et de rayon 2
```

Remarque
Lorsque la valeur du rapport de l'homothétie est un nombre complexe k non réel `homothetie(A,k)` est la similitude de centre le point `A`, de rapport `abs(k)` et d'angle `arg(k)`.

9.18.6 La similitude : `similarity similitude`

Voir aussi : 10.14.6 pour la géométrie 3-d.
`similitude`, en géométrie plane, a trois ou quatre arguments : un point (le centre de rotation), un réel (la valeur du rapport k de la similitude), un réel (la mesure a de l'angle de rotation en radians (ou degrés)) et éventuellement l'objet géométrique à transformer.
Remarque : si le rapport k est négatif, l'angle de la similitude est alors de mesure $-a$ radians (ou degrés).
Lorsque `similitude` a trois arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
s:=similitude(i,2,-pi/2)
```

Puis :

```
s(1+i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
s(cercle(1+i,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le cercle de centre -i et de rayon 2
```

Lorsque `similitude` a quatre arguments, `similitude` dessine et renvoie le transformé du quatrième argument dans la similitude de centre le premier argument de rapport le deuxième argument et d'angle le troisième argument.
On tape :

```
similitude(i,2,-pi/2,1+i)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
similitude(i,2,-pi/2,cercle(1+i,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le cercle de centre -i et de rayon 2
```

Remarque
En 2d la similitude de centre le point `A`, de rapport `k` et d'angle `a` se traduit par :
`similitude(A,k,a)` ou par `homothetie(A,k*exp(i*a))`.

9.18.7 L'inversion : `inversion`

Voir aussi : 10.14.7 pour la géométrie 3-d.
`inversion`, en géométrie plane, a deux ou trois arguments : un point (le centre de l'inversion), un réel (la valeur du rapport de l'inversion) et éventuellement l'objet géométrique à transformer.
Lorsque `inversion` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
Si `inver:=inversion(C,k)` et `A1:=inver(A)`, on a $\overline{CA} * \overline{CA1} = k$.
On tape :

```
inver:=inversion(i,2)
```

Puis :

```
inver(cercle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
inver(cercle(1+i,1/2))
```

On obtient :

```
Le cercle de centre 8/3+i et de rayon 4/3, il passe
par le point 4+i
```

Lorsque `inversion` a trois arguments, `inversion` dessine et renvoie le transformé du troisième argument dans l'inversion de centre le premier argument et de rapport le deuxième argument.
Si `A1:=inversion(C,k,A)` on a $\overline{CA} * \overline{CA1} = k$.
On tape :

```
inversion(i,2,cercle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
inversion(i,2,cercle(1+i,1/2))
```

On obtient :

```
Le cercle de centre 8/3+i et de rayon 4/3 (il passe
par le point 4+i)
```

9.18.8 La projection orthogonale : `projection`

Voir aussi : 10.14.8 pour la géométrie 3-d.
`projection`, en géométrie plane, a un ou deux arguments : un objet géométrique et éventuellement un point.
Lorsque `projection` a un argument, c'est une fonction qui agit sur un point et qui projette orthogonalement ce point sur l'objet géométrique.
On tape :

```
p1:=projection(droite(-1,i))
```

Puis :

```
p1(1+i)
```

On obtient :

```
Le point 1/2+3/2*i apparait avec une croix (x) noire
```

On tape :

```
p2:=projection(cercle(-1,1))
p2(i)
```

On obtient :

```
Le point d'affixe, sqrt(2)/2+(i)*sqrt(2)/2-1, apparait
            avec une croix (x) noire
```

Lorsque `projection` a deux arguments, `projection` dessine et renvoie le transformé du point par la projection orthogonale sur le premier argument.
On tape :

```
projection(droite(-1,i),1+i)
```

On obtient :

```
Le point 1/2+3/2*i apparait avec une croix (x) noire
```

On tape :

```
projection(cercle(-1,1),i)
```

On obtient :

```
Le point d'affixe, -1+sqrt(2)/2+(i)*sqrt(2)/2,
      apparait avec une croix (x) noire
```

9.19 Les propriétés

9.19.1 Savoir si 1 point est sur un objet graphique : `is_element` `est_element`

Voir aussi : 10.13.1 pour la géométrie 3-d.
`est_element` est une fonction booléenne ayant comme argument un point et un objet géométrique.
`est_element` vaut 1 si le point appartient à l'objet géométrique, et vaut 0 sinon.
On tape :

```
est_element(point(-1-i),droite(0,1+i))
```

On obtient :

```
1
```

On tape :

```
est_element(point(i),droite(0,1+i))
```

On obtient :

```
0
```

9.19.2 Savoir si 3 points sont alignés : `is_collinear est_aligne`

Voir aussi : 10.13.6 pour la géométrie 3-d.
`est_aligne` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`est_aligne` vaut 1 si les points sont alignés, et vaut 0 sinon.
On tape :

```
est_aligne(0,1+i,-1-i)
```

On obtient :

```
1
```

On tape :

```
est_aligne(i/100,1+i,-1-i)
```

On obtient :

```
0
```

9.19.3 Savoir si 4 points sont cocycliques : `is_concyclic est_cocyclique`

Voir aussi : 10.13.7 et 10.13.8 pour la géométrie 3-d.
`is_concyclic` ou `est_cocyclique` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`est_cocyclique` vaut 1 si les points sont cocycliques, et vaut 0 sinon.
On tape :

```
est_cocyclique(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
est_cocyclique(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

9.19.4 Savoir si 1 point est à l'intérieur d'un polygone ou d'un cercle : `is_inside est_dans`

`is_inside` ou `est_dans` est une fonction booléenne ayant comme argument un point et un polygone ou un cercle.
`is_inside` ou `est_dans` vaut 1 si le point est à l'intérieur ou sur le bord du polygone ou du cercle, et vaut 0 sinon.
On tape :

```
est_dans((point(0),cercle(-1,1))
```

On obtient :

```
1
```

On tape :

```
est_dans(point(2),polygone([1,2-i,3+i]))
```

On obtient :

```
1
```

On tape :

```
est_dans(point(1-i),triangle([1,2-i,3+i]))
```

On obtient :

```
0
```

9.19.5 Savoir si on a un triangle équilatéral : `is_equilateral est_equilateral`

Voir aussi : 10.13.9 pour la géométrie 3-d.
`est_equilateral` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`est_equilateral` vaut 1 si les trois points forment un triangle équilatéral ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
est_equilateral(0,2,1+i*sqrt(3))
```

On obtient :

```
1
```

On tape :

```
T:=triangle_equilateral(0,2,C);est_equilateral(T[0])
```

On obtient :

```
1
```

En effet `T[0]` désigne un triangle car `T` est une liste composée du triangle et de son sommet `C`.
On tape `affixe(C)` et on obtient `1+i*sqrt(3)`
On tape :

```
est_equilateral(1+i,-1+i,-1-i)
```

On obtient :

```
0
```

9.19.6 Savoir si on a un triangle isocéle : `is_isosceles est_isocele`

Voir aussi : 10.13.10 pour la géométrie 3-d.
`est_isocele` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`est_isocele` vaut 1 (resp 2, 3) si les trois points forment un triangle isocéle ou si l'objet géométrique est un triangle isocéle dont l'angle portant les deux cotés égaux est désigné par le premier (resp second, troisième) argument, ou vaut 4 si les trois points forment un triangle équilatéral, ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
est_isocele(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
T:=triangle_isocele(0,1,pi/4);est_isocele(T)
```

On obtient :

```
1
```

On tape :

```
T:=triangle_isocele(0,1,pi/4,C);est_isocele(T[0])
```

On obtient :

```
1
```

En effet `T[0]` désigne un triangle car `T` est une liste composée du triangle et de son sommet `C`.
On tape `affixe(C)` et on obtient `(sqrt(2))/2+((i)*sqrt(2))/2`
On tape :

```
est_isocele(1+i,-1+i,-i)
```

On obtient :

```
3
```

9.19.7 Savoir si on a un triangle rectangle ou si on a un rectangle : `is_rectangle est_rectangle`

Voir aussi : 10.13.11 pour la géométrie 3-d.
`est_rectangle` est une fonction booléenne ayant comme argument trois ou quatre points ou un objet géométrique.
`est_rectangle` vaut 1 (resp 2 ou 3) si les trois points forment un triangle rectangle, l'angle droit étant désigné par le premier (resp second, troisième) argument ou si l'objet géométrique est un triangle rectangle,
`est_rectangle` vaut 1 (resp 2) si les quatre points forment un rectangle (resp un carré) ou si l'objet géométrique est un rectangle (resp un carré), et vaut 0 sinon.
On tape :

```
est_rectangle(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
est_rectangle(1+i,-2+i,-2-i,1-i)
```

On obtient :

```
1
```

On tape :

```
R:=rectangle(-2-i,1-i,3,C,D);est_rectangle(R[0])
```

On obtient :

```
1
```

En effet `R[0]` désigne un rectangle car `R` est une liste composée du rectangle et de ses sommets `C` et `D`.

9.19.8 Savoir si on a un carré : `is_square est_carre`

Voir aussi : 10.13.12 pour la géométrie 3-d.
`est_carre` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_carre` vaut 1 si les quatre points forment un carré ou si l'objet géométrique est un carré, et vaut 0 sinon.
On tape :

```
est_carre(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
K:=carre(1+i,-1+i);est_carre(K)
```

On obtient :

```
1
```

On tape :

```
K:=carre(1+i,-1+i,C,D);est_carre(K[0])
```

On obtient :

```
1
```

En effet `K[0]` désigne un carré car `K` est une liste composée d'un carré et de ses sommets `C` et `D`.
Si on tape `affixe(C,D)`, on obtient `-1-i,1-i`.
On tape :

```
est_carre(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

9.19.9 Savoir si on a un losange : `is_rhombus est_losange`

Voir aussi : 10.13.13 pour la géométrie 3-d.
`est_losange` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_losange` vaut 1 (rep 2) si les quatre points forment un losange (resp un carré) ou si l'objet géométrique est un losange (resp un carré), et vaut 0 sinon.
On tape :

```
est_losange(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
K:=losange(1+i,-1+i,pi/4);est_losange(K)
```

On obtient :

```
1
```

On tape :

```
K:=losange(1+i,-1+i,pi/4,C,D);est_losange(K[0])
```

On obtient :

```
1
```

En effet `K[0]` désigne un losange car `K` est une liste composée d'un losange et de ses sommets `C` et `D`.
Si on tape `normal(coordonnees(C,D))`, on obtient `[-sqrt(2)-1,-sqrt(2)+1],[-sqrt(2)+1,-s`
On tape :

```
est_losange(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

9.19.10 Savoir si on a un parallélogramme : `is_parallelogram` `est_parallelogramme`

Voir aussi : 10.13.14 pour la géométrie 3-d.
`est_parallelogramme` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_parallelogramme` vaut 1 (resp 2, 3, 4) si les quatre points forment un parallélogramme (resp un losange, un rectangle, un carré) ou si l'objet géométrique est un parallélogramme (resp un losange, un rectangle, un carré), et vaut 0 sinon.
On tape :

```
est_parallelogramme(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

On tape :

```
est_parallelogramme(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
Q:=quadrilatere(1+i,-1+i,-1-i,1-i);est_parallelogramme(Q)
```

On obtient :

```
4
```

Attention
On doit taper :

```
P:=parallelogramme(-1-i,1-i,i,D);est_parallelogramme(P[0])
```

Pour obtenir :

```
1
```

En effet c'est `P[0]` qui désigne un parallélogramme car `P` est une liste composée d'un parallélogramme et de son dernier sommet `D`.
Si on tape `affixe(D))`, on obtient `-2+i`.

9.19.11 Savoir si 2 droites sont parallèles : `is_parallel` `est_parallele`

Voir aussi : 10.13.3 pour la géométrie 3-d.
`est_parallele`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`est_parallele` vaut 1 si les deux droites sont parallèles, et vaut 0 sinon.
On tape :

```
est_parallele(droite(0,1+i),droite(i,-1))
```

On obtient :

```
1
```

On tape :

```
est_parallele(droite(0,1+i),droite(i,-1-i))
```

On obtient :

```
0
```

9.19.12 Savoir si 2 droites 2-d sont perpendiculaires : `is_perpendicular` `est_perpendiculaire`

Voir aussi : 10.13.4 pour la géométrie 3-d.
`est_perpendiculaire`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`est_perpendiculaire` vaut 1 si les deux droites sont perpendiculaires, et vaut 0 sinon.
On tape :

```
est_perpendiculaire(droite(0,1+i),droite(i,1))
```

On obtient :

```
1
```

On tape :

```
est_perpendiculaire(droite(0,1+i),droite(1+i,1))
```

On obtient :

```
0
```

9.19.13 Savoir si 2 cercles sont orthogonaux : `is_orthogonal` `est_orthogonal`

Voir aussi : 10.13.5 pour la géométrie 3-d.
`est_orthogonal` est une fonction booléenne ayant comme argument deux droites ou deux cercles.
`est_orthogonal` vaut 1 si les deux droites sont perpendiculaires ou si les deux cercles sont orthogonaux (si les tangentes en leurs points d'intersection sont orthogonales), et vaut 0 sinon.
On tape :

```
est_orthogonal(droite(0,1+i),droite(i,1))
```

On obtient :

```
1
```

On tape :

```
est_orthogonal(line(2,i), line(0,1+i))
```

On obtient :

```
0
```

On tape :

```
est_orthogonal(cercle(0,1+i),cercle(2,1+i))
```

On obtient :

```
1
```

On tape :

```
est_orthogonal(cercle(0,1),cercle(2,1))
```

On obtient :

```
0
```

9.19.14 Savoir si des éléments sont conjugués : `is_conjugate est_conjugue`

`est_conjugue` permet de savoir si 4 points sont conjugués ou si 2 points ou 2 droites ou 1 droite et 1 point sont conjugués par rapport à un cercle ou deux droites.
`est_conjugue` est une fonction booléenne ayant comme arguments deux points (resp deux droites ou un cercle) suivi de deux points ou de deux droites ou d'une droite et d'un point.
`est_conjugue` vaut 1 si les arguments sont conjugués, et vaut 0 sinon.
On tape :

```
est_conjugue(cercle(0,1+i),point(1-i),point(3+i))
```

On obtient :

```
1
```

On tape :

```
est_conjugue(cercle(0,1),point((1+i)/2),droite(1+i,2))
```

Ou on tape :

```
est_conjugue(cercle(0,1),droite(1+i,2),point((1+i)/2))
```

On obtient :

```
1
```

On tape :

```
est_conjugue(cercle(0,1),droite(1+i,2),
          droite((1+i)/2,0))
```

On obtient :

```
1
```

On tape :

```
est_conjugue(point(1+i),point(3+i),point(i),point(i+3/2))
```

On obtient :

```
1
```

On tape :

```
est_conjugue(droite(0,1+i),droite(2,3+i),
    droite(3,4+i),droite(3/2,5/2+i))
```

On obtient :

```
1
```

9.19.15 Savoir si 4 points forment un division harmonique : `is_harmonic` `est_harmonique`

`est_harmonique` permet de savoir si 4 points forment un division harmonique.
`est_harmonique` est une fonction booléenne ayant comme arguments quatre points.
`est_harmonique` vaut 1 si les 4 points forment un division harmonique et vaut 0 sinon.
On tape :

```
est_harmonique(0,2,3/2,3)
```

On obtient :

```
1
```

On tape :

```
est_harmonique(0,1+i,1,i)
```

On obtient :

```
0
```

9.19.16 Ces droites sont en faisceau ? is_harmonic_line_bundle est_faisceau_droite

est_faisceau_droite a comme argument une liste de droites.
est_faisceau_droite renvoie :
1 si ces droites sont concourantes en un point,
2 si ces droites sont parallèles,
3 si ces droites sont confondues,
et 0 sinon.
On tape :

```
est_faisceau_droite([droite(0,1+i),droite(0,2+i),
          droite(0,3+i),droite(0,1)])
```

On obtient :

```
1
```

9.19.17 Ces cercles sont-ils en faisceau : is_harmonic_circle_bundle est_faisceau_cercle

est_faisceau_cercle a comme argument une liste de cercles.
est_faisceau_cercle renvoie :
1 si ces cercles forment un faisceau (c'est à dire si ils ont 2 à 2 le même axe radical),
2 si ces cercles sont concentriques,
3 si ces cercles sont confondus,
et 0 sinon.
On tape :

```
est_faisceau_cercle([cercle(0,i),cercle(4,i),
          cercle(0,point(1/2))])
```

On obtient :

```
1
```

9.20 La division harmonique, pôles et polaires

9.20.1 Point divisant un segment dans le rapport k : division_point point_div

point_div a trois arguments : deux points (ou deux nombres complexes a et b) et un nombre complexe k.
point_div renvoie et dessine le point d'affixe z tel que :
$\frac{z-a}{z-b} = k$ On tape :

```
point_div(i,2+i,3+i)
```

On obtient :

```
pnt((5+4*i)/(2+i),0) et le dessin de ce point
```

On tape :

```
point_div(point(i),point(2+i),3)
```

On obtient :

```
pnt(3+i,0) et le dessine de ce point
```

Remarque : 0 représente la couleur du point.

9.20.2 Le birapport de 4 points alignés : `cross_ratio birapport`

`birapport` a comme argument 4 nombres complexes a, b, c, d.
`birapport` renvoie le birapport deces 4 nombres à savoir :
$\frac{c-a}{c-b} / \frac{d-a}{d-b}$ On tape :

```
birapport(0,1,2,3)
```

On obtient :

```
4/3
```

On tape :

```
birapport(i,2+i,3/2+i,3+i)
```

On obtient :

```
-1
```

9.20.3 Division harmonique : `harmonic_division div_harmonique`

Quatre points alignés $A,\ B,\ C,\ D$ sont en division harmonique si on a :

$$\frac{\overline{CA}}{\overline{CB}} = -\frac{\overline{DA}}{\overline{DB}} = k$$

On dit aussi que C et D divisent le segment AB dans le rapport k et que le point D est le conjugué harmonique de C par rapport à A et B ou plus rapidement D est le conjugué harmonique de A, B, C.
Quatre droites concourantes ou parallèles $d1,\ d2,\ d3,\ d4$ sont en division harmonique si elles déterminent sur chaque droite sécante une division harmonique. On dit aussi que $d1,\ d2,\ d3,\ d4$ forment un faisceau harmonique.
`div_harmonique` a comme arguments 3 points alignés ou leur 3 affixes (resp 3 droites concourantes ou parallèles) et le nom d'une variable.
`div_harmonique` affecte le dernier argument pour que l'on obtienne une division harmonique et renvoie la liste des 4 points (resp des 4 droites) et dessine les points (resp les droites).
On tape :

```
div_harmonique(0,2,3/2,D)
```

On obtient :

`[0,2,3/2,pnt(3,0,"D")] et seul le point` D `est dessiné`

On tape :

```
div_harmonique(point(0),point(2),point(3/2),D)
```

On obtient :

```
[pnt(0,0),pnt(2,0),pnt(3/2,0), pnt(3,0,"D")] et les 4
points sont dessinés
```

Remarque : 0 représente la couleur du point.
On tape :

```
div_harmonique(droite(i,0),droite(i,1+i),
droite(i,3+2*(i)),D)
```

On obtient :

```
[pnt([[i,0],0]),pnt([[i,1+i],0]),
pnt([[i,3+2*i],0]),pnt([[i,-3+2*i],0,"D"])] et les 4
droites sont dessinées
```

9.20.4 Le conjugué harmonique : `harmonic_conjugate conj_harmonique`

`conj_harmonique` a comme arguments 3 points alignés A, B, C (resp 3 droites concourantes ou parallèles).
`conj_harmonique` renvoie et dessine le conjugué harmonique de C par rapport à A et B.
On tape :

```
conj_harmonique(0,2,3/2)
```

On obtient :

```
pnt(3,0)et dessine ce point
```

On tape :

```
conj_harmonique(droite(0,1+i),droite(0,3+i),droite(0,i))
```

On obtient :

```
pnt([[0,3+2*i],0]) et dessine cette droite
```

9.20.5 Pôle et polaire : `pole` et `polar polaire`

`polaire` a comme argument un cercle C et un point A (ou un nombre complexe).
`polaire` renvoie et dessine la polaire du point A par rapport au cercle C : c'est la droite qui est le lieu des conjugués de A par rapport au cercle C.
`pole` a comme argument un cercle C et une droite d.
`pole` renvoie et dessine le pôle de d par rapport au cercle C : c'est le point A admettant d comme polaire par rapport à C. On tape :

```
polaire(cercle(0,1),(point(1+i))/2)
```

On obtient :

```
pnt([[2,2*i],0]) et dessine cette droite
```

On tape :

```
pole(cercle(0,1),droite(i,1))
```

On obtient :

```
pnt(1+i,0) et dessine ce point
```

9.20.6 Polaire réciproque : `reciprocation polaire_reciproque`

`polaire_reciproque` a comme argument un cercle C et une liste de points et de droites.
`polaire_reciproque` renvoie la liste obtenue en remplacant dans la liste argument un point (resp une droite) par sa polaire (resp son pôle) par rapport au cercle C.
On tape :

```
polaire_reciproque(cercle(0,1),[point((1+i)/2),
                  droite(1,-1+i)])
```

On obtient :

```
[pnt([[2,2*i],0]),pnt(1+2*i,0)] et dessine cette
                  droite et ce point
```

9.21 Les lieux et les enveloppes

9.21.1 Les lieux : `locus lieu`

`lieu` permet de tracer le lieu d'un point qui dépend d'un autre point qui doit être défini avec la fonction `element` et `lieu` permet aussi de tracer l'enveloppe d'une droite qui dépend d'un point qui doit être défini avec la fonction `element` (voir aussi `enveloppe` pour l'enveloppe d'une famille de droites qui dépendent d'un paramètre 9.21.2).

– lieu d'un point.
  `lieu` a 2 à 4 arguments.
  Les deux premiers argument sont deux noms de variables :
  le premier argument est le nom du point dont on veut connaitre le lieu et ce point est fonction du deuxième argument,
  le deuxième argument est le nom du point qui se déplace sur une courbe `C` et qui doit être défini par `element(C)`.
  On peut préciser éventuellement en troisième argument l'intervalle où se trouve le paramètre `t` utilisé pour le paramétrage de `C` lorsque le deuxième argument décrit `C` et en quatrième argument préciser la valeur de `tstep`.
  Remarque
  Pour connaitre le paramétrage de la courbe `C` on utilise la commande `parameq(C)`.

`lieu` dessine le lieu du premier argument quand le deuxième argument se déplace selon ce que l'on a spécifié comme argument de `element`.

L'algorithme

Si on cherche le lieu de N lorsque M se déplace sur une courbe C, l'algorithme cherche les niveaux qui se trouvent entre le niveau où se trouve la définition de M et celui où se trouve la définition de N et choisit un paramétrage rationnel de la courbe où M se déplace, puis effectue les calculs de ces niveaux intermédiaires, puis trace une courbe paramétrée `X(t)+i*Y(t)` qui dépend du t défini dans `cfg`, puis affiche l'équation du lieu par le calcul de `resultant(X(t)-x,Y(t)-y,t)`.
Il ne faut donc pas avoir d'autres instructions `lieu` dans les instructions intermédiaires.

Conseils

Il faut avoir le moins possible d'instructions entre la définition de M et l'instruction `lieu`.
On tape, pour avoir le lieu du centre de gravité G du triangle de sommets A point(-1), B point(1) et P lorsque P décrit la droite d'équation $y = 1$:

```
P:=element(droite(i,1+i))
G:=isobarycentre(-1,1,P)
lieu(G,P)
```

On obtient :

```
La droite parallèle à l'axe des x passant par i/3
```

Attention

Il faut régler correctement le paramètre t, c'est à dire les valeurs `t-` et `t+` de la fenêtre de configuration graphique pour avoir le lieu entièrement ! ! !
On tape, pour avoir le lieu du centre de gravité G du triangle de sommets A point(-1), B point(1) et P lorsque P décrit le segment [-3+i ; 3+i]. Pour connaitre le paramétrage de la droite `d:=droite(i,1+i)` on utilise `parameq(d)` :

```
d:=droite(i,1+i):;parameq(d)
```

On obtient comme paramétrage :

```
t+i
P:=element(d)
G:=isobarycentre(-1,1,P)
lieu(G,P,t=-3..3)
plotparam(t+i,t=-3..3);triangle(-1,1,P)
```

On obtient comme lieu :

```
Le segment [-1+i/3; 1+i/3]
```

On peut préciser la valeur de `tstep` en quatrième argument, on tape :

```
lieu(G,P,t=-3..3,tstep=0.1)
```

– enveloppe d'une droite fonction d'un point qui se déplace sur une courbe.
`lieu` a comme arguments deux noms de variables : le premier argument est le nom de la droite dont on veut connaitre l'enveloppe et cette droite est fonction du deuxième argument. Le deuxième argument est le nom du point qui se déplace et qui doit être défini avec la fonction `element`.
`lieu` dessine l'enveloppe du premier argument quand le deuxième argument se déplace selon ce que l'on a spécifié comme argument de `element`.
On tape, pour avoir le l'enveloppe de la médiatrice de FH lorsque H décrit

la droite d'équation $x = 0$:

```
F:=point(1)
H:=element(droite(x=0))
d:=mediatrice(F,H)
lieu(d,H)
```

On obtient :

```
La parabole de foyer F et de directrice l'axe des
y dont l'èquation est 2*x-y^2-1=0
```

– enveloppe d'une droite donnée par une équation dépendant d'un paramètre (voir aussi la commande `enveloppe` 9.21.2),
Dans ce cas, il faut dire que la paramètre est l'affixe d'un point de la droite $y = 0$.
Par exemple, enveloppe d'une famille de droites d'équation $y + x\tan(t) - 2\sin(t) = 0$ lorsque $t \in \mathbb{R}$. (cf **??**)
On tape :

```
H:=element(droite(y=0));
D:=droite(y+x*tan(affixe(M))-2*sin(affixe(M)))
lieu(D,H)
```

On obtient :

```
L'astroide d'équation paramérique
2*cos(t)^3+2*i*sin(t)^3
```

Si on veut l'enveloppe lorsque $t = 0..\pi$, on tape :

```
lieu(D,H,t=0..pi)
```

On obtient :

La partie au dessus de $y = 0$ de l'astroide
d'équation paramérique 2*cos(t)^3+2*i*sin(t)^3

On peut aussi chercher l'intersection de `D` et de `E` (voir leur définition ci-dessous) pour avoir l'équation paramétrique du lieu.

```
D:=y+x*tan(t)-2*sin(t)
E:=diff(D,t)
M:=linsolve([D=0,E=0],[x,y])
P:=plotparam(affixe(M))
```

On obtient :

```
L'astroide d'équation paramérique
2*cos(t)^3+2*i*sin(t)^3
```

en effet `simplify(M)` renvoie :
`[2*cos(t)^3,2*sin(t)^3]`

9.21.2 Les enveloppes : `envelope enveloppe`

`enveloppe` a 2 arguments : une expression Xpr dépendant de 3 variables x, y, t (resp u, v, t) et t (resp le vecteur $[u, v, t]$).
Les equations $Xpr = 0$ sont considérées comme les équations de courbes de paramètre t.
`enveloppe` renvoie l'enveloppe de ces courbes lorsque t varie.
On tape :

```
enveloppe(y+x*tan(t)-2*sin(t),t)
```

On obtient :

`le dessin d'une astroïde`

On tape :

`enveloppe(v+u*tan(t)-3*sin(t),[u,v,t])`

On obtient :

`le dessin d'une astroïde`

Remarque Si on a une famille de droites `d` qui dépendent d'un paramètre `a`, on peut utiliser soit `enveloppe(equation(d,a))` soit la commande `lieu` (voir 9.21.1).
Voici un exemple simple : enveloppe de la tangente à un cercle.
On tape :

```
c:=cercle(0,2):;
assume(a:=1);
A:=element(c,a);
d:=tangente(cercle(0,2),A);
```

Puis soit :

`enveloppe(equation(d),a)`

soit :

`lieu(d,A)`

9.21.3 La trace d'un objet géométrique : `trace`

`trace` a comme arguments un objet géométrique qui dépend ou non d'un paramètre (voir aussi `trace` d'une matrice 6.46.3).
`trace` dessine la trace de cet objet géométrique lorsqu'on fait varier le paramètre ou lorsqu'on fait bouger cet objet géométrique en mode `Pointeur`.
Exemple Trouver le lieu géométrique des points du plan équidistants d'une droite D et d'un point F. Pour cela on construit un point H sur la droite D et la droite T perpendiculaire à D en H. M de T équidistant de F et de H.
On ouvre un niveau de géométrie (`Alt+g`), puis on tape les instructions suivantes en mettant une instruction par ligne.
On tape :

```
A:=point(-3-i);
B:=point(1/2+2*(i));
D:=droite(A,B,'couleur'=0);
F:=point(4/3,1/2,'couleur'=0);
assume(a=[0.7,-5,5,0.1]);
H:=element(D,a)
T:=perpendiculaire(H,D)
M:=inter_unique(mediatrice(H,F),T);
trace(M);
```

puis on fait bouger le curseur `a` et on obtient la trace de `M`.

Remarque Pour effacer les traces ou pour rajouter des traces ou pour les activer ou les desactiver il faut utiliser le menu `Trace` du bouton `M` situé à droite de l'écran de géométrie.

Chapitre 10

Les fonctions de géométrie 3-d

10.1 Généralités

Les graphes ou les dessins de la géométrie 3-d se font dans un écran graphique 3-d qui s'ouvre automatiquement en réponse d'une commande graphique 3-d.
Les dessins de la géométrie 3-d se font en général dans un écran de `géométrie 3-d` qui est un écran graphique muni d'un éditeur de commandes et d'une barre de menus at que l'on ouvre avec `Alt+h`.
Si on clique dans la fenêtre graphique avec la souris, en dehors du parallélépipéde servant à la représentation, on peut faire tourner les axes x, y ou z, soit avec les touches `x`, `X`, `y`, `Y`, `z`, `Z`, soit en bougeant la souris sans relacher son bouton. Cela modifie l'axe de vision (axe passant par l'observateur et de vecteur directeur la direction de la visée de l'objet) et le plan de vision (plan perpendiculaire à l'axe de vision dont l'équation est inscrite en haut de l'écran). Le plan de vision est matérialisé par son intersection avec le parallélépipéde servant à la représentation, ces droites d'intersection sont dessinée en pointillé.
On peut aussi translater le plan de vision, le long de l'axe de vision grâce à la molette de la souris : les plans successifs sont obtenus par une translation de vecteur parallèle à l'axe de vision.
On peut se servir d'attributs pour faire une représentation graphique 3-d comme la couleur, l'épaisseur, les lignes en pointillé pour cela voir 9.3. Mais, les points ont toujours la forme d'un carré et il faut mettre une epaisseur d'au moins 3 si on veut le voir (`point_width=3`).
On peut faire des dessins en perspective ou en repère orthonormé (en cochant `Proj_ortho` dans la configuration graphique (bouton `cfg`)), les surfaces sont transparentes ou non et peuvent être éclairées par 8 spots que l'on peut placer en diffèrents endroits repérés par leur coordonnées homogènes (on configure ses spots avec les boutons `l0,l1..l7` situés dans la configuration graphique).
Ces dessins sont interactifs : on peut faire bouger, avec la souris, les points situés dans le plan de vision, et aussi déplacer ces points, avec la molette de la souris, sur une parallèle à l'axe de vision .

À noter que l'on peut aussi faire un zoom-in ou un zoom-out à l'aide des boutons `in` et `out` (voir 3.4).

Si dans la configuration du graphique, on coche `hidden3d`, la surface sera tracée sans dessiner les lignes qui sont cachées et si on veut voir les lignes cachées on décoche `hiden3d` (voir aussi 1.6.2).

Pour la traduction Latex de l'écran `3-d` on se reportera à la section 1.10.5.

10.2 Les angles d'Euler

Les angles d'Euler sont utilisés pour modifier le repère de visualisation. **Rappel** Soient deux repères : l'ancien $(Oxyz)$ et le nouveau $(OXYZ)$.
Soit Ou l'intersection du plan (OY, OZ) avec le plan (Ox, Oz) que l'on oriente arbitrairement.
Soient :
- Ra la rotation d'axe Oy et d'angle $a = (\overrightarrow{Oz}, \overrightarrow{Ou})$ qui transforme Ox en Ov et Oz en Ou,
- Rb la rotation d'axe Ou et d'angle $b = (\overrightarrow{Ov}, \overrightarrow{OX})$ qui transforme Ov en OX et Oy en Ow,
- Rc la rotation d'axe OX et d'angle $c = (\overrightarrow{Ou}, \overrightarrow{OZ})$ qui transforme Ow en OY et Ou en OZ.

On définit complètement la mise en place de $(OXYZ)$ par rapport à $(Oxyz)$ en donnant les angles a,b,c de $(OXYZ)$ par rapport à $(Oxyz)$ et en effectuant la composition de ces trois rotations : $Rc@Rb@Ra$.
Les angles d'Euler sont :
$a = (\overrightarrow{Oz}, \overrightarrow{Ou})$,
$b = (\overrightarrow{Ov}, \overrightarrow{OX})$,
$c = (\overrightarrow{Ou}, \overrightarrow{OZ})$.

Les dessins de la géométrie 3-d se font en choissisant comme repère $Oxyz$, Ox horizontal dirigé vers la droite, Oy vertical dirigé vers le haut et l'axe des z qui pointe vers vous.
Les mesures en degré de a, b, c sont mises dans ry, rz, rx.
Selon l'orientation de Ou, les valeurs de a, b, c ne sont pas uniques :
a, b, c et $a + 180, 180 - b, c + 180$ mettent en place le même repère $OXYZ$,
Lorsque b est un angle droit, c'est à dire que l'axe OX et l'axe Oy ont le même support on n'a pas non plus unicité :
$a, 90, c$ et $a + c, 90, 0$ mettent en place le même repère $OXYZ$ et
$a, -90, c$ et $a - c, -90, 0$ mettent en place le même repère $OXYZ$.
On peut donc choisir, l'angle b dans $]-90, 90[$ et les angles a et c dans $]-180, 180]$ ou bien b dans $-90, 90$ $c = 0$ et a dans $]-180, 180]$.

10.3 Les axes

10.3.1 Tracer les vecteurs unitaires : `vecteur_unitaire_Ox_3d Ox_3d_unit_vector`, `vecteur_unitaire_Oy_3d Oy_3d_unit_vector` `vecteur_unitaire_Oz_3d Oz_3d_unit_vector`

`vecteur_unitaire_Ox_3d()`, `Ox_3d_unit_vector()` trace le vecteur unitaire de l'axe des x de écran de géométrie 3-d.
`vecteur_unitaire_Oy_3d()`, `Oy_3d_unit_vector()` trace le vecteur unitaire de l'axe des y de écran de géométrie 3-d.
`vecteur_unitaire_Oz_3d()`, `Oz_3d_unit_vector()` trace le vecteur unitaire de l'axe des z de écran de géométrie 3-d.

Vous pouvez effacer les axes (resp les faire réapparaitre) en cochant (resp décochant) `Montrer les axes` avec le bouton `cfg` de l'écran de géométrie.
Ces commandes n'ont pas de paramètre. On peut toutefois rajouter une légende avec la commande `legende`
On tape :

```
vecteur_unitaire_Ox_3d(),legende(point([1,0,0]),"i",vert)
```

On obtient :

`Le vecteur unitaire de l'axe des` x `de écran de géométrie 3-d avec` i `écrit en vert`

On tape :

```
vecteur_unitaire_Oy_3d(),legende(point([0,1,0]),"j",vert)
```

On obtient :

`Le vecteur unitaire de l'axe des` y `de écran de géométrie 3-d avec` j `écrit en vert`

On tape :

```
vecteur_unitaire_Oz_3d(),legende(point([0,0,1]),"k",vert)
```

On obtient :

`Le vecteur unitaire de l'axe des` z `de écran de géométrie 3-d avec` k `écrit en vert`

10.3.2 Tracer un repère : `repere_3d frame_3d`

`repere_2d() frame_2d()` trace le repère de écran de géométrie 2-d.
Vous pouvez effacer les axes (resp les faire réapparaitre) en cochant (resp décochant) `Montrer les axes` avec le bouton `cfg` de l'écran de géométrie.
Ces commandes n'ont pas de paramètre.
On tape :

```
repere_3d()
```

On obtient :

`Le repère de écran de géométrie 3-d`

10.4 Les points

10.4.1 Définir un point 3-d : point

Voir aussi : 10.4.1 pour la géométrie plane.
Pour obtenir un point il suffit d'être en mode point et de cliquer dans le parallélépipéde servant à la représentation.
On peut aussi utiliser la commande point :
point, en géométrie 3-d, a comme argument 3 nombres réels ou une liste de 3 nombres réels [xa,ya,za].
point([xa,ya,za]) définit le point de coordonnées [xa,ya,za].
On tape :

```
point(1,2,5)
```

Ou

```
point([1,2,5])
```

On obtient :

```
Le tracé du point de coordonnées [1,2,5]
```

10.4.2 Définir un point 3-d au hasard : point3d

point3d a comme argument une séquence de noms de points.
point3d définit au hasard, les coordonnées entières (entre -5 et +5) des points 3-d donnés en argument.
On tape :

```
point3d(A,B,C)
```

Puis on tape :

```
plan(A,B,C)
```

On obtient :

```
Le tracé du plan ABC
```

10.4.3 Un des points d'intersection de deux objets géométriques : single_inter line_inter inter_unique inter_droite

Voir aussi : 9.9.5 pour la géométrie plane.
inter_unique ou inter_droite a 2 ou 3 arguments qui sont deux objets géométriques et éventuellement un 3ième argument qui est soit un point soit une liste de points.
inter_unique renvoie l'un des points d'intersection de ces deux objets géométriques. Si on a mis un point A comme troisième argument inter_unique renvoie le point d'intersection le plus proche de A et si on a mis une liste de points L comme troisième argument inter_unique renvoie le point d'intersection qui ne se trouve pas dans la liste L.
On tape :

```
A:=inter_unique(plan(point(0,1,1),point(1,0,1),point(1,1,0)),droite(point(0,0
```

On tape :

```
coordonnees(A)
```

On obtient :

```
[2/3,2/3,2/3]
```

On tape :

```
B:=inter_unique(sphere(point(0,0,0),1),droite(point(0,0,0),point(1,1,1)))
coordonnees(B)
```

On obtient :

```
[1/(sqrt(3)),1/(sqrt(3)),1/(sqrt(3))]
```

On tape :

```
B1:=inter_unique(sphere(point(0,0,0),1),droite(point(0,0,0),point(1,1,1)),poi
coordonnees(B1)
```

On obtient :

```
[1/(sqrt(3)),1/(sqrt(3)),1/(sqrt(3))]
```

On tape :

```
B1:=inter_unique(sphere(point(0,0,0),1),droite(point(1,0,0),point(1,1,1)))
coordonnees(B1)
```

On obtient :

```
[1,0,0]
```

On tape :

```
B1:=inter_unique(sphere(point(0,0,0),1),droite(point(1,0,0),point(1,1,1)),[po
coordonnees(B1)
```

On obtient :

```
[1/3,2/3,2/3]
```

10.4.4 Les points d'intersection de deux objets géométriques : `inter`

Voir aussi : 9.9.6 pour la géométrie plane.
`inter` a 2 arguments qui sont deux objets géométriques.
`inter` renvoie la liste des points (ou la courbe) d'intersection de ces deux objets géométriques.
On tape :

```
LA:=inter(plan(point(0,1,1),point(1,0,1),point(1,1,0)),droite(point(0
```

```
coordonnees(LA)
```

On obtient :

```
[[2/3,2/3,2/3]]
```

On tape :

```
LB:=inter(sphere(point(0,0,0),1),droite(point(0,0,0),point(1,1,1)))
```

```
coordonnees(LB)
```

On obtient :

```
[[1/(sqrt(3)),1/(sqrt(3)),1/(sqrt(3))],[-(1/(sqrt(3))),-(1/(sqrt(3)))
```

On tape :

```
coordonnees(LB[0])
```

On obtient :

```
[1/(sqrt(3)),1/(sqrt(3))]
```

On tape :

```
coordonnees(LB[1])
```

On obtient :

```
[-(1/(sqrt(3))),-(1/(sqrt(3))),-(1/(sqrt(3)))]
```

On tape :

```
C:=inter(sphere(point(0,0,0),1),droite(point(0,0,0),point(1,1,1)),poi
```

```
coordonnees(C)
```

On obtient :

```
[1/(sqrt(3)),1/(sqrt(3)),1/(sqrt(3))]
```

On tape :

```
LB:=inter(sphere(point(0,0,0),1),plan(point(0,0,0),point(1,0,0),point
```

On obtient :

```
un cercle
```

10.4.5 Le milieu d'un segment : `midpoint milieu`

Voir aussi : 9.9.8 pour la géométrie plane.
`milieu`, en géométrie 3-d, a comme argument 2 points ou une liste de 2 points.
`milieu` renvoie et dessine le point milieu du segment défini par ces deux points.
On tape :

```
milieu(point(1,4,0),point(1,-2,0))
```

On obtient :

```
Le point(1,1,0) est tracé
```

10.4.6 L'isobarycentre de n points : `isobarycenter isobarycentre`

Voir aussi : 9.9.9 pour la géométrie plane.
`isobarycentre`, en géométrie 3-d, a comme argument la liste ou la séquence de n points.
`isobarycentre` renvoie et trace un point qui est l'isobarycentre de ces n points.
On tape :

```
isobarycentre(point(1,4,0),point(1,-2,0))
```

On obtient :

```
Le point (1,1,0) est tracé
```

10.4.7 Point défini comme barycentre de n points : `barycenter barycentre`

Voir aussi : 9.9.10 pour la géométrie plane et 6.12.10.
`barycentre` ou `barycenter`, , en géométrie 3-d, a comme argument des listes de longueur 2 (resp une matrice ayant deux colonnes) :
le premier élément de la liste j (resp le $j$ième élément de la première colonne de la matrice) contient le point A_j, le deuxième élément de la liste j (resp le $j$ième élément de la deuxième colonne) contient le coefficient réel α_j affecté à A_j.
`barycentre` ou `barycenter` renvoie et trace le point qui est le barycentre des points A_j affectés des coefficients réels α_j lorsque $\sum \alpha_j \neq 0$.
Si $\sum \alpha_j = 0$, `barycentre` ou `barycenter` renvoie une erreur.
On tape :

```
barycentre([point(1,4,0),1],[point(1,-2,0),1])
```

Ou on tape :

```
barycentre([[point(1,4,0),1],[point(1,-2,0),1]])
```

On obtient :

```
Le point (1,1,0) est tracé
```

10.5 Les lignes

10.5.1 Définir une droite 3-d : `line droite`

Voir aussi : 3.10.1 et9.10.1 pour la géométrie plane.
`droite`, en géométrie 3-d, a comme argument deux points ou un point et son vecteur directeur ou deux équations de plans.
`droite` renvoie et trace la droite définit par ses arguments.
On tape :

```
droite([0,3,0],point([3,0,3]))
```

On obtient :

```
Le tracé de la droite passant par les points de
coordonnèes [0,3,0] et [3,0,3]
```

On tape :

```
droite([0,3,0],[3,0,3])
```

On obtient :

```
Le tracé de la droite passant par le point de
coordonnèes [0,3,0] et de vecteur directeur [3,0,3]
donc passant par le point de coordonnées [3,3,3]
```

On tape :

```
droite(x=y,y=z)
```

On obtient :

```
Le tracé de la droite intersection des plans x=y et
y=z
```

On tape :

```
point3d(A,B)
```

Puis on tape :

```
droite(A,B)
```

On obtient :

```
Le tracé de la droite AB
```

Remarque
`droite` définit une droite orientée :
Lorsque la droite est donnée par deux points, son orientation est définie par l'ordre des points donnés en argument. Par exemple `droite(A,B)` définit une droite orientée par le vecteur $\overrightarrow{AB}$.
Lorsque la droite est donnée par 1 point et son vecteur directeur, son orientation est définie par le vecteur directeur donné en argument. Par exemple `droite(A,[u1,u2,u3])` définit une droite orientée par le vecteur $[u1, u2, u3]$.
Lorsque la droite est donnée par deux équations de plans, son orientation est définie par le produit vectoriel des normales aux plans (en mettant les équations des plans sous la forme " membredegauche-membrededroite=0" on détermine les normales orientées de ces plans). Par exemple `droite(x=y,y=z)` est orientée par `cross([1,-1,0],[0,1,-1])=[1,1,1]`.

10.5.2 Définir une droite orientée en 3-d : `line droite`

Voir aussi : 9.10.1 pour la géométrie plane.
`droite`, en géométrie 3-d, a comme argument deux points ou deux équations de plans : `a*x+by+cz+d=0` et `a'*x+b'y+c'z+d'=0`.
Attention l'ordre des arguments que l'on donne à droite est important et un ordre différent change l'orientation !
Si la droite est définie par deux points, ces points orientent la droite selon leur position dans les arguments. Par exemple `droite(A,B)` définit une droite orientée par le vecteur $\overrightarrow{AB}$.
Si la droite est définie par deux équations, on écrit ces équations sous la forme "membre_de_gauche-membre_de_droite=0" pour avoir les équations sous la forme `a*x+by+c=0` et `a'*x+b'y+c'z+d'=0`. Alors, le vecteur orientant la droite est le produit vectoriel des 2 vecteurs normaux aux 2 plans definis par les deux équations c'est à dire le vecteur orientant est `cross([a,b,c],[a',b',c'])`. Par exemple `droite(x=2*y,y=3*z)` est orientée par `cross([1,-2,0],[0,1,-3])=[6,3,1]` et
`droite(y=3*z,x=2*y)` est orientée par `cross([0,1,-3],[1,-2,0])=[-6,-3,-1]`
.

10.5.3 La demi-droite en 3-d : `half_line demi_droite`

Voir aussi : 9.10.2 pour la géométrie plane.
`demi_droite`, en géométrie 3-d, a comme argument 2 points.
`demi_droite` renvoie et trace la demi-droite d'origine le premier argument est passant par le deuxième argument.
On tape :

```
demi_droite(point(0,0,0),point(1,1,1))
```

On obtient :

```
La demi-droite d'origine O, passant par le
point(1,1,1)
```

10.5.4 Le segment en 3-d : `segment`

Voir aussi : 9.10.3 pour la géométrie plane.
`segment`, en géométrie 3-d, a comme argument 2 points.
`segment` renvoie et trace le segment défini par les deux arguments.
On tape :

```
segment(point(0,0,0),point(1,1,1))
```

On obtient :

```
Le segment reliant O au point(1,1,1)
```

10.5.5 Le vecteur en 3-d : `vecteur`

Voir aussi : 9.10.5 pour la géométrie plane.
`vecteur`, en géométrie 3-d, a comme arguments soit :
- une liste représentant les coordonnées d'un point A.
  `vecteur` définit et dessine le vecteur $\overrightarrow{OA}$ où O est l'origine du repère.
- deux points A, B ou deux listes représentant les coordonnées de ces points.
  `vecteur` définit et dessine le vecteur $\overrightarrow{AB}$
- un point A (ou une liste représentant les coordonnées de ce point) et un vecteur V (définition récursive).
  `vecteur` définit et dessine le vecteur $\overrightarrow{AB}$ tel que $\overrightarrow{AB} = \overrightarrow{V}$.

On tape :

```
vecteur([1,2,3])
```

On obtient :

```
Le tracé du vecteur d'origine [0,0,0] et d'extrémité
[1,2,3]
```

On tape :

```
vecteur(point([-1,0,0]),point([0,1,2]))
```

Ou on tape :

```
vecteur([-1,0,0],[0,1,2])
```

On obtient :

```
Le tracé du vecteur d'origine [-1,0,0] et d'extrémité
[0,1,2]
```

On tape :

```
V:=vecteur([-1,0,0],[0,1,2])
```

On tape :

```
vecteur(point([-1,2,0]),V)
```

Ou on tape :

```
vecteur([-1,2,0],V)
```

On obtient :

```
Le tracé du vecteur d'origine [-1,2,0] et d'extrémité
[0,3,2]
```

Remarque
En calcul formel, on travaille sur la liste des coordonnées des vecteurs que l'on obtient avec la commande `coordonnees` (cf 10.12.4).

10.5.6 Plan et droites parallèles : `parallel parallele`

Voir aussi : 9.10.6 pour la géométrie plane.
`parallele`, en géométrie 3-d, renvoie et dessine une droite ou un plan selon ses arguments. Si ses arguments sont :

- un point A et 1 droites d, `parallele(A,d)` renvoie et dessine la droite parallèle à la droite d passant par le point A,
- une droite dd et une droite d non parallèles `parallele(dd,d)` renvoie et dessine le plan parallèle à la droite d passant par la droite dd,
- un point A et un plan P, `parallele(A,P)` renvoie et dessine le plan parallèle au plan P passant par le point A,
- un point A et 2 droites d et dd non parallèles, `parallele(A,d,dd)` renvoie et dessine le plan parallèle aux droites d et dd passant par le point A.

On tape :

```
parallele(point(1,1,1),droite(point(0,0,0),point(0,0,1)))
```

On obtient :

```
La droite d'équation x=1,y=1 est tracée
```

On tape :

```
parallele(droite(point(1,0,0),point(0,1,0)),droite(point(0,0,0),point(0,0,1))
```

On obtient :

```
Le plan d'équation x+y-1=0 est tracé
```

On tape :

```
parallele(point(0,0,0),plan(point(1,0,0),point(0,1,0),point(0,0,1)))
```

On obtient :

```
Le plan d'équation x+y+z=0 est tracé
```

On tape :

```
parallele(point(1,1,1),droite(point(0,0,0),point(0,0,1)),droite(point(1,0,0),
```

On obtient :

```
Le plan d'équation x+y=2 est tracée
```

10.5.7 Plans (ou droites) perpendiculaires : `perpendicular perpendiculaire`

Voir aussi : 10.5.8 et, pour la géométrie plane 9.10.7.
`perpendiculaire`, en géométrie 3-d, renvoie et dessine une droite (resp un plan) si le 2-ième argument est une droite (resp un plan).
Plus précisemment, si ses arguments sont :

- un point A et une droite d, `perpendiculaire(A,d)` renvoie et dessine la droite perpendiculaire à la droite d et passant par A,
- une droite dd et un plan P, `perpendiculaire(dd,P)` renvoie et dessine le plan perpendiculaire au plan P et passant par dd.

On tape :

```
perpendiculaire(point(0,0,0),droite(point(1,0,0),point(0,1,0)))
```

On obtient :

```
La droite d'équation y=x, z=0
```

On tape :

```
perpendiculaire(droite([0,0,0],[1,1,0]),plan(point(1,0,0),point(0,1,0
```

On obtient :

```
Le plan d'équation x=y
```

10.5.8 Droite orthogonale à un plan et plan orthogonal à une droite : `orthogonal`

Voir aussi : 10.5.7 et, pour la géométrie plane 9.10.7.
`orthogonal`, en géométrie 3-d, renvoie et dessine une droite (resp un plan) si le 2-ième argument est un plan (resp une droite). Plus précisemment, si ses arguments sont :

- une droite dd et une droite d, `orthogonal(dd,d)` renvoie et dessine un plan orthogonal à la droite d et passant par dd,
- un point A et un plan P, `orthogonal(A,P)` renvoie et dessine une droite orthogonale au plan P et passant par A.

On tape :

```
orthogonal(point(0,0,0),droite(point(1,0,0),point(0,1,0)))
```

On obtient :

```
Le plan d'équation y=x
```

On tape :

```
orthogonal(point(0,0,0),plan(point(1,0,0),point(0,1,0),point(0,0,1)))
```

On obtient :

```
La droite d'équation x=y=z
```

10.5.9 La perpendiculaire commune à deux droites 3-d : common_perpendicular perpendicu- laire_commune

perpendiculaire_commune a comme argument deux droites D1 et D2.
perpendiculaire_commune(D1,D2) dessine la perpendiculaire commune des droites D1 et D2.
On tape :

```
D1:=droite([1,1,0],[0,1,1]);
D2:=droite([0,-1,0],[1,-1,1]);
```

Puis on tape :

```
perpendiculaire_commune(D1,D2)
```

On obtient :

```
La perpendiculaire commune aux deux droites D1 et D2
```

Si on tape :

```
d:=perpendiculaire_commune(D1,D2);0
```

On obtient :

```
expr("pnt(pnt[line[point[1,0,-1],point[-1/3,-2/3,-1/3]],56,"d"])",0),0
```

Ce qui veut dire que la perpendiculaire commune à D1 et D2 passe par les points [1,0,-1] et [-1/3,-2/3,-1/3].
Puis on tape :

```
equation(d)
```

On obtient :

```
2/3-2*x/3+4*y/3=0,-4/3-8*x/9-4*y/9-20*z/9=0
```

10.6 Les plans

10.6.1 Le plan : plane plan

plan a comme argument soit trois points, soit un point et une droite, soit son équation cartésienne.
plan(A,B,C) ou plan(A,droite(B,C)) (resp plan(a*x+b*y+c*z+d=0)) trace le plan ABC (resp le plan d'équation a*x+b*y+c*z+d=0) dans l'espace 3-d.
On tape :

```
plan([0,0,5],[0,5,0],[5,0,0])
```

Ou on tape :

```
plan(x+y+z=5)
```

Ou on tape :

```
plan([0,0,5],droite([0,5,0],[5,0,0]))
```

On obtient :

```
Le plan d'équation x+y+z=5
```

10.6.2 Le plan médiateur : `perpen_bisector mediatrice`

Voir aussi : 9.10.11 pour la géométrie plane.
`mediatrice`, en géométrie 3-d, a comme argument un segment ou deux points `A` et `B`.
`mediatrice(A,B)` trace le plan médiateur du segment `(A,B)`.
On tape :

```
mediatrice(point([0,0,0]),point([4,4,4]))
```

Ou on tape :

```
mediatrice(segment([0,0,0],[4,4,4]))
```

On obtient :

```
Le plan médiateur du segment [0,0,0],[4,4,4]
```

10.6.3 Le plan tangent : `tangent`

Voir aussi : 9.10.8 pour la géométrie plane et 3.10.5 pour les tangentes à un graphe.
`tangent` a comme argument un objet géométrique `G` et un point `A` de `G`.
`tangent` dessine le plan tangent à `G` passant par `A`.
Lorsque `G` est un graphe, `A` peut être soit un point de `G`, soit la liste des coordonnées du projeté sur x0y du point de contact.
On tape :

```
S:=sphere([0,0,0],3)
```

Puis on tape :

```
tangent(S,[2,2,1])
```

On obtient :

```
Le plan tangent à la surface sphère S au point [2,2,1]
```

On tape :

```
G:=plotfunc(x^2+y^2,[x,y])
```

Puis on tape :

```
tangent(G,[2,2])
```

Ou on tape car [2,2] est le projeté du point [2,2,8] qui se trouve sur le graphe $z = x^2 + y^2$ ($8 = 2^2 + 2^2$) :

```
tangent(G,point([2,2,8]))
```

On obtient :

Le plan tangent à la surface $z = x^2 + y^2$ au point [2,2,8]

10.6.4 Plan orthogonal à une droite : `orthogonal`

On utilise la commande `ortogonal` pour avoir un plan orthgonal à une droite ou pour avoir une droite orthgonale à un plan (cf 10.5.8)

10.6.5 Plan perpendiculaire à un plan : `perpendicular perpendiculaire`

On utilise la commande `perpendiculaire` pour avoir un plan perpendiculaire à un plan ou pour avoir une droite perpendiculaire à une droite (cf 10.5.7)

10.7 Les triangles dans l'espace

Le principe est de rajouter, si necessaire, un paramètre pour définir le plan du triangle et définir aussi l'orientation de ce plan. Pour la géométrie plane voir 9.11.

10.7.1 Le triangle quelconque dans l'espace : `triangle`

Voir aussi : 9.11.2 pour la géométrie plane.
`triangle`, en géométrie 3-d, a comme arguments : 3 points `triangle` renvoie et trace le triangle ayant pour sommets ces 3 points.
On tape :

```
A:=point(0,0,0)
```

```
B:=point(3,3,3)
```

```
C:=point(0,3,0)
```

Puis on tape :

```
triangle(A,B,C)
```

On obtient :

```
Le triangle ABC
```

10.7.2 Le triangle isocèle dans l'espace : `isosceles_triangle triangle_isocele`

Voir aussi : 9.11.3 pour la géométrie plane.
`triangle_isocele`, en géométrie 3-d, a trois ou quatre arguments.
Description des arguments :
- si il a trois arguments, ce sont : 3 points (les 2 premiers sommets A et B du triangle) et le troisième argument est soit un point P, soit une liste formée par un point P et un réel c qui désigne la mesure en radians (ou en degrés) de l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$, l'angle $(\overrightarrow{AB}, \overrightarrow{AP})$ étant positif.
Le point P définit le plan du triangle ainsi que l'orientation de ce plan pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AP})$ soit positif.
`triangle_isocele(A,B,P)` renvoie et trace dans le plan ABP orienté par P (l'angle $(\overrightarrow{AB}, \overrightarrow{AP})$ est positif) le triangle ABC isocèle de sommet A ($AB = AC$) et tel que l'angle $(\overrightarrow{AB}, \overrightarrow{AC}) = (\overrightarrow{AB}, \overrightarrow{AP})$, sans définir le point C).
On tape :

```
A:=point(0,0,0)
```

```
B:=point(3,3,3)
```

```
P:=point(0,0,3)
```

Puis on tape :

```
triangle_isocele(A,B,P)
```

Dans le plan ABP, le triangle isocéle de sommets AB,
d'angle $(\overrightarrow{AB},\overrightarrow{AC})=(\overrightarrow{AB},\overrightarrow{AP})$

`triangle_isocele(A,B,[P,c])` renvoie et trace dans le plan ABP orienté par P (l'angle $(\overrightarrow{AB},\overrightarrow{AP})$ est positif) le triangle ABC isocèle de sommet A ($AB=AC$) et tel que l'angle $(\overrightarrow{AB},\overrightarrow{AC})=c$ radians (ou degrés), sans définir le point C).
On tape :

```
A:=point(0,0,0)
```

```
B:=point(3,3,3)
```

```
P:=point(0,0,3)
```

Puis on tape :

```
triangle_isocele(A,B,[P,3*pi/4])
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

Dans le plan ABP, le triangle isocéle de sommets AB,
d'angle $(\overrightarrow{AB},\overrightarrow{AC})=3*pi/4$

- si il a quatre arguments, le dernier argument est le nom d'une variable qui servira à définir le troisième sommet.
On tape :

```
triangle_isocele(A,B,[P,3*pi/4],C)
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

Dans le plan ABP, le triangle isocéle de sommets AB,
d'angle $(\overrightarrow{AB},\overrightarrow{AC})=3*pi/4$

On tape :

```
simplify(coordonnees(C))
```

On obtient :

```
[(-3*sqrt(2)-3)/2,(-3*sqrt(2)-3)/2,(-3*sqrt(2)+6)/2]
```

10.7.3 Le triangle rectangle dans l'espace : `triangle_rectangle`

Voir aussi : 9.11.4 pour la géométrie plane.
`triangle_rectangle`, en géométrie 3-d, peut avoir trois ou quatre arguments.
Description des arguments :
- si il a trois arguments, ce sont : 3 points (les 2 premiers sommets A et B du triangle) et le troisième argument est un point P ou une liste formée par un point P et un réel k non nul.
Le point P définit le plan du triangle ainsi que l'orientation de ce plan pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AP})$ soit positif,

– `triangle_rectangle(A,B,P)` renvoie et trace, dans le plan ABP, le triangle ABC rectangle en A, tel que $AC = AP$.
  On tape :
```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
Q:=point(0,0,-3)
```
  Puis on tape :
```
triangle_rectangle(A,B,P)
```
  On obtient :
  `Dans le plan` ABP`, le triangle` ABC `rectangle en` A
  `tel que` $AC = AP$
– `triangle_rectangle(A,B,[P,k])` renvoie et trace dans le plan ABP, le triangle ABC rectangle en A : ce triangle est direct si $k > 0$, indirect si $k < 0$ et est tel que $AC = |k| * AB$.
  Ainsi si l'angle $(\overrightarrow{BC}, \overrightarrow{BA}) = \beta$ radians (ou degrés), on a $\tan(\beta) = k$.
  On remarquera que si C est le transformé de B dans la similitude de centre A de rapport $|k|$ et d'angle $(k/|k|) * \pi/2$.
  On tape :
```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
Q:=point(0,0,-3)
```
  Puis on tape :
```
triangle_rectangle(A,B,[P,2])
```
  On obtient :
```
Dans le plan ABP, le triangle ABC direct,
rectangle en A tel que AC=2*AB
```
  On tape :
```
triangle_rectangle(A,B,[P,-2])
```
  On obtient :
```
Dans le plan ABP, le triangle ABC indirect,
rectangle en A tel que AC=2*AB
```
  - si il a quatre arguments, le dernier argument est le nom d'une variable qui servira à définir le troisième sommet.
  On tape :
```
triangle_rectangle(A,B,[P,2],C)
```

On obtient :
`Dans le plan ABP, le triangle rectangle de sommets ABC`

On tape :

`simplify(coordonnees(C))`

On obtient :

`[-(3*sqrt(2)),-(3*sqrt(2)),6*sqrt(2)]`

10.7.4 Le triangle équilatèral dans l'espace : `equilateral_triangle` `triangle_equilateral`

Voir aussi : 9.11.5 pour la géométrie plane.
`triangle_equilateral`, en géométrie 3-d, a trois ou quatre arguments.
Description des arguments :
- si il a trois arguments, ce sont 3 points : les 2 premiers sommets A et B du triangle et le troisième point définit le plan du triangle.
`triangle_equilateral(A,B,P)` renvoie et trace dans le demi-plan ABP le triangle équilatéral ABC mais sans définir le point C.
On tape :

```
A:=point(0,0,0)

B:=point(3,3,3)

P:=point(0,0,3)

Q:=point(0,0,-3)
```

Puis on tape :

`triangle_equilateral(A,B,P)`

On obtient :

`Dans le demi-plan ABP,le triangle équilatèral de sommets A et B`

- si il a quatre arguments, le dernier argument est le nom d'une variable qui servira à définir le troisième sommet On tape :

`triangle_equilateral(A,B,P,C)`

On obtient :

`Dans le demi-plan ABP, le triangle équilatèral de sommets A et B`

On tape :

`simplify(coordonnees(C))`

On obtient :

`[(-3*sqrt(6)+6)/4,(-3*sqrt(6)+6)/4,(3*sqrt(6)+3)/2]`

On tape :

```
triangle_equilateral(A,B,Q,D)
```

On obtient :

```
Dans le demi-plan ABQ, le triangle équilatèral de
sommets A et B
```

On tape :

```
simplify(coordonnees(D))
```

On obtient :

```
[(3*sqrt(6)+6)/4,(3*sqrt(6)+6)/4,(-3*sqrt(6)+3)/2]
```

10.8 Les quadrilatères dans l'espace

Voir aussi : 9.12 pour la géométrie plane.
Le principe est de rajouter si necessaire un paramètre pour définir le plan du quadrilatère et définir aussi l'orientation de ce plan.

10.8.1 Le carré dans l'espace : `square carre`

Voir aussi : 9.12.2 pour la géométrie plane.
`carre`, en géométrie 3-d, peut avoir de trois à cinq arguments.
Les arguments sont :
- si il a trois arguments ce sont 3 points de l'espace : les 2 sommets du carré et le troisième point définit le demi-plan du carré.
`carre(A,B,P)` renvoie et trace le carrè $ABCD$ dans le demi-plan ABP contenant P, mais sans définir les points D et C.
On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
Q:=point(0,0,-3)
```

Puis on tape :

```
carre(A,B,P)
```

On obtient :

```
Le carré de sommets A,B dans le demi-plan ABP
```

Puis on tape :

```
carre(A,B,Q)
```

On obtient :

`Le carré de sommets A,B dans le demi-plan ABQ`

- si il a cinq arguments, les 2 derniers paramètres sont le nom de deux variables qui serviront à définir les deux autres sommets. On tape :

```
carre(A,B,P,C,D)
```

On obtient :

`Le carré de sommets A,B,C,D dans le demi-plan ABQ`

On tape :

```
simplify(coordonnees(C))
```

On obtient :

```
[3-3*sqrt(1/2),3-3*sqrt(1/2),3*sqrt(2)+3]
```

On tape :

```
simplify(coordonnees(D))
```

On obtient :

```
[-3*sqrt(2),-3*sqrt(2),3*sqrt(2)]
```

10.8.2 Le losange dans l'espace : `rhombus losange`

Voir aussi : 9.12.3 pour la géométrie plane.
`losange`, en géométrie 3-d, peut avoir de trois à cinq arguments.
Les arguments sont :

– si il a trois arguments ce sont : 2 points (les 2 premiers sommets du losange) et le troisième argument est soit un point P, soit une liste formée par un point P et un nombre réel a. Le point P définit le plan du losange et l'orientation de ce plan.
On a :
- `losange(A,B,P)` renvoie et trace dans le plan ABP, le losange $ABCD$ tel que :
$\overrightarrow{AB}, \overrightarrow{AD}) = \overrightarrow{AB}, \overrightarrow{AP})$, mais sans définir les points C et D.
- `losange(A,B,[P,a])` renvoie et trace dans le plan ABP, le losange $ABCD$ tel que :
$\overrightarrow{AB}, \overrightarrow{AD}) = a$ radians (ou degrés), mais sans définir les points C et D.
On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
```

Puis on tape :

```
losange(A,B,P)
```

On obtient :

`Le losange de sommets` A, B `et d'angle` A `égal à` BAP`, dans le plan` ABP

On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
```

Puis on tape :

```
losange(A,B,[P,pi/3])
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le losange de sommets` A, B `et d'angle` A `égal à pi/3, dans le plan` ABP

– si il a quatre ou cinq arguments, les derniers paramètres sont les noms des variables qui serviront à définir les derniers sommets.

On tape :

```
losange(A,B,[P,pi/3],C,D)
```

On obtient :

`Le losange de sommets` A, B, C, D `et d'angle` A `égal à pi/3, dans le plan` ABP

On tape :

```
simplify(coordonnees(C))
```

On obtient :

```
[(-3*sqrt(6)+18)/4,(-3*sqrt(6)+18)/4,(3*sqrt(6)+9)/2]
```

On tape :

```
simplify(coordonnees(D))
```

On obtient :

```
[(-3*sqrt(6)+6)/4,(-3*sqrt(6)+6)/4,(3*sqrt(6)+3)/2]
```

10.8.3 Le rectangle dans l'espace : `rectangle`

Voir aussi : 9.12.4 pour la géométrie plane.

`rectangle`, en géométrie 3-d, peut avoir de trois à cinq arguments.

Les arguments sont :

- si il a trois arguments ce sont : 2 points (les 2 sommets du rectangle) et le troisième argument est soit un point P soit la liste formée par un point P et un nombre réel k non nul.

Le point P définit le plan du rectangle et l'orientation de ce plan.

`rectangle(A,B,P)` renvoie et trace dans le plan ABP, le rectangle $ABCD$ tel que :

$AD = AP$ et $(\overrightarrow{AB}, \overrightarrow{AD}) = \pi/2$, mais sans définir les points C et D.

On tape :

```
A:=point(0,0,0)

B:=point(3,3,3)

P:=point(0,0,3)
```

Puis on tape :

`rectangle(A,B,P)`

On obtient :

`Le rectangle de sommets`

`rectangle(A,B,[P,k])` renvoie et trace dans le plan ABP, le rectangle $ABCD$ tel que :
$AD = |k| * AB$ et $(\overrightarrow{AB}, \overrightarrow{AD}) = (k/|k|) * \pi/2$, mais sans définir les points C et D.
On tape :

`A:=point(0,0,0)`

`B:=point(3,3,3)`

`P:=point(0,0,3)`

Puis on tape :

`rectangle(A,B,[P,1/2])`

On obtient :

`Le rectangle de sommets`

- si il a cinq arguments, les 2 derniers paramètres sont les noms de deux variables qui serviront à définir les 2 derniers sommets.
On tape :

`rectangle(A,B,P,C,D)`

On obtient :

`Le rectangle de sommets`

On tape :

`simplify(coordonnees(C))`

On obtient :

`[(-sqrt(6)+6)/2,(-sqrt(6)+6)/2,sqrt(6)+3]`

On tape :

`simplify(coordonnees(D))`

On obtient :

`[(-(sqrt(6)))/2,(-(sqrt(6)))/2,sqrt(6)]`

10.8.4 Le parallélogramme dans l'espace : `parallelogram parallelogramme`

Voir aussi : 9.12.5 pour la géométrie plane.
`parallelogramme`, en géométrie 3-d, a trois arguments ou quatre arguments.
Les arguments sont :
- si il a trois arguments ce sont : trois points les sommets A, B, C du parallélogramme $ABCD$.
`parallelogramme(A,B,C)` renvoie et trace dans le plan ABC, le parallélogramme $ABCD$ tel que : $\overrightarrow{AD} = \overrightarrow{BC}$ mais sans définir le point D.
On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
C:=point(0,0,3)
```

On tape :

```
parallelogramme(A,B,C)
```

On obtient :

```
Le parallélogramme ABCD
```

- si il a quatre arguments le dernier paramètre est le nom d'une variable qui servira à définir le sommet manquant.
On tape :

```
parallelogramme(A,B,C,D)
```

On obtient :

```
Le parallélogramme ABCD et le point D
```

On tape :

```
coordonnees(D)
```

On obtient :

```
[-3,-3,0]
```

10.8.5 Les quadrilatères quelconques dans l'espace : `quadrilateral quadrilatere`

Voir aussi : 9.12.6 pour la géométrie plane.
`quadrilatere(A,B,C,D)`, en géométrie 3-d, renvoie et trace la ligne polygonale fermée (ie un quadrilatère non plan si les points ne sont pas coplanaires) $ABCD$.
On tape :

```
quadrilatere(point(0,0,0),point(0,1,0),point(0,2,2)point(1,0,2))
```

On obtient :

```
le quadilatère (de l'espace) de sommets les points
donnés
```

10.9 Les polygones dans l'espace

Voir aussi : 9.13 pour la géométrie plane.
Le principe est de rajouter si necessaire un paramètre pour définir le plan du polygone et définir aussi l'orientation de ce plan.

10.9.1 L'hexagone : `hexagon hexagone`

Voir aussi : 9.13.1 pour la géométrie plane.
Voir aussi : 9.13.2 pour la géométrie plane et 10.9.2 pour la géométrie 3-d.
`hexagone`, en géométrie 3-d, a comme 3 ou 7 arguments qui sont 3 points de l'espace suivi éventuellement de 4 noms de variables. Les trois points sont les 2 sommets de l'hexagone et le troisième point qui définit le plan de l'hexagone et l'orientation du plan et les noms de variables désignent les 4 autres sommets de l'hexagone et servent à définir les sommets manquants.
`hexagone(A,B,P)` renvoie et trace l'hexagone de sommets A, B dans le demi-plan ABP.
On tape :

```
A:=point(0,0,0)
```

```
B:=point(3,3,3)
```

```
P:=point(0,0,3)
```

Puis on tape :

```
hexagone(A,B,P)
```

On obtient :

```
Dans le demi-plan ABP, un hexagone de sommets A et B
```

On aurait pu taper :

```
hexagone(A,B,P,C,D,E,F)
```

pour définir les quatre autres sommets manquants.

10.9.2 Les polygones réguliers dans l'espace : `isopolygon isopolygone`

Voir aussi : 9.13.2 pour la géométrie plane.
`isopolygone`, en géométrie 3-d, a quatre arguments.
Déscription des arguments : Le quatrième argument est un entier n et $abs(n)$ représente le nombre de côtés de l'isopolygone. Selon le signe de n on aura :

- $n > 0$
  les 3 premiers arguments sont 3 points de l'espace : les 2 sommets de l'isopolygone et le troisième point qui définit le plan de l'isopolygone et l'orientation du plan.
  `isopolygone(A,B,P,n)` renvoie et trace l'isopolygone direct de sommets A, B et ayant n côtés dans le plan ABP.
  On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
```

Puis on tape :

```
isopolygone(A,B,P,5)
```

On obtient :

```
Dans le demi-plan ABP, un pentagone de sommets A
et B
```

- $n < 0$
  les 3 premiers arguments sont 3 points de l'espace : le centre et un sommet de l'isopolygone et le troisième point qui définit le plan de l'isopolygone et l'orientation du plan.
  `isopolygone(A,B,P,n)` renvoie et trace l'isopolygone indirect de centre A et de sommet B et ayant $-n$ côtés dans le plan ABP.
  On tape :

```
A:=point(0,0,0)
B:=point(3,3,3)
P:=point(0,0,3)
```

  Puis on tape :

```
isopolygone(A,B,P,-5)
```

  On obtient :

```
Un pentagone de centre A et de sommet B non situé
dans le demi-plan ABP
```

10.9.3 Les polygones quelconques dans l'espace : `polygon polygone`

Voir aussi : 9.13.3 pour la géométrie plane.
`polygone`, en géométrie 3-d, a comme argument un séquence de points de l'espace qui sont les sommets du polygone.
`polygone(A,B,C,D,E,F)` renvoie et trace le polygone $ABCDEF$ si les points A, B, C, D, E, F sont dans un même plan ou sinon trace la ligne polygonale fermée A, B, C, D, E, F, A.
On tape :

```
A:=point(0,0,0)

B:=point(3,3,3)

C:=point(0,0,3)

D:=point(-3,-3,0)

E:=point(-3,-3,-3)
```

Puis on tape :

```
polygone(A,B,C,D,E)
```

On obtient :

```
Le polygone ABCDE dans le plan x=y
```

10.9.4 Ligne polygonale dans l'espace : `open_polygon polygone_ouvert`

Voir aussi : 9.13.4 pour la géométrie plane.
`polygone_ouvert`, en géométrie 3-d, a comme argument une séquence) de n points 3-d.
`polygone_ouvert` renvoie et trace la ligne polygonale ayant pour sommets ces n points.
On tape :

```
polygone_ouvert(point(0,0,0),point(0,1,0),point(0,2,2),point(1,0,2))
```

On obtient :

```
Une ligne polygonale de sommets les points donnés
```

10.10 Les cercles dans l'espace : `circle cercle`

Voir aussi : 9.14.1 pour la géométrie plane.
`cercle`, en géométrie 3-d, a comme argument :

- soit 3 points `A,B,C` non alignés : les deux premiers points définissent un diamètre du cercle et les trois points définissent le plan du cercle.
- soit un point `A`, un vecteur `v` et un point `C` non situé sur la droite définie par `A` et `v` : le point `A` est le centre du cercle, le point `B:=A+v` est un point du cercle et le plan `ABC` est le plan du cercle.

À noter que dans les 2 cas, le premier et le troisième argument peuvent être les coordonnées du point.
On tape :

```
cercle(point(0,0,1),point(0,1,0),point(0,2,2))
```

Ou on tape :

```
cercle([0,0,1],point(0,1,0),point(0,2,2))
```

Ou on tape :

```
cercle([0,0,1],point(0,1,0),[0,2,2])
```

On obtient :

```
Un cercle de diamètre les points [0,0,1] et [0,1,0]
situés dans le plan x=0
```

On tape :

```
cercle(point(0,0,1),[0,1,0],point(0,2,2))
```

Ou on tape :

```
cercle([0,0,1],[0,1,0],point(0,2,2))
```

Ou on tape :

```
cercle([0,0,1],[0,1,0],[0,2,2])
```

On obtient :

```
Un cercle de centre [0,0,1] et passant par le point
[0,1,1] (donc de rayon 1) situés dans le plan x=0
```

10.11 Les coniques dans l'espace

10.11.1 L'ellipse dans l'espace : `ellipse`

Voir aussi : 9.15.1 pour la géométrie plane.
`ellipse`, en géométrie 3-d, a trois paramètres : ses deux foyers et un de ces points non aligné avec les foyers.
`ellipse(F1,F2,A)` trace l'ellipse passant par `A` et de foyers `F1` et `F2`.
On tape :

```
ellipse(point(-1,0,0),point(1,0,0),point(1,1,1))
```

On obtient :

```
L'ellipse passant par (1,1,1), de foyers (-1,0,0) et
(1,0,0)
```

10.11.2 L'hyperbole dans l'espace : `hyperbola hyperbole`

Voir aussi : 9.15.2 pour la géométrie plane.
`hyperbole`, en géométrie 3-d, a trois paramètres : ses deux foyers et un de ces points non aligné avec les foyers.
`hyperbole(F1,F2,A)` trace l'hyperbole passant par `A` et de foyers `F1` et `F2`.
On tape :

```
hyperbole(point(-1,0,0),point(1,0,0),point(1,1,1))
```

On obtient :

```
L'hyperbole passant par (1,1,1), de foyers (-1,0,0) et
(1,0,0)
```

10.11.3 La parabole dans l'espace : `parabola parabole`

Voir aussi : 9.15.3 pour la géométrie plane.
`parabole`, en géométrie 3-d, a comme paramètre trois points : son foyer F, son sommet S et un point P de son plan non aligné avec F et S.
`parabole(F,S,P)` renvoie et dessine dans le plan FSP, la parabole de foyer `F` et de sommet `S`.
On tape :

```
parabole(point(0,0,0),point(-1,0,0),point(1,1,1))
```

On obtient :

```
La parabole de foyer (0,0,0) et de sommet (-1,0,0)
dans le plan passant par (1,1,1) qui est le plan y=z
```

10.12 Les mesures

10.12.1 L'abscisse d'un point 3-d : `abscissa abscisse`

Voir aussi : 9.16.2 pour la géométrie plane.
`abscisse`, en géométrie 3-d, est une fonction ayant comme argument un point 3-d.
`abscisse` renvoie l'abscisse dd ce point.
On tape :

```
abscisse(point([1,2,3]))
```

On obtient :

```
1
```

10.12.2 L'ordonnée d'un point 3-d : `ordinate ordonnee`

Voir aussi : 9.16.3 pour la géométrie plane.
`ordonnee`, en géométrie 3-d, est une fonction ayant comme argument un point 3-d.
`ordonnee` renvoie l'ordonnée de ce point.
On tape :

```
ordonnee(point([1,2,3]))
```

On obtient :

```
2
```

10.12.3 La cote d'un point 3-d : `cote`

`cote` est une fonction ayant comme argument un point 3-d.
`cote` renvoie la cote de ce point.
On tape :

```
cote(point([1,2,3]))
```

On obtient :

```
3
```

10.12.4 Les coordonnées d'un point, d'un vecteur ou d'une droite 3-d : `coordinates coordonnees`

Voir aussi : 9.16.4 pour la géométrie plane.
`coordonnees`, en géométrie 3-d, est une fonction ayant comme argument un point ou un vecteur ou une droite 3-d.
`coordonnees` renvoie la liste de l'abscisse, de l'ordonnée et de la cote du point ou du vecteur ou renvoie une matrice de 2 lignes qui donnent l'abscisse, l'ordonnée et la cote de 2 points de la droite orientée.
On a :

- si le point `A` a pour coordonnées cartésiennes (x_A, y_A, z_A),
  `coordonnees(A)` renvoie $[x_A, y_A, z_A]$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B, z_B),
  `coordonnees(vecteur(A,B))` ou `B-A` renvoie
  $[x_B - x_A, y_B - y_A, z_B - z_A]$
  car `B-A` désigne les coordonnées du vecteur $\overrightarrow{AB}$,
- si le vecteur `V` a pour coordonnées cartésiennes (x_V, y_V, z_V),
  `coordonnees(V)` ou `coordonnees(vecteur(A,V))` renvoie
  $[x_V, y_V, z_V]$,
- si une droite `d` est définie par deux points `A` et `B`,
  `coordonnees(d)` renvoie `[coordonnees(A),coordonnees(B)]`,
- si une droite `d` est définie par son équation,
  `coordonnees(d)` renvoie `[coordonnees(A),coordonnees(B)]`
  où `A` et `B` sont deux points de la droite `d`, le vecteur `AB` ayant même orientation que `d`.

On tape :

```
coordonnees(point([1,2,3]))
```

Ou on tape :

```
coordonnees(point(1,2,3))
```

On obtient :

```
[1,2,3]
```

On tape :

```
coordonnees(vecteur(point([1,2,3]),point([2,4,7])))
```

Ou on tape :

```
coordonnees(vecteur(point(1,2,3),point(2,4,7)))
```

Ou on tape :

```
coordonnees(vecteur([1,2,3],[2,4,7])))
```

Ou on tape :

```
coordonnees(vecteur([1,2,4])))
```

Ou on tape :

```
coordonnees(vecteur([1,2,3],vecteur([1,2,4])))
```

Ou on tape :

```
point([2,4,7])-point([1,2,3])
```

On obtient :

```
[1,2,4]
```

On tape :

```
d:=droite(point(-1,1,0),point(1,2,3))
```

```
coordonnees(d)
```

On obtient :

```
[[-1,1,0],[1,2,3]]
```

On tape :

```
d:=droite(x-2*y+3=0,6*x+3*y-5*z+3=0)
```

```
coordonnees(d)
```

On obtient :

```
[[-1,1,0],[9,6,15]]
```

Attention
`coordonnees` peut aussi avoir comme argument une séquence ou une liste de points. `coordonnees` renvoie alors la séquence ou la liste des listes des coordonnées de ces points, par exemple :
`coordonnees(point([0,1,2]),point([1,2,4]))`
renvoie la séquence :
`[0,1,2],[1,2,4]`
et
`coordonnees([point([0,1,2]),point([1,2,4])])`
renvoie la matrice :
`[[0,1,2],[1,2,4]]` mais
`coordonnees([1,2,4])`
renvoie la matrice :
`[[1,0],[2,0],[4,0]]`
car `[1,2,4]` est considéré comme la liste de 3 points d'affixe `1`, `2` et `4`.

10.12.5 L'équation cartésienne d'un objet géométrique : `equation`

Voir aussi : 9.16.7 pour la géométrie plane.
`equation` permet d'avoir l'équation cartésienne d'un objet géométrique.
Attention ! ! ! il faut auparavant purger les variables `x` et `y` en tapant `purge(x)` et `purge(y)`.
On tape :

```
equation(droite(point(0,1,0),point(1,2,3)))
```

On obtient :

```
(x-y+1=0,3*x+3*y-2*z=0)
```

On tape :

```
equation(sphere(point(0,1,0),2))
```

On obtient :

```
x^2+y^2+-2*y+z^2-3=0
```

qui est l'équation de la sphère de centre (0,1,0) et de rayon `2`.

10.12.6 L'équation paramétrique d'un objet géométrique : parameq

Voir aussi : 9.16.8 pour la géométrie plane.
parameq, en géométrie 3-d, permet d'avoir l'équation paramétrique fonction du paramètre t ou des paramètres u et v, d'un objet géométrique.
Attention ! ! ! il faut auparavant purger ces variables en tapant par exemple : purge(t,u,v).
On tape :

```
parameq(droite(point(0,1,0),point(1,2,3)))
```

On obtient :

```
[-t+1,-t+2,-3*t+3]
```

On tape :

```
parameq(sphere(point(0,1,0),2))
```

On obtient :

```
point[2*cos(u)*cos(v),1+2*cos(u)*sin(v),2*sin(u)]
```

On tape :

```
normal(parameq(ellipse(point(-1,1,1),point(1,1,1),
                point(0,1,2))))
```

On obtient :

```
[sqrt(2)*cos(t),1,sin(t)+1]
```

On peut vérifier, on tape :

```
plotparam([sqrt(2)*cos(t),1,sin(t)+1],t=0..2*pi);
F1:=point(-1,1,1);F2:=point(1,1,1);M:=point(0,1,2)
```

On obtient l'ellipse, ces 2 foyers F_1 et $f-2$ et le point M sur l'ellipse.

10.12.7 La longueur d'un segment : distance longueur

Voir aussi : 9.17.2 pour la géométrie plane.
longueur a comme argument deux points (ou deux listes qui sont les coordonnées de ces points).
longueur renvoie la longueur du segment défini par ces deux points.
On tape :

```
longueur(point(-1,1,1),point(1,1,1))
```

Ou on tape :

```
longueur([-1,1,1],[1,1,1])
```

On obtient :

```
2
```

10.12.8 Le carré de la longueur d'un segment : `distance2` `longueur2`

Voir aussi : 9.17.4 pour la géométrie plane.
`longueur2` a comme argument deux points (ou deux listes qui sont les coordonnées de ces points).
`longueur2` renvoie le carré de la longueur du segment défini par ces deux points.
On tape :

```
longueur2(point(-1,1,1),point(1,1,1))
```

Ou on tape :

```
longueur2([-1,1,1],[1,1,1])
```

On obtient :

```
4
```

10.12.9 La mesure d'un angle : `angle`

Voir aussi : 9.17.5 pour la géométrie plane.
`angle` a comme argument trois points ou deux droites coplanaires ou une droite et un plan.
`angle` renvoie la mesure en radians (ou en degrés) de :
soit de l'angle non orienté de sommet le premier argument, le deuxième argument se trouve sur le premier coté de l'angle et le troisième argument se trouve sur le deuxiéme coté,
soit de l'angle des deux droites coplanaires
soit de l'angle de la droite et du plan.
Ainsi `angle(A,B,C)` désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
On tape :

```
angle((point(0,0,0),point(1,0,0),point(0,0,1)))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
pi/4
```

On tape :

```
angle(droite([0,0,0],[1,1,0]),droite([0,0,0],[1,1,1]))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
acos(2/(sqrt(6)))
```

On tape :

```
angle(droite([0,0,0],[1,1,0]),plan(x+y+z=0))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
acos(2/(sqrt(6)))
```

10.13 Les propriétés

10.13.1 Savoir si 1 objet géométrique est sur un objet graphique : `is_element` `est_element`

Voir aussi : 9.19.1 pour la géométrie plane.
`est_element(a,A)` teste si l'objet géométrique `a` est contenu dans l'objet géométrique `A`.
On tape :

```
P:=plan([0,0,0],[1,2,-3],[1,1,-2])
```

```
est_element(point(2,3,-5),P)
```

On obtient :

```
1
```

On tape :

```
dd:=droite([2,3,-2],[-1,-1,-1])
```

```
R:=plan([-1,-1,-1],[1,2,-3],[1,1,-2])
```

```
est_element(dd,R)
```

On obtient :

```
0
```

10.13.2 Savoir si des points ou /et des droites sont coplanaires : `is_coplanar` `est_coplanaire`

`est_coplanaire` teste si une liste ou une séquence de points ou de droites sont coplanaires.
On tape :

```
est_coplanaire([0,0,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
1
```

On tape :

```
est_coplanaire([-1,2,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
0
```

On tape :

```
est_coplanaire([0,0,0],[1,2,-3],droite([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
est_coplanaire(droite([0,0,0],[1,2,-3]),droite([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
est_coplanaire(droite([-1,2,0],[1,2,-3]),
        droite([1,1,-2],[2,1,-3]))
```

On obtient :

```
0
```

10.13.3 Savoir si droites ou /et plans sont parallèles `is_parallel` `est_parallele`

Voir aussi : 9.19.11 pour la géométrie plane.
`est_parallele`, en géométrie 3-d, teste si deux droites ou, si une droite et un plan ou, si deux plans sont parallèles.
On tape :

```
d:=droite([0,0,0],[-1,-1,-1])
dd:=droite([2,3,-2],[-1,-1,-1])
est_parallele(d,dd)
```

On obtient :

```
0
```

On tape :

```
S:=plan([-1,-1,-1],[1,2,-3],[0,0,0])
est_parallele(S,dd)
```

ou on tape :

```
est_parallele(dd,S)
```

On obtient :

```
1
```

On tape :

```
P:=plan([0,0,0],[1,2,-3],[1,1,-2])
Q:=plan([1,1,0],[2,3,-3],[2,2,-2])
est_parallele(P,Q)
```

On obtient :

```
1
```

10.13.4 Savoir si des droites ou/et plans sont perpendiculaires `is_perpendicular` `est_perpendiculaire`

Voir aussi : 9.19.12 pour la géométrie plane et 10.13.5 pour l'orthogonalité 3-d.
`est_perpendiculaire`, en géométrie 3-d, teste si deux droites ou, si une droite et un plan ou, si deux plans sont perpendiculaires.
En 3-d, deux droites perpendiculaires sont coplanaires et orthogonales.
On tape :

```
est_perpendiculaire(droite([2,3,-2],[-1,-1,-1]),
               droite([1,0,0],[1,2,8]))
```

On obtient :

```
0
```

On tape :

```
est_coplanaire(droite([2,3,-2],[-1,-1,-1]),
               droite([1,0,0],[1,2,8]))
```

On obtient :

```
0
```

On tape :

```
P:=plan([0,0,0],[1,2,-3],[1,1,-2])
```

```
S:=plan([-1,-1,-1],[1,2,-3],[0,0,0])
```

```
est_perpendiculaire(P,S)
```

On obtient :

```
1
```

On tape :

```
dd:=droite([2,3,-2],[-1,-1,-1])
```

```
est_perpendiculaire(dd,P)
```

On obtient :

```
0
```

10.13.5 Orthogonalité de 2 droites ou 2 sphères : `is_orthogonal` `est_orthogonal`

Voir aussi : 9.19.13 pour la géométrie plane et 10.13.4 pour la perpendicularité 3-d.
`est_orthogonal` est une fonction booléenne ayant comme argument deux droites ou deux sphères ou une droite et un plan ou d eux plans.
`est_orthogonal` vaut 1 si les deux droites ou les deux sphères (i.e si les plans tangents en leurs points d'intersection sont orthogonaux), ou la droite et le plan ou les deux plans sont orthogonaux et vaut 0 sinon.
On tape :

```
est_orthogonal(droite([2,3,-2],[-1,-1,-1]),
         droite([1,0,0],[1,2,8]))
```

On obtient :

```
1
```

On tape :

```
est_orthogonal(droite([2,3,-2],[-1,-1,-1]),
      plan([-1,-1,-1],[-1,0,3],[-2,0,0]))
```

On obtient :

```
1
```

On tape :

```
est_orthogonal(plan([0,0,0],[1,2,-3],[1,1,-2]),
      plan([-1,-1,-1],[1,2,-3],[0,0,0]))
```

On obtient :

```
1
```

On tape :

```
est_orthogonal(sphere([0,0,0],sqrt(2)),
         sphere([2,0,0],sqrt(2)))
```

On obtient :

```
1
```

10.13.6 Savoir si 3 points sont alignés : `is_collinear` `est_aligne`

Voir aussi : 9.19.2 pour la géométrie plane.
`est_aligne` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`est_aligne` vaut 1 si les points sont alignés, et vaut 0 sinon.
On tape :

```
est_aligne([2,0,0],[0,2,0],[1,1,0])
```

On obtient :

```
1
```

On tape :

```
est_aligne([2,0,0],[0,2,0],[0,1,1])
```

On obtient :

```
0
```

10.13.7 Savoir si 4 points sont cocycliques : `is_concyclic est_cocyclique`

Voir aussi : 9.19.3 pour la géométrie plane.
`est_cocyclique` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`est_cocyclique` vaut 1 si les points sont cocycliques, et vaut 0 sinon.
On tape :

```
est_cocyclique([2,0,0],[0,2,0],[sqrt(2),sqrt(2),0],
      [0,0,2],[2/sqrt(3),2/sqrt(3),2/sqrt(3)])
```

On obtient :

```
1
```

On tape :

```
est_cocyclique([2,0,0],[0,2,0],[1,1,0],[0,0,2],[1,1,1])
```

On obtient :

```
0
```

10.13.8 Savoir si 5 points sont cospheriques : `is_cospheric est_cospherique`

Voir aussi : 9.19.3 pour la géométrie plane.
`est_cospherique` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`est_cospherique` vaut 1 si les points sont cospheriques, et vaut 0 sinon.
On tape :

```
est_cospherique([2,0,0],[0,2,0],[sqrt(2),sqrt(2),0],
      [0,0,2],[2/sqrt(3),2/sqrt(3),2/sqrt(3)])
```

On obtient :

```
1
```

On tape :

```
est_cospherique([2,0,0],[0,2,0],[1,1,0],[0,0,2],[1,1,1])
```

On obtient :

```
0
```

10.13.9 Savoir si on a un triangle équilatéral : `is_equilateral` `est_equilateral`

Voir aussi : 9.19.5 pour la géométrie plane.
`est_equilateral` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`est_equilateral` vaut 1 si les trois points forment un triangle équilatéral ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
est_equilateral([2,0,0],[0,0,0],[1,sqrt(3),0])
```

Ou on tape :

```
T:=triangle_equilateral([2,0,0],[0,0,0],[1,sqrt(3),0]);
est_equilateral(T)
```

On obtient :

```
1
```

On tape :

```
est_equilateral([2,0,0],[0,2,0],[1,1,0])
```

On obtient :

```
0
```

10.13.10 Savoir si on a un triangle isocéle : `is_isosceles` `est_isocele`

Voir aussi : 9.19.6 pour la géométrie plane.
`est_isocele` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`est_isocele` vaut 1 (resp 2, 3) si les trois points forment un triangle isocéle ou si l'objet géométrique est un triangle isocéle dont le sommet de l'angle portant les deux cotés égaux est désigné par le premier (resp second, troisième) argument, ou vaut 4 si les trois points forment un triangle équilatéral, ou si l'objet géométrique est un triangle équilatéral, et vaut 0 sinon.
On tape :

```
est_isocele([2,0,0],[0,0,0],[0,2,0])
```

On obtient :

```
2
```

On tape :

```
T:=triangle_isocele([0,0,0],[2,2,0],[2,2,2]);
est_isocele(T)
```

On obtient :

```
1
```

On tape :

```
est_isocele([1,1,0],[-1,1,0],[-1,0,0])
```

On obtient :

```
0
```

10.13.11 Savoir si on a un triangle rectangle ou si on a un rectangle : `est_rectangle`

Voir aussi : 9.19.7 pour la géométrie plane.
`est_rectangle` est une fonction booléenne ayant comme argument trois ou quatre points ou un objet géométrique.
`est_rectangle(A,B,C)` vaut 1 (resp 2 ou 3) si les trois points `A,B,C` forment un triangle rectangle, l'angle droit étant désigné par le premier (resp second, troisième) argument ou si l'objet géométrique est un triangle rectangle,
`est_rectangle(A,B,C,D)` vaut 1 (resp 2) si les quatre points `A,B,C,D` sont coplanaires et forment un rectangle (resp un carré) ou si l'objet géométrique est un rectangle (resp un carré), et vaut 0 sinon.
On tape :

```
est_rectangle([2,0,0],[2,2,0],[0,2,0])
```

On obtient :

```
2
```

On tape :

```
est_rectangle([2,2,0],[-2,2,0],[-2,-1,0],[2,-1,0])
```

On obtient :

```
1
```

10.13.12 Savoir si on a un carré : `is_square` `est_carre`

Voir aussi : 9.19.8 pour la géométrie plane.
`est_carre` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_carre` vaut 1 si les quatre points sont coplanaires et forment un carré ou si l'objet géométrique est un carré, et vaut 0 sinon.
On tape :

```
est_carre([2,2,0],[-2,2,0],[-2,-2,0],[2,-2,0])
```

On obtient :

```
1
```

On tape :

```
K:=carre([0,0,0],[2,0,0],[0,0,1]);est_carre(K)
```

`K` est le carre de sommets [0,0,0],[2,0,0],[2,0,2],[0,0,2] (il est situé dans le plan [0,0,0],[2,0,0],[0,0,1]).
On obtient :

```
1
```

On tape :

```
est_carre([2,2,0],[-2,2,0],[-2,-1,0],[2,-1,0])
```

On obtient :

```
0
```

10.13.13 Savoir si on a un losange : `is_rhombus est_losange`

Voir aussi : 9.19.9 pour la géométrie plane.
`est_losange` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_losange` vaut 1 (rep 2) si les quatre points sont coplanaires et forment un losange (resp un carré) ou si l'objet géométrique est un losange (resp un carré), et vaut 0 sinon.
On tape :

```
est_losange([2,0,0],[0,1,0],[-2,0,0],[0,-1,0])
```

On obtient :

```
1
```

On tape :

```
K:=losange([0,0,0],[2,0,0],[[0,0,1],pi/4]);est_losange(K)
```

`K` est le losange `ABCD` de sommets [0,0,0],[2,0,0], [sqrt(2)+2,0,sqrt(2)],[sqrt(2),0,sqrt(2)], il est situé dans le plan [0,0,0],[2,0,0],[0,0,1] et l'angle `BAD` vaut pi/4.
On obtient :

```
1
```

On tape :

```
est_losange([2,2,0],[-2,2,0],[-2,-1,0],[2,-1,0])
```

On obtient :

```
0
```

10.13.14 Savoir si on a un parallélogramme : `is_parallelogram` `est_parallelogramme`

Voir aussi : 9.19.10 pour la géométrie plane.
`est_parallelogramme` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`est_parallelogramme` vaut 1 (resp 2, 3, 4) si les quatre points sont coplanaires et forment un parallélogramme (resp un losange, un rectangle, un carré) ou si l'objet géométrique est un parallélogramme (resp un losange, un rectangle, un carré), et vaut 0 sinon.
On tape :

```
est_parallelogramme([0,0,0],[2,0,0],[3,1,0],[1,1,0])
```

On obtient :

```
1
```

On tape :

```
K:=parallelogramme([0,0,0],[2,0,0],[1,1,0]);est_parallelogramme(K)
```

`K` est le parallélogramme `ABCD` de sommets [0,0,0],[2,0,0],[3,1,0], [1,1,0]. On tape :

```
est_parallelogramme([-1,0,0],[0,1,0],[2,0,0],[0,-1,0])
```

On obtient :

```
0
```

Attention
On doit taper :

```
K:=parallelogramme([0,0,0],[2,0,0],[1,1,0],D);
          est_parallelogramme(K[0])
```

Pour obtenir :

```
1
```

car K est une liste composée d'un parallélogramme et du point D.

10.14 Les transformations

10.14.1 Généralités

Les transformations ci-dessous :
`translation, rotation, homothetie, similitude, symetrie, inversion, projection`
peuvent toujours être considérées, soit comme des fonctions (les arguments sont les paramètres servant à définir la transformation), soit comme agissant sur le dernier

argument (les premiers arguments sont les paramètres servant à définir la transformation et l'objet géométrique à transformer est mis comme dernier paramètre).
L'objet géométrique à transformer peut être de tout type comme point, droite, plan, polygone, polyèdre, cercle (en 3-d un cercle est considéré comme une courbe paramétrée), courbe paramétrée, sphère, surface etc...
Par exemple si `P` est l'objet géométrique à transformer on peut avoir :
`P:=point(1,1,1)` ou `P:=droite(z=0,x=1)` ou `P:=plan(z=0)` ou `P:=sphere(point(1,`
ou `P:=plotfunc(x2-y2,[x,y])` ou `P:=demi_cone([0,0,0],[0,0,1],pi/6)`
etc...

10.14.2 La translation : `translation`

Voir aussi : 9.18.2 pour la géométrie plane.
`translation`, en géométrie 3-d, a un ou deux arguments : le vecteur de translation donné par la liste de ses coordonnées et éventuellement l'objet géométrique à transformer.
Lorsque `translation` a un argument, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
t:=translation([1,1,1])
```

Puis :

```
t(point(1,2,3))
```

On obtient :

```
Le point (2,3,4) est tracé
```

Lorsque `translation` a deux arguments, `translation` dessine et renvoie le transformé du deuxième argument dans la translation de vecteur le premier argument.
On tape :

```
translation([1,1,1], point(1,2,3))
```

On obtient :

```
Le point (2,3,4) est tracé
```

On tape :

```
translation([1,1,1],droite([0,0,0],[1,2,3]))
```

On obtient :

```
La droite passant par les points (1,1,1) et (2,3,4)
```

10.14.3 La symétrie par rapport à un plan, une droite ou un point : `reflection symetrie`

Voir aussi : 9.18.3 pour la géométrie plane.
`symetrie`, en géométrie 3-d, a un ou deux arguments : un point ou une droite ou un plan et éventuellement l'objet géométrique à transformer.
Lorsque `symetrie` a un argument, c'est une fonction qui agit sur un objet géométrique : quand le premier argument est un point il s'agit de la symétrie par rapport à ce point, quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite et quand le premier argument est un plan il s'agit de la symétrie par rapport à ce plan.
On tape :

```
s:=symetrie(point(1,1,1))
```

Puis :

```
s(point(-1,2,4))
```

On obtient :

```
Le point (3,0,-2) est tracé
```

On tape :

```
sd:=symetrie(droite([1,1,0],[-1,-3,0]))
```

Puis :

```
sd(point(-1,2,4))
```

On obtient :

```
Le point (3,0,-4) est tracé
```

On tape :

```
sp:=symetrie(plan([1,1,0],[-1,-3,0],[1,1,1]))
```

Puis :

```
sp(point(-1,2,4))
```

On obtient :

```
Le point (3,0,4) est tracé
```

Lorsque `symetrie` a deux arguments, `symetrie` dessine et renvoie le transformé du deuxième argument dans la symétrie définie par le premier argument : quand le premier argument est un point il s'agit de la symétrie par rapport à ce point, quand le premier argument est une droite il s'agit de la symétrie par rapport à cette droite et quand le premier argument est un plan il s'agit de la symétrie par rapport à ce plan.
On tape :

```
symetrie(point(1,1,1),point(-1,2,4))
```

On obtient :

`Le point (3,0,-2) est tracé`

On tape :

```
symetrie(droite([1,1,0],[-1,-3,0]),point(-1,2,4))
```

On obtient :

`Le point (3,0,-4) est tracé`

On tape :

```
sp:=symetrie(plan([1,1,0],[-1,-3,0],[1,1,1]),point(-1,2,3))
```

On obtient :

`Le point (3,0,4) est tracé`

10.14.4 La rotation : `rotation`

Voir aussi : 9.18.4 pour la géométrie plane et 10.5.2 pour définir un axe.
`rotation`, en géométrie 3-d, a deux ou trois arguments.
Lorsque `rotation` a deux arguments ce sont : une droite orientée par l'ordre de ses arguments ou par le produit vectoriel des normales orientées des plans qui la définissent (l'axe de rotation) et un réel (la mesure de l'angle de rotation). `rotation` est alors une fonction qui agit sur un objet géométrique (point, droite etc...)
On tape :

```
r:=rotation(droite(point(0,0,0),point(1,1,1)),2*pi/3)
```

Puis :

```
r(point(0,0,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

`Le point (1,0,0) est tracé`

Lorsque `rotation` a trois arguments, ce sont : une droite orientée par l'ordre de ses arguments ou par le produit vectoriel des normales orientées des plans qui la définissent (l'axe de rotation), un réel (la mesure de l'angle de rotation) et l'objet géométrique à transformer ; `rotation` dessine et renvoie alors le transformé du troisième argument dans la rotation d'axe le premier argument et d'angle de mesure le deuxième argument.
On tape :

```
rotation(droite(point(0,0,0),point(1,1,1)),2*pi/3,
                    point(0,0,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point (1,0,0) est tracé
```

On tape :

```
rotation(droite(point(0,0,0),point(1,1,1)), 2*pi/3,
        droite(point(1,0,0),point(0,1,0)))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
La droite passant par (0,1,0) et (0,0,1)
```

10.14.5 L'homothétie : `homothety homothetie`

Voir aussi : 9.18.5 pour la géométrie plane.
`homothetie`, en géométrie 3-d, a deux ou trois arguments : un point (le centre de l'homothétie), un réel (la valeur du rapport de l'homothétie) et éventuellement l'objet géométrique à transformer.
Lorsque `homothetie` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
h:=homothetie(point(0,0,0),2)
```

Puis :

```
h(point(0,0,1))
```

On obtient :

```
Le point (0,0,2) est tracé
```

Lorsque `homothetie` a trois arguments, `homothetie` dessine et renvoie le transformé du troisième argument dans l'homothétie de centre le premier argument et de rapport le deuxième argument.
On tape :

```
homothetie(point(0,0,0),2,point(0,0,1))
```

On obtient :

```
Le point (0,0,2) est tracé
```

On tape :

```
homothetie(point(0,0,0),2,sqhere(point(0,0,0),1))
```

On obtient :

```
La sphère de centre (0,0,0) et de rayon 2
```

10.14.6 La similitude : `similarity similitude`

Voir aussi : 9.18.6 pour la géométrie plane et 10.5.2 pour définir un axe.
`similitude`, en géométrie 3-d, a trois ou quatre arguments : une droite orientée par l'ordre de ses arguments ou par le produit vectoriel des normales orientées des plans qui la définissent (l'axe de rotation), un réel (la valeur du rapport k de la similitude), un réel (la mesure a de l'angle de rotation en radians (ou degrés)) et éventuellement l'objet géométrique à transformer.
Remarque : si le rapport k est négatif, l'angle de la similitude est alors de mesure $-a$ radians (ou degrés).
Lorsque `similitude` a trois arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
s:=similitude(droite(point(0,0,0),point(1,1,1)),
                    2,2*pi/3)
```

Puis :

```
s(point(0,0,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point (2,0,0) est tracé
```

On tape :

```
s(sqhere(point(1,0,0),1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
La sphère de centre (0,2,0) et de rayon 2
```

Lorsque `similitude` a quatre arguments, `similitude` dessine et renvoie le transformé du quatrième argument dans la similitude d'axe le premier argument de rapport le deuxième argument et d'angle le troisième argument.
On tape :

```
similitude(droite(point(0,0,0),point(1,1,1)),
           2,2*pi/3,point(0,0,1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
Le point (2,0,0) est tracé
```

On tape :

```
similitude(droite(point(0,0,0),point(1,1,1)),
      2,2*pi/3,sqhere(point(1,0,0),1))
```

On obtient si on a coché `radian` dans la configuration du `cas` (bouton donnant la ligne d'état) :

```
La sphère de centre (0,2,0) et de rayon 2
```

10.14.7 L'inversion : `inversion`

Voir aussi : 9.18.7 pour la géométrie plane.
`inversion`, en géométrie 3-d, a deux ou trois arguments : un point (le centre de l'inversion), un réel (la valeur du rapport de l'inversion) et éventuellement l'objet géométrique à transformer.
Lorsque `inversion` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
Si `inver:=inversion(C,k)` et `A1:=inver(A)`, on a $\overline{CA} * \overline{CA1} = k$.
On tape :

```
inver:=inversion(point(0,0,0),2)
```

Puis :
On tape :

```
inver(point(1,2,-2))
```

On obtient :

```
Le point (2/9,4/9,-4/9)
```

On a en effet :
`[2/9,4/9,-4/9]*[1,2,-2]=2` et les points (1,2,-2) et (2/9,4/9,-4/9) sont alignés avec le point (0,0,0) centre de l'nversion.
On tape

```
inver(sqhere(point(1,0,0),1))
```

On obtient :

```
Le plan d'équation x=1
```

On tape :

```
inver(sqhere(point(1,0,0),1/2))
```

On obtient :

```
La sphère de centre (8/3,0,0) et de rayon 4/3 (elle
    passe par les points (4/3,0,0) et (4,0,0))
```

Lorsque `inversion` a trois arguments, `inversion` dessine et renvoie le transformé du troisième argument dans l'inversion de centre le premier argument et de rapport le deuxième argument.
Si `A1:=inversion(C,k,A)` on a $\overline{CA} * \overline{CA1} = k$.
On tape :

```
inversion(point(0,0,0),2,sphere(point(1,0,0),1))
```

On obtient :

```
Le plan d'équation x=1
```

On tape :

```
inversion(point(0,0,0),2,sphere(point(1,0,0),1/2))
```

On obtient :

```
La sphère de centre (8/3,0,0) et de rayon 4/3
```

10.14.8 La projection orthogonale : `projection`

Voir aussi : 9.18.8 pour la géométrie plane.
`projection`, en géométrie 3-d, a un ou deux arguments : un objet géométrique et éventuellement un point.
Lorsque `projection` a un argument, c'est une fonction qui agit sur un point et qui projette ce point sur l'objet géométrique.
On tape :

```
p1:=projection(droite(point(0,0,0),point(1,1,1)))
```

Puis :

```
p1(point(1,0,0))
```

On obtient :

```
Le point (1/3,1/3,1/3) est tracé
```

On tape :

```
p2:=projection(plan(point(1,0,0),point(0,0,0),
point(1,1,1)))
p2(point(0,0,1))
```

On obtient :

```
Le point (0,1/2,1/2) est tracé
```

On tape :

```
p3:=projection(sqhere(point(1,0,0),1))
p3(point(0,0,1))
```

On obtient :

```
Le point (1-sqrt(2)/2,0,sqrt(2)/2) est tracé
```

Lorsque `projection` a deux arguments, `projection` dessine et renvoie le transformé du point par la projection orthogonale sur le premier argument.
On tape :

```
projection(droite(point(0,0,0),point(1,1,1)),point(1,0,0))
```

On obtient :

```
Le point (1/3,1/3,1/3)
```

On tape :

```
projection(plan(point(1,0,0),point(0,0,0),
point(1,1,1)),point(0,0,1))
```

On obtient :

```
Le point (0,1/2,1/2) est tracé
```

On tape :

```
projection(sqhere(point(1,0,0),1),point(0,O,1))
```

On obtient :

```
Le point (1-sqrt(2)/2,0,sqrt(2)/2) est tracé
```

10.15 Les surfaces

10.15.1 Le cône : `cone`

`cone` a comme argument un point `A`, une direction `v` et un réel `t`.
`cone(A,v,t)` dessine un demi-cône de sommet `A`, direction `v` et de demi-angle au sommet `t`.
On tape :

```
cone([0,1,0],[0,0,1],pi/3)
```

On obtient :

Le cône d'axe paralléle à 0z passant par [0,1,0] et de demi-angle au sommet égal à $\pi/3$

10.15.2 Le demi-cône : `half_cone demi_cone`

`demi_cone` a comme argument un point `A`, une direction `v` et un réel `t`.
`demi_cone(A,v,t)` dessine un demi-cône de sommet `A`, direction `v` et de demi-angle au sommet `t`.
On tape :

```
demi_cone([0,1,0],[0,0,1],pi/3)
```

On obtient :

Le demi-cône supérieur d'axe paralléle à 0z passant par [0,1,0] et de demi-angle au sommet égal à $\pi/3$

10.15.3 Le cylindre : `cylinder cylindre`

`cylindre` a comme argument un point `A`, une direction `v` et un réel `r`.
`cylindre(A,v,r)` dessine un cylindre d'axe (`A`,`v`) et de rayon `r`.
On tape :

```
cylindre([0,1,0],[0,0,1],3)
```

On obtient :

Le cylindre d'axe paralléle à 0z passant par [0,1,0] et de rayon 3 (le cercle de base est dans x0y et a pour equation $x^2 + (y-1)^2 = 9$)

10.15.4 La sphère : `sphere`

`sphere` a comme argument soit deux points (le diamètre de la sphère) soit un point et un réel (son centre et son rayon).
`sphere(A,B)` (resp `sphere(A,r)`) trace la sphère de diamètre `AB` (resp centre `A` et de rayon `r`) dans l'espace 3-d.
On tape :

```
sphere([-2,0,0],[2,0,0])
```

Puis on tape :

```
sphere([0,0,0],2)
```

On obtient :

`La sphère de centre l'origine et de rayon 2`

10.15.5 Surface définie par une fonction : `funcplot`

`funcplot` a comme argument une expression de deux variables et la liste de ses variables.
`funcplot(f(x,y),[x,y])` dessine la surface z=f(x,y).
On tape :

```
funcplot(x^2+y^2,[x,y])
```

On obtient :

`Le paraboloïde` $z = x^2 + y^2$

10.15.6 Surface définie par une équation paramétrique : `paramplot`

`paramplot` a comme arguments la liste formée de l'équation paramétrique d'une surface et de la liste des paramètres utilisés.
`paramplot([f(u,v),g(u,v),h(u,v)],[u,v])` dessine la surface d'équation paramétrique : `x=f(u,v),y=g(u,v),z=h(u,v)`.
On tape :

```
paramplot([u*cos(v),u*sin(v),u],[u,v])
```

On obtient :

`Le dessin du cône de sommet l'origine, d'axe 0z et contenant le cercle définit par` $x^2 + y^2 = 1$ `et` $z = 1$.

10.16 Les solides

10.16.1 Le cube : `cube`

`cube` a comme argument trois points `A,B,C`.
`cube` dessine le cube direct de coté `AB` dont une face est dans le `plan(A,B,C)`.
On tape :

```
cube([0,0,0],[0,4,0],[0,0,1])
```

On obtient :

`Le cube de sommets [0,0,0],[4,0,0], [0,4,0],[0,0,4]...`

On tape :

```
cube([0,0,0],[0,4,0],[0,0,-1])
```

On obtient :

`Le cube symétrique du précédent par rapport à 0y`

10.16.2 Le tétraèdre : `tetrahedron pyramid tetraedre pyramide`

`tetraedre` ou `pyramide` a 3 ou 4 arguments qui sont les points `A,B,C` ou les points `A,B,C,D`.
`tetraedre` ou `pyramide` dessine le tétraèdre régulier direct de coté `AB` dont une face est dans le `plan(A,B,C)` quand il y a 3 arguments et la pyramide `ABCD` quand il y a 4 arguments.
On tape :

```
pyramide([-2,0,0],[2,0,0],[0,2,0])
```

Ou on tape :

```
tetraedre([-2,0,0],[2,0,0],[0,2,0])
```

On obtient :

```
Le tetraèdre régulier de sommets
[-2,0,0],[2,0,0],[0,2*sqrt(3),0] et
[0,2*sqrt(3)/3,4*sqrt(6)/3])
```

On tape :

```
pyramide([-2,0,0],[2,0,0],[0,2,0],[0,0,2])
```

Ou on tape :

```
tetraedre([-2,0,0],[2,0,0],[0,2,0],[0,0,2])
```

On obtient :

```
Le tétraèdre de sommets
[-2,0,0],[2,0,0],[0,2,0],[0,0,2]
```

10.16.3 Le parallélépipède : `parallelepiped parallelepipede`

`parallelepipede` a comme argument quatre points `A,B,C,D`.
`parallelepipede(A,B,C,D)` dessine un parallélépipède de côtés `AB,AC,AD` (les faces sont des parallélogrammes).
On tape :

```
parallelepipede([0,0,0],[5,0,0],[0,5,0],[0,0,5])
```

On obtient :

```
un cube de sommet l'origine et dont 3 faces sont
contenues dans les plans du repère
```

On tape :

```
parallelepipede([0,0,0],[3,2,0],[0,3,2],[2,0,3])
```

On obtient :

```
Un parallélépipède
```

10.16.4 Le prisme : `prism prisme`

`prisme` a comme argument une liste de points `[A,B,C,D...]` d'un même plan et un point `A1`.
`prisme([A,B,C,D,..],A1)` dessine un prisme de base le polygone `A,B,C,D...` et de arêtes parallèles à `AA1` (les faces sont des parallélogrammes).
On tape :

```
prisme([[0,0,0],[5,0,0],[0,5,0],[-5,5,0]],[0,0,5])
```

On obtient :

```
un prisme de base un parallélogramme et d'arrêtes
verticales
```

10.16.5 Les polyèdres : `polyhedron polyedre`

`polyedre` a comme argument une séquence de points.
`polyedre` dessine un polyèdre convexe dont les sommets sont parmi les point de la séquence donnée en argument, les autres points se trouvant à l'intérieur ou sur les faces du polyèdre.
On tape :

```
polyedre([0,0,0],[-2,0,0],[2,0,0],[0,2,0],[0,0,2])
```

On obtient :

```
Le tetraèdre de sommets
[-2,0,0],[2,0,0],[0,2,0],[0,0,2]
```

10.16.6 Les faces : `faces`

`faces` a comme argument un `polyedre`.
`faces` dessine et renvoie la liste des faces du polyèdre.
Une face est une matrice de n lignes et 3 colonnes dont les lignes sont les sommets de la face.
On tape :

```
faces(polyedre([0,0,0],[-2,0,0],[2,0,0],[0,2,0],[0,0,2]))[1]
```

On obtient :

```
Le dessin d'une face et dans l'historique on a
[[0,0,0],[-2,0,0],[2,0,0],[0,0,2]]
```

10.16.7 Les arêtes : `line_segments aretes`

`aretes` a comme argument un `polygone` ou un `polyedre`.
`aretes` dessine et renvoie la liste des arêtes du polyèdre.
Une arête est un segment.
On tape :

`aretes(polyedre([0,0,0],[-2,0,0],[2,0,0],[0,2,0],[0,0,2]))[1]`

On obtient :

```
le dessin du segment (-2,0,0),(2,0,0) et dans
l'historique on a pnt(pnt[group[[-2,0,0],[2,0,0]],0])
```

10.17 Les solides de Platon

Pour les construire, on donne le centre, un sommet et un 3ème point définissant un plan de symétrie.
Pour accélérer les calculs, il peut être utile faire seulement des calculs approchés en utilisant `evalf` dans l'argument : on tapera, par exemple :
`cube_centre(evalf([0,0,0],[3,2,1],[1,1,0]))`

10.17.1 : `centered_tetrahedron tetraedre_centre`

`tetraedre_centre` a comme argument trois points `A,B,C`.
`tetraedre_centre` dessine un tetraèdre de centre `A`, de sommet `B` et tel que le plan `ABC` contient 1 sommet du tetraèdre.
On tape :

```
tetraedre_centre([0,0,0],[0,0,5],[0,1,0])
```

On obtient :

```
Le tetraédre régulier de centre [0,0,0], de sommet
[0,0,5] et ayant un sommet dans le plan y0z
```

10.17.2 : `centered_cube cube_centre`

`cube_centre` a comme argument trois points `A,B,C`.
`cube_centre` dessine un cube de centre `A`, de sommet `B` et tel que le plan `ABC` soit un plan de symetrie du cube.
Ce plan `ABC` contient une arête `BD` du cube issue de `B` et le sommet `D` de cette arête est situé du même côté que `C` par rapport à `AB`.
On tape :

```
cube_centre([0,0,0],[3,3,3],[0,1,0])
```

On obtient :

```
Le cube de centre [0,0,0], de sommet [3,3,3] et ayant
comme sommet D [-1,5,-1]
```

On tape :

```
cube_centre([0,0,0],[3,3,3],[0,-1,0])
```

Ou on tape :

```
cube_centre([0,0,0],[3,3,3],[3,-3,3])
```

On obtient :

```
Le cube de centre [0,0,0], de sommet [3,3,3] et ayant
l'axe det ayant comme sommet D [3,-3,3] et dont les
arêtes sont parallèles aux axes
```

Remarques

– Il existe (cf les exemples ci-dessus) 2 cubes ayant un sommet commun `B`, même centre `A`, et le même plan de symétrie `ABC`.
  En effet, si a est le côté du cube, on a $AB = a * \sqrt{3}$. La donnée de A et de B détermine donc entièrement la sphère S inscrite dans le cube. Le plan ABC coupe S selon un cercle Cs et le cube selon un rectangle $BDEF$ avec $BD = a$ et $BF = a * \sqrt{2}$ car BD est une arête et BF est une diagonale d'une face du cube. Cette face est tangente à S et BF est une tangente au cercle Cs menée par B. Du point B, on peut mener, dans le plan ABC, deux tangentes au cercle Cs. Ces 2 tangentes sont situées de part et d'autre de la droite AB (car A est le centre du cercle Cs). On choisit la tangente située dans le demi-plan ne contenant pas C et on détermine ainsi, une face contenue dans le plan tangent à la sphère passant par cette tangente. Par symétrie par rapport à A, on détermine enfin tout le cube.
  Si le cube a `A` comme centre et `B` comme sommet et si de plus le plan `ABC` contient une arête du cube, on peut aussi dire que le point `C` est dans un plan passant par `A, B` et une arête du cube `BD` issue de `B` et que `C` est du même côté que `D` par rapport à `AB`.

– Pour avoir les coordonnées des sommets du cube, on tape :

```
normal(cube_centre([0,0,0],[3,3,3],[0,1,0])),0
```

  car c'est le dernier argument d'une liste qui détermine le mode d'affichage : ici `0` dit que l'on affiche des expressions.
  On obtient :

```
pnt(pnt[[[[3,3,3],[-1,-1,5],[-5,1,1],[-1,5,-1]],
[[3,3,3],[-1,5,-1],[1,1,-5],[5,-1,-1]],[[3,3,3],[-1,-1,5],[1,-5,1
[5,-1,-1]],[[-1,-1,5],[-5,1,1],[-3,-3,-3],[1,-5,1]],[[-1,5,-1],
[-5,1,1],[-3,-3,-3],[1,1,-5]],[[5,-1,-1],[1,-5,1],[-3,-3,-3],
[1,1,-5]]],0]),0
```

  On peut aussi taper

```
normal(coordonnees(sommets(cube_centre([0,0,0],[3,3,3],[0,1,0])))
```

10.17.3 L'octaèdre : `octahedron octaedre`

`octaedre` a comme argument trois points `A,B,C`.
`octaedre(A,B,C)` dessine un octaèdre de centre `A`, de sommet `B` et tel que le plan `ABC` contient 4 sommets de l'octaèdre.
On tape :

```
octaedre([0,0,0],[0,0,5],[0,1,0])
```

Ou on tape :

```
octaedre([0,0,0],[0,5,0],[0,0,1])
```

Ou on tape :

```
octaedre([0,0,0],[5,0,0],[0,0,1])
```

On obtient :

```
L’octaèdre de centre l’origine et de sommets situés
sur les axes à +5 et -5.
```

10.17.4 Le dodecaèdre : `dodecahedron dodecaedre`

a comme argument trois points `A,B,C`.
`dodecaedre(A,B,C)` dessine un dodecaèdre de centre `A`, de sommet `B` tel que le plan `ABC` contient un axe de symetrie du dodecaedre.
On tape :

```
dodecaedre([0,0,0],[0,0,5],[0,1,0])
```

On obtient :

```
Un dodecaèdre de centre l’origine, de sommet [0,0,5]
et dont un axe de symétrie est dans le plan 0yz
```

On tape :

```
dodecaedre([0,0,0],[0,2,sqrt(5)/2+3/2],[0,0,1])
```

On obtient :

```
Un dodecaèdre de centre l’origine, de sommet
[0,2,sqrt(5)/2+3/2] et dont un axe de symétrie est 0z
```

Attention Les faces (c’est à dire les pentagones) sont dessinées à l’aide d’un triangle et d’un trapèze (on a le dessin du pentagone et d’une de ses diagonales).

10.17.5 L’icosaèdre : `icosahedron icosaedre`

`icosaedre` a comme argument trois points `A,B,C`.
`icosaedre(A,B,C)` dessine un icosaèdre de centre `A`, de sommet `B` tel que le plan `ABC` contienne un sommet parmi les 5 les plus proche de B.
On tape :

```
icosaedre([0,0,0],[0,0,5],[0,1,0])
```

On obtient :

```
Un icosaèdre de centre l’origine, de sommet [0,0,5] et
le plan 0yz contient un sommet
```

On tape :

```
icosaedre([0,0,0],[0,0,sqrt(5)],[2,1,0])
```

On obtient :

```
Un icosaèdre de centre l’origine, de sommet
[0,0,sqrt(5)] et les plans z=1 et z=-1 contiennent les
sommets de l’icosaèdre formant 2 pentagones
```

10.18 Figures et preuves d''exercices avec Xcas

10.18.1 Exercice 1

Soit SABCD une pyramide de sommet S dont la base ABCD est un parallélogramme de centre O.
Soient M et N les milieux respectifs de SA et SD.
Montrer que les plans OMN et SBC sont parallèles.
Pour faire la figure, on tape :

```
// dessin exo1
A:=point(2,2,0);
B:=point(-2,2,0);
C:=point(-3,-2,0);
D:=point(1,-2,0);
S:=point(0,0,4);
polyedre(S,A,B,C,D);
M:=milieu(S,A);
N:=milieu(S,D);
O:=milieu(A,C);
P:=plan(O,M,N);
Q:=plan(S,B,C);
est_parallele(P,Q);
```

Pour faire la démonstration avec Xcas, on tape :

```
// demo exo1
A:=point(a,b,c);
B:=point(d,e,f);
D:=point(u,v,w);
C:=B+(D-A);
//S:=isobarycentre(A,B,C,D)+t*cross(B-A,D-A);
S:=point(k,l,m);
M:=milieu(S,A);
N:=milieu(S,D);
O:=milieu(A,C);
P:=plan(O,M,N);
Q:=plan(S,B,C);
est_parallele(P,Q);
```

On obtient : 1
ce qui veut dire que les plans P et Q sont parallèles.

La démonstration géométrique :
Soit R le milieu de AB.
On a :
M est le milieu de SA et N est le milieu de SD, donc MN est parallèle à AD et 2*MN=AD
M est le milieu de SA et R est le milieu de AB, donc MR est parallèle à SB et 2*MR=SB

O est le milieu de AC et R est le milieu de AB, donc OR est parallèle à CB et 2*OR=CB
ou encore :
$2*\overrightarrow{MN}=\overrightarrow{AD}$
$2*\overrightarrow{MR}=\overrightarrow{SB}$
$2*\overrightarrow{OR}=\overrightarrow{CB}$
La base ABCD est un parallélogramme donc :
overrightarrowAD $=\overrightarrow{BC}$
On en déduit que MNOR est un parallélogramme dont le plan est parallèle à la face SBC.

10.18.2 Exercice 2

Soit SABCD une pyramide de sommet S dont la base ABCD est un carré de centre O et tel que SO est perpendiculaire au plan ABCD.
Le plan passant par C et perpendiculaire à SA coupe SA, SB, SC respectivement en M, N et P.
Montrer que NP est parallèle à BD.
Pour faire la figure, on tape :

```
//dessin exo2
A:=point(2,2,0);
B:=point(-2,2,0);
C:=point(-2,-2,0);
D:=point(2,-2,0);
S:=point(0,0,4);
polyedre(S,A,B,C,D);
Q:=perpendiculaire(C,droite(S,A));
M:=head(inter(Q,droite(S,A)));
N:=head(inter(Q,droite(S,B)));
P:=head(inter(Q,droite(S,D)));
d1:=droite(N,P);
d2:=droite(B,D);
est_parallele(d1,d2)
```

Pour faire la démonstration avec Xcas, on peut taper en supposant que le plan du carré est le plan Oxy :

```
// demo exo2
A:=point(a,b,0);
B:=point(c,f,0);
d:=affixe(rotation(a+i*b,pi/2,point(c+i*f)));
d1:=re(d);
d2:=im(d);
D:=point(d1,d2,0);
C:=normal(B+(D-A));
S:=point((c+d1)/2,(d+d2)/2,g);
Q:=perpendiculaire(C,droite(S,A));
M:=head(inter(Q,droite(S,A)));
```

```
N:=normal(head(inter(Q,droite(S,B))));
P:=normal(head(inter(Q,droite(S,D))));
est_parallele(droite(D,B),droite(N,P));
```

On obtient :
`1`
mais on peut simplifier les calculs, car on peut choisir les axes pour avoir A sur Ox, on tape alors :

```
// demo exo2
A:=point(a,0,0);
B:=point(0,a,0);
C:=point(-a,0,0);
D:=point(0,-a,0);
S:=point(0,0,b);
Q:=perpendiculaire(C,droite(S,A));
M:=head(inter(Q,droite(S,A)));
N:=normal(head(inter(Q,droite(S,B))));
P:=normal(head(inter(Q,droite(S,D))));
est_parallele(droite(D,B),droite(N,P));
```

On obtient :
`1`
La démonstration géométrique :
Les triangles `OAS`, `OBS` et `ODS` rectangles en `O` sont égaux puisque `OA=OB=OD` en tant que demi diagonale d'un carré.
Donc les arêtes `SA` `SB` et `SD` sont égales ainsi que les triangles isocéles `SAB` et `SAD` (la pyramide est donc régulière).
Les triangles `MSN` et `MSP` rectangles en `M` sont égaux (même côté `MS` et leurs angles `S` sont égaux), donc `SN=SP`.
Puisque `SB=SD`, on a donc :
$$\frac{SN}{SB} = \frac{SP}{SD}$$
ce qui montre d'après le théorème de Thalès que `NP` est parallèle à `BD`.

10.18.3 Exercice 3 prolongement de l'exercice 2

Soit `SABCD` une pyramide de sommet `S` dont la base `ABCD` est un losange de centre `O` et tel que `SO` est perpendiculaire au plan `ABCD`.
Le plan passant par `C` et perpendiculaire à `SA` coupe `SA`, `SB`, `SC` respectivement en `M`, `N` et `P`.
Montrer que `NP` est parallèle à `BD`.
Pour faire la démonstration avec `Xcas`, on peut taper en supposant que le plan du losange est le plan Oxy et que lepoint A est sur Ox, on tape alors puisque les diagonales d'un losange sont perpendiculaires en leur milieu :

```
// demo exo3
A:=point(a,0,0);
B:=point(0,b,0);
C:=point(-a,0,0);
```

```
D:=point(0,-b,0);
S:=point(0,0,c);
Q:=perpendiculaire(C,droite(S,A));
M:=head(inter(Q,droite(S,A)));
N:=normal(head(inter(Q,droite(S,B))));
P:=normal(head(inter(Q,droite(S,D))));
est_parallele(droite(D,B),droite(N,P));
```

On obtient :
`1`
La démonstration géométrique :
Les triangles `OBS` et `ODS` rectangles en `O` sont égaux puisque `OB=OD` en tant que demi diagonale d'un losange.
Donc les arêtes `SB` et `SD` sont égales ainsi les triangles `SAB` et `SAD` ont 3 cotés égaux ; ils sont donc égaux.
Les triangles `MSN` et `MSP` rectangles en `M` sont égaux (même côté `MS` et leurs angles `S` sont égaux), donc `SN=SP`.
Puisque `SB=SD`, on a donc :
$\frac{\mathtt{SN}}{\mathtt{SB}} = \frac{\mathtt{SP}}{\mathtt{SD}}$
ce qui montre d'après le théorème de Thalès que `NP` est parallèle à `BD`.

Chapitre 11

Utilisation de `giac` à l'intérieur d'un programme

11.1 Utilisation dans un programme `C++`

On peut utiliser `giac` à l'intérieur d'un programme `C++` en mettant au début du programme par exemple `essai.cc` :
`#include<giac/giac.h>`
puis en compilant le compilant avec :
`c++ -g essai.cc -lgiac -lgmp`
et en l'exécutant en mettant :
`./a.out`
Exemple

```
// -*- compile-command: "g++ -g pgcd.cc -lgiac -lgmp" -*-
#include <giac/giac.h>

using namespace std;
using namespace giac;

gen pgcd(gen a,gen b){
  gen q,r;
  for (;b!=0;){
    r=irem(a,b,q);
    a=b;
    b=r;
  }
  return a;
}

int main(){
  cout << "Entrer 2 entiers";
  gen a,b;
  cin >> a >> b;
  cout << pgcd(a,b) << endl;
  return 0;
```

```
}
```

11.2 Pour définir de nouvelles fonctions de `giac`

On peut définir de nouvelles fonctions qui deviendront des fonctions de `giac`. Pour définir par exemple la fonction de nom `pgcd` (et c'est l'instruction : `const string _pgcd_s("pgcd")` ; qui définit le nom de la fonction), on tape :

```
// -*- mode:C++ ; compile-command: "g++ -I.. -fPIC -DPIC
-g -c pgcd.cpp  -o pgcd.lo && ln -sf pgcd.lo pgcd.o && gcc
-shared pgcd.lo -lc  -Wl,-soname -Wl,libpgcd.so.0 -o
libpgcd.so.0.0.0 &&  ln -sf libpgcd.so.0.0.0 libpgcd.so.0 &&
 ln -sf libpgcd.so.0.0.0 libpgcd.so" -*-
using namespace std;
#include <stdexcept>
#include <cmath>
#include <cstdlib>
#include <giac/giac.h>
#include "pgcd.h"

#ifndef NO_NAMESPACE_GIAC
namespace giac {
#endif // ndef NO_NAMESPACE_GIAC

  gen pgcd(gen a,gen b){
    gen q,r;
    for (;b!=0;){
      r=irem(a,b,q);
      a=b;
      b=r;
    }
    return a;
  }
  gen _pgcd(const gen & args){
    if ((args.type!=_VECT)||(args._VECTptr->size()!=2))
      setsizeerr();
    vecteur &v=*args._VECTptr;
    return pgcd(v[0],v[1]);
  }
  const string _pgcd_s("pgcd");
  unary_function_unary __pgcd(&_pgcd,_pgcd_s);
  unary_function_ptr at_pgcd (&__pgcd,0,true);


#ifndef NO_NAMESPACE_GIAC
} // namespace giac
#endif // ndef NO_NAMESPACE_GIAC
```

On compile avec la commande située après `compile-command` de l'en-tête du programme. Puis, pour l'insérer dans une session `Xcas`, il faut taper la commande `insmod` suivi du chemin absolu complet de la librairie, par exemple :
`insmod("/home/user/giac-0.4.0/doc/en/libpgcd.so").`
Cela suppose que le source de giac a été désarchivé dans le répertoire /home/user).

Le type `gen`

`gen` est le type qui sert à représenter toutes les données de calcul formel : les fonctions de calcul formel travaillent sur des `gen`.
Si g est de type `gen` (`gen g;`), selon le type de `g`, `g.type` aura différentes valeurs qui sont :
`_INT_` si `g.val` est un entier de type `int`,
`_DOUBLE_` si `g._DOUBLE_val` est un réel de type `double`
`_SYMB` si `g._SYMBptr` pointe sur un objet de type `symbolique`
`_VECT` si `g._VECTptr` pointe sur un vecteur de type `vecteur`
`_ZINT` si `g._ZINTptr` pointe sur un entier de type `zint`
`_IDNT` si `g._IDNTptr` pointe sur un identificateur de type `idnt`
`_CPLX` si `g._CPLXptr` pointe sur un complexe de type `complexe`