Calcul formel et Mathématiques avec la calculatrice HP Prime

Renée De Graeve
Maître de Conférence à Grenoble I

0.1 Index

L'index commence page suivante.

Index

Table des matières

Première partie

Pour commencer

0.2 Généralités

Avec la calculatrice HPrime vous avez deux calculatrices en une : une pour faire du calcul formel et exact (touche CAS), l'autre pour faire du calcul approché (touche HOME). Ceci correspond au mariage de deux logiciels dans la calculatrice ; Giac/Xcas pour le CAS et le logiciel développé par HP pour ses gammes de calculatrices scientifiques et graphiques dans HOME. Ces deux logiques sont souvent contradictoires, ce qui a nécessité un important travail de mise en cohérence pour pouvoir utiliser des données de HOME dans CAS et réciproquement, effort qui se poursuit encore aujourd'hui.

Ainsi la logique d'un logiciel de calcul formel est de ne pas avoir de variable préaffectée et de pouvoir stocker n'importe quel type de donnée dans une variable dont le nom est libre (en particulier un nom de variable peut contenir plus qu'une lettre) alors que la logique des calculatrices HP38/39/40 était d'avoir des variables préaffectées dont le nom de variable est une lettre ou une lettre complétée par un chiffre, et ne pouvant stocker qu'un type de donnée : $A, B..Z$ pour les réels, $Z_0, ..Z_9$ pour les complexes, $L_0, L_1..L_9$ pour les listes, $M_0, M_1..M_9$ pour les vecteurs ou matrices etc.... Cela a bien sûr des conséquences importantes, si on écrit `ab` dans CAS, cela désigne une variable dont le nom est en deux lettres, alors que `AB` dans HOME désigne le produit (multiplication implicite) des deux variables `A` et `B`.

Pour éviter les confusions, il est conseillé d'utiliser des noms de variables en minuscules dans le CAS, les noms de commande du CAS sont sauf exceptions en minuscules, alors que les noms de commande dans HOME sont en majuscules. Ce choix est facilité par le blocage du clavier alphabétique en minuscules dans CAS et en majuscules dans HOME. De nombreuses commandes existent dans les deux versions (HOME en majuscule, CAS en minuscules), la plupart du temps elles font la même chose, mais il existe malheureusement des exceptions, par exemple `size` et `SIZE` (voir plus bas).

Notez aussi que dans HOME, on fait la différence entre les listes (`L1:={1,2,3}`) et les vecteurs (`M1:=[1,2,3]`) et la notion de séquence n'existe pas, alors que dans CAS il n'y a pas de différence entre listes et vecteurs (`v:=[1,2,3]` ou `v:={1,2,3}`) et on peut travailler avec des séquence (`s:=1,2,3`).

De plus, **ATTENTION !** l'historique ne reflète pas toujours ce qui a été tapé, dans l'historique de HOME les minuscules sont transformées en majuscules, alors que dans l'historique du CAS, cela dépend si l'on a coché ou non `Affichage Livre` dans la configuration générale (Shift  (Settings)).

Exemple :
Dans l'écran HOME ou dans l'écran CAS, on tape :
`SIZE(1,2,3)` ou `size(1,2,3)`
On obtient dans l'historique `SIZE(1,2,3)` et en réponse : `3`
Dans l'écran HOME, on tape :
`SIZE([1,2,3])` ou `size([1,2,3])`
On obtient dans l'historique `SIZE([1,2,3])` et en réponse : `{3}`
Dans l'écran CAS, on tape :
`SIZE([1,2,3])` ou `size([1,2,3])`
On obtient dans l'historique (si on n'a pas coché `Affichage Livre`) :
`SIZE([1,2,3])` ou `size([1,2,3])` et en réponse : `3`

Dans l'écran HOME, on tape :
`SIZE([[1,2,3],[4,5,6]])` ou `size([[1,2,3],[4,5,6]])`
On obtient dans l'historique `SIZE([[1,2,3],[4,5,6]])` et en réponse : `{2,3}`
Dans l'écran CAS, on tape :
`size([[1,2,3],[4,5,6]])`
On obtient dans l'historique (si on n'a pas coché `Affichage Livre`) :
`size([[1,2,3],[4,5,6]])`
et en réponse : `2`
Mais si on tape dans CAS :
`SIZE([[1,2,3],[4,5,6]])`
On obtient dans l'historique (si on n'a pas coché `Affichage Livre`) :
`SIZE([[1,2,3],[4,5,6]])` et en réponse : `[2,3]`

CONSEIL faites un choix : soit vous travaillez toujours dans HOME soit dans CAS car les commandes de même nom qui ne renvoient pas la même chose dans HOME et dans CAS devient vite un vrai casse-tête !

Remarque pour les utilisateurs de Xcas : la prise en main de la calculatrice mode CAS devrait être rapide. Notez toutefois que :

- il n'y a pas de support pour la traduction des commandes en français, il faut utiliser les noms de commandes en anglais (`factor` et non `factoriser`, `line` et non `droite`, etc.)
- certaines commandes ne sont pas disponibles, HP n'ayant pas souhaité les intégrer (par exemple toutes les commandes sur les permutations)
- certains synonymes ne sont pas disponibles, et le choix fait par HP ne s'est malheureusement pas toujours porté sur la commande native Xcas en minuscules mais sur un nom de commande mixte avec une lettre en majuscules au milieu du nom de commande.
- L'interface pour utiliser le langage de programmation de Xcas est encore perfectible (par exemple le clavier alphabétique se bloque en majuscules même si on sélectionne un programme CAS, l'interface de la fonction de mise au point `debug` est expérimentale...)

0.3 Les touches `CAS` et 

Avec la calculatrice `HPprime` vous pouvez choisir de travailler en mode exact ou en mode approché : il y a deux écrans, l'un pour faire du calcul exact c'est l'écran CAS, l'autre pour faire du calcul approché c'est l'écran HOME.
Dans l'écran CAS ont peut aussi faire du calcul approché par exemple $1/2$ est un nombre exact et `evalf(1/2)=0.5` est un nombre approché. Si dans une expression il y a un nombre approché le résultat sera approché, par exepmle : $1/2 + 1/3$ renvoie $5/6$ alors que $0.5 + 1/3$ renvoie 0.833333333333.
Dans l'écran CAS les commandes sont en minuscules et dans l'écran HOME les commandes sont en majuscules. Si vous appuyez sur `CAS` vous travaillez en mode exact et si vous appuyez sur  vous travaillez en mode aprroché.
Qu'est-ce que cela change ?
On va considérer, par exemple, 2 suites u et v définies par ;
$u_0 = \dfrac{2}{3},\ u_{n+1} = 2u_n - \dfrac{2}{3}$ $(n \geq 0)$ et

$v_0 = \frac{2}{3},\ v_{n+1} = 2(u_n - \frac{1}{3})\ (n \geq 0)$.

Dans l'écran CAS

On appuie sur CAS et on tape pour avoir les premiers termes de u :
`2/3` puis `Enter` et on obtient `2/3`.
On tape :
`2*Ans-2/3` puis `Enter`,`Enter`...
et on obtient `2/3`, `2/3`, `2/3`...
En mode exact, i.e.dans l'écran CAS, la suite u est donc stationnaire et vaut $\frac{2}{3}$.
Le résultat est ici conforme au résultat théorique.
Toujours dans le CAS, on tape pour avoir les premiers termes de v :
`2/3` puis `Enter` et on obtient `2/3`.
On tape :
`2*(Ans-1/3)` puis `Enter`,`Enter`...
et on obtient `2/3`, `2/3`, `2/3`...
En mode exact, i.e. dans l'écran CAS, la suite v est donc stationnaire et vaut $\frac{2}{3}$.
Le résultat est encore ici conforme au résultat théorique.

Dans l'écran HOME

Maintenant on appuie sur ⌂ et pour avoir les premiers termes de u, on tape la valeur de u_0 :
`2/3` puis `Enter` et on obtient `0.666666666667` puis.
on tape :
`2*Ans-2/3` puis `Enter`, `Enter`, `Enter`...
et on obtient `0.666666666663`,`0.666666666663`...
Le résultat est ici presque conforme au résultat théorique.
En mode approché i.e. dans l'écran HOME (touche ⌂), la suite u est donc stationnaire à partir de $n > 0$ et vaut 0.666666666663.
Toujours dans HOME (touche ⌂), on tape pour avoir les premiers termes de v :
`2/3` puis `Enter` et on obtient `0.666666666667`.
On tape :
`2*(Ans-1/3)` puis `Enter`, `Enter`, `Enter`...
et on obtient
v_1=`0.666666666668`, v_2=`0.666666666670`, v_3=`0.666666666674`puis
`0.666666666682`, `0.666666666682`, `0.666666666698`
`0.666666666730`, `0.666666666794`, `0.666666666922`
etc... et apres 51 ou 52 `Enter` on obtient :
v_{40}`1.76617829443` et v_{50}`2 252.46648036` etc.. En mode approché, i.e. dans l'écran HOME (touche ⌂), la suite v tend donc vers $+\infty$.
On voit donc qu'en mode approchè les erreurs de calculs se cumulent et que les résultats affichés ne sont pas toujours conformes aux résultats théoriques !

Comment sont fait les calculs dans HOME

Dans HOME, les nombres réels sont affichés au plus 12 chiffres significatifs mais les calculs sont faits avec plus de chiffres puis sont arrondis pour être affichés par exemple :
1/3 sera représenté par 0.333333333333 (avec 12 fois le chiffre 3)
2/3 sera représenté par 0.666666666667 (avec 11 fois le chiffre 6 et un 7)

4/3 sera représenté par 1.33333333333 (avec 1 puis 11 fois le chiffre 3)
2*0.666666666667 ou 2*0.666666666663 sera représenté par 1.33333333333 (avec 1 puis 11 fois le chiffre 3)
Pour le calcul de u on tape u_0 :
`2/3` on obtient 0.666666666667 puis,
`2*Ans-2/3` on obtient 1.33333333333-0.666666666667=0.666666666663 puis,
`2*Ans-2/3` on obtient puisque 1.33333333333-0.666666666667=0.666666666663
etc...la suite u est donc stationnaire pour $n > 0$ et vaut 0.666666666663.
Pour la suite v le calcul est fait après avoir mis 2 en facteur.
On tape v_0 :
`2/3` on obtient 0.666666666667 puis,
`2*(Ans-1/3)` dans les différentes opérations on a toujours 12 décimales, on obtient :
2*(0.666666666667-0.333333333333)=2*0.333333333334=0.666666666668.
On a donc :
si $A := 0.666666666666$ et $B := 0.333333333333$, on a $A == 2 * B$ et $B == A - B$ mais, $2/3 = A + 10^-12$ et $1/3 = B$
On a donc :
$v_0 = 2/3 = A + 10^-12$ $v_1 = 2 * (A + 10^-12 - B) = 2 * (B + 10^-12) = A + 2 * 10^-12$ puis
$v_2 = 2 * (A + 2 * 10^-12 - B) = A + 2^2 * 10^-12$ puis...
$v_{38} = A + 2^{38} * 10^-12 = 0.94154457361$
$v_{39} = A + 2^{39} * 10^-12 = 1.21642248055$
$v_{40} = A + 2^{40} * 10^-12 = 1.76617829445$
...
$v_{50} = A + 2^{50} * 10^-12 = 1126.56657351$
$v_{51} = A + 2^{51} * 10^-12 = 2252.46648036$
puis la formule risque de n'être plus vraie à cause des erreurs d'arrondis près....
Si on utilise la commande `ITERATE` qui itère, en débutant par la valeur `2/3`, 90 fois la fonction qui à `X` fait correspondre `2*(X-1/3)`, on tape :
`ITERATE(2*(X-1/3),x,2/3,90)` on obtient :
`1.23794003934E15`
et
`ITERATE(2*(X-1/3),x,2/3,91)` on obtient :
`2.4758800788=2*1.23794003934E15`
Donc $v_n = 2^{n-90} * u_{90}$ et quand n tend vers l'infini $v_n = 2^{n-90} * u_{90}$ tend vers l'infini.

0.4 Pour reinitialiser et pour effacer

Pour reinitialiser la calculatrice :

- Enfoncer les touches F O C (sans être en mode `ALPHA`),
- Faire reset avec le trombone en laissant les touches enfoncées,
- Relacher les touches puis choisir 4 FLS Utility, puis 3 Format Disk C, puis Esc puis 9 Reset.

Pour effacer :

- le dernier caractère entré il faut faire `Del` (c'est la grosse flèche noire).

– la ligne d'entrée il faut faire `ON`.

– le dernier résultat ou la dernière commande de l'historique il faut faire `Shift-Del`

– tout l'historique il faut faire `Shift-Esc` (`Clear`).

0.5 L'écran tactile

On remarquera que les menus au bas de l'écran (nommé ici menus du bandeau) ne sont accessibles qu'en les touchant avec le doigt : il n'y a plus de touches `F1..F6` !
L'écran est tactile et cela permet de recopier tres facilement une ligne de commande ou une réponse de l'historique ou de lire ou relire une réponse trop longue, de sélectionner un menu puis une commande de la touche  .
Pour cela :

– il suffit de rechercher la commande ou la réponse à copier en naviguant dans l'historique avec un doigt, puis de sélectionner la commande ou la réponse à copier toujours avec un doigt et d'appuyer sur `Copier` du bandeau quand la ligne est en surbrillance ou on appuie 2 fois rapidement avec le doigt sur la ligne à recopier,
– pour lire une réponse trop longue il suffit de balayer la ligne de la réponse avec un doigt.
– on ouvre un menu avec un doigt ou avec son numéro, on fait de même si il y a un sous-menu, puis on sélectionne la fonction avec un doigt ou avec son numéro et cela provoque l'écriture de la fonction dans la ligne de commande à gauche : il ne reste plus qu'à taper les paramètres de cette fonction et de valider avec `Enter`. Le résultat s'écrit alors à droite.

0.6 Le touches

– `CAS`
Il faut appuyer sur la touche `CAS` pour faire du calcul formel. Les lettres minuscules sont alors accessibles en mode `ALPHA` et la touche `xtθn` permet d'avoir `x` directement.

– 
Il faut appuyer sur la touche  pour quitter le calcul formel et faire du calcul numérique.

– `Apps`
Il faut appuyer sur la touche `Apps` pour utiliser les différentes Applications qui ont chacune 3 vues : une vue symbolique qui contient les commandes demandées (touche `Symb`), une vue graphique qui exécute les commandes graphiques (touche `Plot`) et une vue numérique pour les résultats numériques (touche `Num`).

- Menu
  La touche Menu renvoie un menu spécifique selon ce que l'on fait.
  Par exemple, depuis le CAS ou depuis  vous pouvez echanger des données entre l'écran CAS et l'écran  ,
  depuis l'écran Plot de l'application de géométrie vous pouvez changer la couleur des objets ou faire des figures pleines avec Options du bandeau ou en remplissant de couleur, dans la vue symbolique, le carré situé entre la case qui sert à cocher et le nom de l'objet (en touchant ce carré on ouvr la palette de couleurs).
- Help
  La touche Help donne de l'aide sur les différentes commandes qui se trouvent dans le menu Cmds du bandeau ou dans le menu de la touche  : il faut mettre cette commande en surbrillance avec les flèches puis appuyer sur Help ou bien on tape cette commande et on appuie sur Help.
- Esc
  La touche Esc permet d'annuler la commande en cours

0.7 La configuration générale

On ouvre l'écran de configuration générale avec Shift-  .
Vous pouvez par exemple choisir de :

- Taper les commandes en 2d (choisir Entry : Textbook),
- Avoir les réponses en 2d (cocher Textbook Display),
- que les menus contiennent le nom des commandes plutôt qu'un thème (cocher Menu Display),
- Mettre la calculatrice dans les conditions d'examen

0.8 La configuration du CAS

On tape : Shift - CAS (Settings).
Pour être en mode complexe il faut coché i.
Pour utiliser des variables complexes il faut coché Complex.
Par exemple :
solve(x^3+2*x^2+x+2=0,x) renvoie [-2] en mode réel
solve(x^3+2*x^2+x+2=0,x) renvoie [-2,-i,i] en mode complexe
Pour utiliser des racines carrées dans une factoristion il faut cocher :
Utiliser $\sqrt{\ }$
Par exemple :
factor(x^2+x-1) renvoie x^2+x-1 si Utiliser $\sqrt{\ }$ est décoché,
factor(x^2+x-1) renvoie (x+(-(sqrt(5))+1)/2)*(x+(sqrt(5)+1)/2)
si Utiliser $\sqrt{\ }$ est coché.

0.9 La configuration de la calculatrice Shift-[⌂]

Les touches [Shift]-[⌂](Settings) permettent de faire les réglages de la calculatrice.
Pour avoir dans les menus ou les sous-menus les noms des commandes Menu Display ne doit pas être coché.
Si Menu Display est coché les menus et les sous-menus décrivent les commandes et renvoie la commande quand un menu ou un sous-menu est sélectionné.

0.10 Les fonctions de calcul formel

On a accès aux fonctions de calcul formel en appyant sur la touche [⌨].
Ces fonctions sont classées par thème.
Utiliser [Shift]-[⌂](Settings) et décocher Menu Display pour avoir le nom des fonctions et non la déscription de ces fonctions.

Deuxième partie

Le menu CAS de la touche 

Chapitre 1

Généralités

1.1 Les calculs dans le CAS

Avec le CAS, on fait du calcul exact.
Avec le CAS on peut utiliser les variables de Home qui ont comme noms une des lettres majuscules et qui ont par défaut la valeur 0 mais aussi des variables qui ont comme noms une chaine de lettres minuscules ou de chiffres commencant par une lettre. Ces variables n'ont par défaut aucune valeur : ces variables sont symboliques (sans valeur) tant qu'on ne leur à pas fait une affectation.
Dans le CAS, les commandes sont en général en minuscules, c'est pourquoi la touche ALPHA permet de taper une minuscule et ALPHA,ALPHA bloque le clavier en minuscules (pas besoin de taper SHIFT).
Avec le CAS, les simplifications ne se font pas automatiquement, seules les parenthèses inutiles sont enlevées et les fractions sont simplifiées. Pour avoir la forme simplifiée d'une expression, il faut utiliser la commande simplify.
On remarquera que la réponse peut se faire dans un éditeur d'équations.

1.2 La priorité des opérateurs

Les 4 opérations suivantes sont des opérateurs infixés.
+ désigne l'addition,
− désigne la sostraction,
⋆ désigne l'multiplication,
/ désigne la division.
L'élévation à la puissance est obtenu avec la touche x^y et s'écrit dans l'historique avec ^.
Pour faire les calculs :

- On effectue les calculs mis entre les parenthèses,
- On effectue les puissances,
- On effectue les multiplications et les divitions dans l'ordre de gauche à droite.
- On effectue les additions et les soustractions dans l'ordre de gauche à droite.

1.3 La multiplication implicite

Dans le CAS, pour faire une multiplication, le signe * peut être omis lorsque l'on fait la multiplication d'un nombre par une variable. On a le droit d'écrire 2x mais il faut écrire a*b pour faire le produit de la variable apat b, puisque ab est aussi un nom de variable.
On peut écrire par exemple :
2x+3i+4pi
On ne peut pas écrire :
(2)x, (2)(x+y), (2x+3)(x+y)
il faut écrire :
2x ou 2*(x+y) ou (2x+3)*(x+y)
Attention x2 et xy désignent le nom d'une variable et f(x+1) est la valeur de la fonction f en x+1.

1.4 Le temps de calcul :time

L'évaluation du temps d'un long calcul est écrit en bleu.
Cette 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 le temps chronomèté en secondes.
On tape :

```
time(factor(x^10-1))
```

On obtient en mode réel :

```
0.0045
```

On tape :

```
time(factor(x^100-1))
```

On obtient en mode réel :

```
0.0092
```

On tape :

```
time(factor(x^10-1))
```

On obtient en mode complexe (cocher i dans le CAS Settings) :

```
0.272
```

On tape :

```
time(factor(x^100-1))
```

On obtient en mode complexe :

```
29.794
```

1.5 Les listes et les séquences dans le CAS

Avec le CAS, les listes (resp les vecteurs) sont parenthésées par { } ou par [ ] et les indices sont mis entre des crochets ou entre des parenthèses.
Tous indices commencent à 1.
Par exemple,on tape :
`l:=[1,2,3,4];`
`ll:={1,2,3,4};` `l[2]` ou `ll[2]` renvoie 2
`l(2)` ou `ll(2)` renvoie 2

Avec le CAS, il existe aussi le type séquence qui est une suite d'objets. Les indices d'une séquence commencent aussi à 1.
Par exemple,on tape :
`s:=1,2,3,4`
`s[2]` ou `s(2)` renvoie 2.
Avec ce type séquencela concaténation est aisée.
On tape pour définir la séquence vide :
`s:=NULL;`
On obtient si on n'a pas coché `Textbook` ou `Livre` dans la configuration générale (`Shift HOME`) :
`NULL`
Puis, on tape :
`s:=s,1,2`
On obtient :
`seq[1,2]`
On tape :
`s[1])`
On obtient :
`2`
Alors qu'avec le type liste, on tape pour définir la liste vide :
`l:=[];`
Puis, on tape :
`l:=concat(l,[1,2])`
On obtient :
`[1,2]`
On tape :
`l[1])`
On obtient :
`2`
Pour transformer une liste en séquence on utilise l'opérateur `op`.
On tape :
`op(l)`
On obtient :
`seq[1,2]`
Pour transformer une séquence en liste, il suffit de parenthéser la séquence par `[]`.
On tape :
`[s])`
On obtient :
`[1,2]`

1.6 Différence entre expressions et fonctions

Il faut bien faire la distinction entre expression et fonction.
Une **expression** est une suite de termes séparés par un signe d'opération.
Un terme est un nombre ou un nom de variable ou un produit ou une parenthése contenant une expression.
Convention La multiplication et la division sont prioritaires sur l'addition et la soustraction.
Le signe $*$ est quelqufois omis dans l'écriture, par exemple on écrit : $2x$ au lieu de $2 * x$.
Une **fonction** rèelle f définie sur I partie de $\mathbb{R}$ est une application qui à tout nombre de x de I fait correspondre une expression $f(x)$. La valeur de la fonction en un point x est donc donnée par une expression.
Exemple avec `HPprime` Je tape dans le CAS :
`xpr:=3*x+2`
je définis ainsi l'expression `xpr`
Je tape :
`f(x):=3*x+2`
je définis ainsi la fonction `f`
Je tape :
`subst(xpr,x=1)` et j'obtiens 5
Je tape :
`f(1)` et j'obtiens 5
Je tape :
`plotfunc(3*x+2)` ou,
`plotfunc(xpr)` ou,
`plotfunc(f(x))`
j'obtiens un seul graphe qui est le graphe de la fonction `f`.
Remarque
Le tracé de la plupart des commandes commencant par `plot` ne se fait pas bien depuis l'écran CAS : il est donc préférable d'utiliser l'application de géométrie pour faire les graphes correspondant à ces commandes.

1.6.1 Définir une fonction par une expression

On tape pour définir $f(x) = x\sin(x)$:
`f(x):=x*sin(x)` On tape :
`f(1)`
On obtient :
`sin(1)`
Mais `attention` si on tape :
`xpr:=x*sin(x)`
Puis :
`g(x):=xpr`
cela n'est pas correct car la variable `x` ne figure pas dans `xpr`.
Il faut taper :
`g:=unapply(xpr,x)`
On tape :

```
g(1)
```
On obtient :
```
sin(1)
```
La commande `unapply` renvoie une fonction qui est définie à partir d'une expression et d'une variable : par exemple ici `unapply(xpr,x)` désigne la fonction : $x- > x * sin(x)$.

1.6.2 Définition d'une fonction de une ou plusieurs variables

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)]
```

1.6.3 Définir une fonction par 2 expressions : `when`

On tape : `g(x):=when(x>0,x,-x)`
`g(-2)` renvoie 2
`g(-2)` renvoie 2

1.6.4 Définir une fonction par n valeurs : `PIECEWISE piecewise`

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))
```

1.6.5 Exercice sur les expressions

Voici 6 expressions formées à partir de $T = 1 - x * 2 + x$ en rajoutant des parenthèses :

$$A = (1 - x) * 2 + x$$
$$B = 1 - (x * 2) + x$$
$$C = 1 - x * (2 + x)$$
$$D = (1 - x * 2) + x$$
$$F = 1 - (x * 2 + x)$$
$$G = (1 - x) * (2 + x)$$

1/ Y-a-t-il une (ou des) expression(s) égale à T ?
Si oui, pourquoi ?
2/ Calculer les valeurs de ces expressions pour $x = 1$ et pour $x = -1$.
3/ Parmi les expressions A, B, C, D, F, G :
- Lesquelles sont une somme de 2 termes ?
- Lesquelles sont une différence de 2 termes ?
- Lesquelles sont une somme algébrique de 3 termes ?
- Lesquelles sont un produit de 2 termes ?
- Lesquelles sont égales ?

4/ Simplifier les expressions A, B, C, D, F, G.
5/ Écrire toutes les expressions formées à partir de $S = 1 + x/2 * x$ en rajoutant des parenthèses. **Vérifions avec** `HPprime` On tape :

```
T:=1-x*2+x
A:=(1-x)*2+x
B:=1-(x*2)+x
C:=1-x*(2+x)
```

`D:=(1-x*2)+x`
`F:=1-(x*2+x)`
`G:=(1-x)*(2+x)`
Puis on tape pour connaitre les expressions égales à `T` :
`A==T`, `B==T`, etc...
On trouve que la réponse de `A==T` est `0` ce qui veut dire que l'expression `A` est différente de `T`.
On trouve que la réponse de `B==T` est `1` ce qui veut dire que l'expression `B` est identique à `T` etc...

1.6.6 Exercice (suite) sur les fonctions

1/ Définir 6 fonctions ayant pour valeurs respectives les expressions A, B, C, D, F, G.
2/ Tracer les graphes de ces fonctions et observer les sur un même graphique.
3/ Parmi ces graphes il y a des droites et des paraboles. Retrouver le graphe de chaque fonction. **Vérifions avec** `HPprime` On tape pour définir les 6 fonctions :
`a(x):=(1-x)*2+x`
`b(x):=1-(x*2)+x`
`c(x):=1-x*(2+x)`
`d(x):=(1-x*2)+x`
`f(x):=1-(x*2+x)`
`g(x):=(1-x)*(2+x)`
Puis on tape pour visualiser les graphes :
`plotfunc([a(x),b(x),c(x),d(x),f(x),g(x)])`
On obtient que 5 courbes de couleurs différentes.
On peut taper progressivement :
`plotfunc([a(x)]),plotfunc([a(x),b(x)])` etc...
On voit ainsi que :

- le graphe de `a` est la droite noire,
- le graphe de `b` est une droite rouge,
- le graphe de `c` est la parabole verte,
- le graphe de `d` est la droite jaune qui se supperpose à la droite rouge,
- le graphe de `f` est la droite bleue et,
- le graphe de `g` est la parabole verte.

Chapitre 2

Le menu Algebra

2.1 Simplifier une expression : `simplify`

`simplify` simplifie une expression de façon automatique.
On tape :

```
simplify(x^5+1/((x-1)*4)+1/((x+1)*4)+1/((x+i)*4)+1/((x-i)*4))
```

On obtient :

```
(x^9-x^5+x^3)/(x^4-1)
```

On tape :

```
simplify(3-54*sqrt(1/162))
```

On obtient :

```
-3*sqrt(2)+3
```

Attention `simplify` est plus efficace lorsqu'on est en `radian` pour simplifier des expressions trigonométriques : pour cela on coche `radian` dans la configuration du `CAS`.
On tape :

```
simplify((sin(3*x)+sin(7*x))/sin(5*x))
```

On obtient :

```
4*(cos(x))^2-2
```

2.2 Factoriser un polynôme 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.
On tape :

```
collect(x^3-2*x^2+1)
```

On obtient :

```
(x-1)*(x^2-x-1)
```

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)
```

Voir aussi `factor` selon que dans la configuration du `CAS` on a coché $\sqrt{}$ ou pas.

2.3 Regrouper et simplifier : `regroup`

`regroup` a comme paramètre une expression.
`regroup` effectue les simplifications évidentes sur une expression en regroupant des termes.
On tape :

```
regroup(x+3*x+5*4/x)
```

On obtient :

```
20/x+4*x
```

2.4 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
```

2.5 Développer une expression : expand

expand effectue, sur une expression, la distributivité de la multiplication par rapport à l'addition.
On tape :

```
expand((x+1)*(x+2))
```

On obtient :

```
x^2+3*x+2
```

On tape :

```
expand((a+b)^5)
```

On obtient :

```
5*a^4*b+10*a^3*b^2+10*a^2*b^3+5*a*b^4+b^5+a^5
```

2.6 Multiplier par la quantité conjuguée : mult_conjugate

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)))
```

2.7 Factoriser une expression : `factor`

On tape :

```
factor(x^6-1)
```

On obtient en mode rèel :

```
(x-1)*(x+1)*(x^2-x+1)*(x^2+x+1)
```

On tape :

```
factor(x^6+1)
```

On obtient en mode rèel :

```
(x^2+1)*(x^4-x^2+1)
```

On obtient en mode complexe avec $\sqrt{}$ non coché :

```
(x+i)*(x-i)*(x^2+(i)*x-1)*(x^2+(-i)*x-1)
```

On obtient en mode complexe avec $\sqrt{}$ coché :

```
(x+i)*(x-i)*(x+(-(sqrt(3))-i)/2)*(x+(-(sqrt(3))+i)/2)*
(x+(sqrt(3)-i)/2)*(x+(sqrt(3)+i)/2)
```

On tape :

```
factor(x^6+1,sqrt(3))
```

On obtient en mode complexe avec $\sqrt{}$ coché ou pas :

```
(x+i)*(x-i)*(x+(-(sqrt(3))-i)/2)*(x+(-(sqrt(3))+i)/2)*
(x+(sqrt(3)-i)/2)*(x+(sqrt(3)+i)/2)
```

On tape :

```
factor(x^3-2*x^2+1)
```

On obtient si on a décoché $\sqrt{}$ dans la configuration du `CAS` :

```
(x-1)*(x^2-x-1)
```

On tape :

```
factor(x^3-2*x^2+1)
```

On obtient si on a coché $\sqrt{}$ dans la configuration du CAS :

```
(x+(-(sqrt(5))-1)/2)*(x-1)*(x+(sqrt(5)-1)/2)
```

On tape :

```
factor(expexpand(exp(5*x))-exp(x))
```

On obtient en mode complexe :

```
exp(x)*(-1+exp(x))*(1+exp(x))*(i+exp(x))*(-i+exp(x))
```

2.8 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
```

2.9 Factorisation dans $\mathbb{C}$: cFactor cfactor

cFactor ou cfactor 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)
```

2.10 Substituer une variable par une valeur : `subst`

`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=3)
```

On obtient :

```
10
```

On tape :

```
a:=2;subst(a^2+1,a=3)
```

On obtient :

```
(2,5)
```

On tape :

```
a:=2;purge(a);subst(a^2+1,a=3)
```

On obtient :

```
(2,2,10)
```

2.11 Les fractions

2.11.1 Décomposer en éléments simples : `partfrac`

`partfrac` a comme argument une fraction rationnelle.
`partfrac` renvoie sa décomposition en éléments simples.
On tape :

```
partfrac(x^5+x^3/(x^4-1))
```

On obtient :

```
x^5+1/((x-1)*4)+1/((x+1)*4)+x/((x^2+1)*2)
```

On tape :

```
partfrac(x^5+x^3/(x^4-1))
```

On obtient :

```
x^5+1/((x-1)*4)+1/((x+1)*4)+1/((x+i)*4)+1/((x-i)*4)
```

2.11.2 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))
```

2.11.3 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)
```

2.11.4 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)
```

2.12 Extract

2.12.1 Numérateur d'une fraction après simplification : `numer`

`numer` a comme argument une fraction ou une fraction rationnelle et renvoie le numérateur de cette fraction simplifiée.
On tape :

```
numer(42/12)
```

On obtient :

```
7
```

On tape :

```
numer(x^5+x^3/(x^4-1))
```

On obtient :

```
x^9-x^5+x^3
```

2.12.2 Dénominateur d'une fraction après simplification : `denom`

`denom` a comme argument une fraction ou une fraction rationnelle et renvoie le dénominateur de cette fraction simplifiée.
On tape :

```
denom(42/12)
```

On obtient :

```
2
```

On tape :

```
denom(x^5+1/((x-1)*4)+1/((x+1)*4)+x/((x^2+1)*2))
```

On obtient :

```
x^4-1
```

2.12.3 Numérateur et dénominateur : `f2nd`

`f2nd` a comme argument une fraction ou une fraction rationnelle et renvoie la liste formée par le numérateur et le dénominateur de cette fraction simplifiée.
On tape :

```
f2nd(42/12)
```

On obtient :

```
[7,2]
```

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]
```

2.12.4 Pour avoir le membre de gauche d'une équation : `left`

`left` a comme paramètre une équation ou un intervalle.
`left` renvoie le membre de gauche de l'équation ou la borne gauche de l'intervalle.
On tape :

```
left(a=3)
```

On obtient :

```
a
```

On tape :

```
left(a..2*a+1)
```

On obtient :

```
a
```

2.12.5 Pour avoir le membre de droite d'une équation : `right`

`right` a comme paramètre une équation ou un intervalle.
`right` renvoie le membre de droite de l'équation ou la borne droite de l'intervalle.
On tape :

```
right(a=3)
```

On obtient :

```
3
```

On tape :

```
right(a..2*a+1)
```

On obtient :

```
2*a+1
```

2.12.6 Centre d'un intervalle : `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]
```

2.12.7 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(3,4,5,2,1)
```

On obtient :

```
-1
```

En effet cette permutation se décompse en les cycles :
(1,3,5) et (2,4) c'est à dire en 3 transpositions :
(1,3), (3,5) et (2,4).

Chapitre 3

Le menu Calculus

3.1 Définition d'une fonction : := et -> (Sto▶)

Pour définir par exemple la fonction `f` qui à `x` fait correspondre `x^3+ln(x)`, on tape :
`f:=x-> x^3+ln(x)`
ou on tape :
`f(x):= x^3+ln(x)`

3.2 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
```

3.3 Dérivation

3.3.1 Fonction dérivée d'une fonction : `function_diff`

`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
```

3.3.2 Dérivation : ∂ diff ' "

diff ou ' calcule la dérivée d'une expression ou d'une fonction d'une variable et calcule aussi les dérivées partielles d'une expression de plusieurs variables.
On peut aussi utiliser la touche portant la lettre C et utiliser :
$\frac{\partial\square}{\partial\square}$ et si on a choisit comme configuration une sortie en mode Livre (cf Settings de HOME),il suffit de remplir les $\square$.

- Dérivée d'une expression d'une variable
  On tape :
  ```
  diff(x^3+ln(x))
  ```
  Ou on tape (' est obtenu avec Shift-() (") en effaçant un ') :
  ```
  (x^3+ln(x))'
  ```
  On obtient l'expression de la derivèe de x^3+ln(x) par rapport à x :
  ```
  3*x^2+1/x
  ```
  On tape :
  ```
  diff(y^3+ln(y),y)
  ```
  Ou on tape (' est obtenu avec Shift-() (") en effaçant un ') :
  ```
  (y^3+ln(y),y)'
  ```
  On obtient l'expression de la derivèe de y^3+ln(y) par rapport à y :
  ```
  3*y^2+1/y
  ```
- Dérivée seconde (ou n-ième) d'une expression d'une variable
  On tape :
  ```
  diff(diff(x^3+ln(x)))
  ```
  Ou on tape (" est obtenu avec Shift-()) :
  ```
  (x^3+ln(x))"
  ```
  On obtient l'expression de la derivèe seconde de x^3+ln(x) par rapport à x :
  ```
  3*2*x-1/x^2
  ```
  On tape :
  ```
  diff(diff(diff(diff(x^3+ln(x)))))
  ```
  Ou on tape ("" est obtenu avec Shift-() Shift-()) :
  ```
  (x^3+ln(x))""
  ```
  On obtient l'expression de la derivèe 4-ième de x^3+ln(x) par rapport à x :
  ```
  -2*3/x^4
  ```
- Dérivées partielles d'une expression de plusieurs variables. On tape :
  ```
  diff(x*y*z,{x,y,z})
  ```
  On obtient l'expression les derivèes partielles par rapport à x, par rapport à y et par rapport à z,de x*y*z :
  ```
  {y*z,x*z,x*y}
  ```
- Dérivée d'une fonction
  On définit la fonction f, on tape :
  ```
  f(x):= x^3+ln(x)
  ```
  On obtient :
  ```
  (x)->x^3+ln(x)
  ```
  On tape :
  ```
  g:=diff(f)
  ```

Ou on tape (' est obtenu avec `Shift-()` (") en effaçant un ') :

```
g:=f'
```

On obtient la fonction `g` qui est la fonction dérivée de `f` :

```
x->3*x^2+1/x
```

– Dérivée seconde (ou n-ième) d'une fonction
On définit la fonction `f`, on tape :

```
f(x):= x^3+ln(x)
```

On obtient

```
(x)->x^3+ln(x)
```

On tape :

```
h:=diff(diff(f))
```

Ou on tape (" est obtenu avec `Shift-()`) :

```
h:=f"
```

On obtient la fonction `h` qui est la fonction dérivée seconde de `f` :

```
x->3*2*x-1/x^2
```

3.3.3 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
```

3.4 Intégration

3.4.1 Primitive : `int`

`int` permet de calculer une primitive d'une expression ou d'une fonction ou une intégrale définie.
On peut aussi utiliser la touche portant la lettre `C` et utiliser :
$\int_{\square}^{\square} \square\partial\square$ et si on a choisit comme configuration une sortie en mode `Livre` (cf Settings de HOME), il suffit de remplir les $\square$.

– Primitive d'une expression
On tape :

```
int(x^3+ln(x))
```

On obtient une primitive de `x^3+ln(x)` par rapport à `x` :

```
x*ln(x)-x+x^4/4
```

On tape :

```
int(y^3+ln(y),y)
```

On obtient une primitive de `y^3+ln(y)` par rapport à `y` :

```
y*ln(y)-y+y^4/4
```

– Primitive d'une fonction
On définit la fonction `f`, on tape :

```
f(x):= x^3+ln(x)
```

On obtient :

```
(x)->x^3+ln(x)
```

On tape :

```
g:=int(f)
```

On obtient la fonction `g` qui est une primitive de `f` :

```
(x)->x*ln(x)-x+x^4/4
```

– Intégrale définie
On tape :

```
int(x^3+ln(x),x,1,2)
```

Ou on tape :

```
int(x^3+ln(x),x=1..2)
```

Ou on tape :

```
int(y^3+ln(y),y=1..2)
```

Ou on tape lorsque `f(x):= x^3+ln(x)` et `g:=int(f)` :

```
preval(g(x),1,2)
```

On obtient la valeur de $\int_1^2$ `x^3+ln(x)dx` :

```
2*ln(2)-(-3/4
```

3.4.2 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
```

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

`romberg` 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)` 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
```

3.5 Les limites : `limit`

`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 !).
Lorsqu'on utilise `limit` à partir des menus et si on a choisit comme configuration une sortie en mode `Livre` (cf Settings de HOME) il s'affiche dans la ligne de commande : $\lim_{\square->\square^{\square}}(\square)$ et il suffit de remplir les $\square$.
Par exemple on obtient limit dans le menu CAS ->Analyse->Limite :
$\lim_{x->0^{1}}$`abs(x)/x`
On obtient dans l'historique :

```
limit(abs(x)/x,x,0,1)
```

et comme réponse :

```
1
```

On tape :

```
limit(sin(x)+ln(x))/x,x,1
```

On obtient :

```
sin(1)
```

On tape :

```
limit(1/x,x,0)
```

On obtient :

```
infinity
```

cela veut dire que `abs(1/x)` tend vers $+\infty$ quand x tend vers 0.
On tape :

```
limit(1/x,x,0,1)
```

On obtient :

```
+infinity
```

On tape :

```
limit(1/x,x,0,-1)
```

On obtient :

```
-infinity
```

Remarque
si on tape `limit((-1)^n,n=inf)`, alors le CAS 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(sin(x),x,inf)
```

On obtient :

```
bounded_function(2)
```

On tape :

```
limit(cos(x),x,inf)
```

On obtient :

```
bounded_function(7)
```

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)
```

3.6 Limite et intégrale

On donne ici quelques exemples :
– Déterminer la limite quand a tend vers l'infini de :

$$\int_2^a \frac{1}{x^2}\,dx$$

On tape :

```
limit(int(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(int(x/(x^2-1)+ln((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.

3.7 Les series : `series`

`series` permet de faire le développement limité d'une expression de la variable `Var` en `Var=0` (par défaut en `x=0`) à un ordre donné (par défaut 5).
On tape :

```
series(tan(x))
```

On obtient :

```
x+1/3*x^3+2/15*x^5+x^6*order_size(x)
```

`order_size` désigne une fonction telle que, quelque soit `r` positif :
`x^r*order_size(x)` tend vers zéro quand `x` tend vers zéro.
Donc lorsqu'on a dans la réponse `(x-a)^n*order_size(x-a)`
cela signifie que l'on a un développement limité à l'ordre $n-1$ au voisinage de `x=a`.
On tape :

```
series(tan(x),x=0,9)
```

On obtient :

```
x+1/3*x^3+2/15*x^5+17/315*x^7+62/2835*x^9+x^10*order_size(x)
```

On tape :

```
series(atan(x),x=+infinity,5)
```

On obtient :

```
1/2*pi-1/x+1/3*(1/x)^3-1/5*(1/x)^5+
(1/x)^6*order_size(1/x)
```

ici `(1/x)^6*order_size(1/x)` signifie que l'on a un développement limité à l'ordre $6-1=5$ au voisinage de `1/x=0` i.e. au voisinage de $+\infty$
On tape :

```
series(atan(x),x=-infinity,5)
```

On obtient :

```
-1/2*pi-1/x-1/3*(-1/x)^3+1/5*(-1/x)^5+
(-1/x)^6*order_size(-1/x)
```

ici `(-1/x)^6*order_size(-1/x)` signifie que l'on a un développement limité à l'ordre $6-1=5$ au voisinage de `-1/x=0` i.e. au voisinage de $-\infty$

3.8 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
```

3.9 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.

3.10 Somme indicée finie et infinie et primitive discrète : sum

`sum` effectue la somme indicée finie et infinie ou calcule la primitive discrète d'une expression.
`sum` effectue aussi la somme des éléments d'une liste voir 16.8.

- somme d'une liste ou d'une séquence
  On tape :
  ```
  l:=[1,2,3,4,5,6,7,8]
  ```
  Ou on tape :
  ```
  l:=1,2,3,4,5,6,7,8
  ```
  Puis, on tape :
  ```
  sum(l)
  ```
  On obtient la somme 1+2+..+8=8*9/2 :
  ```
  36
  ```
- somme indicée finie
  On tape :
  ```
  sum(k,k=1..8)
  ```
  Ou on tape :
  ```
  sum(k,k,1,8)
  ```
  On obtient :
  ```
  36
  ```
- somme indicée infinie
  On tape :
  ```
  sum(1/2^k,k,0,inf)
  ```
  On obtient :
  ```
  2
  ```
- primitive discréte d'une expression
  La primitive discrète de l'expression $f(x)$ est la fonction G verifiant :
  $G(x+1)-G(x)=f(x)$

`sum` a alors deux arguments : une expression d'une variable (par exemple $f(x)$) et la variable (par exemple x) :
On tape :

```
sum(x,x)
```

On obtient :

```
(x^2-x)/2
```

Donc :
$4+5+...19 = G(20) - G(4) = 190 - 6 = 184$
On vérifie : `sum(k,k=4..19)` renvoie bien `184`
On tape :

```
sum(1/(x*(x+1)),x)
```

On obtient :

```
-1/x
```

Donc :
$1/(1*2) + 1/(2*3) + ...1/(9*10) = -1/10 + 1 = 9/10$
On vérifie : `sum(seq(1/(k*(k+1)),k,1,9))` renvoie bien `9/10`

3.11 Differential

3.11.1 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.
On tape :

```
curl([x*z,-y^2,2*x^y],[x,y,z])
```

On obtient :

```
[2*ln(x)*x^y,x-2*y*x^(y-1),0]
```

En effet :
`diff(2*x^y,y)-diff(-y^2,z)` renvoie `2*ln(x)*x^y`
`diff(x*z,z)-diff(2*x^y,x)` renvoie `x-2*y*x^(y-1)`
`diff(-y^2,x)-diff(x*z,y)` renvoie `0`
On tape :

```
curl([x*y*z,-y^2,2*x],[x,y,z])
```

On obtient :

```
[0,x*y-2,-x*z]
```

3.11.2 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.
`divergence` désigne la divergence de F.
On tape :

```
divergence([x^2+y,x+z+y,z^3+x^2],[x,y,z])
```

On obtient :

```
2*x+3*z^2+1
```

En effet :
`diff(x^2+y,x)+diff(x+z+y,y)+diff(z^3+x^2,z)` renvoie :
`2*x+1+3*z^2`

3.11.3 Le gradient : `grad`

`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.
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]
```

En effet :
`diff(2*x^2*y-x*z^3,x)` renvoie `4*x*y-z^3`
`diff(2*x^2*y-x*z^3,y)` renvoie `2*x^2`
`diff(2*x^2*y-x*z^3,z)` renvoie `-3*x*z^2`

3.11.4 La 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 à savoir `diff(diff(F,[x,y,z]),[x,y,z])`.
On tape :

```
hessian(2*x^2*y-x*z,[x,y,z])
```

On obtient :

```
[[4*y,4*x,-1],[2*2*x,0,0],[-1,0,0]]
```

En effet :
`diff(diff(2*x^2*y-x*z,x),[x,y,z])` renvoie : `[4*y,4x,-1]`
`diff(diff(2*x^2*y-x*z,y),[x,y,z])` renvoie : `[4x,0,0]`
`diff(diff(2*x^2*y-x*z,z),[x,y,z])` renvoie : `[-1,0,0]`
Remarque
Pour avoir la hessienne aux points critiques, on cherche les points critiques, on tape :

```
solve(diff(2*x^2*y-x*z^3,[x,y,z]),[x,y,z])
```

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]]
```

3.11.5 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
```

3.11.6 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
```

3.11.7 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, la calculatrice 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]
```

3.12 Integral

3.12.1 Primitive et intégrale définie : `integrate`

`integrate` (ou `int`) permettent de calculer une primitive ou une intégrale définie.
On tape :

```
int(exp(x),x,0,1))
```

ou

```
integrate(exp(x),x,0,1)
```

On obtient :

```
exp(1)-1)
```

On tape :

```
evalf(integrate(exp(x^2),x,0,1))
```

Ou on tape :

```
evalf(int(exp(x^2),x,0,1))
```

On obtient :

```
1.46265174591
```

`integrate` (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
  ```

- 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
Dans le CAS `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
```

3.12.2 Intégration par parties : `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.

Valeur renvoyée par `ibpdv` selon ses paramètres :

- `ibpdv(u(x).v'(x),v(x))` (resp `ibpdv([F(x),u(x).v'(x)],v(x))`) renvoie :
  si $v(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),

- `ibpdv(g(x),0)` (resp `ibpdv([F(x),g(x)],0)`) renvoie :
  une primitive $G(x)$ de $g(x)$ (resp $F(x)+G(x)$) où `diff(G(x))=g(x)`.
- `ibpdv(u(x)*v'(x),v(x),x,a,b)` (resp
  `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 $-v(x).u'(x)$ (resp 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 $g(x)$ (resp `ibpdv([F(x),g(x)],0,x,a,b)` renvoie $F(x)+G(b)-G(a)$ où $G(x)$ est une primitive de $g(x)$).

On tape :

```
ibpdv(ln(x),x)
```

On obtient :

```
[x.ln(x),-1]
```

puis on tape

```
ibpdv([x.ln(x),-1],0)
```

On obtient :

```
-x+x.ln(x)
```

On tape :

```
ibpdv(ln(x),x,x,1,2)
```

On obtient :

```
[2*ln(2),-1]
```

On tape :

```
ibpdv(ln(x),x,x,2,3)
```

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis on tape :

```
ibpdv([3*ln(3)-2*ln(2),-1],0,x,2,3)
```

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

3.12.3 Intégration par parties : `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.

Valeur renvoyée par `ibpu` selon ses paramètres :

- `ibpu(u(x).v'(x),u(x))` (resp `ibpu([F(x),u(x).v'(x)],u(x))`) renvoie :
  si $u(x) \neq 0$, une liste formée de $u(x).v(x)$ et de $-v(x).u'(x)$ (resp une liste formée de $F(x) + u(x).v(x)$ et de $-v(x).u'(x)$),
- `ibpu(g(x),0)` (resp `ibpu([F(x),g(x)],0)`) renvoie :
  $G(x)$ une primitive de $g(x)$ (resp $F(x) + G(x)$ où `diff(G(x))=g(x)`).
- `ibpu(u(x)*v'(x),u(x),x,a,b)` (resp `ibpu([F(x),u(x)*v'(x)],u(x),x,a,b)`) renvoie :
  - si $u(x) \neq 0$, une liste formée de $u(b).v(b) - u(a).v(a)$ et de $-v(x).u'(x)$ (resp 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 $g(x)$ (resp $F(x) + G(b) - G(a)$ où $G(x)$ est une primitive de $g(x)$).

On tape :

```
ibpu(ln(x),ln(x))
```

On obtient :

```
[x.ln(x),-1]
```

puis on tape :

```
ibpu([x.ln(x),-1],0)
```

On obtient :

```
-x+x.ln(x)
```

On tape :

```
ibpu(ln(x),ln(x),x,2,3)
```

On obtient :

```
[3*ln(3)-2*ln(2),-1]
```

puis on tape :

```
ibpu([3*ln(3)-2*ln(2),-1],0,x,2,3)
```

On obtient :

```
-1+3*ln(3)-2*ln(2)
```

3.12.4 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
```

On tape :

```
int(ln(x))
```

On obtient :

```
x*ln(x)-x
```

On tape :

```
preval(x*ln(x)-x,2,3)
```

On obtient :

```
3*ln(3)-3-2*ln(2)+2
```

3.13 Limits

3.13.1 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"`.
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 :

$$\int_0^1 x^2 dx = \frac{1}{3}$$

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 :

$$\int_0^1 x^3 dx = \frac{1}{4}$$

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 $\ln(3) - \ln(2) + \ln(2) - \ln(1) + 2 \text{ atan } (1/2) = \ln(3) + 2 \text{ atan } (1/2)$
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 :

```
ln(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 :

$$\int_0^1 \frac{1}{1+x} dx = \ln(1+1) = \ln(2)$$

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 :

$$\int_0^1 \frac{1}{1+x^2} dx = \text{ atan}\,(1) = \frac{\pi}{4}$$

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 :

$$\int_0^1 \frac{1}{\sqrt{1+x^2}} dx = \ln(1+\sqrt{1+1^2} - \ln(0+\sqrt{1+0^2} = \ln(1+\sqrt{2}$$

3.13.2 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 la calculatrice 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)-(sin(1)/2)*(x-1)^2+(x-1)^3*order_size(x-1)
```

3.13.3 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 $\dfrac{A(x)}{B(x)}$ 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.

3.14 Transform

3.14.1 Transformée de Laplace : `laplace`

`laplace` a 1, 2 ou 3 arguments :
l'expression que l'on transforme et éventuellement le nom d'une ou deux 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 donnée comme deuxième argument.
`laplace` est la transformée de Laplace de l'expression donnée comme argument. Le résultat de `laplace` est une expression de variable le troisième argument ou par défaut le second argument ou par défaut x.
On tape :

```
laplace(sin(x))
```

On obtient :

```
1/(x^2+1)
```

Ou on tape :

```
laplace(sin(t),t)
```

On obtient :

```
1/(t^2+1)
```

Ou on tape :

```
laplace(sin(x),x,t)
```

On obtient :

```
1/(t^2+1)
```

Ou on tape :

```
laplace(sin(t),t,s)
```

On obtient :

```
1/(s^2+1)
```

3.14.2 Transformée de Laplace inverse : `invlaplace`

3.14.3 invlaplace

`ilaplace` a 1, 2 ou 3 arguments :
l'expression que l'on transforme et éventuellement le nom d'une ou deux 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 donnée comme deuxième argument.
`invlaplace` est la transformée de Laplace inverse de l'expression donnée comme argument. Le résultat de `invlaplace` est une expression de variable le troisième argument ou par défaut le second argument ou par défaut x.
On tape :

```
invlaplace(1/(x^2+1))
```

On obtient :

```
sin(x)
```

Ou on tape :

```
invlaplace(1/(t^2+1),t)
```

On obtient :

```
sin(t)
```

Ou on tape :

```
invlaplace(1/(t^2+1),t,x)
```

On obtient :

```
sin(x)
```

Remarque :
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 :

$$y'' + p.y' + q.y \ = \ f(x)$$

$$y(0) = a \ y'(0) = b$$

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`.
Exemple :
Résoudre :

$$y'' - 6.y' + 9.y \ = \ x.e^{3.x}, \quad y(0) = c\_0, \quad y'(0) = c\_1$$

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
```

3.14.4 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 :
$\mathtt{fft}([\mathtt{a}_0,..\mathtt{a}_{N-1}])[\mathtt{k}] = \mathtt{b_k} = \sum_{\mathtt{j}=0}^{\mathtt{N}-1} \mathtt{x_j}\omega_{\mathtt{N}}^{-\mathtt{k}\cdot\mathtt{j}}$ avec ω_N racine N-ième de l'unité.
On tape :

```
fft(0,1,1,0)
```

On obtient :

```
[2., -1-i, 0., -1+i]
```

Remarque 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`.

3.14.5 L'inverse de la transformée de Fourier rapide : `ifft`

`ifft` a comme argument une liste ou une séquence $[\mathtt{b}_0,..\mathtt{b}_{N-1}]$ où `N` est une puissance de deux.
`ifft` renvoie la liste $[\mathtt{a}_0,..\mathtt{a}_{N-1}]$ tel que :
$\mathtt{fft}([\mathtt{a}_0,..\mathtt{a}_{N-1}]) = [\mathtt{b}_0,..\mathtt{b}_{N-1}]$.
On tape :

```
ifft([2,-1-i,0,-1+i])
```

Ou on tape :

```
ifft(2,-1-i,0,-1+i)
```

On obtient :

```
[0., 1., 1., 0.]
```

Remarque 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 `ifft`.

Chapitre 4

Le menu Solve

4.1 Résolution d'équations : `solve`

`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)]`. On tape :

```
solve(x^2-3*x+2=0)
```

On obtient :

```
{1,2}
```

On tape :

```
solve(x^4-1=0)
```

On obtient :

```
{-sqrt(2),sqrt(2)}
```

On tape :

```
solve([x+y=3,x*y=2],[x,y])
```

Ou on tape :

```
solve({x+y=3,x*y=2},{x,y})
```

On obtient :

```
{[1,2],[2,1]}
```

On tape :

```
solve([-x^2+y=2,x^2+y=0],[x,y])
```

Ou on tape :

```
solve({-x^2+y=2,x^2+y=0},{x,y})
```

On obtient :

```
{}
```

4.2 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` ou si `i` n'est pas coché dans le Settings du CAS) les zéros seront réels et si on est en mode complexe (`complex_mode:=1` ou si `i` est coché dans le Settings du CAS)) les zéros seront complexes.
On tape :

```
zeros(x^2-3*x+2)
```

On obtient :

```
[2,1]
```

On tape :

```
zeros(x^4-1)
```

On obtient :

```
[1,-1]
```

On tape :

```
zeros([x+y-3,x*y-2],[x,y])
```

Ou on tape :

```
zeros({x+y-3,x*y-2},{x,y})
```

On obtient :

```
[[1,2],[2,1]]
```

On tape :

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

Ou on tape :

```
zeros({-x^2+y-2,x^2+y},{x,y})
```

On obtient :

```
[]
```

4.3 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]]
```

4.4 Résoudre des équations dans $\mathbb{C}$: cSolve csolve

cSolve ou csolve résout une équation ou un système d'équations polynômiales dans $\mathbb{C}$ sans avoir besoin d'être en mode complexe. **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 ou complexe : On tape :

```
cSolve(x^4-1=3)
```

ou

```
csolve(x^4-1=3)
```

On obtient :

```
[sqrt(2),-sqrt(2),sqrt(2)*i,-sqrt(2)*i]
```

On tape :

```
cSolve([-x^2+y=2,x^2+y=0],[x,y])
```

Ou on tape :

```
cSolve({-x^2+y=2,x^2+y=0},{x,y})
```

On obtient :

```
{[i,1],[-i,1]}
```

4.5 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^4-1)
```

On obtient :

```
[1,-1,-i,i]
```

On tape :

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

Ou on tape :

```
cZeros({-x^2+y-2,x^2+y},{x,y})
```

On obtient :

```
[[-i,1],[i,1]]
```

4.6 Équations différentielles

Pour le calcul numérique de solutions déquations différentielles on se reportera à odesolve et pour la représentation graphique de solutions déquations différentielles on se reportera à plotfield, plotode.

4.6.1 Résolution des équations différentielles : `deSolve desolve`

`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 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 ou
  l'équation différentielle suivie de la liste $[x_0, y_0]$ qui donne comme condition initiale $y(x_0) = y_0$.

- quand la variable est x,
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équation différentielle et les conditions initiales) et l'inconnue 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 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),
  les paramètres sont : l'équation différentielle (ou la liste formée par l'équation différentielle et les conditions initiales), la variable t et l'inconnue y ou l'inconnue $y(t)$ (la variable est alors t et l'inconnue est y).
  Dans l'équation différentielle y s'écrit $y(t)$ et $y\prime$ s'écrit `diff(y(t),t)`, y'' s'écrit `diff(y(t),t$2)`.
  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)` ;
  et

```
deSolve([diff(y(t),t$2)+2*diff(y(t),t)+y(t),
         y(0)=1,y'(0)=0],y(t)); ou
deSolve([diff(y(t),t$2)+2*diff(y(t),t)+y(t),
         y(0)=1,y'(0)=0],t,y);
```

On tape (en tapant `Shift-()` pour ") :

```
deSolve(y"+y=cos(x),y)
```

ou encore :

```
deSolve((diff(diff(y))+y)=(cos(x)),y)
```

On trouve :

```
G_0*cos(x)+(x+2*G_1)/2*sin(x)
```

`c_0, c_1` sont les constantes d'intégration : `y(0)=c_0` et `y'(0)=c_1`.
On tape, si veut les solutions vérifiant $y(0) = 1$:

```
deSolve([y''+y=cos(x),y(0)=1],y)
```

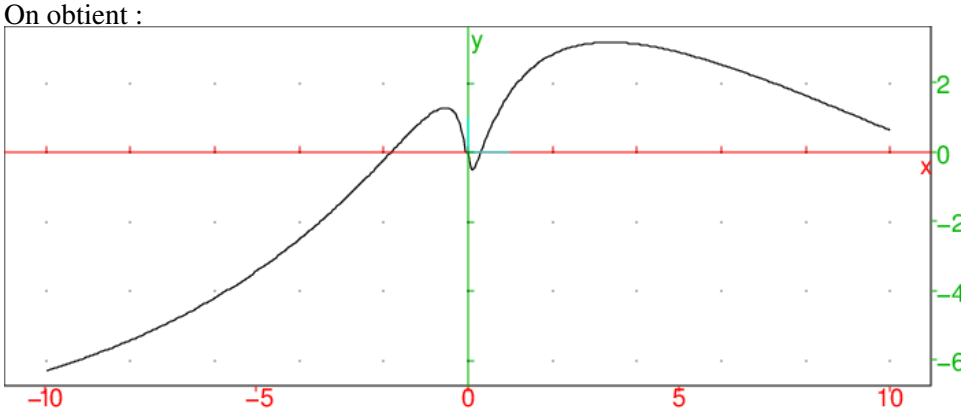
On obtient

```
[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`). **Exercice** Touver les fonctions f dérivables qui vérifient :
$f'(x) = f(-x)$.
La fonction `f'` est donc dérivable et on a :
$f''(x) = -f'(-x) = -f(x)$.
f vérifie donc l'équation différentielle $y'' + y = 0$ qui est facile à intégrer.
Donc f est solution de l'équation différentielle : $y'' + y = 0$.
On tape :
`desolve(y''+y=0)`
On obtient :
`c_0*cos(x)+c_1*sin(x)` Donc `f(x)` est de la forme `c_0*cos(x)+c_1*sin(x)`
Cherchons les valeurs de `c_0` et `c_1` pour avoir `f'(x)-f(-x)=0` pour toutes les valeurs de `x`.
On tape :
`f(x):=c_0*cos(x)+c_1*sin(x)`
`factor(f'(x)-f(-x))`
On obtient :
`(-sin(x)-cos(x))*(c_0-c_1)`
Donc `c_0=c_1`
Donc les fonctions f dérivables qui vérifient $f'(x) = f(-x)$ sont les fonctions égales à :
$c * (\cos(x) + \sin(x))$ où c est une constante arbitraire.
Ou encore $f(x) = k\cos(x - \pi/4)$ où k est une constante arbitraire.
On vérifie en tapant :
`(cos(x)+sin(x))'-(cos(-x)+sin(-x))`
ou en tapant `cos(x-pi/4)'-cos(-x-pi/4)` On obtient bien `0`
Exercice similaire Touver les fonctions f dérivables de $\mathbb{R}^+$ dans $\mathbb{R}$ qui vérifient :
$f'(x) = f(1/x)$.
La fonction `f'` de $\mathbb{R}^*$ dans $\mathbb{R}$ est donc dérivable et on a :
$f''(x) = -f'(1/x)/x^2 = -f(x)/x^2$
Donc f vérifie donc l'équation différentielle $x^2y'' + y = 0$ qui est plus diffficile à intégrer.
On tape :
`factor(desolve(x^2*y''+y=0))`
On obtient :
`sqrt(x)*(c_0*cos(2*sqrt(3)*ln(x)/4)+c_1*sin(2*sqrt(3)*ln(x)/4))`
Cherchons les valeurs de `c_0` et `c_1` pour que f vérifie :
`f'(x)-f(1/x)=0` pour toutes les valeurs de `x`.

On tape :
`f(x):=sqrt(x)*(c_0*cos(sqrt(3)*ln(x)/2)+c_1*sin(sqrt(3)*ln(x)/2))`
`factor(f'(x)-f(1/x))`
On obtient :
`sqrt(x)*(cos(sqrt(3)*ln(x)/2)+sqrt(3)*sin(sqrt(3)*ln(x)/2))*(-c_0-(-(sqrt(3))`
Donc `(-c_0-(-(sqrt(3)))*c_1)=0` c'est à dire `-c_0=c_1*sqrt(3)`.
Donc les fonctions f dérivables qui vérifient $f'(x) = f(1/x)$ sont les fonctions égales à :
$c * (\sqrt{3}\cos(x) + \sin(x))$ où c est une constante arbitraire.
On vérifie en tapant :
`f(x):=sqrt(x)*(sqrt(3)*cos(sqrt(3)*ln(x)/2)+sin(sqrt(3)*ln(x)/2))`
Puis,
`simplify(f'(x)-f(1/x))` renvoie bien `0`.
Pour faire l'intégration à la main on pose $t = \ln(x)$ i.e. $x = e^t$.
On a :
$y'_x = y'_t/x$ et $y''_x = y''_{t^2}/x^2 - y'_t/x^2 = 1/x^2(y''_{t^2} - y'_t)$
Donc $g(t) = f(e^t)$ vérifie l'équation différentielle :
$y''_{t^2} - y'_t + y = 0$ dont l'équation caractéristique est $r^2 - r + 1 = 0$. Donc $g(t) = f(e^t)$ est de la forme :
$e^{t/2}(a\cos(\sqrt{3}t/2) + b\sin(\sqrt{3}t/2))$
i.e. $f(x)$ est de la forme :
$f(x) = e^{\ln(x)/2}(a\cos(\sqrt{3}\ln(x)/2) + b\sin(\sqrt{3}\ln(x)/2))$
Cherchons les valeurs de a et b pour avoir :
$f'(x) = f(1/x)$.
$f'(x) = e^{\ln(x)/2}(a(-\sqrt{3}*\sin(\sqrt{3}\ln(x)/2)+\cos(\sqrt{3}\ln(x)/2))+b(\sin(\sqrt{3}\ln(x)/2)+$
$\sqrt{3}\cos(\sqrt{3}\ln(x)/2)))/(2x)$
et $f'(x) - f(1/x) = 0$ entraine :
$(-a + b\sqrt{3}) = 0$ donc $a = b\sqrt{3}$.
Si on cherche les fonctions f dérivables de $\mathbb{R}^*$ dans $\mathbb{R}$ qui vérifient :
$f'(x) = f(1/x)$.
Il faut poser pour $x < 0$: $x = -\exp(t)$.
On obtient la même équation différentielle mais la relation $y'(x) = y(1/x)$ donne comme condition $c_1 = -c_0\sqrt{3}$.
On pose donc :
`f(x):=exp(ln(abs(x))/2)*(sqrt(3)*cos(sqrt(3)/2*ln(abs(x)))+sin(sqrt(3)/2*ln(a`
`g(x):=exp(ln(abs(x))/2)*(cos(sqrt(3)/2*ln(abs(x)))-sqrt(3)*sin(sqrt(3)/2*ln(a`
Puis :
`h(x):=ifte(x>0,f(x),g(x))`
`k(x):=ifte(x>0,f'(x),g'(x))`
car si on tape `k(x):=h'(x)` on a en réponse :ifte : impossible de faire le test
Erreur : Valeur Argument Incorrecte.
ou bien on tape :
`h(x):=when(x>0,f(x),g(x))` et `k(x):=h'(x)` car `ifte` exécute le test, mais pas `when`, or l'expression est évaluée pour dériver.
Les fonctions `c*h(x)` où `c` est une constante arbitraire vérifient `c*h'(x)=c*k(x)=c*h(1/x)`
On tape :
`plotfunc(h(x))`

On obtient :



On remarquera que :
`limit(f(x),x,0,1)` renvoie `0` et
`limit(g(x),x,0,-1)` renvoie `0` donc
`h(0)=0` mais `h` n'est pas dérivable en `0` car `limit(h(x)/x,x,0)` vaut l'infini.
On tape :
`plotfunc([k(x),h(1/x)])`
On obtient une seule courbe ;

4.6.2 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 donnée 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 :

$$y'' + p.y' + q.y \ = \ f(x)$$

$$y(0) = a \ y'(0) = b$$

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 $y''(x) + p.y'(x) + q.y(x) = f(x)$:

$$\begin{aligned}\mathcal{L}(f)(x) &= \mathcal{L}(y'' + p.y' + q.y)(x)\\ &= -y'(0) - x.y(0) + x^2.\mathcal{L}(y)(x) - p.y(0) + p.x.\mathcal{L}(y)(x)) + q.\mathcal{L}(y)(x)\\ &= (x^2 + p.x + q).\mathcal{L}(y)(x) - y'(0) - (x + p).y(0)\end{aligned}$$

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 :

$$y'' - 6.y' + 9.y = x.e^{3.x}, \quad y(0) = c\_0, \quad y'(0) = c\_1$$

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
```

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

Soit f une fonction de $\mathbb{R}^2$ y'=f(t,y) 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.
- On peut indiquer en paramètre optionnel `curve` pour obtenir la liste des $[t, [y(t)]]$ calculés au lieu de la seule valeur de $y(t1)$.

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 :

```
[1.82241255674]
```

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]]
```

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]],[0.978
```

4.8 Transformée en z et transformée en z inverse

4.8.1 Transformée en z d'une suite : `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 $\frac{z}{(z-1)^2} = \sum_{n=1}^{inf} \frac{n}{z^n}$

4.8.2 Transformée en z inverse d'une fraction rationnelle : `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
```

4.9 Résolution numérique d'équationsnSolve

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

```
nSolve(x^2-2=0,x=1)
```

On obtient :

```
1.41421356237
```

On tape :

```
nSolvex^2-2=0,x=-1)
```

On obtient :

```
-1.41421356237
```

4.10 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 $a..b$.
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]
```

4.11 Les systèmes linéaires

4.11.1 Résoudre un système linéaire : `linsolve`

`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

$$x = -\frac{1}{2}, y = \frac{5}{2}, z = -\frac{1}{2}$$

est la solution du système :

$$\begin{cases} 2x + y + z &= 1 \\ x + y + 2z &= 1 \\ x + 2y + z &= 4 \end{cases}$$

4.11.2 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 $\mathbb{Z}/5\mathbb{Z}$:

$$\begin{cases} 3x + y &= -2 \\ 3x + 2y &= 2 \end{cases}$$

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.

4.12 Les formes quadratiques

4.12.1 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]]
```

4.12.2 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
```

4.12.3 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
```

4.12.4 Procédé de Gramschmidt : `gramschmidt`

`gramschmidt` a un ou deux paramètres :

- une matrice vue comme une liste de vecteurs lignes, le produit scalaire ètant 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)]
```

4.13 Les coniques

4.13.1 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.

4.13.2 Réduction d'une conique : `reduced_conic`

`reduced_conic` 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]`.
`reduced_conic` 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 :

```
reduced_conic(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 :

```
reduced_conic(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
```

Chapitre 5

Le menu Rewrite

5.1 Regrouper les logarithmes : `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)
```

5.2 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)
```

5.3 Linéariser les exponentielles : `lin`

`lin` 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
```

5.4 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
```

5.5 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()) :

5.6 Développer une expression transcendante et de trigo : texpand

texpand a comme argument une expression transcendante et trigonométrique.
texpand est la généralisation de expexpand, lnexpand et trigexpand car elle développe les expressions transcendantes et trigonométriques.
txpand 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 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 a comme argument une expression trigonométrique.
  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 simplify(Ans), on obtient :

```
4*(cos(x))^2-2
```

– `texpand` a comme argument une expression transcendante.
`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
```

5.7 Exp & Ln

5.7.1 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)`

5.7.2 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))
```

5.7.3 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))
```

ou

```
exp2trig(exp(i*x))
```

On obtient si `Complex` n'est pas coché dans la configuration du CAS (`Shift-CAS`) :

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

On obtient si `Complex` est coché dans la configuration du CAS :

```
exp(im(x))*(cos(re(x))+(i)*sin(re(x)))
```

On tape :

```
sincos(exp(i*x)+exp(-i*x))
```

ou

```
exp2trig(exp(i*x)+exp(-i*x))
```

On obtient :

```
cos(x)+i*sin(x)+cos(x)-i*sin(x)
```

puis on sélectionne cette réponse et on appuie sur `simplify` On obtient :

```
2*cos(x)
```

5.7.4 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
```

5.7.5 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
```

5.7.6 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)
```

5.8 Sinus

5.8.1 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
```

5.8.2 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))
```

5.8.3 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)
```

5.9 Cosinus

5.9.1 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
```

5.9.2 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))
```

5.9.3 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)
```

5.10 Tangente

5.10.1 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))
```

5.10.2 Transformer les arctan en arcsin : `atan2asin`

`atan2asin` a comme argument une expression trigonométrique.
`atan2asin` transforme cette expression en remplaçant :
$\arctan(x)$ par $\arcsin(\dfrac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2asin(atan(x))
```

On obtient :

```
asin(x/sqrt(1+x^2))
```

5.10.3 Transformer les arctan en arccos : `atan2acos`

`atan2acos` a comme argument une expression trigonométrique.
`atan2acos` transforme cette expression en remplaçant :
$\arctan(x)$ par $\dfrac{\pi}{2} - \arccos(\dfrac{x}{\sqrt{1+x^2}})$.
On tape :

```
atan2acos(atan(x))
```

On obtient :

```
pi/2-acos(x/sqrt(1+x^2))
```

5.10.4 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 $\frac{\sin(x)}{\cos(x)}$.
On tape :

```
tan2sincos(tan(2*x))
```

On obtient :

```
sin(2*x)/cos(2*x)
```

5.10.5 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)
```

5.11 Trigonométrie

5.11.1 Simplifier en privilégiant les sinus : `trigsin`

`trigsin` a comme argument une expression trigonométrique.
`trigsin` simplifie cette expression à l'aide des formules :
$\sin(x)^2 + \cos(x)^2 = 1$, $\tan(x) = \frac{\sin(x)}{\cos(x)}$ et en privilégiant les sinus.
On tape :

```
trigsin(cos(x)^2+1)
```

On obtient :

```
-sin(x)^2+2
```

5.11.2 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+2)
```

On obtient :

```
cos(x)^4-2*cos(x)^2+3
```

5.11.3 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
```

5.11.4 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 `simplify(Ans)` on a :

```
(2*tan(x)^4+3*tan(x)^2+2)/(tan(x)^4+2*tan(x))^2+1)
```

5.11.5 Linéariser une expression trigonométrique : `tlin`

`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)
```

5.11.6 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`

5.11.7 Rassembler les sinus et cosinus de même angle : `tcollect`

`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)
```

5.11.8 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)
```

5.11.9 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)
```

Chapitre 6

Le menu Integer

6.1 Test de parité : `even`

`even` a comme argument un entier `n`.
`even` renvoie `1` si `n` est pair et renvoie `0` si `n` est impair.
On tape :

```
even(148)
```

On obtient :

```
1
```

On tape :

```
even(149)
```

On obtient :

```
0
```

6.2 Test de non parité : `odd`

`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 obtient :

```
0
```

On tape :

```
odd(149)
```

On obtient :

```
1
```

6.3 Les diviseurs d'un nombre entier : `idivis`

`idivis` renvoie le vecteur de composantes les diviseurs d'un nombre entier.
On tape :

```
idivis(45)
```

On obtient :

```
[1,3,9,5,15,45]
```

6.4 Décomposition en facteurs premiers d'un entier : `ifactor`

`ifactor` renvoie la décomposition en facteurs premiers d'un nombre entier.
On tape :

```
ifactor(20!)
```

On obtient :

```
2^18*3^8*5^4*7^2*11*13*17*19
```

6.5 Liste des facteurs premiers et de leur multiplicité : `ifactors`

`ifactors` renvoie la liste des facteurs premiers d'un entier avec leur multiplicité.
On tape :

```
ifactors(45)
```

On obtient :

```
[3,2,5,1]
```

en effet $45 = 3^2 * 5^1$

6.6 PGCD de deux ou plusieurs entiers : `gcd`

`gcd` renvoie le PGCD de deux ou plusieurs entiers (voir 7.12 pour le PGCD de polynômes).
On tape :

```
gcd(45,10)
```

On obtient :

```
5
```

On tape :

```
gcd(40,12,16,24)
```

On obtient :

```
4
```

6.6.1 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
```

6.7 PPCM de deux ou plusieurs entiers : lcm

lcm renvoie le PPCM de deux ou plusieurs entiers.
On tape :

```
lcm(45,10)
```

On obtient :

```
90
```

On tape :

```
lcm(45,10,25,30)
```

On obtient :

```
450
```

6.7.1 Identité de Bézout : iegcd

iegcd(a,b) désigne le PGCD étendu (identité de Bézout) de deux entiers.
iegcd(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 :

$$2 \cdot 48 + (-3) \cdot 30 = 6$$

6.7.2 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]
```

6.8 Primalité

6.8.1 Test pour savoir si un nombre est premier : `isPrime isprime`

`isPrime(n)` ou `isprime` renvoie `true` si `n` est premier et sinon renvoie `false`.
On tape :

```
isPrime(1234567)
```

On obtient :

```
false
```

On tape :

```
isPrime(1234547)
```

On obtient :

```
true
```

6.8.2 Le N-ième nombre premier : `ithprime`

`ithprime(n)` renvoie le `n`-ième nombre premier.
On tape :

```
ithprime(10)
```

On obtient :

```
29
```

En effet, les 10 premiers nombres premiers sont : $2, 3, 5, 7, 11, 13, 17, 19, 23, 29$.
On tape :

```
ithprime(100)
```

On obtient :

```
541
```

541 est donc le 100-ième nombre premier.

6.8.3 `nextprime`

`nextprime(n)` renvoie le nombre premier `p` qui se trouve juste après `n` (`p>n`).
On tape :

```
nextprime(11)
```

On obtient :

```
13
```

On tape :

```
nextprime(1234567)
```

On obtient :

```
1234577
```

6.8.4 `prevprime`

`prevprime(n)` renvoie le nombre premier `p` qui se trouve juste avant `n` (`p<n`).
On tape :

```
prevprime(11)
```

On obtient :

```
7
```

On tape :

```
prevprime(1234567)
```

On obtient :

```
1234547
```

6.8.5 Indicatrice d'Euler : `euler`

`euler(n)` renvoie le cardinal de l'ensemble des nombres inférieurs à `n` qui sont premiers avec `n`.
`euler(n)` désigne donc l'indicatrice d'Euler de l'entier `n`.
On tape :

```
euler(18)
```

On obtient :

```
6
```

En effet l'ensemble :
E={5,7,11,13,15,17} correspond aux nombres inférieurs à 18 qui sont premiers avec 18, et E a comme cardinal 6. Avec la fonction `euler`, on a la généralistion du petit théorème de Fermat (qui dit que "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(5,6,18)
```

On obtient :

```
1
```

6.8.6 Symbole de Legendre : `legendre_symbol`

Lorsque n est premier, on définit le symbole de Legendre de a noté $\left(\frac{a}{n}\right)$ par :

$$\left(\frac{a}{n}\right) = \begin{cases} 0 & \text{si } a = 0 \mod n \\ 1 & \text{si } a \neq 0 \bmod n \text{ et si } a = b^2 \bmod n \\ -1 & \text{si } a \neq 0 \bmod n \text{ et si } a \neq b^2 \bmod n \end{cases}$$

Quelques propriétés

– Si n est premier :

$$a^{\frac{n-1}{2}} = \left(\frac{a}{n}\right) \bmod n$$

–

$$\begin{aligned}\left(\frac{p}{q}\right).\left(\frac{q}{p}\right) &= (-1)^{\frac{p-1}{2}}.(-1)^{\frac{q-1}{2}} \text{ si } p \text{ et } q \text{ sont impairs et positifs}\\ \left(\frac{2}{p}\right) &= (-1)^{\frac{p^2-1}{8}}\\ \left(\frac{-1}{p}\right) &= (-1)^{\frac{p-1}{2}}\end{aligned}$$

`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
```

6.8.7 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
```

6.8.8 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 $17 = 4^2 + 1^2$

6.9 Division

6.9.1 Quotient de la division euclidienne : `iquo`

`iquo(a,b)` renvoie le quotient de la division euclidienne de `a` par `b` lorsque `a` et `b` sont des entiers.
On tape :

```
iquo(45,10)
```

On obtient :

```
4
```

en effet $45 = 4 * 10 + 5$

6.9.2 Reste de la division euclidienne : `irem`

`irem(a,b)` renvoie le reste de la division euclidienne de `a` par `b` lorsque `a` et `b` sont des entiers.
On tape :

```
irem(45,10)
```

On obtient :

```
5
```

en effet $45 = 4 * 10 + 5$

6.9.3 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]
```

6.9.4 Restes chinois pour des entiers : `ichinrem`

`ichinrem([a,n],[b,p])` renvoie le vecteur `[c,lcm(p,q)]` formée de deux entiers.
Le premier nombre `c` est tel que

$$\forall k \in \mathbb{Z}, \quad d = c + k \times \text{lcm}(p, q)$$

l vérifie

$$d = a \pmod{p}, \quad d = b \pmod{q}$$

Si `n` et `p` sont premiers alors il y a toujours une solution et `q=n*p`
Exemple :
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 :

```
ichinrem({[3,5},{9,13})
```

On obtient :

```
[-17,65]
```

Donc les solutions sont `x=-17+k*65)` avec `k`$\in \mathbb{Z}$.
On a en effet $-17 = -5 * 4 + 3 = 3 \bmod 5$ et $-17 = -2 * 13 + 9 = 9 \bmod 13$
`ichinrem` renvoie Restes chinois pour des entiers.
On tape :

```
ichinrem({2,7},{3,5})
```

Ou on tape :

```
ichinrem([2,7],[3,5])
```

On obtient :

```
[-12,35]
```

On a bien $-12 + 35k = 2 \bmod 7$ et $-12 + 35k = 3 \bmod 5$ pour $k \in \mathbb{Z}$

6.9.5 Calcul de $a^n \mod p$: `powmod`

`powmod(a,n,p)` calcule a^n modulo p avec la méthode de la puissance rapide).
On tape :

```
powmod(5,21,13)
```

On obtient :

```
5
```

$5^2 = 25 = -1 \bmod 13$ donc $5^{21} = 5 * 5^{20} = 5 \bmod 13$
On tape :
`powmod(37,25,11)`
On obtient :
`1`
en effet $37 = 4 \bmod 11$ et $4^5 = 4^2 * 4^2 * 4 = 25 * 4 = 1 \bmod 11$

6.10 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 $\mathbb{Z}/p\mathbb{Z}[x]$.
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`.

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`.

6.10.1 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
```

6.10.2 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
```

6.10.3 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
```

6.10.4 Multiplication dans $\mathbb{Z}/p\mathbb{Z}$ ou $\mathbb{Z}/p\mathbb{Z}[x]$: ⋆

Pour réaliser une multiplication 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 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
```

6.10.5 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 $-3*4=-6*2=1 \mod 13$

6.10.6 Reste : `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 $-3*4=-6*2=1 \mod 13$

6.10.7 Quotient et reste : `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 **??** et 6.11.4).
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]
```

6.10.8 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)
```

6.10.9 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 :
$10 = -3 \pmod{13}$ $40 = 1 \pmod{13}$ $80 = 2 \pmod{13}$ $32 = 6 \pmod{13}$.

6.10.10 Calcul de $a^n \mod p$ ou de $A(x)^n \bmod ¶(x), p$: powmod

- 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 $A(x)^n \bmod ¶(x), p$ 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 6.10.9)
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
```

6.10.11 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 : $3 \times -4 = -12 = 1 \pmod{13}$

6.10.12 Transformer un entier en sa fraction modulo p : `fracmod`

`fracmod` 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 : $3 * -4 = -12 = 1 \pmod{13}$ donc $3 = -1/4 \% 13$. On tape :

```
fracmod(13,121)
```

On obtient :

```
-4/9
```

En effet : $13 \times -9 = -117 = 4 \pmod{121}$ donc $13 = -4/9 \% 13$.

6.10.13 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 7.12 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
```

6.10.14 Factorisation dans $\mathbb{Z}/p\mathbb{Z}[x]$: `factor`

`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)
```

6.10.15 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`).

6.10.16 Inverse d'une matrice de $\mathbb{Z}/p\mathbb{Z}$: `inv`

`inverse` (ou `inv`) a comme argument une matrice A à coefficients dans $\mathbb{Z}/p\mathbb{Z}$.
`inv` renvoie, l'inverse de la matrice A dans $\mathbb{Z}/p\mathbb{Z}$.
On tape :

```
inv([[1,2,9]% 13,[3,10,0]% 13,[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 $Z/13Z$.

6.10.17 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 **??**).
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 $A_1 x = B_1$ est équivalent à $Ax = B$.
Résoudre dans $Z/13Z$

$$\begin{cases} x + 2 \cdot y &= 9 \\ 3 \cdot x + 10 \cdot y &= 0 \end{cases}$$

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`.

6.10.18 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, la calculatrice 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)
```

6.10.19 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
```

6.11 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.

6.11.1 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)]]
```

6.11.2 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)
```

6.11.3 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
```

6.11.4 Quotient et reste euclidien : `quorem`

`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]
```

6.11.5 PGCD de polynômes par l'algorithme d'Euclide : `gcd igcd`

`gcd` ou `igcd` 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 6.6 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 ètè calculé dans $\mathbb{Z}[X]$.

6.11.6 Choisir l'algorithme du PGCD de deux polynômes : `ezgcd` `modgcd`

`ezgcd` et `modgcd` désignent 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,
`modgcd` est calculé avec l'algorithme modulaire,
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

```
modgcd(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

```
modgcd((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

```
modgcd((x+1)^4-y^4,(x+1-y)^2)
```

On obtient :

```
x-y+1
```

6.11.7 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 **??** 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
```

6.11.8 Idendité de Bézout : `egcd`

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]
```

6.11.9 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]]
```

6.11.10 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]
```

Chapitre 7

Le menu Polynomial

7.1 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
```

7.2 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 du polynôme.
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 :

```
[-1,0.5+0.866025403784*i,0.5-0.866025403784*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]
```

Pour chercher les racines numériques de $P(x) = x^3 - 5 * x^2 + 8 * x - 4$, on tape :

```
proot([1,-5,8,-4])
```

ou on tape :

```
proot(x^3-5x^2+8x-4)
```

On obtient :

```
[1.,2.,2.]
```

7.3 Racines exactes d'un polynôme

7.3.1 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 vecteurs de coordonnées la valeur des racines complexes et exactes du polynôme et leur multiplicité ou de coordonnées un intervalle (les bornes de l'intervalle sont les sommets opposés d'un rectangle de côtés parallèles aux axes et contenant 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 :

```
[[-1,1],[[(4-7*i)/8,(8-13*i)/16],1],[[(8+13*i)/16,(4+7*i)/8],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 :

```
[[-1,1],[[(8+13*i)/16,(4+7*i)/8],1]]
```

7.3.2 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]
```

7.4 Fraction rationnelle, ses racines et ses pôles exacts

7.4.1 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.
La calculatrice 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]
```

7.5 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 :
$x^2+2x+1=(x-2)^2+6(x-2)+9$

7.6 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 $\frac{1}{\alpha}$
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 :
$\frac{1}{\alpha} = -(\alpha)^3 - 10.\alpha$

– 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)
```

7.7 Coefficients d'un polynôme : `coeff`

`coeff` 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` renvoie le coefficient du polynôme de degré spécifié.
On tape :

```
coeff(x^3-5x^2+8x-4,2)
```

On obtient :

```
-5
```

On tape :

```
coeff(-x^4+3*x*y^2+x,y,2)
```

On obtient :

```
3*x
```

On tape :

```
coeff(-x^4+3*x*y^2+x,[x,y],[1,2])
```

On obtient :

```
3
```

7.8 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 $(x-1)(x-2)(x^2)(x-3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

7.9 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.

7.10 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(2t^2-2)
```

On obtient :

```
[1,2,t-1,2*(t-1),t+1,2*(t+1),(t-1)*(t+1),2*(t-1)*(t+1)]
```

On tape :

```
divis([t^2,x^2-1])
```

On obtient :

```
[[1,t,t^2],[1,x+1,x-1,(x-1)*(x+1)]]
```

7.11 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]]
```

7.12 PGCD de polynômes par l'algorithme d'Euclide : `gcd`

`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 6.6 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+2*x+1,x^2-1)
```

On obtient :

```
x+1
```

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]
```

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, car `%%` sert ici à désigner un nombre modulaire :

```
gcd((x^2+2*x+2) %% 5,(x^2-1) %% 5)
```

On obtient :

```
(1 %% 5)*x -1 %% 5
```

Mais si on tape :

```
gcd(x^2+2*x+2,x^2-1) %% 5
```

On obtient :

```
1 %% 5
```

car l'opération modulaire se fait après le calcul du PGCD qui a ètè calculé dans $\mathbb{Z}[X]$.

7.13 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 **??** 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
```

7.14 Créer

7.14.1 Transformer un polynôme en une liste (format interne récursif dense) : `symb2poly`

`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).
`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 :

```
symb2poly(x^2-1)
```

Ou on tape :

```
symb2poly(x^2-1,x)
```

Ou on tape :

```
symb2poly(y^2-1,y)
```

On obtient :

```
[1,0,-1]
```

On tape :

```
symb2poly(x*y^2+2y-1,x)
```

On obtient :

```
[y^2,2y-1]
```

On tape :

```
symb2poly(x*y^2+2y-1,y)
```

On obtient :

```
[x,2,-1]
```

7.14.2 Transformer le format interne creux distribué du polynôme en une écriture polynômiale : `poly2symb`

`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 $cx^{px}y^{py}z^{pz}$).
`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 :

```
poly2symb([1,0,-1])
```

Ou on tape :

```
poly2symb([1,0,-1],x)
```

On obtient :

```
x*x-1
```

On tape :

```
poly2symb([1,0,-1],y)
```

On obtient :

```
y*y-1
```

7.14.3 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 $(x-1)(x-2)(x^2)(x-3) = x^5 - 6x^4 + 11x^3 - 6x^2$.

7.14.4 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)^-1
```

7.14.5 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
```

7.14.6 Évaluation d'un polynôme : `polyEval`

`polyEval` a comme argument un polynôme `p` donné par la liste de ses coefficients et un réel `a`.
`polyEval` renvoie la valeur numérique ou exacte de `p(a)`.
On tape :

```
polyEval([1,0,-1],sqrt(2))
```

On obtient :

```
sqrt(2)*sqrt(2)-1
```

Puis :

```
normal(sqrt(2)*sqrt(2)-1)
```

On obtient :

```
1
```

On tape :

```
polyEval([1,0,-1],1.4)
```

On obtient :

```
0.96
```

7.14.7 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]
```

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

On obtient :

```
[1,-3,3,-1]
```

On tape :

```
pmin([[1,1,0],[0,1,1],[0,0,1]],x)
```

On obtient :

```
x^3-3*x^2+3*x-1
```

Donc le polynôme minimal de [[1,1,0],[0,1,1],[0,0,1]] est $x^3-3*x^2+3*x-1$.
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.

7.14.8 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]]
```

7.14.9 Polynômes aléatoires : `randpoly randPoly`

`randpoly` `randPoly`) a comme paramètre un entier `n`.
`randPoly` renvoie les coefficients d'un polynôme de degré `n` et dont les coefficients sont des entiers aléatoires équirépartis sur -99..+99.
On tape dans HOME ou dans CAS :

```
randPoly(4)
```

ou

```
randpoly(4)
```

On obtient par exemple :

```
[4,53,-45,80,-99)
```

7.14.10 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 !

7.15 Algebra

7.15.1 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)
```

7.15.2 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
```

7.15.3 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
```

7.15.4 Valuation d'un polynôme : `valuation`

`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
```

7.15.5 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
```

7.15.6 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)
```

7.15.7 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)
```

ou :

```
content([6,-3,9]))
```

On obtient :

```
3
```

7.15.8 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
```

7.15.9 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 :

7.15.10 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 $G = \text{gcd}(P, \text{diff}(P))$.
- 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`.

7.15.11 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`.

7.15.12 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
```

7.15.13 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 :

$$A = (ax; ay) = (2a\frac{1-t^2}{1+t^2}; 2a\frac{2t}{1+t^2})$$

On pose $F1F2 = 2c$ et on note I le milieu de $AF2$. On a :
$F2 = (2c, 0)$ et

$$I = (c + ax/2; ay/2) = (c + a\frac{1-t^2}{1+t^2}; a\frac{2t1-t^2}{1+t^2})$$

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 :
$A=(ax;ay)=(2a\frac{1-t^2}{1+t^2};2a\frac{2t}{1+t^2})$
On pose $F1F2=2c$ et on note I le milieu de $AF2$. On a :
$F2=(2c,0)$ et
$I=(c+ax/2;ay/2)=(c+a\frac{1-t^2}{1+t^2};a\frac{2t1-t^2}{1+t^2})$
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 :

$$\frac{X^2}{a^2}+\frac{Y^2}{b^2}=1$$

7.15.14 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]
```

7.16 Special

7.16.1 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

$$(x^{10} - 1) * (x^2 + 1) * (x^8 - x^6 + x^4 - x^2 + 1) = x^{20} - 1$$

7.16.2 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]
```

7.16.3 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`.

7.16.4 Polynôme de Hermite : `hermite`

`hermite` a comme argument un entier n et éventuellement 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
```

7.16.5 Interpolation de Lagrange : `lagrange`

`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 :

```
1/2*(x-1)
```

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 :

```
1/2*(y-1)
```

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 de `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`

7.16.6 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 :

$$\forall 0 \leq j \leq l - 1, \quad B^{(j)}(x_1) - A^{(j)}(x_1) = 0$$

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 :
$s|_{[a,x_2]} = a_0 + a_1x + ...a_lx^l + \alpha_1q_1(x)$.

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 :

$$\forall 0 \leq j \leq l - 1, \quad C^{(j)}(x_2) - B^{(j)}(x_2) = 0$$

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 : $s|_{[a,x_3]} = a_0 + a_1x + ...a_lx^l + \alpha_1q_1(x) + \alpha_2q_2(x)$
Et ainsi de suite, on définit les fonctions :

$$\forall 1 \leq j \leq n - 1, \mathrm{q}_j(x) = \begin{cases} 0 & \text{sur} \quad [a, x_j] \\ (x - x_j)^l & \text{sur} \quad [x_j, b] \end{cases}$$

ainsi,
$s|_{[a,b]} = a_0 + a_1 x + ...a_l x^l + \alpha_1 q_1(x) + ... + \alpha_{n-1} q_{n-1}(x)$ 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 :

$$[((3\pi^3+(-7\pi^2)x+4x^3)1/3)/(\pi^3),$$

$$((15\pi^3+(-46\pi^2)*+36\pi x^2-8x^3)1/12)/(\pi^3)]$$

7.16.7 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 :
$<f,g>=\int_0^{+\infty}f(x)g(x)x^ae^{-x}dx$
On tape :

```
laguerre(2)
```

On obtient :

```
1/2*a^2-a*x+3/2*a+1/2*x^2-2*x+1
```

On tape :

```
laguerre(2,y)
```

On obtient :

```
1/2*a^2-a*y+3/2*a+1/2*y^2-2*y+1)
```

On tape il faut que `b` soit purgé (`b:='b'`) :

```
laguerre(2,y,b)
```

On obtient :

```
1/2*b^2-b*y+3/2*b+1/2*y^2-2*y+1
```

7.16.8 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 :

```
35/8*x^4+-15/4*x^2+3/8
```

On tape :

```
legendre(4,y)
```

On obtient :

```
35/8*y^4+-15/4*y^2+3/8
```

7.16.9 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 :
$\cos(4.x)=Re((\cos(x)+i.\sin(x))^4)$
$\cos(4.x)=\cos(x)^4-6.\cos(x)^2.(1-\cos(x)^2)+(1-\cos(x)^2)^2.$
$\cos(4.x)=T(4,\cos(x)).$

7.16.10 Polynôme de Tchebychev de 2-nde 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 :
$\sin(4.x) = \sin(x) * (8 * \cos(x)^3 - 4.\cos(x)) = \sin(x) * U(3, \cos(x))$.

Chapitre 8

Le menu Plot

Remarque
Le tracé de la plupart des commandes commencant par `plot` ne se fait pas bien depuis l'écran CAS : il est donc préfierable d'utiliser l'application de géométrie pour faire les graphes correspondant à ces commandes.

8.1 Graphe d'une fonction : `plotfunc`

On ouvre l'application de géométrie et on appuie sur la touche `Plot`. Puis on utilise `Cmds` du bandeau. On choisit `6 Tracé` puis `1 Fonction`. `plotfunc(` s'écrit dans la ligne de commande et il suffit de compléter par l'expression $f(x)$)
dont on veut avoir le graphe.
La vue symbolique contient alors une ligne supplémentaire contenant la commande tapée par exemple : `plotfunc(f(x))`.
`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
```

Ou bien
On tape :

```
gf:=plotfunc(x^2-2)
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$$\sqrt{}\texttt{GC:=gf}$$

Puis on appuie sur la touche Plot pour avoir la vue graphique de l'application de géométrie, on obtient :

le graphe de x^2-2

8.2 Courbe en paramétrique : plotparam

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 $t1 \geq t \geq t2$).
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 :

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

ou encore :

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

Ou on tape si dans la configuration du graphique t va de -4 à 1 :

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

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

8.3 Courbe en polaire : plotpolar

plotpolar(f(t),t) trace la représentation polaire de la courbe définie par : $\rho = f(t)$.
On tape :

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

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
```

8.4 Tracé d'une suite récurrente : plotseq

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))
```

8.5 Courbe implicite en 2-d : plotimplicit

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

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

- `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 tape :

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

Ou on tape :

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

Ou on tape :

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

On obtient :

```
circle(point(O,O),1)
```

On tape :

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

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le dessin du cercle unité
```

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

`plotdensity(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.

8.7 Le champ des tangentes : plotfield

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 $x(t)' = -y, y(t)' = x$.

8.8 Lignes de niveaux : `plotcontour`

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

```
g:=plotcontour(x^2+y^2,[x=-3..3,y=-3..3],[1,2,3],
      display=[vert,rouge,noir]+[filled$3])
```

On obtient :

```
[polygon(point(....))..]
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

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

On tape :

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

On obtient :

```
[polygon(point(-4.8,-5),point(-3.9,-4)....)]
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$\sqrt{}$`GC:=`

On complète en

$\sqrt{}$`GC:=g`

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

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

8.9 Tracé de solutions d'équation différentielle : plotode

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

Pour visualiser les valeurs de la solution, se reporter à la section 4.7.
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)

8.10 Ligne polygonale : plotlist

`plotlist` a pour argument une liste `l` ou une matrice ayant deux colonnes.
`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 :

```
a:=plotlist([0,1,4,9,16])
```

Ou on tape :

```
a:=plotlist([[0,0],[1,1],[2,4],[3,9],[4,16]])
```

Puis dans la vue symbolique de l'application de géométrie (`Symb`), on fait `New` il s'écrit par exemple :

$$\sqrt{}\texttt{GD:=}$$

On complète en

$$\sqrt{}\texttt{GD:=a}$$

Puis dans la vue graphique de l'application de géométrie (`Plot`), on obtient :

```
le dessin des 5 points ((0,0),(1,1),...(4,16)) reliés
                    par 4 segments
```

Attention
`plotlist([0,1,2,3,4],[0,1,4,9,16])` ou
`plotlist([[0,1,2,3,4],[0,1,4,9,16]])` n'est pas valide !

Troisième partie

Le menu MATH de la touche 

Chapitre 9

Les fonctions sur les réels

9.1 les constantes de HOME

- `e` pour $\exp(1)$
- `i` pour le nombre complexe de module 1 et d'argument $\pi/2$ ou (0,1)
- `MAXREAL` c'est $+\infty$
- `MINREAL` c'est 0
- `Pi` ou `pi` ou `PI` pour π

9.2 Les constantes symboliques du CAS : `e pi i infinity inf 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)` et
`evalf(euler_gamma)` renvoie `0.577215664902`

9.3 Les booléens

9.3.1 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`.
Dans HOME, `TRUE` est remplacé dans l'historique par `1` et `FALSE` est remplacé dans l'historique par `0`.
Les tests ou les conditions sont des fonctions booléennes.

9.3.2 Les tests : ==, !=, >, >=, <, <=

==, !=, >, >=, <, <= sont des opérateurs infixés.
>=, <=, != sont obtenus avec les touches $\geq$, $\leq$ et $\neq$.
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.

9.3.3 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
```

9.4 Les opérateurs bit à bit

9.4.1 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`.

9.4.2 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`.

9.5 Les fonctions usuelles

Les fonctions usuelles sont accessibles en appuyant sur les touches corespondantes.
On a besoin de leurs noms en programmation, par exemple :
`id` désigne la fonction identité,
`sq` désigne la fonction carrée,
`sqrt` désigne la fonction racine carrée,
`neg` désigne la fonction $x->-x$,
`inv` désigne la fonction $x->\frac{1}{x}$.

9.6 Le plus petit entier >= à l'argument : CEILING ceiling

CEILING(a) ou ceiling(a) renvoie le plus petit entier supérieur ou égal à l'argument a.
On tape :

```
CEILING(45/8)
```

On obtient :

```
6
```

On tape :

```
CEILING(-45/8)
```

On obtient :

```
-5
```

On tape :

```
CEILING(2.5)
```

On obtient :

```
3
```

9.7 Partie entière d'un réel : FLOOR floor

FLOOR(a) ou floor(a) renvoie le plus grand entier inférieur ou égal à l'argument a.
On tape :

```
FLOOR(45/8)
```

On obtient :

```
5
```

On tape :

```
FLOOR(-45/8)
```

On obtient :

```
-6
```

On tape :

```
FLOOR(2.5)
```

On obtient :

```
2
```

9.8 Argument sans sa partie fractionnaire : IP

IP(a) renvoie l'argument réel a sans sa partie fractionnaire.
On tape :

```
IP(45/8)
```

On obtient :

```
5
```

On tape :

```
IP(-45/8)
```

On obtient :

```
-5
```

9.9 Partie fractionnaire : FP

FP(a) renvoie la partie fractionnaire de l'argument réel a.
On tape dans  :

```
FP(45/8)
```

On obtient :

```
0.625
```

On tape dans CAS :

```
FP(45/8)
```

On obtient :

```
5/8
```

On tape dans  :

```
FP(-45/8)
```

On obtient :

```
-0.625
```

On tape dans CAS :

```
FP(-45/8)
```

On obtient :

```
-5/8
```

9.10 Arrondir avec n décimales un réel ou un complexe : ROUND round

ROUND(a) ou round(a) (resp ROUND(a,n) ou round(a,n)) arrondit le réel a selon un entier (resp selon un nombre décimal ayant n décimales) le plus proche.
ROUND(a) (resp ROUND(a,n)) arrondit le complexe a à selon un élément de $\mathbb{Z}[i]$ le plus proche, (resp avec n décimales). On tape :

```
ROUND(45/8)
```

ou

```
round(45/8)
```

On obtient :

```
6
```

On tape :

```
ROUND(45/8,2)
```

ou

```
round(45/8,2)
```

On obtient :

```
5.63
```

On tape :

```
ROUND(-45/8)
```

ou

```
round(-45/8)
```

On obtient :

```
-6
```

On tape :

```
ROUND(-45/8,2)
```

ou

```
round(-45/8,2)
```

On obtient :

```
-5.62
```

On tape :

```
ROUND(0.5+i*pi)
```

ou

```
round(0.5+i*pi)
```

On obtient :

```
1+3*i
```

On tape :

```
ROUND(0.5+i*pi,4)
```

ou

```
round(0.5+i*pi,4)
```

On obtient :

```
0.5+3.1416*i
```

9.11 Tronquer avec n décimales un réel ou un complexe : TRUNCATE trunc

`TRUNCATE` ou `trunc` renvoie l'argument tronqué avec n décimales (par défaut n=0) . On tape dans HOME ou dans le CAS :

```
TRUNCATE(45/8)
```

ou

```
trunc(45/8)
```

On obtient :

```
5
```

On tape dans HOME ou dans le CAS :

```
TRUNCATE(45/8,2)
```

ou

```
trunc(45/8,2)
```

On obtient :

```
5.62
```

On tape dans HOME ou dans le CAS :

```
TRUNCATE(-45/8)
```

ou

```
trunc(-45/8)
```

On obtient :

```
-5
```

On tape dans HOME ou dans le CAS :

```
TRUNCATE(-45/8,2)
```

ou

```
trunc(-45/8,2)
```

On obtient :

```
-5.63
```

On tape dans HOME ou dans le CAS :

```
TRUNCATE(sqrt(2)+i*sqrt(5),4)
```

ou

```
trunc(sqrt(2)+i*sqrt(5),4)
```

On obtient :

```
1.4142+2.236*i
```

Attention Dans le CAS `truncate(P,n` tronque le polynôme `P` à l'ordre `n`. On tape :

```
truncate(x^5+x^4+x^2+x+1,2)
```

On obtient :

```
x^2+x+1
```

9.12 La partie fractionnaire d'un réel : `frac`

`frac` d'un réel renvoie sa partie fractionnaire.
On tape dans CAS ou dans HOME :

```
frac(22/7))
```

On obtient :

```
1/7
```

On tape dans CAS ou dans HOME :

```
frac(sqrt(2))
```

On obtient :

```
sqrt(2)-1
```

9.13 Le Réel sans sa partie fractionnaire : `iPart`

) `iPart` d'un réel renvoie un réel qui est égal au réel argument sans sa partie fractionnaire,
On tape :

```
iPart(sqrt(2))
```

On obtient :

```
1.0
```

9.14 Mantisse d'un réel : `MANT`

`MANT(a)` renvoie $|a|/10^n$ où l'entier n vérifie :
$10^n \leq |a| < 10^{n+1}$.
`MANT(a)` renvoie donc la mantisse d'un réel a c'est à dire les chiffres significatifs de a.
On tape :

```
MANT(45/8)
```

On obtient :

```
5.625
```

On tape :

```
MANT(-45/8)
```

On obtient :

```
5.625
```

9.15 Partie entière du logarithme à base 10 d'un réel : XPON

`XPON(a)` renvoie l'entier n tel que $10^n \leq |a| < 10^{n+1}$.
On tape :

```
XPON(45/8)
```

On obtient :

```
0
```

On tape :

```
XPON(45000/8)
```

On obtient :

```
3
```

On tape :

```
XPON(1234*sqrt(2))
```

On obtient :

```
3
```

en effet $10^3 < 1234 * \sqrt{2} \simeq 1745.13953597 < 10^4$

Chapitre 10

Arithmétique

10.1 Maximum de 2 ou plusieurs valeurs : `MAX max`

`MAX` ou `max` renvoie le maximum des éléments d'une séquence ou d'une liste de réels.
On tape :

```
MAX(4,5,8,2,6)
```

ou

```
max(4,5,8,2,6)
```

On obtient :

```
8
```

10.2 Minimum de 2 ou plusieurs valeurs : `MIN min`

`MIN` ou `min` renvoie le minimum des éléments d'une séquence ou d'une liste de réels.
On tape :

```
MIN(4,5,8,2,6)
```

ou

```
min(4,5,8,2,6)
```

On obtient :

```
2
```

10.3 `MOD`

`MOD` est une fonction infixée.
`a MOD b` renvoie le reste de la division euclidienne de a par b.
On tape :

```
22 MOD 5
```

On obtient :

```
2
```

10.4 FNROOT

FNROOT renvoie une racine approchée de l'expression donnée en premier argument pour la variable donnée en second argument et qui soit proche du 3ième argument.
On tape en mode réel (i non coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4)
```

On obtient :

```
-1.74295920217,1.
```

On tape en mode réel (i non coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4,x,-2)
```

On obtient :

```
-1.74295920217
```

On tape en mode complexe (i coché dans le Shift-CAS (Settings)):

```
FNROOT(x^4+3x-4)
```

On obtient :

```
[-1.74295920217,0.371479601083+1.46865601291*i,
      0.371479601083-1.46865601291*i,1.0]
```

10.5 NTHROOT surd

NTHROOT est une fonction infixée alors que surd est une commande préfixé du CAS .
NTHROOT s'obtient avec la touche shiftée $\sqrt[n]{}$ (Shift x^y).
p NTHROOT n renvoie la valeur de $n^{1/p}$
surd(n,p) renvoie $n^{1/p}$
On tape en mode réel dans HOME (i non coché dans le Shift-CAS (Settings)):

```
3 NTHROOT 8
```

On obtient :

```
2
```

On tape en mode complexe dans HOME (i coché dans le Shift-CAS (Settings)):

```
3 NTHROOT -1+i
```

On obtient :

```
0.85502540378-0.5*i
```

On tape en mode réel dans le CAS (i non coché dans le Shift-CAS (Settings)):

```
3 NTHROOT 8
```

ou

```
surd(8,3)
```

On obtient :

```
2
```

On tape en mode complexe dans le CAS :

```
3 NTHROOT -1+i
```

ou

```
surd(-1+i,3)
```

On obtient :

```
exp(ln(-1+i)/3)
```

10.6 %

`%(a,b)` renvoie $\frac{a}{100} * b$ (a pour cent de b).
On tape :

```
%(5,70)
```

On obtient :

```
3.5
```

On tape :

```
%(5,90)
```

On obtient :

```
4.5
```

10.7 Complexe

10.7.1 La touche i

La touche `i` est une touche shiftée (c'est `Shift 2`).
`i` est le nombre complexe de module 1 et d'argument $\pi/2$.
On tape :

```
1+3*i
```

ou

```
1+3i
```

On obtient :

```
1+3*i
```

10.7.2 ARG arg

ARG ou arg renvoie l'argument du nombre complexe donné en argument (en degrés ou en radians selon la configuration choisi).
On tape dans HOME :
ARG(2+6*i)
On obtient si on est en radians :
1.2490457724 On obtient si on est en degrés :
71.5650511771
On tape dans CAS :
ARG(2+6*i)
On obtient si on est en radians :
atan(3) On obtient si on est en degrés :
71.5650511771

10.7.3 CONJ conj

CONJ ou conj renvoie le conjugué du nombre complexe donné en argument.
On tape :

```
CONJ(1+3*i)
```

ou

```
conj(1+3*i)
```

On obtient :

```
1-3*i
```

10.7.4 IM im

IM ou im renvoie la partie imaginaire du nombre complexe donné en argument.
On tape :

```
IM(1+3*i)
```

On obtient :

```
3
```

10.7.5 RE re

RE ou re renvoie la partie réelle du nombre complexe donné en argument.
On tape :

```
RE(1+3*i)
```

On obtient :

```
1
```

10.7.6 SIGN sign

SIGN ou sign renvoie le nombre complexe donné en argument divisé par son module.
On tape :

```
SIGN(1+3*i)
```

On obtient :

```
(1+3*i)/sqrt(10)
```

10.7.7 La touche Shift-+/− : ABS abs

La touche |x| est une touche shiftée (c'estShift-+/−).
La touche |x| renvoie ABS(x) qui vaut :

- la valeur absolue d'un réel,
- le module d'un nombre complexe,
- la longueur d'un vecteur v_j ($(\sum_{j=1}^{n} |v_j|^2)^{1/2}$),
- la norme de Schur ou de Frobenius d'une matrice $a_{j,k}$ ($(\sum_{j,k=1}^{n} |a_{j,k}|^2)^{1/2}$).

On tape :

```
ABS(1+3*i)
```

On obtient :

```
3.1622776602
```

On utilise la touche |x| dans le CAS :

```
ABS(1+3*i)
```

ou on tape :

```
abs(1+3*i)
```

On obtient :

```
sqrt(10)
```

10.7.8 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
```

10.7.9 Multiplier par le complexe conjugué : `mult_c_conjugate`

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))
```

10.8 Exponentielle et Logarithmes

10.8.1 La fonction logarithme népérien `LN ln log`

`LN` ou `ln` ou `log` désigne la fonction logarithme népérien.
`LN` (ou dans CAS `ln`) peut s'obtenir avec la touche `LN`.
Attention Le log népérien dans HOME c'est `LN` et dans CAS c'est `ln` ou `log`.
On tape dans HOME : On tape :

```
LN(e)
```

On obtient :

```
1
```

On tape :

```
LN(2)
```

On obtient :

```
0.69314718056
```

Mais dans le CAS, on tape :

```
ln(e)
```

ou

```
log(e)
```

On obtient :

```
1
```

On tape :

```
ln(2)
```

ou

```
log(2)
```

On obtient :

```
ln(2)
```

10.8.2 La fonction logarithme à base 10 : `LOG log10`

Attention Le log à base 10 dans HOME c'est `LOG` et dans CAS c'est `log10` et `log` désigne le log népérien.
`LOG` ou `log10` désigne la fonction logarithme à base dix,
`LOG` (ou dans CAS `log10`) peut s'obtenir avec la touche `LOG`.
On tape dans HOME :

```
LOG(10)
```

On obtient :

```
1
```

On tape :

```
LOG(7)
```

On obtient :

```
0.84509804001
```

On tape dans CAS :

```
log10(10)
```

On obtient :

```
1
```

On tape :

```
log10(7)
```

On obtient :

```
ln(7)/ln(10)
```

10.8.3 La fonction logarithmeà base b : `logb`

`logb` désigne la fonction logarithme à base donnée comme deuxième argument :
On tape dans le CAS :

```
logb(7,7)
```

On obtient :

```
1
```

On tape dans CAS :

```
logb(7,10)
```

On obtient :

```
ln(7)/ln(10)
```

`logb(7,10)=log10(7)=log(7)/log(10),`

10.8.4 La fonction anti-logarithme :`ALOG alog10`

`alog10` désigne la fonction anti-logarithme à base dix, c'est la fonction :
$x- > 10^x$.
On tape dans HOME :

```
ALOG(3/2)
```

On obtient :

```
31.6227766017
```

c'est la valeur approchée de `sqrt(10)*10` à 10^{-10} près.
On tape dans CAS :

```
ALOG(3/2)
```

ou

```
alog(3/2)
```

ou On obtient :

```
sqrt(10)*10^1
```

On tape :

```
alog10(10)
```

On obtient :

```
10000000000
```

10.8.5 La tfonction exponentielle `EXP exp`

`EXP` ou `exp` désigne la fonction exponentielle.
`EXP` (ou dans CAS `exp`) peut s'obtenir avec la touche `EXP`.
On tape dans HOME :

```
EXP(2)
```

On obtient :

```
7.38905609893
```

Mais dans le CAS, on tape :

```
exp(2)
```

On obtient :

```
exp(2)
```

10.8.6 `EXPM1`

`EXPM1` désigne la fonction $x->EXP(x)-1$.
On tape :

```
EXPM1(4)
```

On obtient :

```
EXP(4)-1
```

On tape :

```
EXPM1(2.*10^-4)
```

On obtient :

```
0.00020002000133
```

MAIS si on tape :

```
EXP(2.*10^-4)-1
```

On obtient :

```
0.00020002
```

10.8.7 `LNP1`

`LNP1` désigne la fonction $x->LN(x+1)$
On tape :

```
LNP1(4)
```

On obtient :

```
LN(5)
```

On tape :

```
LNP1(2.*10^-4)
```

On obtient :

```
1.99980002666E-4
```

Chapitre 11

Fonctions trigonométriques

11.1 Les touches des fonctions trigonométriques

`SIN` ou `sin` désigne la fonction sinus.
On tape dans le CAS :

```
SIN(pi/3)
```

ou

```
sin(pi/3)
```

On obtient :

```
sqrt(3)/2
```

`ASIN` ou `asin` désigne la fonction arcsinus.
On tape dans le CAS :

```
ASIN(1/2)
```

ou

```
asin(1/2)
```

On obtient :

```
pi/6
```

`COS` ou `cos` désigne la fonction cosinus.
On tape dans le CAS :

```
COS(pi/3)
```

ou

```
cos(pi/3)
```

On obtient :

```
1/2
```

`ACOS` ou `acos` désigne la fonction arccosinus.
On tape dans le CAS :

```
ACOS(1/2)
```

ou

```
acos(1/2)
```

On obtient :

```
pi/3
```

`TAN` ou `tan` désigne la fonction tangente.
On tape dans le CAS :

```
TAN(pi/3)
```

ou

```
tan(pi/3)
```

On obtient :

```
sqrt(3)
```

`ATAN` ou `atan` désigne la fonction arcsinus.
On tape dans le CAS :

```
ATAN(sqrt(3)/3)
```

ou

```
atan(sqrt(3)/3)
```

On obtient :

```
pi/6
```

11.2 CSC csc

`CSC(x)` ou `csc` renvoie `1/SIN(x)` : c'est la fonction cosécante.
On tape dans le CAS :

```
CSC(pi/3)
```

ou

```
csc(pi/3)
```

On obtient après simplification :

```
2*sqrt(3)/3
```

11.3 ACSC acsc

`ACSC(x)` ou `acsc` renvoie `ASIN(1/x)` : c'est la fonction réciproque de la fonction cosécante.
On tape dans le CAS :

```
ACSC(sqrt(2))
```

ou

```
acsc(sqrt(2))
```

On obtient après simplification :

```
pi/4
```

11.4 SEC sec

`SEC(x)` ou `sec` renvoie `1/COS(x)` : c'est la fonction sécante.
On tape dans le CAS :

```
SEC(pi/3)
```

ou

```
sec(pi/3)
```

On obtient :

```
2
```

11.5 ASEC asec

`ASEC(x)` ou `asec` renvoie `ACOS(1/x)` : c'est la fonction réciproque de la fonction sécante.
On tape dans le CAS :

```
ASEC(2)
```

ou

```
asec(2)
```

On obtient :

```
1/3*pi
```

11.6 COT cot

`COT(x)` ou `cot` renvoie `COS(x)/SIN(x)` : c'est la fonction cotangente.
On tape dans le CAS :

```
COT(pi/3)
```

ou

```
cot(pi/3)
```

On obtient après simplification :

```
sqrt(3)/3
```

11.7 ACOT acot

`ACOT(x)` ou `acot` renvoie `pi/2-ATAN(x)` : c'est la fonction réciproque de la fonction cotangente.
On tape dans le CAS :

```
ACOT(sqrt(3))
```

ou

```
acot(sqrt(3))
```

On obtient après simplification :

```
pi/6
```

Chapitre 12

Fonctions hyperboliques

12.1 Sinus hyperbolique : SINH sinh

SINH(x) ou sinh(x) renvoie : $\frac{\exp(x)-\exp(-x)}{2}$: c'est la fonction sinus hyperbolique,
On tape dans le CAS :

```
hyp2exp(SINH(ln(2)))
```

On tape dans le CAS :

```
hyp2exp(sinh(ln(2)))
```

On obtient :

```
3/4
```

En effet la commande hyp2exp transforme les fonctions hyperboliques en exponentielles.

12.2 Arc sinus hyperbolique : ASINH asinh

ASINH ou asinh est la fonction réciproque de la fonction sinus hyperbolique,
On tape dans le CAS :

```
ASINH(3/4)
```

ou

```
asinh(3/4)
```

On obtient :

```
ln(2)
```

12.3 Cosinus hyperbolique : COSH cosh

COSH(x) ou cosh renvoie : $\frac{\exp(x)+\exp(-x)}{2}$: c'est la fonction cosinus hyperbolique,
On tape dans le CAS :

```
COSH(0)
```

ou

```
cosh(0)
```

On obtient :

```
1
```

12.4 Arc cosinus hyperbolique : ACOSH acosh

ACOSH ou acosh est la fonction réciproque de la fonction cosinus hyperbolique,
On tape dans le CAS :

```
hyp2exp(ACOSH(1))
```

ou

```
hyp2exp(acosh(1))
```

On obtient :

```
0
```

12.5 Tangente hyperbolique :TANH tanh

TANH(x) ou tanh renvoie : $\frac{\exp(2x)-1}{\exp(2x)+1}$, c'est la fonction tangente hyperbolique,
On tape dans le CAS :

```
hyp2exp(TANH(ln(3)))
```

ou

```
hyp2exp(tanh(ln(3)))
```

On obtient :

```
4/5
```

En effet la commande hyp2exp transforme les fonctions hyperboliques en exponentielles.

12.6 Arc tangente hyperbolique : ATANH atanh

ATANH ou atanh est la fonction réciproque de la fonction tangente hyperbolique,
On tape dans le CAS :

```
ATANH(4/5)
```

ou

```
atanh(4/5)
```

On obtient :

```
1/2*ln(9)
```

12.7 Autres fonctions

12.7.1 Liste des variables : lname

lname a comme paramètre une expression.
lname 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]
```

12.7.2 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)]
```

12.7.3 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]]
```

12.7.4 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
```

12.7.5 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 le CAS évalue toujours les expressions, sans avoir besoin de la commande eval : le niveau d'évaluation est indiqué dans la case Evaluation récursive de la Configuration du CAS (Shift CAS) et vaut par défaut vaut 5.
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 :

```
i*r+1
```

- on tape :

```
eval(a,2)
```

  On obtient :

```
i*(p+1)+1
```

- on tape :

```
eval(a,3)
```

  On obtient :

```
1-3*i
```

12.7.6 Pour ne pas évaluer une expression : `QUOTE quote '`

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`.
Remarque Lorsqu'on tape par exemple `a:=quote(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)`.
On tape :

```
a:=2;quote(2+3*a)
```

ou

```
a:=2;'2+3*a'
```

On obtient :

```
(2,2+3.a)
```

12.7.7 Évaluation numérique : `evalf approx`

`evalf` ou `approx` a comme paramètre une expression numérique ou une matrice. `evalf` renvoie la valeur numérique de l'expression ou de la matrice.
En mettant un deuxième argument `n` à `evalf` (ou `approx`), on peut spécifier le nombre de chiffres significatifs de l'approximation.
On tape :

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

On obtient :

```
1.41421356237
```

On tape :

```
evalf(sqrt(2),5)
```

On obtient :

```
1.4142
```

On tape :

```
evalf([[1,sqrt(2)],[0,1]])
```

On obtient :

```
[[1.0,1.41421356237],[0.0,1.0]]
```

On tape :

```
evalf([[1,sqrt(2)],[0,1]],5)
```

On obtient :

```
[[1.0,1.4142],[0.0,1.0]]
```

12.7.8 Approximation rationnelle : `exact`

`exact` a comme paramètre une expression numérique réelle.
`exact` donne une approximation rationnelle de tous les nombres décimaux r contenus dans l'expression à moins de `epsilon` c'està dire $|r - \texttt{exact}(r)| < \epsilon$ où ϵ est définit par `epsilon` dans la configuration du `CAS` (touche Shift `CAS`.
On tape :

```
exact(1.5)
```

On obtient :

```
3/2
```

On tape :

```
exact(1.414)
```

On obtient :

```
707/500
```

On tape :

```
exact(1.41421356237^2)
```

On obtient :

```
2
```

Chapitre 13

Fonctions de probabilité

13.1 Factorielle : `factorial !`

`!` est une fonction post fixée alors que `factorial` est une fonction préfixée.
`n!` ou `factorial(n)` renvoie la factorielle de n si n est entier et
`a!` renvoie valeur de la fonction Gamma en a+1 si a est réel.
On tape dans le CAS :

```
20!
```

ou

```
factorial(20)
```

On obtient :

```
2432902008176640000
```

On tape :

```
5.2!
```

ou

```
factorial(5.2)
```

On obtient :

```
169.406099462
```

13.2 Nombre de combinaisons de p objets pris parmi n : `COMB comb`

`COMB(n,p)` ou `comb(n,p)` renvoie le nombre de combinaisons de p éléments pris parmi n (n et p sont des entiers).
On a : `COMB(n,p)` renvoie $\frac{n!}{p!(n-p)!}$.
On tape :

```
COMB(5,2)
```

On obtient :

```
10
```

13.3 Nombre d'arrangements de p objets pris parmi n : PERM perm

`PERM(n,p)` ou `perm(n,p)` renvoie le nombre d'arrangements de p éléments pris parmi n (n et p sont des entiers).
On a : `PERM(n,p)` renvoie $\frac{n!}{(n-p)!}$.
On tape :

```
PERM(5,2)
```

On obtient :

```
20
```

13.4 Nombres aléatoires

13.4.1 Nombre aléatoire (réel ou entier) : RANDOM

- Pour avoir un nombre aléatoire réel entre 0 et 1, on ne met pas d'argument.
  On tape :
  ```
  RANDOM
  ```
  On obtient un nombre réel de 0,1, par exemple :
  ```
  0.291424166081
  ```
- Pour avoir un nombre aléatoire entier a entre 1 et n ($1 \leq a \leq n$) on met n comme argument sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3
  ```
  On obtient 1,2 ou 3, par exemple :
  ```
  1
  ```
- Pour avoir un nombre aléatoire réel a entre b et c ($b \leq a \leq c$), on met b, c comme arguments sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3,5
  ```
  On obtient un nombre réel de 3,5 par exemple :
  ```
  4.81731227506
  ```
- Pour avoir k nombres aléatoires entiers a entre p et n ($p \leq a \leq n$) on met k, p, n comme arguments sans mettre de parenthèse.
  On tape :
  ```
  RANDOM 3,2,5
  ```
  On obtient 3 nombres entiers compris entre 2 et 5, par exemple :
  ```
  [5,3,2]
  ```

13.4.2 Nombre entier aléatoire RANDINT

- Pour avoir un nombre aléatoire entier a entre 1 et n ($1 \leq a \leq n$) on met n comme argument.
  On tape :
  ```
  RANDINT(4)
  ```
  On obtient 0,1,2,3 ou 4, par exemple :

```
2
```

– Pour avoir un nombre aléatoire entier a entre b et c ($b \leq a \leq c$), on met b et c comme arguments.
On tape :

```
RANDINT(4,6)
```

On obtient 4,5 ou 6, par exemple :

```
5
```

– Pour avoir k nombres aléatoires entiers a entre p et n ($p \leq a \leq n$) on met k, p et n comme arguments. On tape :

```
RANDINT(4,2,6)
```

On obtient 4 nombres compris entre 2 et 6, par exemple :

```
[2,6,2,6]
```

13.4.3 La fonction `rand` du CAS

Tirage équiréparti sur $[0,1[$ **:** `rand()`

`rand()` renvoie au hasard, de façon équiprobable, un nombre réel de $[0,1[$.
On tape :

```
rand()
```

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)`

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
```

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)` 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)`

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)
```

On obtient :

```
1
```

ou on obtient :

```
0
```

On tape :

```
rand(-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)
```

On obtient par exemple :

```
8
```

Tirage aléatoire sans remise de p objets parmi n : `rand`

`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]
```

13.4.4 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]
```

13.4.5 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 **??**, **??** et **??**).
`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 la calculatrice 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)'`
`'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 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
```

13.4.6 Tirage selon une loi multinomiale : avec des programmes

On écrit les programmes `randmult` et `randmultinom` qui simulent la loi multinomiale.
`randmult(n,P)` choisit au hasard `n` nombres parmi `1...k` selon la loi multinomiale de probabilité `P` (`k=size(P)`). Cela veut dire qu'on effectue un tirage avec remise de `n` objets parmi `k=size(P)` objets. `randmult(n,P)` renvoie une liste `R` de `k=size(P)` éléments où `R[j]` est le nombre d'objets de probabilité `P[j]` qui ont été tirés.
On tape pour le programme `randmult` :

```
(n,p)->BEGIN
  local k,j,l,r,x,y;
  k:=size(p);
  x:=cumSum(p);
  if x[k]!=1 then return "erreur"; end;
    y:=makelist(0,1,k)
  for j from 1 to n do
    r:=rand(0,1);
    l:=1;
    while r>x[l] do
      l:=l+1;
    end;
  y[l]:=y[l]+1
  end;
  return (y);
END;
```

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.
Pour simuler un tirage, on tape : On tape :

```
randmult(6,[1/2,1/3,1/6])
```

On obtient par exemple :

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

randmultinom(n,P,C) choisit au hasard n objets parmi les éléments de la liste C. Si k=size(C), l'objet C[j] a la probabilité P[j] dêtre tiré pour (j=1..k). On doit avoir k=size(C)=size(P) et sum(P)=1.
randmultinom(n,P,C) renvoie la séquence de k listes constituées du nom des objets et de leur nombre d'apparition.
On tape pour le programme randmultinom :

```
(n,p,c)->BEGIN
  local k,j,l,r,x,y;
  k:=size(p);
  if size(c)!=k then return "erreur"; end;
  x:=cumSum(p);
  if x[k]!=1 then return "erreur"; end;
    y:=MAKELIST([c[j],0],j,1,k);
   for j from 1 to n do
    r:=rand(0,1);
    l:=1;
    while r>x[l] do
      l:=l+1;
    end;
  y[l,2]:=y[l,2]+1
  end;
 return y;
END;
```

On effectue 6 fois le tirage d'un objet parmi 3 objets ["A","B","C"] (tirage avec remise). Chaque objet a la probabilité [1/2,1/3,1/6] d'être tiré.
Pour simuler un tirage, on tape :

```
randmultinom(6,[1/2,1/3,1/6],["A","B","C"])
```

On obtient par exemple :

```
[["A",3],["B",1],["C",2]]
```

13.4.7 Tirage selon une loi normale : RANDNORM randNorm

RANDNORM(mu,sigma) ou randNorm(mu,sigma) renvoie un réel aléatoirement distribué selon la loi normale $N(mu, sigma)$ (par défaut $mu = 0$ et $sigma = 1$).
On tape :

```
RANDNORM()
```

Ou on tape :

```
RANDNORM(0,1)
```

On obtient, par exemple :

```
1.2440525851
```

On tape :

```
RANDNORM(1,2)
```

On obtient, par exemple :

```
-1.8799815939
```

13.4.8 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
```

13.4.9 Pour initialiser la suite de nombres aléatoires : `RANDSEED RandSeed srand`

`RANDSEED` ou `RandSeed` ou `srand` initialise la suite des nombres aléatoires fournie par `RANDOM`. Si on ne met pas de paramètre, `RANDSEED` utilise la valeur du temps comme paramètre.
On tape :

```
RANDSEED()
```

On obtient :

```
1
```

On tape :

```
RANDSEED(pi)
```

On obtient :

```
1
```

13.4.10 `UTPC`

`UTPC(n,x0)` renvoie la probabilité qu'une variable aléatoire Chisquare avec `n` degrés de liberté soit supérieure à `x0`.
On tape :
`UTPC(2,6.1)`
On obtient :
`0.0473589243911`

13.4.11 UTPF

`UTPF(n,m,x0)` renvoie la probabilité qu'une variable aléatoire Fisher-Snedecor avec `n,m` degrés de liberté soit supérieure à `x0`.
on tape :
`UTPF(4,10,3.5)`
On obtient :
`0.0491881403249`

13.4.12 UTPN

`UTPN(mu,v,x0)` renvoie la probabilité qu'une variable aléatoire Normale soit supérieure à `x0` avec `mu` la moyenne et `v` la variance (par défaut $mu = 0$ et $v = 1$).
On tape :
`UTPN(1.96)` ou
`UTPN(1,4,4.92)`
On obtient :
`0.0249978951482`
en effet $(x-1)/\sqrt{4} > 1.96$ est équivalent à $x > 4.92$.
On tape :
`UTPN(0.98)`
ou
`UTPN(1,4,2.96)`
On obtient :
`0.163543059328`
en effet $(x-1)/\sqrt{4} > 0.98$ est équivalent à $x > 2.96$.

13.4.13 UTPT

`UTPT(n,x0)` renvoie la probabilité qu'une variable aléatoire de Student avec `n` degrés de liberté soit supérieure à `x0`.
On tape :
`UTPT(3,2.35)`
On obtient :
`0.050152899407`
On tape :
`UTPT(3,-2.35)`
On obtient :
`0.949847100593`

13.5 Densité de probabilité

13.5.1 Densité de probabilité de la loi normale : NORMALD normald

`NORMALD(x)` ou `normald(x)` est la densité de probabilité de la loi normale centrée réduite (de moyenne 0 et d'écart-type 1).
`NORMALD(`μ,σ`,x)` ou `normald(`μ,σ`,x)` est la densité de probabilité de la loi normale de moyenne μ et d'écart-type σ).
On tape :

```
NORMALD(0.5)
```

Ou on tape :

```
NORMALD(0,1,0.5)
```

On obtient :

```
0.352065326764
```

On tape :

```
NORMALD(1,2,0.5)
```

On obtient :

```
0.193334058401
```

13.5.2 Densité de probabilité de la loi de Student : `STUDENT student`

`STUDENT(n,x)` ou `student(n,x)` est la densité de probabilité de la loi de Student ayant n degrés de liberté.
On tape :

```
STUDENT(3,5.2)
```

On obtient :

```
0.00366574413491
```

13.5.3 Densité de probabilité du χ^2 : `CHISQUARE chisquare`

`CHISQUARE(n,x)` ou `chisquare(n,x)` est la densité de probabilité de la loi du χ^2 ayant n degrés de liberté.
On tape :

```
CHISQUARE(2,3.2)
```

On obtient :

```
0.100948258997
```

13.5.4 Densité de probabilité de la loi de Fisher : `FISHER fisher snedecor`

`FISHER(n,m,x)` ou `fisher(n,m,x)` ou `snedecor(n,m,x)` renvoie la densité de probabilité en x de la loi de Fisher-Snedecor (n et m sont les nombres de degrés de liberté).
On tape :

```
FISHER(4,10,2.1)
```

On obtient :

```
0.141167840452
```

13.5.5 Densité de probabilité de la loi binomiale : `BINOMIAL binomial`

`BINOMIAL(n,k,p)` ou `binomial(n,k,p)` renvoie `COMB`$(n,k) * p^k * (1 - p)^(n - k)$ et
`BINOMIAL(n,k)` ou `binomial(n,k,p)` renvoie `COMB`(n,k) si il n'y a pas de 3ème argument.
On tape :

```
BINOMIAL(4,2)
```

On obtient :

```
6
```

On tape :

```
BINOMIAL(4,2,0.5)
```

On obtient :

```
0.375
```

On tape dans le CAS :

```
binomial(4,2)
```

On obtient :

```
6
```

```
binomial(4,2,1/2)
```

On obtient :

```
3/8
```

13.5.6 Densité de probabilité de la loi de Poisson : `POISSON 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)` ou `poisson(mu,k)` renvoie $\frac{\exp(-mu) * mu^k}{k!}$.
On tape :

```
POISSON(0.5,2)
```

On obtient :

```
0.0758163324641
```

13.6 Fonction de répartition

13.6.1 Fonction de répartition de la loi normale : NORMALD_CDF normald_cdf

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, on a :
$Proba(X \leq x)$= NORMALD_CDF(x)=normald_cdf(x) et
$Proba(x \leq X \leq y)$= NORMALD_CDF(x,y)=normald_cdf(x,y).
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, on a :
$Proba(X \leq x)$= NORMALD_CDF(μ,σ,x).
$Proba(x \leq X \leq y)$= NORMALD_CDF(μ,σ,x,y).
On tape :

```
NORMALD_CDF(0.96)
```

Ou on tape :

```
NORMALD_CDF(0,1,0.96)
```

On obtient :

```
0.831472392533
```

On tape :

```
NORMALD_CDF(1.96)
```

On obtient :

```
0.975002104852
```

On tape :

```
NORMALD_CDF(0,1.96)
```

On obtient :

```
0.475002104852
```

car NORMALD_CDF(0)=1/2 et 0.975002104852-0.5=0.475002104852
On tape :

```
NORMALD_CDF(1,2,1.96)
```

On obtient :

```
0.684386303484
```

On tape :

```
NORMALD_CDFnormal_cdf(1,2,1.1,2.9)
```

On obtient :

```
0.309005067853
```

13.6.2 Fonction de répartition de la loi de Student : `STUDENT_CDF` `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)`=`student_cdf(n,x)`.
$Proba(x \leq X \leq y)$=`STUDENT_CDF(n,x,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
```

13.6.3 Fonction de répartition de la loi du χ^2 `CHISQUARE_CDF` `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)`=`chisquare_cdf(n,x)`.
$Proba(x \leq X \leq y)$=`CHISQUARE_CDF(n,x,y)`=`chisquare_cdf(n,x,y)`.
On tape :

```
CHISQUARE_CDF(5,11)
```

On obtient :

```
0.948620016517
```

On tape :

```
CHISQUARE_CDF(5,3)
```

On obtient :

```
0.300014164121
```

On tape :

```
CHISQUARE_CDF(5,3,11)
```

On obtient :

```
0.648605852396
```

car 0.948620016517-0.300014164121=0.648605852396

13.6.4 La fonction de répartition de la loi de Fisher-Snédécor : `FISHER_CDF` `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)=fisher_cdf(n1,n2,x).`
$Proba(x \leq X \leq y)$=`FISHER_CDF(n1,n2,x,y)=`
`fisher_cdf(n1,n2,x,y)=snedecor_cdf(n1,n2,x,y).`
On tape :

```
FISHER_CDF(5,3,9)
```

On obtient :

```
0.949898927032
```

On tape :

```
FISHER_CDF(3,5,9.)
```

On obtient :

```
0.981472898262
```

On tape :

```
FISHER_CDF(3,5,2.)
```

On obtient :

```
0.767376082
```

On tape :

```
FISHER_CDF(3,5,2.,9.)
```

On obtient :

```
0.214096816262
```

car 0.981472898262-0.767376082=0.214096816262

13.6.5 Fonction de répartition de la loi binomiale : `BINOMIAL_CDF` `binomial_cdf`

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n, p)$.
On a :
`BINOMIAL_CDF(n,p,x)=binomial_cdf(n,p,x)=` $Proba(X \leq x)$=
`BINOMIAL(n,0,p)+...+BINOMIAL(n,floor(x),p).`
`BINOMIAL_CDF(n,p,x,y)=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
```

On tape dans le CAS :

```
binomial_cdf(4,1/2,2))
```

On obtient :

```
11/16
```

On tape :

```
binomial_cdf(2,3/10,1)
```

On obtient :

```
91/100
```

On tape :

```
binomial_cdf(2,3/10,1,2)
```

On obtient :

```
51/100
```

13.6.6 Fonction de répartition de la loi de Poisson : POISSON_CDF 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)=poisson_cdf(mu,x) avec $X \in \mathcal{P}(mu)$. et
$Proba(x \leq X \leq y)$=POISSON_CDF(mu,x,y)=poisson_cdf(mu,x,y)
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
```

13.7 Fonction de répartition inverse

13.7.1 Fonction de répartition inverse normale : NORMALD_ICDF normald_icdf

Lorsqu'une variable aléatoire X suit une loi normale centrée réduite, si on a NORMALD_ICDF(x)=norm
c'est que l'on a :
$Proba(X \leq h)$=x=NORMALD_CDF(h)=normald_cdf(h).
Lorsqu'une variable aléatoire X suit une loi normale de moyenne μ et d'écart-type σ, si on a :
NORMALD_ICDF(μ,σ,x)=normald_icdf(μ,σ,x)=h
c'est que l'on a :
$Proba(X \leq h)$=x=NORMALD_CDF(μ,σ,h)=normald_cdf(μ,σ,h).
On tape :

```
NORMALD_ICDF(0.95)
```

Ou on tape :

```
NORMALD_ICDF(0,1,0.95)
```

On obtient :

```
1.64485362695
```

On tape :

```
NORMALD_ICDF(0.975)
```

On obtient :

```
1.95996398454
```

On tape :

```
NORMALD_ICDF(1,2,0.495)
```

On obtient :

```
0.974933060984
```

On tape :

```
NORMALD_ICDF(1,2,NORMALD_CDF(1,2,0.975))
```

On obtient :

```
0.975
```

On tape :

```
NORMALD_CDF(1,2,NORMALD_ICDF(1,2,0.495))
```

On obtient :

```
0.495
```

On tape :

```
NORMALD_ICDF(1,2,2.96*sqrt(2))
```

On obtient :

```
0.944423950497
```

13.7.2 Fonction de répartition inverse de Student : STUDENT_ICDF student_icdf

Lorsqu'une variable aléatoire X suit une loi de Student ayant n degrés de liberté, si on a `STUDENT_ICDF(n,x)=student_icdf(n,x)=h` c'est que : $Proba(X \leq h)$=`x=STUDENT_CDF(n,h)=student_cdf(n,h)`.
On tape :

```
STUDENT_ICDF(5,0.95)
```

On obtient :

```
2.01504837333
```

13.7.3 Fonction inverse de la fonction de répartition de la loi du χ^2 CHISQUARE_ICDF chisquare_icdf

Lorsqu'une variable aléatoire X suit une loi du χ^2 ayant n degrés de liberté, si on a `CHISQUARE_ICD(n,x)=chisquare_icdf(n,x)=h` c'est que : $Proba(X \leq h)$=`x= CHISQUARE_CDF(n,h)=chisquare_cdf(n,h)`.
On tape :

```
CHISQUARE_ICDF(5,0.95)
```

On obtient :

```
11.0704976935
```

13.7.4 Inverse de la fonction de répartition de la loi de Fisher-Snédécor : FISHER_ICDF 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)=fisher_icdf(n1,n2,x)=snedecor_icdf(n1,n2,x)=h`
c'est que :
$Proba(X \leq h)$=`x=FISHER_CDF(n1,n2,h)=`
`fisher_cdf(n1,n2,h)= snedecor_cdf(n1,n2,h`
On tape :

```
FISHER_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)`

13.7.5 Fonction de répartition inverse de la loi binomiale : BINOMIAL_ICDF binomial_icdf

Lorsqu'une variable aléatoire X suit une loi binomiale $\mathcal{B}(n, p)$, si on a :
`BINOMIAL_ICDF(n,p,x)=binomial_icdf(n,p,x)=h` c'est que
$Proba(X \leq h) = x$=`BINOMIAL_ICDF(n,p,h)=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
```

On tape dans le CAS :

```
binomial_icdf(4,1/2,0.9)
```

On obtient :

```
3
```

On tape :

```
binomial_icdf(2,3/10,0.95)
```

On obtient :

```
2
```

13.7.6 Fonction de répartition inverse de Poisson :POISSON_ICDF 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)=poisson_icdf(mu,t)= h équivaut à
$Proba(X \leq h)$ =t= poisson_cdf(mu,h)=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
```

Chapitre 14

Les fonctions de statistique

14.1 Les fonctions 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 `A1`, ou
- en prenant comme série statistique la liste `A1` avec comme effectifs encore la liste `A1`.

On tape :
`A1:=[0,1,2,3,4,5,6,7,8,9,10,11]`
On pourra se reporter aussi à 15.1 lorsque les arguments sont des listes et à 15.1.1 lorsque les arguments sont des matrices.

14.1.1 La moyenne : `mean`

`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

14.1.2 L'écart-type : `stddev`

`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)
```

14.1.3 L'écart-type de la population : `stddevp stdDev`

`stddevp` 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 :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

stddevp(A1)
```

On obtient :

```
sqrt(13)
```

En effet : `n=size(A1)=12` et `12/11*stddev(A1)^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(A1,A1)
```

On obtient :

```
sqrt(22/3)
```

En effet `sum(A1)=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` 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`.

14.1.4 La variance : `variance`

`variance` calcule la variance numérique des éléments d'une liste.
On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

variance(A1)
```

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 :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

variance(A1,A1)
```

On obtient :

```
65/9
```

On tape :

```
variance([[1,2],[3,4]])
```

On obtient :

```
[1,1]
```

14.1.5 La médiane : `median`

`median` calcule la médiane des éléments d'une liste.
On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

median(A1)
```

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 :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

median(A1,A1)
```

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.

14.1.6 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 :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

quartiles(A1)
```

On obtient :

```
[[0.0],[2.0],[5.0],[8.0],[11.0]]
```

On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

quartiles(A1,A1)
```

On obtient :

```
[1,6,8,10,11]
```

14.1.7 Le premier quartile : `quartile1`

`quartile1` renvoie le premier quartile des éléments d'une liste.
On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

quartile1(A1)
```

On obtient le premier quartile de `A1` :

```
2.0
```

`quartile1` calcule le premier quartile des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
quartile1(A1,A1)
```

On obtient le premier quartile de `A1` pondérée par `A1` :

14.1.8 Le troisième quartile : `quartile3`

`quartile3` renvoie le troisième quartile des éléments d'une liste.
On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

quartile3(A1)
```

On obtient le troisième quartile de `A1` :

```
8.0
```

`quartile3` calcule le troisième quartile des éléments d'une liste pondérée par une autre liste donnée comme deuxième argument.
On tape :

```
quartile3(A1,A1)
```

On obtient le premier quartile de `A1` ponderée par `A1` :

```
10
```

14.1.9 Les déciles : `quantile`

`quantile(L1,p)` où `L1` est la série statistique et `p` un réel de [0,1[, indique la valeur du caractère à partir de laquelle la fréquence cumulée de `L1` atteint ou dépasse `p`.
On tape :

```
A1:=[0,1,2,3,4,5,6,7,8,9,10,11]

quantile(A1,0.1)
```

On obtient le premier décile :

```
1.0
```

On tape :

```
quantile(A1,0.25)
```

On obtient le premier quartile :

```
2.0
```

On tape :

```
quantile(A1,0.5)
```

On obtient la médiane :

```
5.0
```

On tape :

```
quantile(A1,0.75)
```

On obtient le troisième quartile :

```
8.0
```

On tape :

```
quantile(A1,0.9)
```

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(A1,A1,0.25)
```

On obtient le premier quartile de la liste `A` pondérée par `A` :

```
6
```

14.1.10 L'histogramme : `histogram`

`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,1,100),0,100)
```

Ici on a choisi `class_min=0` et `class_size=100`. On tape :

```
histogram(seq(rand(10),k,1,100),0,1)
```

Ici on a choisi `class_min=0` et `class_size=1`.

14.1.11 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 `A1:=[0,1,2,3,4,5,6,7,8,9,10,11]`):

```
covariance(A1,A1^2)
```

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 $N[j,k]$ ($j = 1..p, k = 1..q$), `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 :

$$M = \begin{bmatrix} a \setminus b & b[1] & ... & b[q] \\ a[1] & N[1,1] & ... & N[1,q] \\ ... & ... & ... & ... \\ a[p] & N[p,0] & ... & N[p,q] \end{bmatrix}$$

`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
```

14.1.12 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 `A1:=[0,1,2,3,4,5,6,7,8,9,10,11]`):

```
correlation(A1,A1^2)
```

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 $N[j,k]$ ($j = 1..p, k = 1..q$), `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 :

$$M = \begin{bmatrix} a \setminus b & b[1] & ... & b[q] \\ a[0] & N[1,1] & ... & N[1,q] \\ ... & ... & ... & ... \\ a[p] & N[p,1] & ... & N[p,q] \end{bmatrix}$$

`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))
```

14.1.13 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 `A1:=[0,1,2,3,4,5,6,7,8,9,10,11]`):

```
covariance_correlation(A1,A1^2)
```

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))]
```

14.1.14 Ligne polygonale : `polygonplot`

`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`).

14.1.15 Ligne polygonale : `plotlist`

`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. `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 :

```
plotlist([0,1,4,9,16])
```

Ou on tape :

```
plotlist([[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 !

14.1.16 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`).

14.1.17 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 `A1` 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(A1[0]);max(A1[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],[3.0,4.0,4.5,5.0,5.5,6.0,8.0,10
```

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],[3.0,4.0,6.0,8.0,10.0,12.0,10.0
```

14.1.18 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 :

```
X1:=[0,1,2,3,4,5,6,7,8,9,10]

Y1:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22,

47.31,61.78,80.68,105]

Z1:=log(Y1)

linear_regression(X1,Z1)
```

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.

14.1.19 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 :

```
X1:=[0,1,2,3,4,5,6,7,8,9,10]

Y1:=[7.3,9.53,12.47,16.3,21.24,27.73,36.22,47.31,

61.78,80.68,105]

exponential_regression(X1,Y1)
```

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(X1,ln(Y1))]
```

On obtient :

```
1.30568684451,7.30853268031
```

14.1.20 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 :

```
X1:=[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]

Y1:=[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(X1,Y1)
```

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(X1),Y1)
```

On obtient :

```
2.83870854646,0.843078064152
```

et le coefficient de corrélation est :

```
correlation(ln(X1),Y1)
```

On obtient :

```
0.977939822434
```

On peut aussi taper pour chercher une meilleur approximation :

```
logarithmic_regression(X1,log(Y1))
```

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(X1),ln(Y1))
```

On obtient :

```
0.732351031846,0.467599676658
```

et le coefficient de corrélation est :

```
correlation(ln(X1),ln(Y1))
```

On obtient :

```
0.999969474543
```

14.1.21 Régression polynômiale : `polynomial_regression`

Pour approcher les données par une fonction polynômiale de degré $\leq n$ d'équation $y = a_0x^n + ..+ a_n$, 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`.

14.1.22 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 :

```
X1:=[1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8]

Y1:=[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(X1),ln(Y1))
```

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(X1),ln(Y1))
```

On obtient :

```
0.999969474543
```

14.1.23 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 `L1` 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 la calculatrice arrive à l'estimer...

`logistic_regression(L1,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 `L1`, la calculatrice 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+3.1415926535
 -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`

Chapitre 15

Les statistiques

15.1 Fonctions statistiques sur une liste : `mean, variance, stddev, stddevp, median, quantile, quartiles, quartile1, quartile3`

Voir aussi 15.1.1 and 14.

Fonctions utiles pour les statistiques dont les donn'ees sont des listes :

- `mean` 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` 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` 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
```

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 15.1.1 et pour les listes pondérées au chapitre 14.

15.1.1 Fonctions statistiques sur les colonnes d'une matrice : mean, stddev, variance, median, quantile, quartiles

Voir aussi 15.1 and 14.
Fonctions utiles pour les statistiques dont les donn'ees sont les colonnes d'une matrice :

- mean 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 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]]
```

15.2 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 CAS, les indices d'une table peuvent être n'importe quels objets du CAS. 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.

Chapitre 16

Les listes

16.1 `MAKELIST` et `makelist`

Dans HOME, `MAKELIST` crée une liste à partir d'une expression symbolique.
Par exemple, on crèe une liste à partir de X^2+1, en faisant varier la variable X de 2 à 6 avec un pas de 1 (1 peut être omis), on tape :

```
MAKELIST(X^2+1,X,2,6)
```

On obtient :

```
{5,10,17,26,37}
```

On tape :

```
MAKELIST(0,X,1,10)
```

On obtient :

```
{0,0,0,0,0,0,0,0,0,0}
```

On crèe une liste à partir de X^2+1, en faisant varier la variable X de 2 à 6 avec un pas de 2, on tape :

```
MAKELIST(X^2+1,X,2,6,2)
```

On obtient :

```
{5,17,37}
```

Dans CAS on peut utiliser `MAKELIST` et `makelist`. `makelist` a une fonction comme premier argument, le deuxième argument représente la valeur initilale de la variable et le troisième argument représente sa valeur finale. On peut mettre un quatrième argument qui représente le pas de la variable.
Attention les indices commencent aussi à 1.
On tape :

```
makelist(x->x^2,1,10)
```

On obtient :

```
[1,4,9,16,25,36,49,64,81,100]
```

On tape :

```
makelist(x->x^2,1,10,2)
```

On obtient :

```
[1,9,25,49,81]
```

16.2 SORT sort

SORT ou sort trie dans l'ordre croissant les composantes d'une liste.
On tape :

```
SORT([12,2,31,24,15])
```

On obtient :

```
[2,12,15,24,31]
```

On tape :

```
SORT({12,2,31,24,15})
```

On obtient :

```
{2,12,15,24,31}
```

16.3 REVERSE

REVERSE crée une liste en inversant l'ordre des éléments.
On tape :

```
REVERSE([1,22,3,4,5])
```

On obtient :

```
[5,4,3,22,1]
```

On tape :

```
REVERSE({1,22,3,4,5})
```

On obtient :

```
{5,4,3,22,1}
```

16.4 CONCAT concat

CONCAT ou concat concaténe deux listes ou deux vecteurs ou deux chaines de caractères ou deux matrices (les 2 matrices doivent avoir le même nombre de lignes et seront concaténées ligne par ligne.
On tape :

```
CONCAT([1,2,3],[4,5])
```

On obtient :

```
[1,2,3,4,5]
```

On tape :

```
CONCAT({1,2,3},{4,5})
```

On obtient :

```
{1,2,3,4,5}
```

On tape :

```
CONCAT("BON","JOUR")
```

On obtient :

```
"BONJOUR"
```

On tape :

```
2=>A
```

```
CONCAT([1,A,3],[4,5])
```

On obtient :

```
[1,2,3,4,5]
```

On tape :

```
CONCAT([[1,2],[3,4]],[[4,5,6],[6,7,8]])
```

On obtient :

```
[[1,2,4,5,6],[3,4,6,7,8]]
```

On tape :

```
2=>A
```

```
CONCAT({1,A,3},{4,5})
```

On obtient :

```
{1,2,3,4,5}
```

Pour concaténer une chaine et une liste en une liste on utilise `CONCAT`.
On tape :

```
2=>A

CONCAT([1,A,3]),"L1"
```

On obtient :

```
[1,2,3,"L1"]
```

On tape :

```
2=>A

CONCAT("L1",[1,A,3])
```

On obtient :

```
["L1",1,2,3]
```

Attention
Pour concaténer une chaine et une liste en une chaine on utilise +.
On tape :

```
2=>A

"L1="+[1,A,3]
```

On obtient :

```
"L1=[1,2,3]"
```

On tape :

```
2=>A

[1,A,3]+"L1="
```

On obtient :

```
"[1,2,3]L1="
```

On tape :

```
2=>A

"L1="+{1,A,3}
```

On obtient :

```
"L1={1,2,3}"
```

16.4.1 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]]
```

16.4.2 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]
```

16.5 POS

POS renvoie la position d'un élément dans une liste c'est à dire POS renvoie l'indice de la première occurence de l'élément ou 0 si l'élément n'est pas dans la liste.
On tape :

```
POS([4,3,1,2,3,4,5],4)
```

Ou on tape :

```
POS({4,3,1,2,3,4,5},4)
```

On obtient :

```
1
```

On tape :

```
POS([4,3,1,2,3,4,5],2)
```

Ou on tape :

```
POS({4,3,1,2,3,4,5},2)
```

On obtient :

```
4
```

On tape :

```
POS([4,3,1,2,3,4,5],6)
```

Ou on tape :

```
POS({4,3,1,2,3,4,5},6)
```

On obtient :

```
0
```

16.6 DIM dim SIZE size length

`SIZE` ou `size` ou `DIM` ou `dim` ou `length` renvoie la longueur de la liste (ou de la chaîne) donnée en argument.
Attention !
Dans HOME et dans CAS `SIZE` renvoie la dimension d'une matrice alors que dans CAS `size` renvoie le nombre de ligne d'une matrice.
On tape dans HOME :

```
SIZE({1,2,3})
```

On obtient :

```
3
```

On tape :

```
SIZE([[1,2,3],[4,5,6]])
```

On obtient :

```
{2,3}
```

On tape dans CAS :

```
size([1,2,3])
```

On obtient :

```
3
```

On tape :

```
size([[1,2,3],[4,5,6]])
```

On obtient :

```
2
```

Attention
On tape dans CAS :

```
SIZE([[1,2,3],[4,5,6]])
```

On obtient :

```
[2,3]
```

16.6.1 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]
```

16.6.2 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]
```

16.6.3 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]
```

16.6.4 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]
```

16.6.5 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)`

16.6.6 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); return
remove(x->(ord(x)==97),seq(chn[k],k,1,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"
```

16.6.7 Partie droite et gauche d'une liste : `right, left`

- `right(l,n)` renvoie les n derniers éléments d'une liste `l`.
  On tape :
  ```
  right([1,2,3,4,5,6],4)
  ```
  On obtient :
  ```
  [3,4,5,6]
  ```
- `left(l,n)` renvoie les n premiers éléments d'une liste `l`.
  On tape :
  ```
  left([1,2,3,4,5,6,7,8],3)
  ```
  On obtient :
  ```
  [1,2,3]
  ```

16.6.8 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 :
`"l'indice de sa première apparition"`.
Attention à l'ordre des paramètres, c'est pour des raisons de compatibilité !
On tape :

```
member(2,[1,2,3,4,2])
```

On obtient :

```
2
```

On tape :

```
member(2,% {1,2,3,4,2% })
```

On obtient :

```
2
```

16.6.9 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 :
`"l'indice de sa première apparition"`.
Attention, à l'ordre des paramètres, c'est pour des raisons de compatibilité !
On tape :

```
contains([1,2,3,4,2],2)
```

On obtient :

```
2
```

On tape :

```
contains(% {1,2,3,4,2% },2)
```

On obtient :

```
2
```

16.6.10 Compter les éléments d'une liste ou d'une matrice vérifiant une propriété : `count`

Selon ses paramètres : `count` peut compter dans une liste `l` le nombre d'éléments égaux à `a` avec `count(x->x==a,l)`,
`count` peut compter dans une liste `l` le nombre d'éléments supérieurs à `a` avec `count(x->x>a,l)`,
`count` peut compter dans une liste `l` le nombre d'éléments inférieurs à `a` avec `count(x->x<a,l)`.`count` peut compter dans une liste `l` le nombre d'éléments de `l` avec `count(x->1,l)` En effet `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
```

En effet, `(x)->x<12` est une fonction booléenne qui vaut `1` si `x<12` et `0` sinon. On a donc $1+0+0+1+1+0=3$.
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
```

En effet, `(x)->x==12` est une fonction booléenne qui vaut `1` si `x==12` et `0` sinon. On obtient donc le nombre de termes égals à 12. Ici c'est 1.
On tape :

```
count((x)->x>12,[2,12,45,3,7,78])
```

On obtient :

```
2
```

En effet, `(x)->x>12` est une fonction booléenne qui vaut `1` si `x>12` et `0` sinon. On a donc $0+0+1+0+0+1=2$.
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.

16.6.11 Sélectionner des éléments d'une liste : select

Dans CAS 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]
```

16.7 Liste des différences de termes consécutifs : ΔLIST deltalist

Dans HOME (resp CAS) ΔLIST (resp deltalist) renvoie la liste des différences entre les composantes de la liste donnée en argument.
On tape dans HOME :

```
ΔLIST([1,21,34,41,52])
```

On obtient :

```
[20,13,7,11]
```

On tape dans HOME :

```
ΔLIST({1,21,34,41,52})
```

On obtient :

```
{20,13,7,11}
```

On tape dans CAS :

```
deltalist([1,21,34,41,52])
```

On obtient :

```
[20,13,7,11]
```

16.8 Somme des éléments d'une liste : ΣLIST sum

Dans HOME ΣLIST renvoie la somme des composantes de la liste donnée en argument.
On tape :

```
ΣLIST([1,2,3,4,5])
```

On obtient :

```
15
```

On tape :

```
ΣLIST({1,2,3,4,5})
```

On obtient :

```
15
```

Dans CAS `sum` renvoie la somme des composantes de la liste donnée en argument.
On tape :

```
sum([1,2,3,4,5])
```

On obtient :

```
15
```

16.9 Produit des éléments d'une liste : ΠLIST product

Dans HOME,ΠLIST renvoie le produit des composantes de la liste donnée en argument.
On tape :

```
ΠLIST([1,2,3,4,5])
```

On obtient :

```
120
```

On tape :

```
ΠLIST({1,2,3,4,5})
```

On obtient :

```
120
```

Dans CAS,`product` renvoie le produit des composantes de la liste donnée en argument.
On tape :

```
product([1,2,3,4,5])
```

On obtient :

```
120
```

16.9.1 Appliquer une fonction d'une variable aux éléments d'une liste : map apply

`map` ou `apply` sert à appliquer une fonction aux éléments d'une liste. Mais ces 2 instructions ne sont pas des synonymes. On a :

- `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]
```

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 :

```
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

```
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

```
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 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]]
```

Remarque
Si `l1,l2,l3` sont des listes :
`sizes([l1,l2,l3])=map(size,[l1,l2,l3])`

16.9.2 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]]
```

16.10 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).

16.11 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]
```

16.12 Fonctions utiles pour les listes et les composantes d'un vecteur

16.12.1 Les normes d'un vecteur : `maxnorm l1norm l2norm norm`

Voir aussi **??** 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 :
  ```
  4
  ```
  En effet : `x=3, y=-4, z=2` et `4=max(|x|,|y|,|z|)`.
- `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|`.
- `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 $29 = |x|^2 + |y|^2 + |z|^2$.

16.12.2 Pour normaliser les composantes d'un vecteur : `normalize`

`normalize` 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 $50 = |x|^2 + |y|^2 + |z|^2$.

16.12.3 Sommes cumulées des éléments d'une liste : `cumSum`

`cumSum` permet de faire les sommes cumulées des éléments d'une liste ou d'une séquence de nombres réels ou de décimaux ou de chaîne de caractères.
`cumSum` a comme argument une liste ou une séquence.
`cumSum` renvoie une liste ou une séquence, l'élément d'indice k étant obtenu en faisant la somme des éléments d'indice $1...k$.
Si `l` est une liste, `cumSum` renvoie la liste `lr` égale à `[sum(l[j],j=1..k)$(k=1..size(l))]`.
Si `l` est une séquence, `cumSum` renvoie la séquence `lr` égale à `sum(l[j],j=1..k)$(k=1..size(l))`.
On tape :

```
L:=cumSum(1,2,3)
```

On obtient :

```
1,3,6
```

On tape :

```
L:=cumSum([1,2,3])
```

On obtient :

```
[1,3,6]
```

On tape :

```
c[2]
```

On obtient :

```
3
```

16.12.4 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]])
```

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]
```

16.12.5 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 '-'.

16.12.6 Produit terme à terme de deux listes : .*

Voir aussi `product` pour les listes et les matrices (cf **??** et **??**)
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]]
```

16.12.7 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]
```

Chapitre 17

Les chaînes de caractères

17.1 Écriture d'une chaîne ou d'un caractère : "

Les chaînes de caractères s'écrivent en utilisant les guillemets comme délimiteurs (" " c'est la touche ALPHA 0).
Un caractère est une chaîne ayant un caractère ; en effet les délimiteurs ' ' ou (quote touche Shift ()) 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 (les indices qui commencent à 1).
Exemple :
On tape :

```
"bonjour"[2]
```

On obtient :

```
"o"
```

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?"
```

17.1.1 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"
```

17.1.2 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 $1...k-1$) avec la chaîne d'indice k.
Si `l` est une liste de k chaînes `cumSum` renvoie la liste `lr` égale à `[sum(l[j],j=1..k)$(k=1..siz`
On tape :

```
c:=cumSum("Madame ","bon","jour")
```

On obtient :

```
"Madame ","Madame bon","Madame bonjour"
```

On tape :

```
c[2]
```

On obtient :

```
"Madame bon"
```

17.1.3 Repérer un caractère dans une chaîne : INSTRING 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 0 si c n'est pas dans S et,
sinon renvoie "l'indice de sa première apparition".
On tape :

```
inString("abcded","d")
```

On obtient :

```
4
```

On tape :

```
inString("abcd","e")
```

On obtient :

```
0
```

17.2 ASC asc

ASC ou asc renvoie la liste des codes ASCII des caractères de la chaine.
On tape les " " à l'aide de la touche ALPHA 0.
On tape dans HOME :

```
ASC("A")
```

On obtient :

```
[65]
```

On tape dans HOME :

```
ASC("ABC")
```

On obtient :

```
[65,66,67]
```

On tape dans CAS :

```
asc("A")
```

On obtient :

```
[65]
```

On tape dans CAS :

```
asc("ABC")
```

On obtient :

```
[65,66,67]
```

17.3 CHAR char

CHAR ou char renvoie la chaine correspondant aux caractères ayant comme code ASCII ceux de l'argument.
On tape dans HOME :

```
char(65)
```

ou

```
char({65})
```

On obtient :

```
"A"
```

On tape :

```
char([65,66,67])
```

ou

```
char({65,66,67})
```

On obtient :

```
"ABC"
```

On tape dans CAS :

```
char(65)
```

ou

```
char([65])
```

ou

```
char({65})
```

On obtient :

```
"A"
```

On tape :

```
char([65,66,67])
```

ou

```
char({65,66,67})
```

On obtient :

```
"ABC"
```

17.3.1 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"
```

17.4 Pour utiliser une chaîne comme un nombre ou une commande : expr

17.4.1 Pour utiliser une chaîne comme un nombre

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.
expr renvoie ce nombre entier On tape :

```
expr("123")+1
```

On obtient :

```
124
```

On tape :

```
expr("45.67")+2.12
```

On obtient :

```
47.79
```

expr permet aussi 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 :

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 $1 * 16^2 + 2 * 16 + 15 = 303$

17.4.2 Pour utiliser une chaîne comme nom de commande

`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` **??**)
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
```

17.5 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('(ifactor(6)'))
```

On obtient :

```
"ifactor(6)"
```

17.6 inString

`inString(l,c)` teste si c est dans la chaine l et renvoie l'indice de c ou 0.
On tape :

```
inString"ABCDEF","C"
```

On obtient :

```
3
```

On tape :

```
inString"ABCDEF","G"
```

On obtient :

```
0
```

17.7 left

`left(l,n)` renvoie la partie gauche de longueur n de la chaine l.
On tape :

```
left("ABCDEF",3)
```

On obtient :

```
"ABC"
```

17.8 right

`right(l,n)` renvoie la partie droite de longueur n de la chaine l.
On tape :

```
right("ABCDEF",2)
```

On obtient :

```
"EF"
```

17.9 mid

`mid(l,d,n)` renvoie la chaine extraite de la chaine l, d'ebutant par le caractère d'indice d, et de longueur n (par défaut n=`dim`$(l) - d$). On tape :

```
mid("ABCDEF",2,3)
```

On obtient :

```
"BCD"
```

On tape :

```
mid("ABCDEF",2)
```

On obtient :

```
"BCDEF"
```

17.10 rotate

`rotate` renvoie la chaine obtenue en mettant le dernier caractère en premier.
On tape :

```
rotate("ABC")
```

On obtient :

```
("CAB")
```

17.11 Longueur d'une chaîne : dim DIM size SIZE length

`DIM` (ou `dim` ou `size` ou `SIZE` ou `length` renvoie la longueur de la chaîne (ou de la liste).
On tape dans HOME ou dans CAS :

```
DIM("ABC")
```

On obtient :

On peut aussi utiliser `SIZE`
On tape :

```
SIZE("ABC")
```

On obtient :

```
3
```

Remarque Dans HOME `DIM` et `SIZE` sont équivalents pour les matrices.
On tape dans HOME :

```
DIM([[1,2,3],[4,5,6]])
```

On obtient :

```
{2,3}
```

On tape dans HOME :

```
SIZE([[1,2,3],[4,5,6]])
```

On obtient :

```
{2,3}
```

Dans le CAS `dim` et `size` ne sont pas équivalents pour les matrices.
`dim` renvoie la liste donnant la dimension d'une matrice alors que `size` renvoie la longueur de la liste On tape dans le CAS :

```
dim([[1,2,3],[4,5,6]])
```

On obtient :

```
[2,3]
```

On tape dans le CAS :

```
size([[1,2,3],[4,5,6]])
```

On obtient :

```
2
```

17.12 +

+ de concaténe 2 chaines.
On tape :

```
"ABC"+"DEF"
```

On obtient :

```
"ABCDEF"
```

On peut aussi utiliser `CONCAT`
On tape :

```
CONCAT("ABC","DEF")
```

On obtient :

```
"ABCDEF"
```

17.13 Avoir la liste ou la chaîne privée de son premier élément : `tail`

`tail(s)` renvoie la liste ou la chaîne `s` 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)`
On tape :

```
tail("abcdef")
```

On obtient :

```
"bcdef"
```

`l:=tail("abcdef")` est équivalent à `l:=suppress("abcdef","a")`

17.14 Début d'une liste ou d'une chaîne : `head`

`head(s)` renvoie le premier élément de la liste `s` ou le premier caractère de la chaîne `s`.
On tape :

```
head([0,1,2,3])
```

On obtient :

```
0
```

On tape :

```
head("abcdef")
```

On obtient :

```
"a"
```

Chapitre 18

Les polynômes

18.1 POLYCOEF

POLYCOEF renvoie les coefficients d'un polynôme connaissant ses racines.
On tape :
`POLYCOEF([2,1])`
ou on tape :
`POLYCOEF(2,1)`
On obtient :
`poly1[1,-3,2]`
cela représente le polynôme $X^2 - 3X + 2 = (X - 2) * (X - 1)$
On tape :
`POLYCOEF([2,-1,3,-4])`
On obtient :
`poly1[1,0,-15,10,24]`
cela représente le polynôme $X^4 - 15X^2 + 10X + 24$

18.2 POLYEVAL

POLYEVAL renvoie l'écriture symbolique d'un polynôme donné par la liste de ses coefficients ou POLYEVAL évalue en un point, un polynôme donné par par la liste de ses ses coefficients.
On tape :
`POLYEVAL({1,0,-15,10,24})`
ou on tape :
`POLYEVAL([1,0,-15,10,24])`
On obtient :
`X^4-15*X^2+10*X+24`
On tape :
`POLYEVAL({1,0,-15,10,24},4)`
ou on tape :
`POLYEVAL([1,0,-15,10,24],4)`
On obtient :
`80`
car $4^4 - 15 * 4^2 + 10 * 4 + 24 = 80$

18.3 POLYFORM

POLYFORM développe un polynôme donné par une expression d'une ou plusieurs variables.
POLYFORM permet aussi de factoriser un polynôme d'une ou plusieurs variables et de faire la décomposition en éléments simples d'une fraction rationnelle.
On tape :
```
POLYFORM((X+2)^3+5)
```
ou `POLYFORM((X+2)^3+5,X)`
On obtient :
```
X^3+6*X^2+12*X+13
```
On tape :
```
POLYFORM((X+Y)^3+5)
```
ou
```
POLYFORM((X+Y)^3+5,X,Y)
```
On obtient :
```
X^3+3*X^2*Y+3*X*Y^2+Y^3+5
```
On tape :
```
POLYFORM((X+Y)^3+5,Y,X)
```
On obtient :
```
Y^3+3*Y^2*X+3*Y*X^2+X^3+5
```
On tape :
```
POLYFORM((X+2)^2+5,X)
```
On obtient :
```
X^2+4*X+8
```
On tape :
```
POLYFORM((X+Y)^2+5)
```
On obtient :
```
X^2+2*X*Y+Y^2+5
```
On tape :
```
POLYFORM((X^2+2*X+1)*(X-1)))
```
On obtient :
```
X^3+X^2-X-1)
```
Pour factoriser on tape :
```
POLYFORM(X^3+X^2-X-1,'*')
```
On obtient :
```
(X-1)*(X+1)^2
```
On tape :
```
POLYFORM(X^2-Y^2,'*')
```
On obtient :
```
(X-Y)*(X+Y)
```
On tape :
```
POLYFORM(1/(X^2-Y^2),'*')
```
On obtient :
```
1/((X-Y)*(X+Y))
```
Pour faire la décomposition en éléments simples d'une fraction rationnelle, on tape :
```
POLYFORM(1/(1-X^2)^2),'+'
```

On obtient :
```
-1/4/(X-1)^2-1/4/(X-1)+1/4/(X+1)^2+1/4/(X+1)
```
On tape :
```
POLYFORM(1/(X^2-Y^2)),'+'
```
On obtient :
```
1/(2*Y)/(X+Y)-1/(2*Y)/(X-Y)
```
Remarque
On peut aussi employer `STO` pour obtenir une réécriture d'une expression :
- `STO STO` pour évaluer formellement une expression,
- `STO +` pour développer ou pour faire une décomposition en éléments simples
- `STO *` pour factoriser

On tape depuis `Home` :
```
X*SIN(X) STO STO
```
On obtient :
```
X*SIN(X)
```
On tape :
```
∂(X*SIN(X),X) STO STO
```
On obtient :
```
SIN(X)+X*COS(X)
```
On tape :
```
∫(SIN(X),X) STO STO
```
On obtient :
```
-COS(X)
```
On tape :
```
(X+Y)^2+5 STO +
```
On obtient :
```
X^2+2*X*Y+Y^2+5
```
On tape :
```
1/(1-X^2)^2 STO +
```
On obtient :
```
-1/4/(X-1)^2-1/4/(X-1)+1/4/(X+1)^2+1/4/(X+1)
```
On tape :
```
X^3+X^2-X-1 STO *)
```
On obtient :
```
(X-1)*(X+1)^2
```

18.4 POLYROOT

`POLYROOT` renvoie les racines d'un polynôme connaissant ses coefficients.
On tape :
```
POLYROOT({1,0,-15,10,24})
```
Ou on tape :
```
POLYROOT([1,0,-15,10,24])
```
On obtient :
```
[-1,2,3,-4]
```
qui sont les 4 racines du polynôme $X^4 - 15X^2 + 10X + 24$

Chapitre 19

Les suites récurrentes

19.0.1 Valeurs d'une suite récurrente ou d'un système de suites récurrentes : `seqsolve`

Voir aussi `rsolve` 19.0.2.
`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) (cf19.0.2)
- 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]
```

19.0.2 Valeurs d'une suite récurrente ou d'un système de suites récurrentes : `rsolve`

Voir aussi `seqsolve` 19.0.1.

`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]]
  ```

Chapitre 20

Les matrices

20.1 Généralités

Pour écrire une matrice on met entre des crochets la suite des vecteurs lignes, par exemple : [[1,2],[3,4]].
Les matrices numériques sont stockées dams les variables $M_0, M_1 ... M_9$.
Les indices des lignes et des colonnes d'une matrice commencent à 1 et on met l'indice entre des crochets ou entre des parenthèses.

20.2 Définition

Pour définir la matrice M_1 égale à [[1,2],[3,4]], on tape :
`[[1,2],[3,4]]=>M1` ou
on utilise l'éditeur de matrices (SHIFT 4 (Matrix))
On obtient :
`[[1,2],[3,4]]`
Pour définir la matrice M_2 égale à [[1,2],[3,4],[5,6]], on tape :
`[[1,2],[3,4],[5,6]]=>M2` ou
on utilise l'éditeur de matrices (SHIFT 4 (Matrix))
On obtient :
`[[1,2],[3,4],[5,6]]`
Pour avoir l'élément `3` de M_2 situé au début de la deuxième ligne : ce sera l'élément sur la ligne d'indice 2 et sur la colonne d'indice 1 si on le désigne par `M2[2,1]` ou `M2(2,1)`.
On tape :
`M1[2,1]`
On obtient :
`3` On tape :
`M1(2,1)`
On obtient :
`3`

20.2.1 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]
```

20.2.2 Nombre de lignes : `rowDim`

`rowDim` a comme argument une matrice A.
`rowDim` renvoie le nombre de lignes de la matrice A.
On tape :

```
rowDim([[1,2,3],[3,4,5]])
```

On obtient :

```
2
```

20.2.3 Nombre de colonnes : `colDim`

`colDim` a comme argument une matrice A.
`colDim` renvoie le nombre de colonnes de la matrice A.
On tape :

```
colDim([[1,2,3],[3,4,5]])
```

On obtient :

```
3
```

20.3 Opérations sur les lignes et les colonnes utiles en programmation

20.3.1 Rajouter une colonne à une matrice : `ADDCOL`

`ADDCOL(M1,col,n` rajoute la colonne `col` qui sera la colonne d'indice `n` à la matrice `M1`.
On tape :

```
[[1,2],[3,4]]=>M1
ADDCOL(M1,[5,6],1)
```

On obtient la nouvelle matrice M_1 :

```
[[5,1,2],[6,3,4]]
```

On tape :

```
M1
```

On obtient la nouvelle matrice M_1 :

`[[5,1,2],[6,3,4]]`
Pour rajouter une dernière colonne (qui sera ici la colonne d'indice 3) à la matrice M_1, on tape :
`[[1,2],[3,4]]=>M1`
`ADDCOL(M1,[5,6],3)`
On obtient la nouvelle matrice M_1 :
`[[1,2,5],[3,4,6]]`
On tape :
`M1`
On obtient la nouvelle matrice M_1 :
`[[1,2,5],[3,4,6]]`

20.3.2 Échanger deux lignes : `SWAPROW rowSwap`

`SWAPROW` ou `rowSwap` a trois arguments : une matrice et deux entiers $n1$ et $n2$.
`SWAPROW` ou `rowSwap` renvoie la matrice obtenue en échangeant dans la matrice argument les lignes $n1$ et $n2$.
On tape dans HOME (les indices commencent à 1) :

```
SWAPROW([[1,2],[3,4]],1,2)
```

On obtient :

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

On tape dans CAS (les indices commencent aussi à 1) :

```
SWAPROW([[1,2],[3,4]],1,2)
```

ou

```
rowSwap([[1,2],[3,4]],1,2)
```

On obtient :

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

20.3.3 Échanger deux colonnes : `SWAPCOL colSwap`

`SWAPCOL` ou `colSwap` a trois arguments : une matrice et deux entiers $n1$ et $n2$.
`SWAPCOL` ou `colSwap` renvoie la matrice obtenue en échangeant dans la matrice argument les colonnes $n1$ et $n2$.
On tape dans HOME ou dans CAS (les indices commencent à 1) :

```
SWAPCOL([[1,2],[3,4]],1,2)
```

On obtient :

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

On tape dans CAS

```
SWAPCOL([[1,2],[3,4],[5,6]],1,2)
```

ou

```
colSwap([[1,2],[3,4],[5,6]],1,2)
```

On obtient :

```
[[2,1],[4,3],[6,5]]
```

20.3.4 Extraire des lignes d'une matrice : `row`

`row` permet d'extraire une ou plusieurs lignes d'une matrice.
`row` a 2 arguments : une matrice et un entier n ou un intervalle $n_1..n_2$.
`row` renvoie la ligne d'indice n de la matrice donnée en argument, ou la séquence des lignes d'indice allant de n_1 à n_2 de cette matrice.
On tape dans HOME ou dans CAS (les indices commencent à 1) : :

```
row([[1,2,3],[4,5,6],[7,8,9]],1)
```

On obtient :

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

On tape :

```
row([[1,2,3],[4,5,6],[7,8,9]],1..2)
```

On obtient :

```
[[1,2,3],[4,5,6]]
```

20.3.5 Extraire des colonnes d'une matrice : `col`

`col` permet d'extraire une ou plusieurs colonnes d'une matrice.
`col` a 2 arguments : une matrice, et un entier n ou un intervalle $n_1..n_2$.
`col` renvoie la colonne d'indice n de la matrice donnée en argument, ou la séquence des colonnes d'indice allant de n_1 à n_2 de cette matrice.
On tape dans HOME ou danc CAS (les indices commencent à 1) :

```
col([[1,2,3],[4,5,6],[7,8,9]],1)
```

On obtient :

```
[1,4,7]
```

On tape :

```
col([[1,2,3],[4,5,6],[7,8,9]],1..2)
```

On obtient :

```
([1,4,7],[2,5,8])
```

20.3.6 Supprimer des colonnes d'une matrice : `DELCOL delcols`

Dans HOME, `DELCOL(M1,n)` enlève la colonne d'indice `n` à la matrice `M1`.
Dans CAS, `delcols` a 2 arguments : une matrice A, et un entier n ou un intervalle $n_1..n_2$.
`delcols` renvoie la matrice obtenue en supprimant la colonne n ou les colonnes n_1 jusqu'à n_2 de la matrice A.
On tape :

```
[[1,2,5],[3,4,6]]=>M1
DELCOL(M1,2)
```

ou
`DELCOL(M1,2..2)`
On obtient :
`[[1,5],[3,6]]`
Pour enlever les colonnes d'indices 2 et 3 à la matrice M_1, on tape :
`[[1,2,5],[3,4,6]]=>M2`
`DELCOL(M1,2..3)`
On obtient :
`[[1],[3]]` On tape :

`delcols([[1,2,3],[4,5,6],[7,8,9]],2)`

On obtient :

`[[1,3],[4,6],[7,9]]`

On tape :

`delcols([[1,2,3],[4,5,6],[7,8,9]],1..2)`

On obtient :

`[[3],[6],[9]]`

20.3.7 Supprimer des lignes d'une matrice : `DELROW delrows`

Dans HOME, `DELROW(M1,n)` enlève la ligne d'indice `n` à la matrice `M1`
Dans CAS, `delrows` a 2 arguments : une matrice A, et un entier n ou un intervalle $n_1..n_2$.
`delrows` renvoie la matrice obtenue en supprimant la ligne n ou les lignes n_1 jusqu'à n_2 de la matrice A.
On tape :
`[[1,2],[3,4],[5,6]]=>M1`
`DELROW(M1,2)`
ou
`DELROW(M1,2..2)`
On obtient la nouvelle matrice M_1 :
`[[1,2],[5,6]]`
Pour enlever les lignes d'indice 2 et 3 à la matrice M_1, on tape :
`[[1,2],[3,4],[5,6]]=>M1`
`DELROW(M1,2..3)`
On obtient la nouvelle matrice M_1 :
`[[1,2]]` On tape :

`delrows([[1,2,3],[4,5,6],[7,8,9]],2)`

On obtient :

`[[1,2,3],[7,8,9]]`

On tape :

`delrows([[1,2,3],[4,5,6],[7,8,9]],1..2)`

On obtient :

`[[7,8,9]]`

20.3.8 Extraire une sous-matrice d'une matrice : `SUB subMat`

Dans HOME, `SUB` a 3 arguments une matrice $M1$, et 2 listes d'indices $\{nl1, nc1\}$, $\{nl2, nc2\}$.
Attention les indices commencent à 1.
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.
`SUB(M1,{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 tape :

```
SUB([[3,4,5],[1,2,6]],{1,2},{2,3})
```

On obtient :

```
[[4,5],[2,6]]
```

Dans CAS, `subMat` a 3 arguments une matrice A, et 2 listes d'indices $[nl1, nc1]$, $[nl2, nc2]$ ou $\{nl1, nc1\}$, $\{nl2, nc2\}$.
Attention les indices commencent aussi à 1. 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:=[[1,2,3],[4,5,6],[7,8,9]]
```

On tape :

```
subMat(A,[1,2],[2,3])
```

On obtient :

```
[[2,3],[5,6]]
```

20.3.9 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 on 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]
```

20.3.10 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 dans HOME :

```
REPLACE([[1,2,3],[4,5,6]],{1,1},[[5,6],[7,8]])
```

Ou on tape dans CAS :

```
REPLACE([[1,2,3],[4,5,6]],[1,1],[[5,6],[7,8]])
```

On obtient :

```
[[5,6,3],[7,8,6]]
```

On tape dans HOME :

```
REPLACE([[1,2,3],[4,5,6]],{1,2},[[7,8],[9,0]])
```

Ou on tape dans CAS :

```
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 dans HOME ou dans CAS :

```
REPLACE([1,2,3,4],2,[5,6])
```

On obtient :

```
[1,5,6,4]
```

On tape dans HOME ou dans CAS :

```
REPLACE([1,2,3,4],2,[5,6,7,8])
```

On obtient :

```
[1,5,6,7]
```

20.3.11 Rajouter une ligne à une matrice : ADDROW

`ADDROW(M1,row,n)` rajoute la ligne `row` qui sera la ligne d'indice `n` à la matrice `M1`.
On tape :
```
[[1,2],[3,4]]=>M1
ADDROW(M1,[5,6],1)
```
On obtient la nouvelle matrice M_1 :
```
[[5,6],[1,2],[3,4]]
```
Pour rajouter une dernière ligne (qui sera ici la ligne d'indice 3) à la matrice M_1, on tape :
```
[[1,2],[3,4]]=>M1
ADDROW(M1,[5,6],3)
```
On obtient la nouvelle matrice M_1 :
```
[[1,2],[3,4],[5,6]]
```

20.3.12 Ajouter une ligne à une autre : rowAdd

Dans le CAS, `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]],1,2)
```

On obtient :

```
[[1,2],[4,6]]
```

20.3.13 Multiplier une ligne par une expression : SCALE mRow

Attention `SCALE` et `mRow` n'ont pas leurs arguments dans le même ordre !
`SCALE` a trois arguments : une matrice A, une expression et un entier n.
`mRow` a trois arguments : une expression, une matrice A et un entier n.
`SCALE` ou `mRow` renvoie la matrice obtenue en remplacant dans A la ligne n par la multiplication de la ligne n par l'expression.
On tape :

```
SCALE([[1,2],[3,4]],12,2)
```

Ou on tape :

```
mRow(12,[[1,2],[3,4]],2)
```

On obtient :

```
[[1,2],[36,48]]
```

20.3.14 Ajouter k fois une ligne à une autre : SCALEADD mRowAdd

Attention SCALEADD et mRowAdd n'ont pas leurs arguments dans le même ordre !
SCALEADD a quatre arguments : une matrice A, un réel k et deux entiers $n1$ et $n2$.
mRowAdd a quatre arguments : un réel k, une matrice A et deux entiers $n1$ et $n2$.
mRowAdd 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 :

```
SCALEADD([[5,7],[3,4],[1,2]],1.1,2,3)
```

Ou on tape :

```
mRowAdd(1.1,[[5,7],[3,4],[1,2]],2,3)
```

On obtient :

```
[[5,7],[3,4],[4.3,6.4]]
```

20.4 Création et arithmétique des matrices

20.4.1 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]]
```

20.4.2 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]]
```

20.4.3 Elévation d'une matrice à une puissance entière : ^ &^

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)/
```

20.4.4 Produit de Hadamard (version infixée) : .*

Voir aussi **??** et **??**.
.* 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]]
```

20.4.5 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]]
```

20.4.6 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]]
```

20.5 Matrice transposée : `transpose`

Dans CAS, `transpose` renvoie la transposée de la matrice donnée enargument.
On tape :

```
transpose([[i,2],[4,5-i]])
```

On obtient :

```
[[i,4],[2,5-i]]
```

20.6 Matrice transposée de la conjuguée : `TRN` ou `trn`

Dans CAS, `TRN` renvoie la transposée de la conjuguée de la matrice donnée en argument. On tape :

```
TRN([[i,2],[4,5-i]])
```

Ou on tape :

```
trn([[i,2],[4,5-i]])
```

On obtient :

```
[[-i,4],[2,5+i]]
```

20.7 Déterminant : `DET` ou `det`

Dans HOME, `DET` renvoie le déterminant d'une matrice carrée.
On tape :

```
DET([[1/2,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
-1.5
```

Dans CAS, `DET` ou `det` renvoie le déterminant d'une matrice carrée.
On tape :

```
DET([[1/2,2,4],[4,5,6],[7,8,9]])
```

Ou on tape :

```
det([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
-3/2
```

20.7.1 Polynôme caractéristique : `charpoly`

`charpoly` a un (resp deux)argument(s).
`charpoly` a comme argument une matrice A d'ordre n (resp une matrice A d'ordre n et un nom de variable formelle).
`charpolyp` 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 :

```
charpoly([[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):;charpoly([[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.

20.8 Espace vectoriel et application linéaire

20.8.1 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]]
```

20.8.2 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]]
```

20.8.3 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]]
```

20.8.4 Noyau d'une application linéaire : `ker`

`ker` a comme argument la matrice d'une application linéaire f dans la base canonique.
`ker` 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]`.

20.9 Résolution d'un système linéaire : RREF ou rref

Dans HOME, RREF permet de résoudre un systéme linéaire de matrice M_1 et de second membre $M3$.
ADDCOL(M1,M3)=>M2 et RREF(M2) renvoie la forme réduite échelonnée de M_2.
Par exemple, on veut résoudre le système : $\{3x + y = 2, 3x + 2y = -2\}$ par rapport à x, y.
On tape :

```
RREF([[3,1,2],[3,2,-2]])
```

On obtient :

```
[[1,0,2],[0,1,-4]]
```

et on en déduit que $x = 2$ et $y = -4$.
On veut résoudre le système :
$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ par rapport à x, y, z.
On tape :

```
RREF([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

On obtient :

```
[[1,0,-0.333333333333,4],[0,1,-0.666666666667,1],[0,0,0,0]]
```

et on en déduit que $x = 4 + z/3$, $y = 1 + 2z/3$ et $z = z$.

Dans CAS, rref ou RREF renvoie la forme réduite échelonnée de la matrice argument.
Par exemple, on veut résoudre le système : $\{3x + y = 2, 3x + 2y = -2\}$ par rapport à x, y.
On tape :

```
rref([[3,1,2],[3,2,-2]])
```

Ou on tape :

```
RREF([[3,1,2],[3,2,-2]])
```

On obtient :

```
[[1,0,2],[0,1,-4]]
```

et on en déduit que $x = 2$ et $y = -4$.
On veut résoudre le système :
$\{x + y - z = 5, 2x - y = 7, x - 2y + z = 2\}$ par rapport à x, y, z.
On tape :

```
rref([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

Ou on tape :

```
RREF([[1,1,-1,5],[2,-1,0,7],[1,-2,1,2]])
```

On obtient :

```
[[1,0,-1/3,4],[0,1,-2/3,1],[0,0,0,0]]
```

et on en déduit que $x = 4 + z/3$, $y = 1 + 2z/3$ et $z = z$.

20.9.1 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 6.10.17) :

```
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

$$\begin{cases} 3x + y &= 1 \\ 3x + 2y &= 2 \end{cases}$$

20.10 Création de matrices

20.10.1 Créer une matrice à partir d'une expression : `MAKEMAT` et `makemat`

Dans HOME, `MAKEMAT(Expr(I,J),n,p)` crée une matrice à partir d'une expression selon les variables I et J.
I représente l'indice des lignes et J représente l'indice des colonnes et l"indice I va de 1 à n et l'indice J va de 1 à p.
`MAKEMAT(Expr(I,J),n,p)` renvoie la matice $M_{I,J} = Expr(I, J)$ pour $I = 1..n$ et $J = 1..p$.
On tape :

```
MAKEMAT(I*J,2,3)
```

On obtient :

```
[[2,3,4],[3,4,5]]
```

Dans CAS on peut utiliser `MAKEMAT` et `makemat`. `makemat` a une fonction comme premier argument : la première variable est l'indice de la ligne et la deuxième variable est l'indice de la colonne. Le deuxième argument représente le nombre de ligne et le troisième argument représente le nombre de colonne.
Attention les indices commencent aussi à 1.
On tape :

```
MAKEMAT((I+J),2,3)
```

Ou on tape :

```
makemat((j,k)->(j+k),2,3)
```

On obtient :

```
[[0,1,2],[1,3,5]]
```

20.10.2 Matrice de zéros : `matrix`

`matrix(n,p)` renvoie la matrice de `n` lignes et de `p` colonnes formée par des zéros.
On tape :

```
matrix(4,3)
```

On obtient :

```
[[0,0,0],[0,0,0],[0,0,0],[0,0,0]]
```

20.10.3 Matrice identité : `IDENMAT` ou `identity`

Dans HOME, `IDENMAT(n)` crée la matrice identité d'ordre n.
On tape :

```
IDENMAT(3)
```

On obtient la matrice identité d'ordre 3 :

```
[[1,0,0],[0,1,0],[0,0,1]]
```

Dans CAS,`IDENMAT(n)` crée la matrice identité d'ordre n.
On tape :

```
IDENMAT(3)
```

Ou on tape :

```
identity(3)
```

On obtient la matrice identité d'ordre 3 :

```
[[1,0,0],[0,1,0],[0,0,1]]
```

20.10.4 Matrice aléatoire : `RANDMAT` et `randMat randmatrix ramn`

Dans HOME, `RANDMAT(M1,n,p)` crée une matrice aléatoire M_1 de n lignes et p colonnes et formée d'entiers entre -99 et +99.
On tape :

```
RANDMAT(M1,2,3)
```

On obtient :

```
[[-24.0,-67.0,38.0],[-73.0,-3.0,72.0]]
```

On tape :

```
M1
```

On obtient :

```
[[-24,-67,38],[-73,-3,72]]
```

Dans CAS on n'a que 2 arguments : `RANDMAT(n,p)` crée une matrice aléatoire de n lignes et p colonnes et formée d'entiers entre -99 et +99.
On tape :

```
RANDMAT(2,3)
```

Dans CAS ou dans HOME, `randMat(n,p)` ou `randmatrix` ou `ranm(n,p)` crée une matrice aléatoire de n lignes et p colonnes et formée d'entiers entre -99 et +99.
On tape :

```
randMat(2,3)
```

Ou on tape :

```
randmatrix(2,3)
```

Ou on tape :

```
ranm(2,3)
```

On obtient par exemple :

```
[[-57,17,39],[-61,23,4]]
```

20.10.5 JordanBlock

Dans HOME, on l'utilise sous la forme `CAS.JordanBlock(a,n)` car c'est une fonction du CAS.

Dans CAS, `JordanBlock(a,n)` renvoie une matrice carrée d'ordre n avec a sur la diagonale principale, 1 au-dessus de cette diagonale et 0 ailleurs.
On tape :

```
JordanBlock(7,3)
```

On obtient :

```
[[7,1,0],[0,7,1],[0,0,7]]
```

20.10.6 N-ième matrice de Hilbert hilbert

`hilbert` s'utilise dans le CAS (dans HOME il faut mettre `CAS.hilbert`).
`hilbert(n)` renvoie la n-ième matrice de Hilbert c'est à dire :
$H_{j,k} = \dfrac{1}{j+k+1}$ pour $j = 1..n$ et $k = 1..n$.
On tape :

```
hilbert(3)
```

On obtient :

```
[[1,1/2,1/3],[1/2,1/3,1/4],[1/3,1/4,1/5]]
```

20.10.7 Matrice d'une isométrie : mkisom

`mkisom` s'utilise dans le CAS (dans HOME il faut mettre `CAS.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]]
```

On tape pour avoir la matrice en dimension 2 d'une rotation de centre O et d'angle 1 :

```
mkisom(1,1)
```

On obtient :

```
[[cos(1),-sin(1)],[sin(1),cos(1)]]
```

20.10.8 Matrice de Vandermonde : `vandermonde`

`vandermonde` s'utilise dans le CAS (dans HOME il faut mettre `CAS.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-1} ($k = 1..n$).
Attention !
La calculatrice numérote les lignes et les colonnes à partir de 1.
On tape :

```
vandermonde([a,2,3])
```

On obtient (si `a` n'est pas affecté) :

```
[[1,1,1],[a,2,3],[a*a,4,9]]
```

20.11 Basique

20.11.1 Norme de Schur ou de Frobenius d'une matrice : `ABS`

On rappelle que `ABS` permet d'avoir :
- la valeur absolue d'un réel,
- le module d'un nombre complexe,
- la longueur d'un vecteur v_j ($(\sum_{j=1}^{n} |v_j|^2)^{1/2}$),
- la norme de Schur ou de Frobenius d'une matrice $a_{j,k}$: $(\sum_{j,k=1}^{n} |a_{j,k}|^2)^{1/2}$.

On tape :

```
ABS([[1,2],[3,4]])
```

On obtient :

```
sqrt(30)
```

en effet $\sqrt{1+4+9+16)} = \sqrt{30}$
On tape :

```
ABS([[1,2],[3,4],[5,11]])
```

On obtient :

```
4*sqrt(11)
```

en effet $\sqrt{1+4+9+16+25+121)} = \sqrt{176} = 4\sqrt{11}$
Dans CAS, on tape :

```
abs([[i,2],[4,5+i]])
```

Ou on tape `||` avec la touche située à côté de la malette :

```
|[[i,2],[4,5+i]]|
```

On obtient :

```
sqrt(47)
```

car $\sqrt{1+4+16+25+1} = \sqrt{47}$
Dans HOME, on tape :

```
ABS([[i,2],[4,5+i]])
```

Ou on tape `||` avec la touche située à côté de la malette :

```
|[[i,2],[4,5+i]]|
```

On obtient :

```
6.8556546004
```

car $\sqrt{1+4+16+25+1} = \sqrt{47} \simeq 6.8556546004$

20.11.2 Maximum des normes des lignes d'une matrice : `ROWNORM` ou `rownorm`

Dans HOME, `ROWNORM` a comme argument une matrice.
`ROWNORM` renvoie le maximum des normes des lignes de cette matrice (la norme d'une ligne étant la somme des valeurs absolues des composantes de la ligne).
On tape :

```
ROWNORM([[1,-2,3],[4,5,-6]])
```

On obtient :

```
15
```

en effet on a $1+2+3 = 6 < 4+5+6 = 15$
Dans CAS, `ROWNORM` ou `rownorm` a comme argument une matrice.
`ROWNORM` ou `rownorm` renvoie le maximum des normes des lignes de cette matrice (la norme d'une ligne étant la somme des valeurs absolues des composantes de la ligne).
On tape :

```
ROWNORM([[1,-2,3],[4,5,-6]])
```

Ou on tape :

```
rownorm([[1,-2,3],[4,5,-6]])
```

On obtient :

```
15
```

en effet on a $1+2+3 = 6 < 4+5+6 = 15$

20.11.3 Max des normes des colonnes d'une matrice : COLNORM ou colnorm

Dans HOME, COLNORM a comme argument une matrice.
COLNORM renvoie le maximum des normes des colonnes de cette matrice (la norme d'une colonne étant la somme des valeurs absolues des composantes de la colonne).
On tape :

```
COLNORM([[1,-2,3],[4,5,-6]])
```

On obtient :

```
9
```

en effet on a $1+4<2+5<3+6=9$
Dans CAS, COLNORM ou colnorm a comme argument une matrice.
COLNORM ou colnorm renvoie le maximum des normes des colonnes de cette matrice (la norme d'une colonne étant la somme des valeurs absolues des composantes de la colonne).
On tape :

```
COLNORM([[1,-2,3],[4,5,-6]])
```

Ou on tape :

```
colnorm([[1,-2,3],[4,5,-6]])
```

On obtient :

```
9
```

en effet on a $1+4<2+5<3+6=9$

20.11.4 Norme spectrale d'une matrice : SPECNORM

Dans HOME, SPECNORM sécrit CAS.SPECNORM et a comme argument une matrice M_1.
SPECNORM renvoie la norme spectrale de cette matrice M_1 : c'est la plus grande valeur singulière de la matrice M_1 i.e. la racine de la plus grande valeur propre de la matrice symétrique M_1*TRN(M_1).
On tape :

```
CAS.SPECNORM([[1,1],[0,2]])
```

On obtient :

```
2.28824561127
```

car
`eigenvals([[1,1],[0,2]]*trn([[1,1],[0,2]]))=[sqrt(5)+3,-sqrt(5)+3]`
`EIGENVAL([[1,1],[0,2]]*TRN([[1,1],[0,2]]))=[0.7639320225,5.2360679775]`
et
`SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]` et

$\sqrt{\sqrt{5.}+3} \sim 2.28824561127$,

Dans CAS, SPECNORM a comme argument une matrice A.
SPECNORM renvoie la norme spectrale de cette matrice A : c'est la plus grande valeur singulière de la matrice A i.e. la racine de la plus grande valeur propre de la matrice symétrique $A*$TRAN(A).
On tape :

```
SPECNORM([[1,1],[0,2]])
```

On obtient :

```
2.28824561127
```

car
`eigenvals([[1,1],[0,2]]*trn([[1,1],[0,2]]))=[sqrt(5)+3,-sqrt(5)+3]`
`EIGENVAL([[1,1],[0,2]]*TRN([[1,1],[0,2]]))=[0.7639320225,5.2360679775`
et
`SVL([[1,1],[0,2]])=[sqrt(sqrt(5)+3),sqrt(-sqrt(5)+3)]` et
$\sqrt{\sqrt{5.}+3} \sim 2.28824561127$. En effet `SVL(A)`), renvoie la liste des valeurs singulières (i.e. les racines carrées positives des valeurs propres de $A * trn(A)$) de la matrice A numérique réelle donnée en argument.

20.11.5 Rayon spectral d'une matrice carrée : SPECRADC

Dans HOME, SPECRAD sécrit CAS.SPECRAD et a comme argument une matrice carrée.
SPECRAD renvoie le rayon spectral de cette matrice carrée : le rayon spectral est ègal à la plus grande valeur propre en valeur absolue.
On tape :

```
CAS.SPECRAD([[1,1],[0,-2]])
```

On obtient :

```
2.
```

Dans CAS, SPECNORM a comme argument une matrice A.
SPECNORM renvoie le rayon spectral de cette matrice carrée : le rayon spectral est ègal à la plus grande valeur propre en valeur absolue.
On tape :

```
SPECRAD([[1,1],[0,-2]])
```

On obtient :

```
2.
```

20.11.6 Conditionnement d'une matrice carrée inversible : COND cond

Dans HOME (resp dans CAS), COND (resp cond) a comme argument une matrice carrée inversible et un deuxième argument 1 ou 2 ou ∞ (obtenu avec Shift 9) (resp 1 ou 2 ou inf) et par défaut c'est 1.
désignant la norme utilisée (l_1 ou l_2 ou l^∞).
COND renvoie le conditionnement de cette matrice carrée inversible pour la norme spécifiée :

- Si deuxième argument est 1 le conditionnement d'une matrice carrée inversible est le produit de la norme colonne (c'est colnorm) de cette matrice par la norme colonne de son inverse.
  La norme colonne de M_1 de dimension p, q est :
  $\texttt{MAX}_{1\leq k\leq q}(\sum_{j=1}^{p}\texttt{ABS(M1[j,k])})$.
  On tape :

  ```
  COND([[1,2],[5,6]])
  ```

  ou

  ```
  cond([[1,2],[5,6]])
  ```

  On obtient :

  ```
  22
  ```

  On a en effet :
  Norme colonne de $[[1,2],[5,6]]$ est 2+6=8
  Norme colonne de $[[1,2],[3,-4]]^{-1} = [[-1.5,0.5],[1.25,-0.25]]$ est 1.5+1.25=2.75.
  On a bien 2.75*8=22.

- Si deuxième argument est 2 le conditionnement d'une matrice carrée inversible est le produit de la norme pectrale (c'est SPECNORM et c'est aussimax(SVL())) de cette matrice par la norme pectrale de son inverse.
  On tape :

  ```
  COND([[1,2],[3,-4]],2)
  ```

  ou

  ```
  cond([[1,2],[3,-4]],2)
  ```

  On obtient :

  ```
  2.6180339888
  ```

  On a en effet :
  SPECNORM(INV([[1,2],[3,-4]]))=5.116672736
  SPECNORM(INV([[1,2],[3,-4]]))=0.5116672736.
  On a bien 5.116672736*0.5116672736=2.6180339888.

- Si deuxième argument est inf le conditionnement d'une matrice carrée inversible est le produit de la norme ligne (c'est rownorm) de cette matrice par la norme ligne de son inverse.
  La norme ligne de M_1 de dimension p, q est :
  $\texttt{MAX}_{1\leq j\leq p}(\sum_{k=1}^{q}\texttt{ABS(M1[j,k])})$.
  On tape :

  ```
  COND([[1,2],[5,6]],inf)
  ```

  ou

```
cond([[1,2],[5,6]],inf)
```

On obtient :

```
22.0
```

On a en effet :
Norme ligne de $[[1,2],[5,6]]$ est 5+6=11
Norme ligne de $[[1,2],[3,-4]]^{-1} = [[-1.5,0.5],[1.25,-0.25]]$ est 1.5+0.5=2.
On a bien 2*11=22.

20.11.7 Rang d'une matrice : `RANK` ou `rank`

Dans HOME, `RANK` renvoie le rang de la matrice donnée en argument.
On tape :

```
RANK([[1,2,3],[4,5,6]])
```

On obtient :

```
2
```

Dans CAS, `RANK` ou `rank` renvoie le rang de la matrice donnée en argument.
On tape :

```
RANK([[1,2,3],[4,5,6]])
```

Ou on tape :

```
rank([[1,2,3],[4,5,6]])
```

On obtient :

```
2
```

20.11.8 Étape de la réduction de Gauss-Jordan d'une matrice : `pivot`

`pivot` s'utilise dans le CAS (dans HOME il faut mettre `CAS.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]]
```

20.11.9 Trace d'une matrice carrée : TRACE ou trace

Dans HOME, TRACE renvoie la trace de la matrice carrée donnée en argument.
On tape :

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

On obtient :

```
14.5
```

On tape :

```
TRACE([[i,2],[4,5-i]])
```

On obtient :

```
5
```

Dans CAS, TRACE ou trace renvoie la trace de la matrice carrée donnée en argument.
On tape :

```
TRACE([[1/2,2,3],[4,5,6],[7,8,9]])
```

Ou on tape :

```
trace([[1,2,3],[4,5,6],[7,8,9]])
```

On obtient :

```
29/2
```

On tape :

```
TRACE([[i,2],[4,5-i]])
```

Ou on tape :

```
trace([[i,2],[4,5-i]])
```

On obtient :

```
5
```

20.12 Avancée

20.12.1 Valeurs propres : EIGENVAL et eigenvals

Dans HOME, EIGENVAL renvoie le vecteur des valeurs propres calculables d'une matrice numérique diagonalisable.
On tape :

```
EIGENVAL([[1,1],[0,2]])
```

On obtient :

```
[2,1]
```

Dans CAS `eigenvals` renvoie le vecteur des valeurs propres calculables d'une matrice.
On tape :

```
eigenvals([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[1,1,1]
```

On tape :

```
eigenvals([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

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

On tape :

```
eigenvals([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[1,1,1]
```

On tape :

```
eigenvals([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

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

20.12.2 Vecteurs propres : `EIGENVV` et `eigenvects`

Dans HOME, `EIGENVV` renvoie la liste de deux matrices celle des vecteurs propres et celle des valeurs propres calculables d'une matrice numérique diagonalisable.
On tape :

```
EIGENVV([[1,1],[0,2]])
```

On obtient :

```
{[[0.707106781187,-1.41421356237],[0.707106781187,0]],[[2,0],[0,1]]}
```

Dans CAS `eigenvects` ou `eigVc` renvoie la matrice des vecteurs propres d'une matricediagonalisable.
On tape :

```
eigenvects([[1,1],[0,2]])
```

ou

```
eigVc([[1,1],[0,2]])
```

On obtient :

```
[[1,-1],[1,0]]
```

On tape :

```
eigenvects([[1,1,2],[0,1,1],[0,0,1]])
```

ou

```
eigVc([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
"Non diagonalisable, Try Jordan"
```

On tape :

```
eigenvects([[1,1,2],[0,2,1],[0,0,3]])
```

ou

```
eigVc([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[[3,-1,1],[2,-1,0],[2,0,0]],[[3,0,0],[0,2,0],[0,0,1]]
```

20.12.3 Matrice de Jordan : `eigVl`

`eigVl`) a comme argument une matrice d'ordre n.
`eigVl` renvoie la matrice de Jordan associée à cette matrice.
Remarque : Si la matrice est symbolique, on peut avoir en réponse des valeurs propres numériques car il faut que le CAS 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]]
```

20.12.4 Matrice de Jordan et sa matrice de passage : `jordan`

`jordan` s'utilise dans le CAS (dans HOME il faut mettre `CAS.jordan` et le résultat sera exact).
`jordan` renvoie la liste formée par la matrice de passage et la forme de Jordan d'une matrice.
On tape :

```
jordan([[1,1,2],[0,2,1],[0,0,3]])
```

On obtient :

```
[[3,-1,1],[2,-1,0],[2,0,0]]
```

On tape :

```
jordan([[1,1,2],[0,1,1],[0,0,1]])
```

On obtient :

```
[[1,2,0],[0,1,0],[0,0,1]],[[1,1,0],[0,1,1],[0,0,1]]
```

Si `A=[[1,1,2],[0,1,1],[0,0,1]]`, `P=[[1,2,0],[0,1,0],[0,0,1]]` et `B=[[1,1,0],[0,1,1],[0,0,1]]`, on a :
`inv(P)*A*P` renvoie `B`.

20.12.5 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]]`

20.12.6 Matrice diagonale et sa diagonale : `diag`

`diag` s'utilise dans le CAS (dans HOME il faut mettre `CAS.diag`).
Lorsque `diag` a comme argument une matrice, `diag` renvoie le vecteur form̀ par les éléments de sa diagonale.
Lorsque `diag` a comme argument un vecteur, `diag` renvoie la matrice diagonale de diagonale les éléments de ce vecteur. On tape :

```
diag([[1,0],[0,2]])
```

On obtient :

```
[1,2]
```

On tape :

```
diag([1,2])
```

On obtient :

```
[[1,0],[0,2]]
```

20.12.7 Matrice de Cholesky : `cholesky`

`cholesky` s'utilise dans le CAS (dans HOME il faut mettre `CAS.cholesky`).
`cholesky` a comme argument une matrice symétrique `A`.
`cholesky` renvoie la matrice `L` tel que `A=L*tran(L)`
On tape :

```
cholesky([[3,1],[1,4]])
```

On obtient :

```
[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]
```

et on a bien `[[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]]*tran([[3/(sqrt(3)),0],[1/(sqrt(3)),(sqrt(33))/3]])` renvoie `[[3,1],[1,1/3+11/3]]`

20.12.8 Forme normale de Hermite d'une matrice : `ihermite`

`ihermite` calcule la forme normale de Hermite pour une matrice A à coefficients entiers.
`ihermite` renvoie U, B tels que U est inversible dans $\mathbb{Z}$, B est triangulaire supérieure et $B = U * A$.
On tape :

```
ihermite([[1,2],[2,3]])
```

On obtient :

```
[[-3,2],[2,-1]],[[1,0],[0,1]]
```

20.12.9 Réduction de Hessenberg d'une matrice : `hessenberg`

`hessenberg` s'utilise dans le CAS (dans HOME il faut mettre `CAS.hessenberg`).
`hessenberg` a comme premier argument une matrice A et comme deuxième argument 0, -1 ou -2 ou $n > 1$ et n premier.
`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.
- Avec comme deuxième argument -1, les calculs sont approchés et la matrice B est triangulaire.
- 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.
- Avec comme deuxième argument $n > 1$ et n premier, les calculs sont modulo n et la matrice B est triangulaire.

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]])
```

On obtient :

```
[[[1,0,0],[0,4/7,1],[0,1,0]],
 [[1,29/7,2],[7,39/7,8],[0,278/49,3/7]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-1)
```

On obtient :

```
[[[-0.293737737475,0.802770468103,0.518919759814],
  [-0.69005396727,-0.553745992027,0.466037443312],
  [-0.661470833702,0.221189854777,-0.716611041155]],
   [[12.4541647409,-2.25953233593,-4.26290461387],
  [8.03071937292e-17,-0.379762185881,0.849798726727],
[4.52383345971e-20,-9.66694414605e-19,-5.07440255497]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],-2)
```

On obtient :

```
[[[1,0.0,0.0],[0,0.496138938357,0.868243142124],
      [0,0.868243142124,-0.496138938357]],
      [[1.0,3.59700730309,0.248069469178],
    [8.0622577483,8.01538461538,6.27692307692],
      [0,4.27692307692,-2.01538461538]]]
```

On tape :

```
hessenberg([[1,2,3],[4,5,6],[7,8,1]],3)
```

On obtient :

```
[[[1,0,0],[0,1,0],[0,1,1]],[[1,-1,0],[1,-1,0],[0,1,1]]]
```

20.12.10 ismith

`ismith` s'utilise dans le CAS (dans HOME il faut mettre `CAS.ismith`).
`ismith(A)` calcule la forme normale de Smith de la matrice A à coefficients entiers et renvoie les matrices U, B, V avec U et V inversibles dans $\mathbb{Z}$ et où B est une matrice diagonale telle que $B[j,j]$ divise $B[j+1,j+1]$ et $B = U * A * V$.
On tape :

```
ismith([[1,2],[2,3]])
```

On obtient :

```
[[1,0],[2,-1]],[[1,0],[0,1]],[[1,-2],[0,1]]
```

On tape :

```
ismith([[9,-36,30],[-36,192,-180],[30,-180,180]])
```

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]]
```

20.13 Factorisation

20.13.1 Décomposition LQ d'une matrice : LQ

Dans HOME LQ(M1), (resp dans CAS LQ(A)), renvoie la décomposition LQ d'une matrice numérique M_1 (resp A) de dimension $m \times n$ en une matrice triangulaire inférieure M_2 (resp L) de dimension $m \times n$, une matrice orthogonale M_3 (resp Q) de dimension $n \times n$ et une matrice de permutation M_4 (resp P) de dimension $n \times n$ telles que l'on ait $M_4 * M_1 = M_2 * M_3$ (resp $P * A = L * Q$).
0n tape :

```
LQ([[4,0,0],[8,-4,3]])
```

On obtient :

```
[[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]]
```

0n tape :

```
LQ([[0.8,0.6],[2.2,0.4]])
```

On obtient :

```
[[1.0,0],[2.0,-1.0]],[[0.8,0.6],
[-0.6,0.8]],[[1,0],[0,1]]
```

0n tape :

```
LQ([[4,3],[11,2]])
```

On obtient :

```
[[5.0,0],[10.0,-5.0]],[[0.8,0.6],
[-0.6,0.8]],[[1,0],[0,1]]
```

0n tape :

```
LQ([[1,2],[3,4]])
```

On obtient :

```
[[2.2360679775,0.],[4.9193495505,0.894427191]],
[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]],
[[1,0],[0,1]]]
```

ce qui signifie que :
`[[1,2],[3,4]]=[[2.2360679775,0.0],[4.9193495505,0.894427191]]*`
`[[0.4472135955,0.894427191],[0.894427191,-0.4472135955]]`
0n tape :

```
[[1,2,3],[3,4,5],[5,6,7]]=>M3
```

```
LQ(M3)
```

On obtient :

```
[[3.74165738677,0,0],[6.94879228972,1.30930734142,0],
          [10.1559271927,2.61861468283,1]],
   [[0.267261241912,0.534522483825,0.801783725737],
   [0.872871560944,0.218217890236,-0.436435780472],
[-9.09494701773e-13,-2.27373675443e-13,6.8212102633e-13]],
             [[1,0,0], [0,1,0],[0,0,1]]
```

20.13.2 Norme minimale du système linéaire `A*X=B` : `LSQ`

`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 $sqrt\sum |M(j,k)|^2$).

- 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)

0n tape :

```
LSQ([[1,2],[3,4]],[[5,-1],[11,-1]])
```

On obtient :

```
[[1,1],[2,-1]]
```

0n tape :

```
LSQ([[1,2]],[[5,-1]])
```

On obtient :

```
[[1,-1.2],[2,-0.4]]
```

0n tape :

```
LSQ([[1,2],[3,4],[3,6]],[[5,-1],[11,-1],[15,-3]])
```

On obtient :

```
[[1,1],[2,-1]]
```

0n tape :

```
LSQ([[1,2],[3,4],[3,6]],[[5,-1],[11,-1],[15,-1]])
```

On obtient :

```
[[1,-0.2],[2,-0.1]]
```

20.13.3 Décomposition LU d'une matrice carrée : LU

Attention LU et lu ne renvoient pas la même chôse.
Dans HOME LU(M1), renvoie la décomposition LU d'une matrice carrée M_1 (resp $A1$) en une matrice M_2 (resp L) triangulaire inférieure (de diagonale 1) et une matrice M_3 (resp U) triangulaire supérieure telles que si M_4 (resp P) est une matrice de permutation on ait $M_4 * M_1 = M_2 * M_3$ (resp $P * A1 = L * U$).
0n tape :

```
LU([[1,2],[1,4]])
```

On obtient :

```
{[[1,0],[1,1]],[[1,2],[0,2]],[[1,0],[0,1]]}
```

0n tape :

```
LU([[1,2],[3,4]])
```

On obtient :

```
{[[1,0],[0.333333333333,1]],
 [[3,4],[0,0.666666666667]], [[0,1],[1,0]]}
```

car on a choisit de mettre des 1 sur la diagonale de L.
Cela signifie que :
`[[0,1],[1,0]]*[[1,2],[3,4]])= [[3,0],[1,0.666666]]*[[1,1.333333],[0,1]]`
On tape :

```
LU([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
[[[1,0,0],[4,1,0],[7,2,1]],
 [[1,2,4],[0,-3,-10],[0,0,1]],[0,1,2]]
```

ce qui signifie que :
`[[1,2,4],[4,5,6],[7,8,9]]=`
`[[1,0,0],[4,1,0],[7,2,1]]*[[1,2,4],[0,-3,-10],[0,0,1]]`
car la matrice associée à la permutation [0,1,2] est la matrice identité d'ordre 3.
0n tape :

```
LU([[6,12,18],[5,14,31],[3,8,18]])
```

On obtient :

```
[[[1,0,0],[2,1,0],[5/3,(-1)/6,1]],[[3,8,18],[0,-4,-18],[0,0,-2]],[2,0,1]]
```

ce qui signifie que :
`[[0,0,1],[1,0,0],[0,1,0]]*[[6,12,18],[5,14,31],[3,8,18]]=`
`[[1,0,0],[2,1,0],[5/3,(-1)/6,1]]*[[3,8,18],[0,-4,-18],[0,0,-2]]`
car la matrice associée à la permutation [2,0,1] est la matrice [[0,0,1],[1,0,0],[0,1,0]].

20.13.4 Décomposition LU : `lu`

Attention `LU` et `lu` ne renvoient pas la même chôse.
Dans HOME (resp dans CAS) `lu` a comme argument une matrice carrée $M1$ (resp $A1$) d'ordre n (numérique ou symbolique).
`lu(M1)` renvoie une permutation p de $1..n$ (car dans HOME les indices commencent à 1), une matrice triangulaire inférieure L avec des 1 sur sa diagonale et une matrice triangulaire supérieure U.
`lu(A)` renvoie une permutation p de $1..n$ (car dans CAS les indices commencent aussi à 1), une matrice triangulaire inférieure L avec des 1 sur sa diagonale et une matrice triangulaire supérieure U.
Ces matrices sont telles que :

- $P*M1 = L*U$ (resp $P*A1 = L*U$), où P est la matrice de permutation associée à p (que l'on peut calculer dans CAS avec `P:=permu2mat(p)`),
- l'équation $A*x = B$ équivaut à :

$$L*U*x = P*B = p(B) \text{ where } p(B) = [b_{p(1)}, b_{p(2)}..b_{p(n)}], \quad B = [b_1, b_2..b_n]$$

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 dans HOME :

```
(p,L,U):=lu([[3,5],[4,5]])
```

On obtient :

```
[1,2],[[1,0],[1.3333333333,1]],[[4,5],[0,-1.6666666667]]
```

On tape dans CAS :

```
(p,L,U):=lu([[3,5],[4,5]])
```

On obtient :

```
[1,2],[[1,0],[4/3,1]],[[3,5],[0,-5/3]]
```

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)*A1; 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 dans CAS :

```
lu([[1,2],[3,4]])
```

On obtient :

```
[1,2],[[1,0],[3,1]],[[1,2],[0,-2]]
```

20.13.5 Décomposition QR d'une matrice carrée : `QR qr`

Dans HOME `QR(M1)`, (resp dans CAS `qr(A)`), renvoie la décomposition QR d'une matrice carrée M_1 (resp A) en une matrice Q orthogonale et une matrice R triangulaire supérieure telles que si P est une matrice de permutation on ait $M_1 * P = Q * R$ (resp $A * P = Q * R$).
0n tape :

```
QR([[4,11,-2],[3,2,11]])
```

ou

```
qr([[4,11,-2],[3,2,11]])
```

On obtient :

```
{[[0.8,-0.6],[0.6,0.8]],[[5.0,10.0,5.0],[0,-5.0,10.0]],
                    [[1,0],[0,1]]}
```

0n tape :

```
QR([[1,2],[3,4]])
```

ou

```
qr([[1,2],[3,4]])
```

On obtient :

```
          [[0.316227766017,0.948683298051],
          [0.94868329805,-0.316227766017]],
[[3.16227766017,4.42718872424], [0.0,0.632455532034]],
                    [[1,0],[0,1]]
```

ce qui signifie que :
`[[0.316227766017,0.948683298051],[0.94868329805,-0.316227766017]]*`
`[[3.16227766017,4.42718872424],[0.0,0.632455532034]]=`
`[[1,2],[3,4]]`
0n tape :

```
QR([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient :

```
              [[[1,0,0],[4,1,0],[7,2,1]],
        [[1,2,4],[0,-3,-10],[0,0,1]],[0,1,2]]
```

ce qui signifie que :
`[[1,2,4],[4,5,6],[7,8,9]]=`
`[[1,0,0],[4,1,0],[7,2,1]]*[[1,2,4],[0,-3,-10],[0,0,1]]`

20.13.6 Réduction de Hessenberg d'une matrice : SCHUR schur

Dans HOME SCHUR(M1), (resp dans CAS schur(A)), renvoie les matrices numériques $[P, B]$ tel que $B = inv(P) * M1 * P$ (resp $B = inv(P) * A * P$) avec B triangulaire. On a SCHUR(A)=hessenberg(A,-1). B est la matrice de Hessenberg semblable à la matrice M_1 (res A).
On tape :

```
SCHUR([[1,2,3],[4,5,6],[-1,3,-2]])
```

On obtient deux matrices P (matrice orthogonale de passage tran(P)=inv(P)) et B (matrice triangulaire semblable à l'argument) :

```
   [[[0.353452714748,-0.31069680265,0.882348386557],
   [0.905760021954,-0.122092619725,-0.405822836763],
     [0.23381608385,0.94263507734,0.238262774897]],
     [[8.10977222864,3.79381095693,2.32899008373],
 [0.0,-3.0,-3.03031411127],[0.0,0.0,-1.10977222865]]]
```

et on a :
B∼ inv(P)*[[1,2,3],[4,5,6],[-1,3,-2]]*P.
On tape :

```
SCHUR([[1,2,4],[4,5,6],[7,8,9]])
```

On obtient deux matrices P (matrice orthogonale de passage tran(P)=inv(P)) et B (matrice triangulaire semblable á l'argument) :

```
 [[[-0.275726630766,-0.921788330317,-0.272545591008],
  [-0.518148584403,-0.0962872203049,0.849853408352],
  [-0.809627611663,0.375546329096,-0.451074367633]],
    [[16.5037894003,3.99680014234,-0.803622341526],
 [-4.55776517517e-20,-1.61625693617,0.616262731649],
[4.1752316389e-20,-2.72155736143e-15,0.112467535861]]]
```

et on a :
MB∼ inv(P)*[[1,2,3],[4,5,6],[-1,3,-2]]*P

20.13.7 Singular value decomposition : SVD et svd

Dans HOME SVD(M1) renvoie 1 matrice M2, 1 vecteur M3, 1 matrice M4.
Cela donne la factorisation de la matrice rectangulaire numérique réelle M_1 (de dimension $m * n$) en M2*diag(M3)*TRN(M4) où M2 est une matrice orthogonale $m * m$,M4 est une matrice orthogonale $n * n$, et diag(M3) est une matrice diagonale de dimension $m * n$ ayant comme diagonale les valeurs singulières M3 de M_1.
Dans CAS svd(A) renvoie 1 matrice U, 1 vecteur S, 1 matrice Q.
Cela donne la factorisation de la matrice rectangulaire numérique réelle A (de dimension $m*n$) en U*diag(S)*TRN(Q) où U est une matrice orthogonale $m*m$, Q est une matrice orthogonale $n * n$, et diag(S) est une matrice diagonale de dimension $m * n$ ayant comme diagonale les valeurs singulières S de A.
On tape dans HOME :

```
M2,M3,M4:=SVD([[1,2],[2,1]])
```

On obtient :

```
{[[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]],
                              [3,1],
[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]]}
```

et on a (ici $M4$ est symétrique) :
`M2*diag(M3)*TRN(M4).`
On tape dans CAS :

```
U,S,Q:=SVD([[1,2],[2,1]])
```

Ou on tape dans CAS :

```
U,S,Q:=svd([[1,2],[2,1]])
```

On obtient :

```
{[[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]],
                             [3.,1.],
[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]]}
```

et on a (Q est symétrique) :
`[[0.707106781187,-0.707106781187],[0.707106781187,0.707106781187]]*`
`[[3,0],[0,1]]*`
`[[0.707106781187,0.707106781187],[0.707106781187,-0.707106781187]]=`
`[[1,2],[2,1]]`
On tape dans CAS :

```
SVD([[1,2],[3,4]])
```

Ou on tape dans CAS :

```
svd([[1,2],[3,4]])
```

On obtient :

```
[[-0.404553584834,-0.914514295677],[-0.914514295677,0.404553584834]],

             [5.46498570422,0.365966190626]


[[-0.576048436767,0.81741556047],[-0.81741556047,-0.576048436766]],
```

car $[[-0.404553584834, -0.914514295677], [-0.914514295677, 0.404553584834]]*$
$[[5.46498570422, 0], [0, 0.365966190626]]*$
$TRN([[-0.576048436767, 0.81741556047], [-0.81741556047, -0.576048436766]]) =$
$[[1.0, 2.0], [3.0, 4.0]]$

20.13.8 Valeurs singulières : `SVL svl`

Dans HOME `SVL(M1)`, (resp dans CAS `svl(A)`), renvoie la liste des valeurs singulières de la matrice numérique réelle M_1 (resp A) i.e. les racines carrés positives des valeurs propres de la matrice réelle symétrique M_1*TRN(M_1) (resp(A*TRN(A)).
On tape :

```
SVL([[1,4],[1,1]])
```

Ou :

```
svl([[1,4],[1,1]])
```

On obtient :

```
[4.30277563773,0.697224362268]
```

car `eigenvals([[1,4],[1,1]]*[[1,1],[4,1]])` renvoie ;
`(5*sqrt(13)+19)/2,(-5*sqrt(13)+19)/2` et
`sqrt((5*sqrt(13)+19)/2.),sqrt((-5*sqrt(13)+19)/2.)` renvoie
`4.30277563773,0.697224362268`
On tape :

```
SVL([[1,2],[2,1]])
```

ou

```
svl([[1,2],[2,1]])
```

On obtient :

```
[3,1]
```

car `EIGENVAL([[1,2],[2,1]]*[[1,2],[2,1]])` renvoie `[9,1]`.
On tape :

```
SVL([[1,2],[3,4]])
```

ou

```
svl([[1,2],[3,4]])
```

On obtient :

```
[5.46498570422,0.365966190626]
```

car `EIGENVAL([[1,2],[3,4]]*[[1,3],[2,4]])` renvoie
`[29.8660687473,0.133931252682]`
qui sont les valeurs approchées de $\sqrt{221}+15$ et $-\sqrt{221})15$.
On a :
$\sqrt{\sqrt{221}+15} \sim 5.46498570422$, $\sqrt{-\sqrt{221}+15} \sim 0.365966190626$

20.14 Vecteur

20.14.1 Produit vectoriel : `CROSS` ou `cross`

Dans HOME, `CROSS` renvoie le produit vectoriel des deux vecteurs.
On tape :

```
CROSS([1,2,3],[4,5,6])
```

ou

```
CROSS({1,2,3},{4,5,6})
```

On obtient :

```
[-3,6,-3]
```

car $2*6-5*3=-3, 4*3-1*6=6, 5-4*2=-3$
Dans CAS, `CROSS` ou `cross` renvoie le produit vectoriel des deux vecteurs.
On tape :

```
CROSS([1,2,3],[4,5,6])
```

ou

```
cross([1,2,3],[4,5,6])
```

On obtient :

```
[-3,6,-3]
```

car $2*6-5*3=-3, 4*3-1*6=6, 5-4*2=-3$

20.14.2 Produit scalaire : `DOT` ou `dot`

Dans HOME, `DOT` renvoie le produit scalaire des deux vecteurs.
On tape :

```
DOT([1,2,3],[3,4,5])
```

On obtient :

```
26
```

car $1*3+2*4+3*5=26$
Dans CAS `DOT` renvoie le produit scalaire des deux vecteurs.
On tape :

```
DOT([1,2,3],[3,4,5])
```

Ou on tape :

```
dot([1,2,3],[3,4,5])
```

On obtient :

```
26
```

car $1*3+2*4+3*5=26$

20.14.3 Norme l^2 :l2norm

l2norm s'utilise dans le CAS (dans HOME il faut mettre CAS.l2norm et la réponse sera exacte).
l2norm calcule la norme l^2 d'un vecteur : c'est la racine carrée de la somme des carrés de ses coordonnées.
On tape :

```
l2norm([3,-4,2])
```

Ou on tape :

```
l2norm(vecteur(3,-4,2))
```

On obtient :

```
sqrt(29)
```

En effet : x=3, y=-4, z=2 et $29 = |x|^2 + |y|^2 + |z|^2$.

20.14.4 Norme l^1 :l1norm

l1norm s'utilise dans le CAS (dans HOME il faut mettre CAS.l1norm et la réponse sera exacte).
l1norm calcule 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|.

20.14.5 Norme du maximum :maxnorm

maxnorm s'utilise dans le CAS (dans HOME il faut mettre CAS.maxnorm et la réponse sera exacte).
maxnorm calcule 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 :

```
4
```

En effet : x=3, y=-4, z=2 et 4=max(|x|,|y|,|z|).

Chapitre 21

Les fonctions spéciales

21.1 La fonction β : `Beta`

`Beta` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Beta` et la réponse sera exacte).
`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 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`

21.2 La fonction Γ : `Gamma`

`Gamma` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Gamma` et la réponse sera exacte).
`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 :

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)`

21.3 Les derivées de la fonction DiGamma : `Psi`

`Psi` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Psi` et la réponse sera exacte).
`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
```

21.4 La fonction ζ : `Zeta`

`Zeta` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Zeta` et la réponse sera exacte).
`Zeta` a comme argument un réel x.
`Zeta` renvoie pour $x > 1$: $\sum_{n=1}^{+\infty} \frac{1}{n^x}$.
On tape :

```
Zeta(2)
```

On obtient :

```
pi^2/6
```

On tape :

```
Zeta(4)
```

On obtient :

```
pi^4/90
```

21.5 La fonction erf : `erf`

`erf` s'utilise dans le CAS (dans HOME il faut mettre `CAS.erf` et la réponse sera exacte).
`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$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = \frac{1}{2} +$ erf$(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

21.6 La fonction $erfc$: erfc

`erfc` s'utilise dans le CAS (dans HOME il faut mettre `CAS.erfc` et la réponse sera exacte).
`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` :
`normal_cdf`$(x) = 1 - \frac{1}{2}$`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$(x) = \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-u^2} du = 1 - \frac{1}{2}$erfc$(\frac{x}{\sqrt{2}})$
On vérifie en tapant :
`normal_cdf(1)=0.841344746069`

21.7 La fonction exponentielle integrale Ei : `Ei`

`Ei` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Ei` et la réponse sera exacte).
`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 :

```
0.57721566490153286060606507
```

21.8 La fonction sinus integral Si : Si

Si s'utilise dans le CAS (dans HOME il faut mettre CAS.Si et la réponse sera exacte).
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
```

21.9 La fonction cosinus integral Ci : `Ci`

`Ci` s'utilise dans le CAS (dans HOME il faut mettre `CAS.Ci` et la réponse sera exacte).
`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
```

21.10 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
```

21.11 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)
```

Chapitre 22

Lesconstantes et les calculs avec des unités

22.1 La touche shiftée `Units`

Avec la touche shiftée `Units` puis `Outils` du bandeau, on obtient les fonctions qui permettent de faire des calculs avec des unités :
`convert`, `mksa`, `ufactor`, `usimplify`.
Avec la touche shiftée `Units` puis `Unités` du bandeau, on obtient les unités classées par thème avec en 1 les préfixe disponibles.
Avec la touche shiftée `Units` puis `Const` du bandeau, on obtient les constantes Mathématiques, Chimiques, Physiques et Quantiques.

22.2 Les unités

22.2.1 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
```

22.2.2 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`.

22.2.3 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
```

22.3 Les Outils

22.3.1 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
```

22.3.2 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)
```

22.3.3 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)
```

22.3.4 Simplifier une unité : `usimplify`

`usimplify` permet de simplifier une unité dans un objet-unité.
On tape :

```
usimplify(3_(W*s))
```

On obtient :

```
3_J
```

22.4 Les constantes physiques

Avec la touche shiftée `Units` puis `Const` du bandeau, puis `3: Physique` on obtient les constantes physiques classées par thème. Avec la touche shiftée `Units` puis `Unit` du bandeau on obtient les unités classées par thème.Les constantes physiques et les unités sont classées par thème, se trouvent dans le menu `Phys`.

22.5 Les unités

22.5.1 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
```

22.5.2 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
```

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

`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
```

22.5.4 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)
```

22.5.5 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
```

22.5.6 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)
```

22.5.7 Simplifier une unité : `usimplify`

`usimplify` permet de simplifier une unité dans un objet-unité.
On tape :

```
usimplify(3 _(W*s))
```

On obtient :

```
3 _J
```

22.6 Les constantes

22.6.1 La notation des constantes chimiques ou physiques ou quantiques

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 ou chimique.
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
```

22.6.2 Bibliothèque des constantes physiques

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 23

Les fonctions de géométrie 3D

23.1 La perpendiculaire commune à deux droites 3-d : common_perpendicular

`common_perpendicular` a comme argument deux droites `D1` et `D2`.
`common_perpendicular(D1,D2)` dessine la perpendiculaire commune des droites `D1` et `D2`.
On tape :

```
D1:=line([1,1,0],[0,1,1]);
D2:=line([0,-1,0],[1,-1,1]);
```

Puis on tape :

```
d:=common_perpendicular(D1,D2)
```

On obtient :

```
pnt([[1,0,-1],[-1/3,-2/3,-1/3]],0,d)
```

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]
```

Quatrième partie

Les Applications et la touche Apps

Chapitre 24

Le menu Geometry

ATTENTION La documentation sur l'application de géométrie n'est pas définitive ! ! ! ! !

24.1 Généralités

Nous allons décrire ici l'`Applications` de `Géométrie` qui permet de faire de la géométrie interactive.
Comme toutes les `Apps`, l'`Applications` de `Géométrie` plane a 3 vues : la vue symbolique, la vue graphique et la vue numérique.
Dans le bandeau de ces 3 vues figure `Cmds` qui contient les commandes de géométrie classées par thèmes utiles dans chacune des vues :
`Point Ligne Polygone Courbe Tracé Transformation` pour la vue symbolique,
`Zoom Point Ligne Polygone Courbe Tracé Transformation Cartésien, Mesure,Tests` pour la vue graphique,
`Cartésien, Mesure,Tests` pour la vue numérique.
Ces 3 vues sur un exemple

1. la touche `Symb` ouvre la vue symbolique.
   À l'aide de `Edit` ou `Nouv` du bandeau, vous mettez dans la case `GC` : `circle(1,2)`
   Pour cela :
   – vous tapez `circle` en toutes lettres ou
   vous utilisez `Cmds` du bandeau (`Cmds->courbe->cercle`).
   Si c'est votre première commande son nom sera `GA` : vous pouvez modifier ce nom en sélectionnant `GA` et en l'éditant avec `Edit` du bandeau, vous changez le `A` en `C` et vous appuyer sur `Enter`.
   – vous tracez directement un cercle dans la vue graphique pour que `circle(point(GA),GB-GA)` s'inscrive en `GC` dans la vue symbolique. Vous pouvez alors modifier les valeurs de GA et $GB - GA$ par `point(1)` et `2`
   Remarques

   (a) Pour que la commande contenue dans `GC` soit exécutée il faut cocher la case située devant `GC`.

(b) Pour mettre en surbrillance une ligne de commande il faut utiliser le curseur.

(c) Pour changer l'ordre des commandes il faut utiliser les flèches ↑ et ↓ du bandeau : avec ces flèches vous déplacez la ligne mise en surbrillance.

(d) Pour insérer une nouvelle commande là où vous le souhaitez il faut utiliser `Inser` du bandeau.

(e) Pour changer le nom d'une variable il faut, dans la vue symbolique, sélectionner ce nom puis, il faut l'éditer avec `Edit` du bandeau et faire la modification et valider avec `Enter`.

(f) Pour effacer une ligne de commande il faut la mettre en surbrillance, puis d'appuyer sur puis la touche d'effacement.

2. la touche `Plot` ouvre la vue graphique.
Pour tracer directement un cercle dans la vue graphique vous utilisez `Cmds` du bandeau (`Cmds->courbe->cercle`) puis vous montrez avec votre doigt le point qui sera le centre (vous pouvez ensuite affiner avec le curseur) et vous appuyez sur `Enter` pour confirmer vôtre choix : la commande `point(2.375,1.24)` (par exemple) a été enregistée automatiquement en `GA` dams la vue symbolique.
Puis vous montrez avec votre doigt un second point qui doit se trouver sur le cercle (vous pouvez ensuite affiner avec le curseur) et vous appuyez sur `Enter` pour confirmer vôtre choix.
La commande `point(2.375,3.24)` (par exemple) s'inscrit en `GB` et la commande `circle(point(GA),GB-GA)` s'inscrit en `GC` dans la vue symbolique.
Le cercle une fois dessiné vous avez la possibilité :

(a) de le dessiner en couleur,
pour cela, vous approchez votre doigt du contour du cercle pour faire apparaitre `Options` dans le bandeau puis `Options->Choisir une couleur` et `Enter`. La palette des couleurs s'ouvre et vous sélectionnez une couleur avec votre doigt et vous confirmez avec `Enter`.

(b) de le dessiner en forme pleine
pour cela, vous approchez votre doigt du contour du cercle pour faire apparaitre `Options` dans le bandeau puis `Options->Rempli` et `Enter` a pour effet de remplir le cercle avec la couleur choisie.

(c) de ne pas faire apparaitre son nom
pour cela, vous approchez votre doigt du contour du cercle pour faire apparaitre `Options` dans le bandeau puis `Options->Hide Label` : si il y a une ambiguité la machine vous proposera plusieurs choix.

(d) de le translater
pour cela, vous approchez votre doigt (vous affinez avec le curseur) du nom du cercle pour faire apparaitre `Options` dans le bandeau puis `Enter`. Vous pouvez effectuer la translation de votre choix avec votre

doigt ou avec le curseur. Vous confirmer avec Enter.

Remarque
Il faut noter que les points ou les objets géométriques créés directement dans l'écran graphique sont notés de façon automatique :
les objets géométriques sont nommés et désignés par A, B, C... mais sont sauvés dans les variables GA, GB, GC... variables que l'on retrouve dans la vue symbolique. Il faut noter que le point F n'existe pas car GF est une commande du CAS.

3. la touche Num ouvre la vue mumérique.
   Si vous voulez l'équation du cercle ainsi définit vous tapez : equation(GC) et vous obtenez :
   (x-1.0)^2+y^2=4.0
   La vue numérique permet d'utiliser les commandes numériques liées au graphique et d'avoir des résultats numériques.
   Les commandes donnant un résultat numérique (coordonnées des points, équation des droites ou des courbes...) peuvent être exécutées dans lécran Num en utilisant Nouv puis Cmds du bandeau de lécran Num.

Lorsqu'on est dans le CAS ou dans l'Application de Géométrie, les commandes de géométrie sont en anglais et sont classées par thème dans le menu Apps->Géométrie de la touche  . On retrouve les 9 thèmes contenu dans Cmds du bandeau des différantes vues :
Point, Ligne, Polygone, Courbe, Tracé, Transformation, Cartésien, Mesure, Tests.

Remarque
Toutes les fonctions de géométrie peuvent être utilisées depuis le CAS, mais alors la réponse sera par exemple point(1,2) ou line(x=1) mais ne sera pas dessinée.
On peut donc faire de la géométrie analytique $2-d$ et même $3-d$ depuis le CAS : on peut par exemple taper depuis le CAS :
g:=line(x=1) puis rajouter dans l'écran Symb GK:=g.

Attention
On peut retrouver la valeur de ces variables dans la vue symbolique ou dans la vue numérique ou dans l'écran du CAS. Si par exemple dans l'application de géométrie on a : GA:=point(4.16+2.13*i) et que dans le CAS, on tape GA:=5, GA sera égal à 5 tant que l'on a pas utilisé Plot de l'application de géométrie car après son utilisation on aura à nouveau GA:=point(4.16+2.13*i).
Il est donc dangereux d'utiliser les variables GA, GB... dans le CAS.

24.2 Point

24.2.1 Point défini comme barycentre de n points : barycenter

barycenter, en géométrie plane, a comme argument n listes de longueur 2 (resp une matrice ayant n lignes et 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 ou le nombre complexe a_j représentant l'affixe de ce point, 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.
`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 :

```
barycenter([1+i,1],[1-i,1])
```

Ou on tape :

```
barycenter([point(1,1),1],[point(1,-1),1])
```

Ou on tape :

```
barycenter([[1+i,1],[1-i,1]])
```

Ou on tape :

```
barycenter([[point(1,1),1],[point(1,-1),1]])
```

On obtient dans l'Application de géométrie :

```
Le point d'affixe 1 est tracé avec une croix avec son
nom
```

On obtient dans le CAS :

```
point(1)
```

Attention dans l'Application de géométrie pour avoir comme réponse un nombre complexe il faut demander l'affixe du barycentre, sinon on a le tracé du point barycentre.
On tape :

```
affix(barycenter([1+i,1],[1-i,1]))
```

Ou on tape :

```
affix(barycenter([[1+i,1],[1-i,1]]))
```

On obtient :

```
1
```

Dans l'écran CAS `barycenter` est aussi utilisable en géométrie $3-d$ et a comme argument n listes de longueur 2 (ou une matrice ayant n lignes et 2 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.
`barycenter` renvoie `point([a,b,c])` où `[a,b,c]` sont les coordonnées du barycentre de ces n points.
On tape :

```
barycenter([point(0,0,0),1],[point(3,3,3),2])
```

On obtient :

```
pnt(pn[(point[2,2,2],0)])
```

24.2.2 Le point en géométrie : point

Dans la vue graphique, pour obtenir un point il suffit d'être en mode point (i.e. Point du bandeau, puis sélectionner point et Enter) et d'amener le curseur à l'endroit voulu (avec le doigt ou les flèches) puis de valider avec Enter : 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.
Attention
Si a,b est un couple de 2 nombres complexes dont 1 est non réel, GK:=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, GA:=point(a,b) renvoie et dessine le point ayant pour affixe a+ib.
On tape :

```
GA:=point(1+i)
```

On obtient :

```
Le point A d'affixe 1+i est tracé avec une croix
```

On tape :

```
GB:=point(-2,1)
```

On obtient :

```
Le point B d'affixe -2+i est tracé avec une croix
```

On tape :

```
GC:=point(-2,i)
```

On obtient :

```
Les 2 points d'affixe -2 et i sont tracés avec une
                croix et sont notés C
```

Remarque Lorqu'on fait une affectation par exemple GA:=point(-2+i) cela a pour effet de stocker le point(-2+i) dans la variable GA, de dessiner le point avec une croix et de lui mettre comme légende le nom qui est situé à gauche de := en supprimant la lettre G : ici A.
Si on fait plusieurs affectations avec un seul signe := par exemple :
GD,GE:=point(-2+i),point(2+i) la variable GD contient le point(-2+i), la variable GE contient point(-2+i) mais on ne pourra pas deplacer ces points en mode pointeur.
Pour éviter cela on doit taper :
GL:=point(-2+i),point(2+i):;GD:=L[0];GE:=L[1]
ou
GL:=point(-2+i,2+i):;GD:=L[0];GE:=L[1] ce qui définit le point D d'affixe -2 et le point E d'affixe i (car l'affixe de GD n'est pas réel !).

24.2.3 Le milieu d'un segment : `midpoint`

`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).
`midpoint` renvoie et dessine le point milieu du segment défini par ces deux points.
On tape :

```
midpoint(-1,1+i)
```

On obtient dans l'Application de géométrie :

```
Le point d'affixe i/2 est tracé avec son nom
```

Dans l'écran CAS `midpoint` est utilisable en géométrie $3-d$ et renvoie le point milieu du segment défini par deux points.
On tape :

```
midpoint(point(0,0,0),point(2,2,2))
```

On obtient :

```
point(1,1,1)
```

24.2.4 L'isobarycentre de n points : `isobarycenter`

`isobarycenter` a comme argument la liste (ou la séquence) de n points ou de n nombres complexes représentant l'affixe de ces points.
`isobarycenter` renvoie et trace un point qui est l'isobarycentre de ces n points.
On tape :

```
isobarycenter(0,2,2*i)
```

On obtient :

```
Le point d'affixe 2/3+2*i/3 est tracé avec une croix
          dans l'Application de géométrie
```

Dans l'écran CAS `isobarycenter` est utilisable en géométrie $3-d$ a comme argument la liste (ou la séquence) de n points.
`isobarycenter` renvoie `point([a,b,c])` où `[a,b,c]` sont les coordonnées de l'isobarycentre de ces n points.
On tape :

```
isobarycenter(point(0,0,0),point(3,3,3))
```

On obtient :

```
pnt(pn[(point[3/2,3/2,3/2],0])
```

24.2.5 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 déplacé.

24.2.6 Le point en polaire en géométrie plane : polar_point

polar_point(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)
```

On obtient :

```
Le tracé du point d'affixe 2*exp(i*pi/4)
```

24.2.7 Un des points d'intersection de deux objets géométriques : single_inter

single_inter 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.
single_inter renvoie l'un des points d'intersection de ces deux objets géométriques.
Si on a mis un point GA (ou son affixe) comme troisième argument single_inter renvoie le point d'intersection le plus proche de GA et si on a mis une liste de points l (ou une liste d'affixe) comme troisième argument single_inter renvoie le point d'intersection qui ne se trouve pas dans la liste l.
On tape :

```
GA:=single_inter(line(0,1+i),line(1,i))
```

On obtient :

```
Le point d'affixe 1/2+i/2 est tracé avec une croix et
                    est noté A
```

On tape :

```
GB:=single_inter(circle(0,1),line(-1,i))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté B
```

On tape :

```
GB1:=single_inter(circle(0,1),line(-1,i),[i])
```

On obtient :

```
Le point d'affixe -1 est tracé avec une croix et est
noté B1
```

On tape :

```
GB2:=single_inter(circle(0,1),line(-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 :

```
GC:=single_inter(circle(1,sqrt(2)),circle(0,1))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C
```

On tape :

```
GC1:=single_inter(circle(1,sqrt(2)),circle(0,1),[i])
```

On obtient :

```
Le point d'affixe -i est tracé avec une croix et est
noté C1
```

On tape :

```
GC2:=single_inter(circle(1,sqrt(2)),circle(0,1),i/2)
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C2
```

24.2.8 Les points d'intersection de deux objets géométriques : `inter`

`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 :

```
GA:=inter(line(0,1+i),line(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 :

```
GB:=inter(circle(0,1),line(1,i))[0]

GC:=inter(circle(0,1),line(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 dans CAS :

```
inter(circle(0,1),line(1,i))
```

On obtient :

```
[point(1),point(i)]
```

On tape en géométrie dans la vue symbolique :

```
GL inter(circle(0,1),line(1,i))
```

On obtient dans la vue graphique :

```
Les points d'affixe i et 1 sont tracés avec une croix
                et sont notés L
```

On tape en géométrie plane :

```
GD:=inter(circle(0,1),line(1,i),point(1/2))
```

On obtient :

```
Le point d'affixe 1 est tracé avec une croix et est
                      noté D
```

24.2.9 Orthocentre d'un triangle : `orthocenter`

`orthocenter` a comme argument un triangle ou trois points ou trois nombres complexes désignant l'affixe de trois points.
`orthocenter` 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 :

```
orthocenter(triangle(0,1+i,-1+i))
```

On obtient :

```
Le point d'affixe 0 est tracé avec une croix
```

On tape dans la vue symbolique de l'application de gépmétrie :

```
GT triangle(-i,2+i,-1+i);GH orthocenter(T)
```

On obtient :

```
Le triangle T et le point H d'affixe 0 sont tracés
```

24.2.10 Les sommets d'un polygone : `vertices`

`vertices` a comme argument un polygone.
`vertices` 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 :

```
vertices(equilateral_triangle(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 :

```
GC:=vertices(equilateral_triangle(0,2))[2]
```

On obtient :

```
Le point d'affixe 1+i*√3 est tracé avec une croix et
                est noté C
```

Attention
Si on tape :

```
GT:=equilateral_triangle(0,2,C);
```

On obtient :

```
          le triangle T et le point C
```

Alors que si on tape

```
GT:=equilateral_triangle(0,2,C):;vertices(GT[0])
```

On obtient :

```
les sommets de T tracés avec une croix sans nom
```

24.2.11 Les sommets d'un polygone : `vertices_abca`

`vertices_abca` a comme argument le nom d'un polygone.
`vertices_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 :

```
vertices_abca(equilateral_triangle(0,2))
```

On obtient :

```
[pnt(0,0),pnt(2,0),pnt((2*(sqrt(3)*(i)+1))/2,0),pnt(0,0)]
```

24.2.12 Point sur un objet géométrique : `element`

`element` peut avoir différents types d'arguments :

1. 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, on tape dans la vue symbolique :
   `GC:=element(-pi..pi)` ou
   `GC:=element(-pi..pi,pi/2)` ou
   `GC:=element(-pi..pi,pi/2,pi/100.0)`
   cela signifie que `GC` peut prendre une valeur quelconque de $[-\pi;\pi]$ et le deuxième argument $\pi/2$ donne la valeur qui définit `GC` au début et $\pi/100.0$ est le pas choisi.
   Puis dans la vue graphique :
   – on a en bas et à droite on a `Pointeur x1,y1` où `x1,y1` sont les coordonnées du pointeur. On tape `C` en mode `Alpha`,
   – en bas et à droite on a alors `Pointeur GC` on appuie sur `Enter` pour valider.
   – maintenant on pointe sur `C`.
   Un trait apparait en haut de l'écran graphique : c'est un curseur qui permet de changer la valeur de `GC` avec les flèches de direction ($\leftarrow$ et $\rightarrow$) et en bas et à droite on a `Déplacer GC` À gauche de ce trait on a `GC=xc` (`xc` est la valeur de `GC`). Les flèches de direction $\leftarrow$ et $\rightarrow$ permettent de changer la valeur `xc` de `GC`.
   Exemple.
   On définit `GC` comme ci-dessus :
   On tape dans la vue symbolique :
   `GC:=element(-pi..pi,pi/2)`
   `GD:=line(y+x*TAN(GC)-2*SIN(GC)=0)`
   `GD` est donc une droite de paramètre `GC`.
   Quand on fait bouger le curseur `GC`, la doite `GD` bouge.
   On peut conserver la trace de cette droite `GD` en tapant dans la vue symbolique : `trace(GD)` ou en utilisant dans le menu de la vue graphique `Point->Plus->trace` qui permet de choisir le nom de l'objet dont on veut la trace et ainsi `trace(GD)` s'écrit automatiquement dans la vue symbolique.

On peut ainsi voir que l'enveloppe de ces droites est une astroide.
Remarque
Pour effacer la trace ou pour l'arrêter, il faut utiliser le menu `Point->Plus`.

2. un objet géométrique et un réel (par défaut ce réel vaut $1/2$), par exemple : `GA:=element(circle(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 de ce cercle et le deuxième argument 1 donne la valeur du paramètre t pour définir `GA`).
   Par exemple, `GA:=element(circle(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 `affix(GA)` = $2 * \exp(i/2)$). Lorsque ensuite on déplacera `A` avec les flèches, `A` se déplacera sur l'objet géométrique.
   Attention c'est la projetion du curseur sur le cercle qui définit le point `A` : il faut donc veiller à déplacer le curseur dans la fenêtre graphique de façon à ce qu'il définisse un point `A`.
3. un objet géométrique et un nom de variable (par exemple `GC`) défini auparavant par la commande `element` : par exemple `GC:=element(0..pi)`. Si on tape `GD:=element(circle(0,2),GC)`, alors `GC` est la variable de paramétrage de l'obget géométrique défini par le premier argument de `element`, c'est à dire que `GD` se trouve sur le cercle de centre 0 et de rayon 2 et que `GD` a comme affixe $2 * \exp(i * GC)$, car $2 * \exp(i * t)$ est l'équation paramétrique du circle(0,2). Il est donc obligatoire dans ce cas de définir auparavant le deuxième argument (ici `GC`) comme étant l'élément d'un intervalle.
   On tape par exemple :

```
GC:=element(0..pi)
```

   puis

```
GD:=element(circle(0,2),GC)
```

   Puis on pointe le curseur sur `GC` (`Pointeur GC`) puis `Enter` cela a pour effet d'avoir en haut un curseur noté `GC` que l'on peut faire bouger avec les flèches de direction ($\leftarrow$ et $\rightarrow$) de 0 à π, avec à gauche de ce curseur un nombre égal à la valeur de 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 :

```
GA:=point(1);GB:=point(2+i)
GC:=element(0..2)
```

   puis

```
GD:=element(line(GA,GB),GC)
```

   `D` est un point de la droite `AB` et on a `M=A+t*(B-A)` i.e. `M=(1-t)*A+t*B` pout parcourir le segment `AB` , il faut mettre `GC:=element(0..1)` ou encore `GD:=element(segment(GA,GB),GC)` qui aura pour effet de laisser `D` en `A` si `t<0` et de laisser `D` en `B` si `t>1`.

4. une ligne polygonale GP et [floor(GC),frac(GC)] avec GC défini auparavant par la commande element : par exemple GC:=element(0..5) si GP a 5 côtés.
   Les côtés de la ligne polygonale GP ont comme numèro : 0,1....
   Si par exemple GP a 5 côtés et a pour sommets S(0),...S(4),S(5)=S(0), on tapera :
   GC:=element(0..5)
   GD:=element(GP,[floor(GC),frac(GC)])
   Ainsi selon les valeurs de GC, D va parcourir les 5 côtés de GP : D sera situé sur le côté de numéro n=floor(GC) et on aura :
   D=frac(GC)*S(n)+(1-frac(GC))*S(n+1).
   Par exemple :

```
GA:=point(0);
GB:=point(4);
GC:=point(4*i);
Gd:=element(0..3);
GT:=triangle(A,B,C);
GM:=element(GT,[floor(GD),frac(GD)]);
```

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 : GN:=GM+(GB-GA) à condition que M ne soit pas défini comme élément d'une courbe C. En effet si on a tapé GM:=element(GC) il faut définir N en tapant : GN:=affix(GM)+GB-GA ou GN:=GM+GB-GA (sans parenthèses) car GN:=GM+GB-GA est interprété en GN:=(GM+GB)-GA car il n'y a pas de règle de priorité entre + et − alors que
GN:=GM+(GB-GA) renvoie un élément de la courbe C qui est le projeté de N sur C.
On a donc, si on tape :
GA:=point(-2,2);GB:=point(1,3);GC:=circle(0,1);
GM:=element(GC);GN:=affix(GM)+GB-GA;(ou GN:=GM+GB-GA;)
GN n'est pas sur la courbe C
mais si on tape :
GP:=GM+(GB-GA) (ou GP :=projection(GC,GN) ;) P est sur la courbe C.

24.2.13 Point divisant un segment : division_point

division_point a trois arguments : deux points (ou deux nombres complexes a et b) et un nombre complexe k.
division_point renvoie et dessine le point d'affixe z tel que :
$\frac{z-a}{z-b} = k$ On tape :

```
GA:=division_point(i,2+i,3+i)
```

On obtient :

```
le point A d'affixe (5+4*i)/(2+i)
```

car `csolve(z-i=(3+i)*(z-2-i),z)` renvoie `[(14+3*i)/5]` et `(5+4*i)/(2+i)` renvoie `(14+3*i)/5`
On tape :

```
GB:=division_point(point(i),point(2+i),3)
```

On obtient :

```
le point B d'affixe 3+i
```

car `csolve(z-i=3*(z-2-i),z)` renvoie `[3+i]`

24.2.14 Division harmonique : `harmonic_division`

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.
`harmonic_division` a comme arguments 3 points alignés ou leur 3 affixes (resp 3 droites concourantes ou parallèles) et le nom d'une variable.
`harmonic_divisiondiv_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 :

```
harmonic_division(0,2,3/2,GD)
```

On obtient :

`[0,2,3/2,pnt(3,0,"GD")]` et seul le point D est dessiné

On tape :

```
harmonic_division(point(0),point(2),point(3/2),GD)
```

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 :

```
harmonic_division(line(i,0),line(i,1+i),
line(i,3+2*(i)),GD)
```

On obtient :

```
[pnt([[i,0],0]),pnt([[i,1+i],0]),
pnt([[i,3+2*i],0]),pnt([[i,-3+2*i],0,"GD"])] et les 4
droites sont dessinées
```

24.2.15 Le conjugué harmonique : `harmonic_conjugate`

`harmonic_conjugat` a comme arguments 3 points alignés GA, GB, GC (resp 3 droites concourantes ou parallèles).
`harmonic_conjugate` renvoie et dessine le conjugué harmonique de GC par rapport à GA et GB.
On tape :

```
harmonic_conjugate(0,2,3/2)
```

On obtient :

```
pnt(3,0)et dessine ce point
```

On tape :

```
harmonic_conjugate(line(0,1+i),line(0,3+i),line(0,i))
```

On obtient :

```
pnt([[0,3+2*i],0]) et dessine cette droite
```

24.2.16 Pôle et polaire : `pole` et `polar`

`polar` a comme argument un cercle GC et un point GA (ou un nombre complexe).
`polar` renvoie et dessine la polaire du point GA par rapport au cercle GC : c'est la droite qui est le lieu des conjugués de GA par rapport au cercle GC.
`pole` a comme argument un cercle GC et une droite Gd.
`pole` renvoie et dessine le pôle de Gd par rapport au cercle GC : c'est le point GA admettant Gd comme polaire par rapport à GC. On tape :

```
polaire(circle(0,1),(point(1+i))/2)
```

On obtient :

```
pnt([[2,2*i],0]) et dessine cette droite
```

On tape :

```
pole(circle(0,1),line(i,1))
```

On obtient :

```
pnt(1+i,0) et dessine ce point
```

24.2.17 Polaire réciproque : `reciprocation`

`reciprocation` a comme argument un cercle GC et une liste de points et de droites.
`reciprocation` 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 GC.
On tape :

```
reciprocation(circle(0,1),[point((1+i)/2),
                line(1,-1+i)])
```

On obtient :

```
la droite d'équation y=(-x+2) et le point d'affixe
                    1+2i
```

24.2.18 Le centre d'un cercle : `center`

`center` a comme argument le nom d'un cercle (voir la définition du cercle **??**). `center` renvoie et trace le centre de ce cercle.
On tape :

```
GC:=center(circle(0,point(2*i)))
```

On obtient :

```
Le point d'affixe i est tracé avec une croix et est
noté C
```

On tape :

```
GM:=center(circle(point(1+i),1))
```

On obtient :

```
Le point d'affixe 1+i est tracé avec une croix et est
noté M
```

24.3 Line

24.3.1 Droite définie par un point et une pente : `DrawSlp`

`DrawSlp(a,b,m)` dessine la droite de pente `m` `passant par le point (a,b)`
On tape :

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

On obtient :

```
La droite passant par le point d'affixe 1+2i et de
pente -1
```

24.3.2 Tangente au graphe de $y = f(x)$ en $x = a$: `LineTan`

`LineTan(f(x),x=a)` trace la tangente au graphe de $y = f(x)$ en $x = a$.
On tape :

```
LineTan(sin(x),pi/6)
```

Ou on tape :

```
LineTan(sin(t),t,pi/6)
```

On tape :

```
LineTan(sin(t),t=pi/6)
```

On obtient :

```
Le tracé de la tangente à la courbe y = sin(x) au point
d'abscisse x = π/6)
```

24.3.3 Hauteur d'un triangle : altitude

altitude(GA,GB,GC) trace la hauteur du triangle ABC issue de A.
On tape :

```
altitude(1,0,1-i)
```

On obtient dans l'Application de géométrie :

Le tracé de la hauteur du triangle $(1,0,1-i)$ issue du point d'affixe 1

On tape :

```
altitude(0,1,2-i)
```

On obtient dans l'Application de géométrie :

Le tracé de la hauteur du triangle $(0,1,2-i)$ issue du point d'affixe 0

On tape dans l'écran CAS :

```
a:=altitude(1,0,1-i)
```

On obtient :

```
line(y=x-1)
```

On tape dans l'écran CAS :

```
a:=altitude(1,0,1-i)
```

On obtient :

```
line(y=x)
```

24.3.4 Bissectrice intérieure d'un angle : bisector

bisector(GA,GB,GC) trace la bissectrice intérieure de l'angle $\widehat{BAC}$.
On tape :

```
bisector(0,1,i)
```

On obtient dans l'Application de géométrie :

Le tracé de la bissectrice intérieure de l'angle $\widehat{(0,1,i)}$

On tape dans l'écran CAS :

```
bisector(0,1,i)
```

On obtient :

```
line(y=x)
```

24.3.5 Bissectrice extérieur d'un angle : `exbisector`

`exbisector(GA,GB,GC)` trace la bissectrice intérieure de l'angle $\widehat{BAC}$.
On tape :

```
exbisector(0,1,i)
```

On obtient dans l'Application de géométrie :

`Le tracé de la bissectrice extérieure de l'angle` $\widehat{(0,1,i)}$

On tape dans l'écran `CAS` :

```
exbisector(0,1,i)
```

On obtient :

```
line(y=-x)
```

24.3.6 Demi-droite : `half_line`

`half_line(GA,GB)` trace la demi droite A, B.
On tape :

```
half_line(1,2+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé de la demi-droite d'origine le point d'affixe
        1 et contenant le point d'affixe 2+i.
```

On tape dans l'écran `CAS` :

```
half_line(1,2+i)
```

On obtient :

```
line(y=x-1)
```

24.3.7 La droite et la droite orientée : `line`

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 `slope=m` 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.
`line(GA,GB)` trace la droite A, B.
Remarque`slope` est aussi une commande donnant la pente d'une droite. Il est préférable d'utiliser `DrawSlp` pour définir une droite avec un point et sa pente (`DrawSlp(a,b,m)` définit la droite `line(point(a,b),slope=m)`).
Dans l'écran `CAS` :
On tape :

```
line(1,2+i)
```

On obtient :

```
line(y=x-1)
```

Dans l'Application de géométrie :
On tape :

```
line(1,2+i)
```

On obtient :

```
Le tracé de la droite passant par le point d'affixe 1
           et par le point d'affixe 2+i.
```

On tape :

```
line(0,1+i)
```

On obtient :

```
La droite d'équation y=x est tracée
```

On tape :

```
line(i,slope=2)
```

Ou on tape :

```
DrawSlp(0,1,2)
```

On obtient :

```
La droite d'équation y=2x+1 est tracée
```

On tape :

```
line(y-x=0)
```

On obtient :

```
La droite d'équation y=x est tracée
```

Remarque : l'orientation de la droite

- Lorsque la droite est donnée par deux points, son orientation est définie par l'ordre des points donnés en argument. Par exemple `ine(GA,GB)` 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 `line(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 $B = A + 1 + i * m$.

24.3.8 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
```

24.3.9 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
```

24.3.10 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
```

24.3.11 Le vecteur en géométrie plane : `vector`

`vector`, en géométrie plane, a comme arguments soit :

- deux points GA, GB ou deux nombres complexes représentant l'affixe de ces points ou deux listes représentant les coordonnées de ces points.
  `vector` définit et dessine le vecteur $\overrightarrow{GAGB}$
- un point GA (ou un nombre complexe représentant l'affixe de ce point ou une liste représentant les coordonnées de ce point) et un vecteur GV (définition récursive).
  `vector` définit et dessine le vecteur $\overrightarrow{AB}$ tel que $\overrightarrow{GAGB} = \overrightarrow{GV}$. Si `GW:=vector(GA,GV)`, alors le point `GB` tel que $\overrightarrow{GAGB} = \overrightarrow{GV}$ est `point(GW[1,1])` ou `point(coordinates(GV)` ou `GA+(affix(V)[1]-affix(GV)[0])`.

On tape :

```
vector(point(-1),point(i))
```

Ou on tape :

```
vector(-1,i)
```

Ou on tape :

```
vector([-1,0],[0,1])
```

On obtient :

```
Le tracé du vecteur d'origine -1 et d'extrémité i
```

On tape :

```
GV:=vector(point(-1),point(i))
```

On tape :

```
vector(point(-1+i),GV)
```

Ou on tape :

```
vector(-1+i,GV)
```

Ou on tape :

```
vector([-1,1],GV)
```

On tape :

```
point([-1,1],coordinates(GV))
```

On obtient :

```
Le tracé du vecteur d'origine -1+i et d'extrémité 2*i
```

On tape :

```
GD:=point([-1,1]+coordinates(GV))
```

On obtient :

```
GD le point(2*i)
```

Remarque
En calcul formel, on travaille sur la liste des coordonnées des vecteurs que l'on obtient avec la commande `coordinates` (cf **??**).

24.3.12 Médiane d'un triangle : `median_line`

`median_line(GA,GB,GC)` trace la médiane du triangle ABC issue de A.
On tape :

```
median_line(0,1,2+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé de la droite passant par le point d'affixe 0
 et par le point d'affixe (3+i)/2 (milieu du segment
                      (0,2+i))
```

On tape dans l'écran `CAS` :

```
median_line(0,1,2+i)
```

On obtient :

```
line(y=x/3)
```

24.3.13 Droites parallèles : `parallel`

`parallel(GA,GD)` trace la parallèle à la droite D passant par A.
On tape :

```
parallel(0, line(1,i))
```

On obtient dans l'Application de géométrie :

```
le tracé de la droite d'équation y=-x
```

On tape dans l'écran `CAS` :

```
parallel(0, line(1,i))
```

On obtient :

```
line(y=(-x))
```

24.3.14 Médiatrice : `perpen_bisector`

`perpen_bisector(GA,GB)` trace la médiatrice du segment AB On tape :

```
perpen_bisector(1,i)
```

On obtient dans l'Application de géométrie :

```
le tracé de la droite d'équation y=x
```

On tape dans l'écran `CAS` :

```
perpen_bisector(1,i)
```

On obtient :

```
line(y=x)
```

24.3.15 Perpendiculaire à une droite : `perpendicular`

`perpendicular(GA,GB,GC)` ou `perpendicular(GA,line(GB,GC))` trace la perpendiculaire à la droite BC droite passant par le point A.
On tape :

```
perpendicular(1,1,2-i)
```

On obtient dans l'Application de géométrie :

```
le tracé de la perpendiculaire à la droite (1,2-i)
        etpassant par le point d'affixe 1
```

On tape dans l'écran `CAS` :

```
perpendicular(1,1,2-i)
```

On obtient :

```
line(y=(x-1))
```

24.3.16 Segment : segment

segment(GA,GB) trace le segment AB.
On tape :

```
segment(0,1+i)
```

On obtient dans l'Application de géométrie :

```
Le tracé du segment (0,1+i)
```

On tape dans l'écran CAS :

```
segment(0,1+i)
```

On obtient :

```
segment(point(0),point(1+i))
```

24.3.17 Les tangentes à un objet géométrique ou la tangente en un point d'un graphe : tangent

tangent a deux arguments : un objet géométrique et un point A.

- l'objet géométrique est le graphe G d'une fonction 2-d
  Dans ce cas, le deuxième argument peut être soit, un nombre réel x_0, soit un point A situé sur G.
  Par exemple si on a défini la fonction g, on tape :
  GG:=plotfunc(g(x),x)
  tangent(GG, 1.2)
  trace la tangente au graphe G de la fonction g au point d'abscisse x=1.2,
  ou on tape :
  GA:=point(1.2+i*g(1.2))
  tangent(GG, GA)
  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 dans le CAS :
  ```
  g(x):=x^2
  ```
  Puis dans Symb, on tape :
  ```
  GG:=plotfunc(g(x),x)
  GT:=tangent(G,1)
  ```
  ou on tape :
  ```
  GT:=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 dans Num :
  ```
  equation(GT)
  ```
- l'objet géométrique n'est pas 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(circle(0,1),point(1+i))
```

On obtient :

```
La droite d'équation x=1 et la droite d'équation
                        y=1
```

On tape :

```
tangent(element(circle(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)

24.3.18 Axe radical de deux cercles : `radical_axis`

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 :

```
radical_axis(circle(0,1+i),circle(1,1+i)))
```

On obtient :

Le tracé de la droite d'équation $x = 1/2$

En effet : la droite $x = 1/2$ est la médiatrice du segment [0 ;1]

24.4 Polygon

24.4.1 Le triangle quelconque : `triangle`

`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
```

24.4.2 Triangle équilatèral : `equilateral_triangle`

`equilateral_triangle`, en géométrie plane, a deux ou trois arguments :

- 2 arguments
  Ces 2 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
  `equilateral_triangle(GA,GB)` 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 `GT:=equilateral_triangle(0,2)`) et utiliser la commande `vertices(GT)` qui renvoie la liste des sommets de T. On définira alors `GC:= vertices(GT)[2]` mais il est plus simple de rajouter GC le nom du dernier sommet comme troisième argument.

- 3 arguments
  Ces 3 arguments sont les 2 arguments précédents et le 3ième 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,GC)
```

  On obtient :

```
le triangle équilatèral de sommets les points
d'affixe 0,2,1+i*sqrt(3)
```

  On tape :

```
normal(affix(GC))
```

  On obtient :

```
1+i*sqrt(3)
```

24.4.3 Triangle rectangle : `right_triangle`

`right_triangle`, en géométrie plane, a trois ou quatre arguments :

- 3 arguments,
  ces 3 arguments sont : 2 points A et B (ou 2 nombres complexes représentant l'affixe de ces points) et un réel k non nul.
  `right_triangle(GA,GB,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 :

```
right_triangle(i,-i,2)
```

  On obtient :

```
Le triangle rectangle de sommets i, -i, 4+i
```

  On tape :

```
right_triangle(i,-i,-2)
```

On obtient :

```
Le triangle rectangle de sommets i, -i, -4+i
```

– 4 arguments
ces 4 arguments sont les 3 arguments précédents et un dernier argument qui est le nom d'une variable qui servira à définir le troisième sommet.
On tape :

```
right_triangle(i,-i,2,GD)
```

On obtient :

```
Le triangle rectangle de sommets i, -i, 4+i
```

On tape :

```
normal(affix(GD))
```

On obtient :

```
4+i
```

24.4.4 isosceles_triangle

isosceles_triangle, en géométrie plane, a trois ou quatre arguments :

– 3 arguments
Ces 3 arguments 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 $(\overrightarrow{AB}, \overrightarrow{AC})$.
isosceles_triangle(GA,GB,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 :

```
isosceles_triangle(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, -√2+i
```

– 4 arguments
Ces 3 arguments sont les 3 arguments précédents et le 4ième argument est le nom d'une variable qui servira à définir le troisième sommet.
On tape :

```
isosceles_triangle(i,1,-3*pi/4,GC)
```

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, -√2+i
```

On tape :

```
normal(affix(GC))
```

On obtient :

```
-sqrt(2)+i
```

24.4.5 Losange : rhombus

rhombus, en géométrie plane, peut avoir de trois à cinq arguments :

– 3 arguments,
ces 3 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe

de ces points et un nombre réel a.
`rhombus(GA,GB,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 :

```
rhombus(-2*i,sqrt(3)-i,pi/3)
```

On obtient si on a coché `radian` dans la configuration du `CAS` ou de [⌂](`Shift-CAS` ou `Shift-`[⌂])

`Le losange de sommets -2*i, √3-i, √3+i, 0`

– 4 (resp 5) arguments,
ces 4 (resp 5) arguments sont les 3 arguments précédents, 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 :

```
rhombus(-2*i,sqrt(3)-i,pi/3,E,F)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

`Le losange de sommets -2*i, √3-i, √3+i, 0`

On tape :

```
normal(affix(E))
```

On obtient :

```
sqrt(3)+i
```

On tape :

```
normal(affix(F))
```

On obtient :

```
0
```

24.4.6 `rectangle`

`rectangle`, en géométrie plane, peut avoir de trois à cinq arguments :

– 3 arguments,
ces trois arguments sont : deux points (ou deux nombres complexes représentant l'affixe de ces points) et un nombre réel k non nul.
`rectangle(GA,GB,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 :
$affix(GD) = affix(GA) + k * \exp(i * \pi/2) * (affix(GB) - affix(GA))$
mais sans définir les points C et D.
Remarque Si k est complexe, on a :
$affix(GD) = affix(GA) + k * \exp(i * \pi/2) * (affix(GB) - affix(GA))$
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
```

$$-1+i=(1+i)*\exp(i*\pi/2)$$

– 5 arguments, ces cinq arguments sont les 3 arguments précédents et les 2 derniers arguments sont les noms de deux variables qui serviront à définir les 2 derniers sommets.
On tape :

```
rectangle(0,1+i,-1/2,GG,GH)
```

On obtient :

```
Le rectangle de sommets 0,1+i,3/2+i/2,1/2-i/2
```

On tape :

```
normal(affix(GG))
```

On obtient :

```
(3+i)/2
```

On tape :

```
normal(affix(GH))
```

On obtient :

```
(1-i)/2
```

24.4.7 square

`square`, en géométrie plane, peut avoir de deux à quatre arguments :

– 2 arguments
Ces 2 arguments sont : 2 points ou 2 nombres complexes représentant l'affixe de ces points (ou encore une liste de 2 points ou de 2 complexes).
`squareGA,GB)` renvoie et trace le carrè $ABCD$ de sens direct, mais sans définir les points D et C.
On tape :

```
square(0,1+i)
```

On obtient :

```
Le carré de sommets 0, 1+i, 2*i, -1+i
```

– 3 (resp 4) arguments
Ces 3 (resp 4) arguments sont les 2 arguments orécédents suivi par le le nom d'une (resp deux) variable(s) qui serviront à définir l'avant-dernier sommet (resp les deux autres sommets). On tape :

```
square(0,1+i,GC,GD)
```

On obtient :

```
Le carré de sommets 0, 1+i, 2*i, -1+i
```

On tape :

```
affix(GC)
```

On obtient :

```
2*i
```

On tape :

```
affix(GD)
```

On obtient :

```
-1+i
```

24.4.8 quadrilateral

quadrilateral(GA,GB,GC,GD), en géométrie plane, renvoie et trace le quadrilatère $ABCD$.
On tape :

```
quadrilateral(0,1,1+i,-1+2*i)
```

On obtient :

```
Le "cerf-volant" de sommets 0, 1, 1+i, 1+2*i
```

24.4.9 parallelogram

parallelogram, en géométrie plane, a trois arguments ou quatre arguments :

– 3 arguments,
Ces 3 arguments sont : trois points (ou trois nombres complexes représentant l'affixe de ces points).
parallelogram(GA,GB,GC) renvoie et trace le parallélogramme $ABCD$ tel que : $\overrightarrow{AD} = \overrightarrow{BC}$ mais sans définir le point D.
On tape :

```
parallelogram(0,1,2+i)
```

On obtient :

```
Le parallélogramme de sommets 0,1,2+i,1+i
```

On tape :

```
parallelogram(1,0,-1+i)
```

On obtient :

```
Le parallélogramme de sommets 1,0,-1+i,i
```

– 4 arguments,
Ces 4 arguments sont les 3 arguments précédent et le 4ième argument est le nom d'une variable qui servira à définir le sommet manquant.
On tape :

```
parallelogram(0,1,2+i,GF)
```

On obtient :

```
Le parallélogramme de sommets 0,1,2+i,1+i et le
point F d'affixe 1+i
```

On tape :

```
normal(affix(GF))
```

On obtient :

```
1+i
```

24.4.10 isoploygon

isoploygon, 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 $k > 0$, isopolygon trace le polygone régulier direct ayant k cotés et comme sommets consécutifs les deux premiers arguments.
On tape :

```
isopolygon(0,1,4)
```

On obtient :

```
Le carré de sommets 0,1,1+i,i
```

Lorsque $k < 0$, `isopolygon` trace le polygone régulier direct ayant $-k$ cotés, comme centre le premier argument, et comme sommet le deuxième argument.
On tape :

```
isopolygon(0,1,-4)
```

On obtient :

```
carré de sommets 1,i,-1,-i
```

24.4.11 L'hexagone : `hexagon`

Voir aussi : ?? pour la géométrie 2-d.
`hexagon`, 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).
`hexagon(A,B)` renvoie et trace le hexagone $ABCDEF$ de sens direct, mais sans définir les points D,C,E et F.
On tape :

```
hexagon(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 :

```
hexagon(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 :

```
affix(C)
```

On obtient :

```
3/2+i*sqrt(3)/2
```

On tape :

```
affix(D)
```

On obtient :

```
1+i*sqrt(3)
```

On tape :

```
affix(E)
```

On obtient :

```
i*sqrt(3)
```

On tape :

```
affix(F)
```

On obtient :

```
-1/2+i*sqrt(3)/2
```

24.4.12 Le polygone : `polygon`

`polygon`, 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.
`polygon` renvoie et trace le polygone ayant pour sommets ces n points.
On tape :

```
polygon(-1,-1+i/2,i,1+i,-i)
```

On obtient :

```
Le polygone de sommets -1,-1+i/2,i,1+i,-i
```

On tape :

```
polygon(makelist(x->exp(i*pi*x/3),0,5,1))
```

On obtient :

L'hexagone de sommets $1, e^{\frac{i\pi}{3}}, e^{\frac{2i\pi}{3}}, .., e^{\frac{5i\pi}{3}}$

24.4.13 La ligne polygonale : `open_polygon`

`open_polygon`, 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.
`open_polygon` renvoie et trace la ligne polygonale ayant pour sommets ces n points.
On tape :

```
open_polygon(-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 :

```
open_polygon(makelist(x->exp(i*pi*x/3),0,5,1))
```

On obtient :

La ligne polygonale de sommets $1, e^{\frac{i\pi}{3}}, e^{\frac{2i\pi}{3}}, .., e^{\frac{5i\pi}{3}}$

24.4.14 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 `polygon` sur le résultat de `convexhull` pour obtenir le tracé de l'enveloppe convexe.
On tape

```
polygon(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).

24.5 Courbes

24.5.1 Le cercle et ses arcs : `circle`

`circle` a un ou deux arguments pour dessiner un cercle et de quatre à six arguments pour dessiner un arc de cercle :

- Avec un argument :
  l'argument de `cercle` est alors l'équation du cercle ayant comme variables x et y,
- Avec deux arguments :
  le premier argument de `cercle` est un point ou un nombre complexe considéré comme l'affixe d'un point.
  le deuxième argument 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).
  Ainsi :

  - `circle(GC,r)` où `GC` 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 `circle(GA,GB-GA)`.

  - `circle(GA,GB)` où `A` est un point ou un nombre complexe et `B` un point, trace le cercle de diamètre `AB`.

  On tape :

  ```
  circle(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 :

  ```
  circle(-1,i)
  ```

  On obtient :

  ```
  Le cercle de centre -1 et de rayon 1 est tracé
  ```

On tape :

```
circle(-1, point(i))
```

On obtient :

```
Le cercle de diamétre -1,i
```

– 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 :

```
circle(-1,1,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

L'arc AB (GA:=point(0) et GB:=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 :

```
circle(-1,i,0,pi/4,A,B)
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

L'arc AB (GA:=point(-1+i) et GB:=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 :

```
circle(-1, point(i),0,pi/4,A,B)
```

On obtient :

L'arc AB (GA:=point(i) et GB:=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.

24.5.2 Les arcs de cercle : `arc ARC`

Voir aussi : 24.5.1 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 `circle` a quatre arguments, `circle` dessine aussi un arc de cercle (cf **??**).

24.5.3 Le cercle circonscrit : `circumcircle`

`circumcircle` a trois paramètres qui définissent les sommets d'un triangle.
`circonscrit` dessine et renvoie le cercle circonscrit de ce triangle.
On tape :

```
circumcircle(-1,i,1+i)
```

On obtient :

```
Le cercle circonscrit du triangle(-1,i,1+i)
```

24.5.4 Tracé d'une conique : `conic`

`conic` 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 `reduced_conic` pour avoir l'équation paramétrique de la conique : On tape :

```
reduced_conic(2*x^2+2*x*y+2*y^2+6*x)[4]
```

On obtient :

```
[-2+i+(1+i)*(cos(t)+sqrt(3)*i*sin(t)),t,0,2π,2π/60]
```

24.5.5 L'ellipse : `ellipse`

`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(GF1,GF2,GA)` trace l'ellipse passant par `A` et de foyers `F1` et `F2` ou,
  `ellipse(GF1,GF2,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
```

24.5.6 Le cercle exinscrit : `excircle`

`excircle` 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 :

```
excircle(-1,i,1+i)
```

On obtient :

```
Le cercle exinscrit dans l'angle de sommet -1 du
triangle(-1,i,1+i) est tracé
```

24.5.7 L'hyperbole : `hyperbola`

`hyperbola`, 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(GF1,GF2,GA)` trace l'hyperbole passant par `A` et de foyers `F1` et `F2` ou,
  `hyperbole(GF1,GF2,a)` où `a` est un nombre réel, trace l'hyperbole de foyers `F1` et `F2` et de demi-grand axe `|a|`.

On tape :

```
hyperbola(-i,i,1+i)
```

On obtient :

```
L'hyperbole de foyers -i, i et passant par 1+i
```

On tape :

```
hyperbola(-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
```

24.5.8 Le cercle inscrit : `incircle`

`incircle` a trois paramètres qui définissent les sommets d'un triangle.
`inscrit` dessine et renvoie le cercle inscrit de ce triangle.
On tape :

```
incircle(-1,i,1+i)
```

On obtient :

```
Le cercle inscrit du triangle(-1,i,1+i)
```

24.5.9 Lieu et enveloppe : `locus`

`locus` permet de tracer le lieu d'un point qui dépend d'un autre point qui doit être défini avec la fonction `element`.
`locus` permet aussi de tracer l'enveloppe d'une droite qui dépend d'un point qui doit être défini avec la fonction `element`

- lieu d'un point.
  `locus` a 2 à 4 arguments.
  Les deux premiers argument sont deux noms de variables :
  le premier argument est le nom du point (par exemple `B`) dont on veut connaitre le lieu et ce point est fonction du deuxième argument,
  le deuxième argument est le nom du point (par exemple `A`) qui se déplace sur une courbe `C` et qui doit être défini par `GA:=element(GC)`.
  On peut préciser éventuellement en troisième argument l'intervalle où se trouve le paramètre 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)`.
  `locus` dessine le lieu du premier argument quand le deuxième argument se déplace selon ce que l'on a spécifié comme argument de `element`.
  Conseils
  Il faut avoir le moins possible d'instructions entre la définition de M et l'instruction `locus`.
  On tape, pour avoir le lieu du centre de gravité B du triangle de sommets point(-1), point(1) et A lorsque A décrit la droite d'équation $y = 1$:

```
GA:=element(line(i,1+i))
GB:=isobarycenter(-1,1,GA)
GC:=locus(GB,GA)
```

On obtient :

```
La droite parallèle à l'axe des x passant par i/3
```

On tape dans la vue numérique :

```
equation(GC)
```

On obtient :

```
equation(GC):y=1/3
```

– enveloppe d'une droite fonction d'un point mobile sur une courbe.
`locus` 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`.
`locus` 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 $x = 0$:

```
GF:=point(1)
GH:=element(line(x=0))
GD:=perpend_bisector(GF,GH)
locus(GD,GH)
```

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,
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 1)
On tape :

```
GH:=element(line(y=0));
GD:=line(y+x*tan(affix(M))-2*sin(affix(M)))
locus(GD,GH)
```

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 :

```
locus(GD,GH,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 `GD` et de `GE` (voir leur définition ci-dessous) pour avoir l'équation paramétrique du lieu.

```
GD:=y+x*tan(t)-2*sin(t)
GE:=diff(GD,t)
GM:=linsolve([GD=0,GE=0],[x,y])
GP:=plotparam(affix(simplify(GM)),t)
```

On obtient :

```
L'astroide d'équation paramérique
```

```
2*cos(t)^3+2*i*sin(t)^3
```

en effet `simplify(GM)` renvoie :
`[2*cos(t)^3,2*sin(t)^3]`

24.5.10 La parabole : `parabola`

`parabola`, 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(GF,GS)` renvoie et dessine la parabole de foyer `F` et de sommet `S`.
  `parabole(GS,c)` renvoie et dessine la parabole de sommet `S`=$x_s + iy_s$ et d'équation $y = y_S + c * (x - x_S)^2$. Il faut savoir que si p est le paramètre de la parabole, on a $FS = p/2$ et $c = 1/(2 * p)$.

On tape :

```
parabola(0,i)
```

On obtient :

```
La parabole de foyer 0 et de sommet i
```

On tape :

```
parabola(0,1)
```

On obtient :

La parabole de sommet 0 et déquation $y = x^2$

On tape :

```
parabola(x^2-y-1)
```

ou on tape :

```
parabola(-i,1)
```

ou on tape :

```
parabola(i,-i)
```

On obtient :

```
La parabole de sommet -i et de foyers i
```

24.5.11 Puissance d'un point par rapport à un cercle : `powerpc`

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 :

```
powerpc(circle(0,1+i),3+i)
```

On obtient :

```
8
```

En effet : $r = \sqrt{2}$ et $d = \sqrt{10}$ donc $d^2 - r^2 = 8$

24.6 Transformation

24.6.1 L'homothétie : `homothety`

`homothety` 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 `homothety` a deux arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
h:=homothety(i,2)
```

Puis :

```
h(1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

Lorsque `homothety` a trois arguments, `homothety` dessine et renvoie le transformé du troisième argument dans l'homothétie de centre le premier argument et de rapport le deuxième argument.
On tape :

```
homothety(i,2,1+i)
```

On obtient :

```
Le point 2+i tracé avec une croix (x) noire
```

On tape :

```
homothety(i,2,circle(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 `homothety(GA,k)` est la similitude de centre le point `A`, de rapport `abs(k)` et d'angle `arg(k)`.

24.6.2 L'inversion : `inversion`

`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 `GF:=inversion(GC,k)` et `GB:=GF(GA)`, on a $\overline{CA} * \overline{CB} = k$.
On tape :

```
GF:=inversion(i,2)
```

Puis :

```
GF(circle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
GF(circle(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,circle(1+i,1))
```

On obtient :

```
La droite verticale d'équation x=1
```

On tape :

```
inversion(i,2,circle(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
```

24.6.3 La projection orthogonale : `projection`

`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(line(-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(circle(-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 donné en 2ième argument par la projection orthogonale sur le premier argument.
On tape :

```
projection(line(-1,i),1+i)
```

On obtient :

```
Le point 1/2+3/2*i apparait avec une croix (x) noire
```

On tape :

```
projection(circle(-1,1),i)
```

On obtient :

```
Le point d'affixe, -1+sqrt(2)/2+(i)*sqrt(2)/2,
apparait avec une croix (x) noire
```

24.6.4 La symétrie droite et la symétrie point : `reflection`

`reflection`, en géométrie plane, a un ou deux arguments : un point ou une droite et éventuellement l'objet géométrique à transformer.
Lorsque `reflection` 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:=reflection(-1)
```

Puis :

```
sp(1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
sd:=reflection(line(-1,i))
```

Puis :

```
sd(1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

Lorsque `reflection` a deux arguments, `reflection` 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 :

```
reflection(-1,1+i)
```

On obtient :

```
Le point -3-i tracé avec une croix (x) noire
```

On tape :

```
reflection(line(-1,i),1+i)
```

On obtient :

```
Le point 2*i tracé avec une croix (x) noire
```

24.6.5 La rotation : `rotation`

`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` :

```
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` :

```
Le point 0 tracé avec une croix (x) noire
```

On tape :

```
rotation(i,-pi/2,line(1+i,-1))
```

On obtient si on a coché `radian` dans la configuration du `CAS` :

```
La droite passant par 0 et -1+2*i
```

24.6.6 La similitude : `similarity`

`similarity`, 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 `similarity` a trois arguments, c'est une fonction qui agit sur un objet géométrique.
On tape :

```
GS:=similarity(i,2,-pi/2)
```

Puis :

```
GS(1+i)
```

On obtient si on a choisi radian dans la configuration du CAS :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
GS(circle(1+i,1))
```

On obtient si on a choisi radian dans la configuration du CAS :

```
Le cercle de centre -i et de rayon 2
```

Lorsque similarity a quatre arguments, similarity 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 :

```
similarity(i,2,-pi/2,1+i)
```

On obtient si on a choisi radian dans la configuration du CAS :

```
Le point -i tracé avec une croix (x) noire
```

On tape :

```
similarity(i,2,-pi/2,circle(1+i,1))
```

On obtient si on a choisi radian dans la configuration du CAS :

```
Le cercle de centre -i et de rayon 2
```

Remarque
En 2d la similitude de centre le point GA, de rapport k et d'angle a se traduit par : similarity(GA,k,a) ou par homothety(GA,k*exp(i*a)).

24.6.7 La translation : translation

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 :

```
GA:=point(1);GB:=point(2+i);translation(vector(GA,GB),-2)
```

Ou on tape :

```
translation(1+i,-2)
```

Ou on tape :

```
GA:=point(1);GB:=point(2+i);translation(GB-GA,-2)
```

On obtient :

```
Le point -1+i tracé avec une croix (x) noire
```

On tape :

```
translation(1+i,line(-2,-i))
```

On obtient :

```
La droite passant par -1+i et 1
```

24.7 Mesure et graphique

24.7.1 Affichage de la mesure d'un angle : `angleat`

`angleat` a comme argument le nom de 3 points et 1 point (ou 1 nombre complexe qui est l'affixe de ce point).
Attention Il faut que les 3 premiers arguments soient des noms.
`angleat` 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 `anglat(GA,GB,GC,GD)` 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 cette commande dans La vue symbolique.
On tape :

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);
segment(GA,GB);segment(GA,GC);
angleat(GA,GB,GC,0.2i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` (`Shift-CAS`) :

```
αA=atan(1/3) s'écrit au point(0.4i)
```

24.7.2 Affichage de la mesure d'un angle : `angleatraw`

`angleatraw` a comme argument 4 points (ou 4 nombres complexes qui sont l'affixe de ces points).
`angleatraw` 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 `angleatraw(GA,GB,GC,GD)` 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 :

```
GA:=point(-1);GB:=point(1+i);GC:=point(i);

segment(GA,GB);segment(GA,GC);

angleatraw(GA,GB,GC,0.2i)
```

On obtient si on a coché `radian` dans la configuration du `CAS` (`Shift-CAS`) :

```
atan(1/3) s'écrit au point(0.4i)
```

24.7.3 Affichage de l'aire d'un polygone : `areaat`

`areaat` a comme arguments le nom d'un cercle ou d'un polygone étoilé par rapport à son premier sommet et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`areaat` 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 premier argument soit le nom d'un cercle ou d'un polygone.
On tape :

```
t:=triangle(0,1,i)

areaat(t,(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2 avec la légende
```

On tape :

```
cc:=circle(0,2)

areaat(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
c:=square(0,2)

areaat(c,2.2)
```

On obtient :

```
4 s'écrit au point(2.2) avec une légende
```

On tape :

```
h:=hexagon(0,1)

areaat(h,1.2)
```

On obtient :

```
3*sqrt(3)/2 s'écrit au point(1.2) avec une légende
```

24.7.4 Affichage de l'aire d'un polygone : `areaatraw`

`areaatraw` a 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 de ce point). `areaatraw` 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 :

```
areaatraw(triangle(0,1,i),(1+i)/2)
```

On obtient :

```
1/2 s'écrit au point(1+i)/2
```

On tape :

```
areaatraw(circle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
areaatraw(square(0,2),2.2)
```

On obtient :

```
4 s'écrit au point(2.2)
```

On tape :

```
areaatraw(hexagon(0,1),1.2)
```

On obtient :

```
3*sqrt(3)/2 s'écrit au point(1.2)
```

24.7.5 Affichage de la longueur d'un segment : `distanceat`

`distanceat` est une commande qui permet d'afficher en un point la longueur d'un segment avec une légende.
On tape cette commande dans La vue symbolique.
`distanceat` 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 de point.
`distanceat` 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 donner le nom des objets) :

```
GA:=point(-1);GB:=point(1+i);

distanceat(GA,GB,0.4i)
```

On obtient :

```
"GAB=sqrt(5)" s'écrit au point(0.4i)
```

On tape (on doit donner le nom des objets) :

```
GC:=point(0);GD:=line(-1,1+i)

distanceat(GC,GD,i/2)
```

On obtient :

```
"GCD=sqrt(5)/5" s'écrit au point(i/2)
```

On tape (on doit donner le nom des objets) :

```
GK:=circle(0,1); GL:=line(-2,1+3i)

distanceat(GK,GL,0)
```

On obtient :

```
"GKL=sqrt(2)-1" s'écrit au point(0)
```

24.7.6 Affichage de la longueur d'un segment : `distanceatraw`

`distanceatraw` est une commande qui permet d'afficher en un point la longueur d'un segment mais sans légende.
On tape cette commande dans La vue symbolique.
`distanceatraw` 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` 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 cette longueur à l'endroit du 3-ième point.
On tape :

```
GA:=point(-1);GB:=point(1+i);

distanceatraw(GA,GB,0.4i)
```

Ou on tape directement :

```
distanceatraw(point(-1),point(1+i),0.4i)
```

On obtient :

```
sqrt(5) s'écrit au point(0.4i)
```

On tape :

```
GC:=point(0);GD:=line(-1,1+i)

distanceatraw(GC,GD,i/2)
```

Ou on tape directement :

```
distanceatraw(point(0),line(-1,1+i),0.4i)
```

On obtient :

```
sqrt(5)/5 s'écrit au point(i/2)
```

On tape :

```
GK:=circle(0,1); GL:=line(-2,1+3i)

distanceatraw(GK,GL,0)
```

Ou on tape directement :

```
distanceatraw(circle(0,1),line(-2,1+3i),0.4i)
```

On obtient :

```
sqrt(2)-1 s'écrit au point(0)
```

24.7.7 Affichage du périmètre d'un polygone :`perimeterat`

`perimeterat` a comme argument le nom d'un cercle ou d'un polygone et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`perimeterat` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point avecune légende.
On tape cette commande dans La vue symbolique.
Attention Il faut que le premier argument soit le nom d'un cercle ou d'un polygone.
On tape :

```
t:=triangle(0,1,i)

perimeterat(t,(1+i)/2)
```

On obtient :

```
2+sqrt(2) s'écrit au point((1+i)/2) avec une légende
```

On tape :

```
c:=square(0,2)

perimeterat(c,2.2)
```

On obtient :

```
8 s'écrit au point(2.2) avec une légende
```

On tape :

```
cc:=circle(0,2)

perimeterat(cc,2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2) avec une légende
```

On tape :

```
h:=hexagon(0,1)

perimeterat(h,1.2)
```

On obtient :

```
6 s'écrit au point(1.2) avec une légende
```

24.7.8 Affichage du périmètre d'un polygone : `perimeteratraw`

`perimeteratraw` a comme argument un cercle un polygone et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`perimeteratraw` renvoie le point, calcule le périmètre du cercle ou du polygone et affiche ce périmètre à l'endroit du point.
On tape cette commande dans La vue symbolique.
On tape :

```
perimeteratraw(triangle(0,1,i),(1+i)/2)
```

On obtient :

```
2+sqrt(2) s'écrit au point((1+i)/2)
```

On tape :

```
perimeteratraw(circle(0,2),2.2)
```

On obtient :

```
4*pi s'écrit au point(2.2)
```

On tape :

```
perimeteratraw(hexagon(0,1),1.2)
```

On obtient :

```
6 s'écrit au point(1.2)
```

On tape :

```
perimeteratraw(square(0,2),2.2)
```

On obtient :

```
8 s'écrit au point(2.2)
```

24.7.9 Affichage de la pente d'une droite : `slopeat`

`slopeat` est une commande qui permet d'afficher en un point la pente d'une droite ou d'un segment avec une légende.
On tape cette commande dans La vue symbolique.
`slopeat` a 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).
`slopeat` 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 Il faut que le premier argument soit le nom d'une droite ou d'un segment.
On tape :

```
GD:=line(1,2i)
```

Ou on tape :

```
GD:=segment(1,2i),i)
slopeat(GD,i)
```

On obtient :

```
"sD=-2" s'écrit au point(i)
```

On tape :

```
GP:=line(2y-x=3),2*i)
slopeat(GP,2*i)
```

On obtient :

```
"sP=1/2" s'écrit au point(2*i)
```

On tape :

```
GT:=tangent(plotfunc(sin(x)),pi/4)
```

Ou on tape :

```
GT:=LineTan(sin(x),pi/4)
```

Puis :

```
slopeat(GT,i)
```

On obtient :

```
"sT=(sqrt(2))/2" s'écrit au point(i)
```

24.7.10 Affichage de la pente d'une droite : `slopeatraw`

`slopeatraw` est une commande qui permet d'afficher en un point la pente d'une droite ou d'un segment mais sans légende.
On tape cette commande dans La vue symbolique.
`slopeatraw` a 2 arguments une droite (ou un segment) et 1 point (ou 1 nombre complexe qui est l'affixe d'un point).
`slopeatraw` renvoie le point, calcule la pente de la droite (ou du segment) et affiche cette pente à l'endroit du point.
On tape :

```
GD:=line(1,2i)
slopeatraw(GD,i)
```

Ou on tape directement :

```
slopeatraw(line(1,2i),i)
```

On obtient :

```
-2 s'écrit au point(i)
```

On tape :

```
GE:=segment(1,2i),i)
slopeatraw(GE,1)
```

Ou on tape directement :

```
slopeatraw(segment(1,2i),1)
```

On obtient :

```
-2 s'écrit au point(1)
```

On tape :

```
GP:=line(2y-x=3,2*i)
slopeatraw(GP,2*i)
```

Ou on tape directement :

```
slopeatraw(line(2y-x=3,2*i),2*i)
```

On obtient :

```
1/2 s'écrit au point(2*i)
```

On tape :

```
GT:=tangent(plotfunc(sin(x)),pi/4)
slopeat(GT,i)
```

Ou on tape directement :

```
slopeatraw(tangent(plotfunc(sin(x)),pi/4),i)
```

On obtient :

```
(sqrt(2))/2 s'écrit au point(i)
```

24.8 Measure

24.8.1 L'abscisse d'un point ou d'un vecteur : `abscissa`

`abscissa`, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
`abscissa` renvoie l'abscisse du point ou du vecteur :
- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `abscisse(GA)` renvoie x_A,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `abscisse(GA-GB)` renvoie $x_A - x_B$ (car `GA-GB` désigne le vecteur $\overrightarrow{BA}$).

On tape :

```
abscissa(point(1+2*i))
```

On obtient :

```
1
```

On tape :

```
abscissa(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
abscissa(1+2*i)
```

On obtient :

```
1
```

On tape :

```
abscissa([1,2])
```

On obtient :

```
1
```

24.8.2 L'affixe d'un point ou d'un vecteur : `affix`

`affix` est une fonction qui a comme argument soit un point soit un vecteur soit les coordonnées d'un point ou d'un vecteur 2-d.
`affix` renvoie l'affixe du point ou du vecteur :

- si le point `A` a pour coordonnées cartésiennes (x_A, y_A), `affix(GA)` renvoie $x_A + i * y_A$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `affix(GA-GB)` ou `affix(vector(GB,GA))` renvoie $x_A - x_B + i * (y_A - y_B)$ (car `GA-GB` désigne le vecteur $\overrightarrow{BA}$) et `coordinates(vector(GB,GA))` renvoie $[x_A + i * y_A, x_B + i * y_B]$.

On tape :

```
affix(point(i))
```

On obtient :

```
i
```

On tape :

```
affix(point(i)-point(1+2*i))
```

On obtient :

```
-1-i
```

24.8.3 La mesure d'un angle : angle

angle a comme argument trois points (ou trois nombres complexes qui sont l'affixe de ces points) 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.
Ainsi :
angle(GA,GB,GC) désigne la mesure de l'angle en radians (ou en degrés) de $(\overrightarrow{AB}, \overrightarrow{AC})$.
angle(GA,GB,GC,"") trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté.
angle(GA,GB,GC,"a") trace l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ avec comme légende un petit arc orienté noté a.
angle(GA,GB,GC,"")[0] ou angle(GA,GB,GC,"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 choisi radian dans la configuration du CAS :

```
pi/4
```

On tape :

```
angle(0,1,1+i,"")
```

On obtient si on a coché radian dans la configuration du CAS :

```
[pi/4,circle(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` :

```
[pi/4,circle(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` :

```
[pi/2,polygon(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.

24.8.4 Longueur d'un arc de courbe : `arcLen`

`arcLen` a soit 1 soit quatre paramètres.
Attention il ne faut pas être en mode complexe !

- le paramètre est soit un cercle ou un arc de cercle, soit un polygone.
  On tape
  ```
  arcLen(circle(0,1,0,pi/2))
  ```
  On obtient :
  ```
  pi/4
  ```
  On tape
  ```
  arcLen(hexagon(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
  ```

24.8.5 Aire d'un polygone : `area`

`area` 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(square(0,2))
```

On obtient :

```
4
```

24.8.6 Les coordonnées d'un point, d'un vecteur ou d'une droite : `coordinates`

`coordinates`, en géométrie plane, est une fonction ayant comme argument un point ou un nombre complexe ou un vecteur ou une droite.

`coordinates` 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), `coordinates(GA)` renvoie $[x_A, y_A]$,
- si le point `B` a pour coordonnées cartésiennes (x_B, y_B), `coordinates(vector(GA,GB))` ou `coordinates(GB-GA)` 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), `coordinates(GV)` ou `coordinates(vector(GA,GV))` renvoie $[x_V, y_V]$,
- si une droite `D` est définie par deux points `A` et `B`, `coordinates(GD)` renvoie `[affix(GA),affix(GB)]`, si `D` est définie par son équation, `coordinates(GD)` renvoie `[affix(GA),affix(GB)]` où `A` et `B` sont deux points de la droite `D`, le vecteur `AB` ayant même orientation que `d`.

On tape :

```
coordinates(point(1+2*i))
```

Ou on tape :

```
coordinates(1+2*i)
```

On obtient :

```
[1,2]
```

On tape :

```
coordinates(point(1+2*i)-point(i))
```

Ou on tape :

```
coordinates(point(1+2*i)-point(i))
```

On obtient :

```
[1,1]
```

On tape :

```
coordinates(vector(point(i),point(1+2*i)))
```

Ou on tape :

```
coordinates(vector(i,1+2*i))
```

Ou on tape :

```
coordinates(vector([0,1],[1,2]))
```

On obtient :

```
[1,1]
```

On tape :

```
coordinates(1+2*i)
```

Ou on tape :

```
coordinates(vector(1+2*i))
```

Ou on tape :

```
coordinates(vecteur(point(i),vecteur(1+2*i)))
```

On obtient :

```
[1,2]
```

On tape :

```
coordinates(point(i),vector(1+2*i))
```

On obtient :

```
[1,2]
```

On tape :

```
d:=line(-1+i,1+2*i)
```

Ou on tape

```
d:=line(point(-1,1),point(1,2))
```

Puis,

```
coordinates(d)
```

On obtient :

```
[-1+i,1+2*i]
```

On tape :

```
d:=line(y=(1/2*x+3/2))
```

On obtient :

```
[(3*i)/2,1+2*i]
```

On tape :

```
d:=line(x-2*y+3=0)
```

On obtient :

```
[(3*i)/2,(-4+i)/2]
```

Attention
`coordinates` peut aussi avoir comme argument une séquence ou une liste de points. `coordinates` renvoie alors la séquence ou la liste des listes des coordonnées de ces points, par exemple :
`coordinates(i,1+2*i)` ou `coordinates(point(i),point(1+2*i))` renvoie la séquence :
`[0,1],[1,2]`
et
`coordinates([i,1+2*i])` ou `coordinates([point(i),point(1+2*i)])` renvoie la matrice :
`[[0,1],[1,2]]` donc
`coordinates([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`.

24.8.7 Les coordonnées rectangulaire d'un point : `rectangular_coordinates`

`rectangular_coordinates` renvoie la liste de l'abscisse et de l'ordonnée d'un point donné par la liste de ses coordonnées polaires.
On tape :

```
rectangular_coordinates(2,pi/4)
```

Ou on tape :

```
rectangular_coordinates(polar_point(2,pi/4))
```

On obtient :

```
[2/(sqrt(2)),2/(sqrt(2))]
```

24.8.8 Les coordonnées polaire d'un point : `polar_coordinates`

`polar_coordinates` renvoie la liste du module et de l'argument de l'affixe d'un point ou d'un nombre complexe ou de la liste de coordonnées rectangulaires.
On tape :

```
polar_coordinates(1+i)
```

Ou on tape :

```
polar_coordinates(point(1+i))
```

Ou on tape :

```
polar_coordinates([1,1])
```

On obtient :

```
[sqrt(2),pi/4]
```

24.8.9 La longueur d'un segment et distance entre les deux objets géométriques : distance

distance a comme argument deux points (ou deux nombres complexes qui sont l'affixe de ces points) ou deux objets géométriques.
distance renvoie la longueur du segment défini par ces deux points ou la distance entre les deux objets géométriques.
On tape :

```
distance(-1,1+i)
```

On obtient :

```
sqrt(5)
```

On tape :

```
distance(0,line(-1,1+i))
```

On obtient :

```
sqrt(5)/5
```

On tape :

```
distance(circle(0,1),line(-2,1+3i))
```

On obtient :

```
sqrt(2)-1
```

24.8.10 Le carré de la longueur d'un segment : distance2

distance2 a comme argument deux points (ou 2 points et 1 nombre complexe qui est l'affixe d'un point).
distance2 renvoie le carré de la longueur du segment défini par ces deux points.
On tape :

```
distance2(-1,1+i)
```

On obtient :

```
5
```

24.8.11 L'équation cartésienne d'un objet géométrique : equation

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) ou x:='x' et y:='y'.
On tape :

```
equation(line(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`.

24.8.12 Avoir comme réponse la valeur d'une mesure affichée : `extract_measure`

`extract_measure` permet d'obtenir la valeur d'une mesure qui a été affichée.
`extract_measure` a pour argument la commande qui a affiché cette mesure.
On tape :

```
GA:=point(-1);GB:=point(1+i);GC:=segment(GA,GB)
```

```
extract_measure(distanceat(GA,GB,i))
```

On obtient :

```
sqrt(5)
```

On tape :

```
extract_measure(distanceatraw(GA,GB,i))
```

On obtient :

```
sqrt(5)
```

On tape :

```
extract_measure(slopeat(GC,i))
```

On obtient :

```
1/2
```

On tape :

```
extract_measure(slopeatraw(GC,i))
```

On obtient :

```
1/2
```

24.8.13 L'ordonnée d'un point ou d'un vecteur : ordinate

ordinate, en géométrie plane, est une fonction ayant comme argument un point ou un vecteur ou un nombre complexe.
ordinate renvoie l'ordonnée du point ou du vecteur :
- si le point A a pour coordonnées cartésiennes (x_A, y_A), ordinate(GA) renvoie y_A,
- si le point B a pour coordonnées cartésiennes (x_B, y_B), ordinate(GA-GB) renvoie $y_A - y_B$ (A-B désigne le vecteur $\overrightarrow{BA}$).
On tape :

```
ordinate(point(1+2*i))
```

On obtient :

```
2
```

On tape :

```
ordinate(point(i)-point(1+2*i))
```

On obtient :

```
-1
```

On tape :

```
ordinate(1+2*i)
```

On obtient :

```
2
```

On tape :

```
ordinate([1,2])
```

On obtient :

```
2
```

24.8.14 L'équation paramétrique d'un objet géométrique : parameq

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(line(-1,i))
```

On obtient :

```
-t+(1-t)*(i)
```

On tape :

```
parameq(circle(-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)
```

24.8.15 Périmètre d'un polygone : `perimeter`

`perimeter` calcule le périmètre d'un cercle ou d'un polygone. Voir aussi la commande `arcLen`.
On tape :

```
perimeter(triangle(0,1,i))
```

On obtient :

```
2+sqrt(2)
```

On tape :

```
perimeter(square(0,2))
```

On obtient :

```
8
```

24.8.16 Le rayon d'un cercle : `radius`

`radius` a comme argument un cercle.
`radius` renvoie la longueur du rayon de ce cercle.
On tape :

```
radius(circle(-1,i))
```

On obtient :

```
1
```

On tape :

```
radius(circle(-1,point(i)))
```

On obtient :

```
sqrt(2)/2
```

24.8.17 Pente d'une droite : `slope`

`slope` est soit une commande soit un attribut de la commande `droite` pour cela voir 24.3.7
Lorsque `slope` est une commande son argument est une droite ou un segment ou deux points ou deux nombres complexes.
`slope` renvoie la pente de la droite définie par le segment ou par les 2 points ou leurs affixes.
On tape :

```
slope(line(1,2i))
```

Ou on tape :

```
slope(segment(1,2i))
```

Ou on tape :

```
slope(point(1),point(2i))
```

Ou on tape :

```
slope(1,2i)
```

On obtient :

```
-2
```

On tape :

```
slope(line(2y-x=3))
```

On obtient :

```
1/2
```

On tape :

```
slope(tangent(plotfunc(sin(x)),pi/4))
```

Ou on tape :

```
slope(LineTan(sin(x),pi/4))
```

On obtient :

```
(sqrt(2))/2
```

24.9 Test

24.9.1 Savoir si 3 points sont alignés : `is_colinear`

`is_colinear` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`is_colinear` vaut 1 si les points sont alignés, et vaut 0 sinon.
On tape :

```
is_colinear(0,1+i,-1-i)
```

On obtient :

```
1
```

On tape :

```
is_colinear(i/100,1+i,-1-i)
```

On obtient :

```
0
```

24.9.2 Savoir si 4 points sont cocycliques : `is_concyclic`

`is_concyclic` est une fonction booléenne ayant comme argument une liste ou une séquence de points.
`is_concyclic` vaut 1 si les points sont cocycliques, et vaut 0 sinon.
On tape :

```
is_concyclic(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
is_concyclic(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

24.9.3 Savoir si des éléments sont conjugués : `is_conjugate`

`is_conjugate` 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.
`is_conjugate` 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.
`is_conjugate` vaut 1 si les arguments sont conjugués, et vaut 0 sinon.
On tape :

```
is_conjugate(circle(0,1+i),point(1-i),point(3+i))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(circle(0,1),point((1+i)/2),line(1+i,2))
```

Ou on tape :

```
is_conjugate(circle(0,1),line(1+i,2),point((1+i)/2))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(circle(0,1),line(1+i,2), line((1+i)/2,0))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(point(1+i),point(3+i),point(i),point(i+3/2))
```

On obtient :

```
1
```

On tape :

```
is_conjugate(line(0,1+i),line(2,3+i),
     line(3,4+i),line(3/2,5/2+i))
```

On obtient :

```
1
```

24.9.4 Savoir si des points ou /et des droites sont coplanaires : `is_coplanar`

`is_coplanar` teste si une liste ou une séquence de points ou de droites sont coplanaires.
On tape :

```
is_coplanar([0,0,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
1
```

On tape :

```
is_coplanar([-1,2,0],[1,2,-3],[1,1,-2],[2,1,-3])
```

On obtient :

```
0
```

On tape :

```
is_coplanar([0,0,0],[1,2,-3],line([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
is_coplanar(line([0,0,0],[1,2,-3]),line([1,1,-2],[2,1,-3]))
```

On obtient :

```
1
```

On tape :

```
is_coplanar(line([-1,2,0],[1,2,-3]),
      line([1,1,-2],[2,1,-3]))
```

On obtient :

```
0
```

24.9.5 Savoir si 1 point est sur un objet graphique : `is_element`

`is_element` est une fonction booléenne ayant comme argument un point et un objet géométrique.
`is_element` vaut 1 si le point appartient à l'objet géométrique, et vaut 0 sinon.
On tape :

```
is_element(point(-1-i),line(0,1+i))
```

On obtient :

```
1
```

On tape :

```
is_element(point(i),line(0,1+i))
```

On obtient :

```
0
```

24.9.6 Savoir si on a un triangle équilatéral : `is_equilateral`

`is_equilateral` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`is_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 :

```
is_equilateral(0,2,1+i*sqrt(3))
```

On obtient :

```
1
```

On tape :

```
GT:=equilateral_triangle(0,2,GC);is_equilateral(GT[0])
```

On obtient :

```
1
```

En effet `GT[0]` désigne un triangle car `GT` est une liste composée du triangle et de son sommet `GC`.
On tape `affix(GC)` et on obtient `1+i*sqrt(3)`
On tape :

```
is_equilateral(1+i,-1+i,-1-i)
```

On obtient :

```
0
```

24.9.7 Savoir si on a un triangle isocéle : `is_isosceles`

`is_isosceles` est une fonction booléenne ayant comme argument trois points ou un objet géométrique.
`is_isoceles` 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 :

```
is_isosceles(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
GT:=isosceles_triangle(0,1,pi/4);is_isoceles(GT)
```

On obtient :

```
1
```

On tape :

```
GT:=isosceles_triangle(0,1,pi/4,GC);is_isoceles(GT[0])
```

On obtient :

```
1
```

En effet `GT[0]` désigne un triangle car `GT` est une liste composée du triangle et de son sommet `C`.
On tape `affix(GC)` et on obtient `(sqrt(2))/2+((i)*sqrt(2))/2`
On tape :

```
is_isosceles(1+i,-1+i,-i)
```

On obtient :

```
3
```

24.9.8 Orthogonalité de 2 droites ou 2 cercles : `is_orthogonal`

`is_orthogonal` est une fonction booléenne ayant comme argument deux droites ou deux cercles.
`is_orthogonal` vaut 1 si les deux droites ou les deux cercles (i.e si les tangentes en leurs points d'intersection sont orthogonales), et vaut 0 sinon.
On tape :

```
is_orthogonal(line(1,i), line(0,1+i))
```

On obtient :

```
1
```

On tape :

```
is_orthogonal(line(2,i), line(0,1+i))
```

On obtient :

```
0
```

On tape :

```
is_orthogonal(circle(0,1),circle(sqrt(2),1))
```

On obtient :

```
1
```

On tape :

```
is_orthogonal(circle(0,1),circle(2,1))
```

On obtient :

```
0
```

24.9.9 Savoir si 2 droites sont parallèles : `is_parallel`

`is_parallel`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`is_parallel` vaut 1 si les deux droites sont parallèles, et vaut 0 sinon.
On tape :

```
is_parallel(line(0,1+i),line(i,-1))
```

On obtient :

```
1
```

On tape :

```
is_parallel(line(0,1+i),line(i,-1-i))
```

On obtient :

```
0
```

24.9.10 Savoir si on a un parallélogramme : `is_parallelogram`

`is_parallelogram` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_parallelogram` 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 :

```
is_parallelogram(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

On tape :

```
is_parallelogram(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GQ:=quadrilateral(1+i,-1+i,-1-i,1-i);is_parallelogram(GQ)
```

On obtient :

```
4
```

Attention
On doit taper :

```
GP:=parallelogram(-1-i,1-i,i,GD);is_parallelogram(GP[0])
```

Pour obtienir :

```
1
```

En effet c'est `GP[0]` qui désigne un parallélogramme car `GP` est une liste composée d'un parallélogramme et de son dernier sommet `D`.
Si on tape `affix(GD))`, on obtient `-2+i`.

24.9.11 Savoir si 2 droites sont perpendiculaire : is_perpendicular

`is_perpendicular`, en géométrie plane, est une fonction booléenne ayant comme argument deux droites.
`is_perpendicular` vaut 1 si les deux droites sont perpendiculaires, et vaut 0 sinon.
On tape :

```
is_perpendicular(line(0,1+i),line(i,1))
```

On obtient :

```
1
```

On tape :

```
is_perpendicular(line(0,1+i),line(1+i,1))
```

24.9.12 Savoir si on a un triangle rectangle ou si on a un rectangle : is_rectangle

`is_rectangle` est une fonction booléenne ayant comme argument trois ou quatre points ou un objet géométrique.
`is_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,
`is_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 :

```
is_rectangle(1,1+i,i)
```

On obtient :

```
2
```

On tape :

```
is_rectangle(1+i,-2+i,-2-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GR:=rectangle(-2-i,1-i,3,GC,GD);is_rectangle(GR[0])
```

On obtient :

```
1
```

En effet `GR[0]` désigne un rectangle car `GR` est une liste composée du rectangle et de ses sommets `C` et `D`.

24.9.13 Savoir si on a un losange : `is_rhombus`

`is_rhombus` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_rhombus` 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 :

```
is_rhombus(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GK:=rhombus(1+i,-1+i,pi/4);is_rhombus(GK)
```

On obtient :

```
1
```

On tape :

```
GK:=rhombus(1+i,-1+i,pi/4,GC,DD);is_rhombus(GK[0])
```

On obtient :

```
1
```

En effet `GK[0]` désigne un losange car `GK` est une liste composée d'un losange et de ses sommets `GC` et `GD`.
Si on tape : `normal(coordinates(GC,GD))`,
on obtient `[-sqrt(2)-1,-sqrt(2)+1],[-sqrt(2)+1,-sqrt(2)+1]`.
On tape :

```
is_rhombus(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

24.9.14 Savoir si on a un carré : `is_square`

`is_square` est une fonction booléenne ayant comme argument quatre points ou un objet géométrique.
`is_square` vaut 1 si les quatre points forment un carré ou si l'objet géométrique est un carré, et vaut 0 sinon.
On tape :

```
is_square(1+i,-1+i,-1-i,1-i)
```

On obtient :

```
1
```

On tape :

```
GK:=square(1+i,-1+i);is_square(GK)
```

On obtient :

```
1
```

On tape :

```
GK:=square(1+i,-1+i,C,D);is_square(GK[0])
```

On obtient :

```
1
```

En effet `GK[0]` désigne un carré car `GK` est une liste composée d'un carré et de ses sommets `C` et `D`.
Si on tape `affix(GC,GD)`, on obtient `-1-i,1-i`.
On tape :

```
is_square(i,-1+i,-1-i,1-i)
```

On obtient :

```
0
```

24.9.15 Savoir si 4 points forment un division harmonique : `is_harmonic`

indexis_harmonic|textbf `est_harmonique` permet de savoir si 4 points forment un division harmonique. `is_harmonic` est une fonction booléenne ayant comme arguments quatre points.
`is_harmonic` vaut 1 si les 4 points forment un division harmonique et vaut 0 sinon.
On tape :

```
is_harmonic(0,2,3/2,3)
```

On obtient :

```
1
```

On tape :

```
is_harmonic(0,1+i,1,i)
```

On obtient :

```
0
```

24.9.16 Ces droites sont en faisceau ? is_harmonic_line_bundle

is_harmonic_line_bundle a comme argument une liste de droites.
is_harmonic_line_bundle 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 :

```
is_harmonic_line_bundle([line(0,1+i),line(0,2+i),
             line(0,3+i),line(0,1)])
```

On obtient :

```
                              1
```

24.9.17 Ces cercles sont-ils en faisceau : is_harmonic_circle_bundle

is_harmonic_circle_bundle a comme argument une liste de cercles.
is_harmonic_circle_bundle 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 :

```
is_harmonic_circle_bundle([circle(0,i),circle(4,i),
              circle(0,point(1/2))])
```

On obtient :

```
                              1
```

24.10 Exercices de géométrie

24.10.1 Les transformations

- La translation Un pavage :tout quadrilatère plan non croisé pave le plan.
  On définit 4 points A,B,C,D au hasard :
  menu Points->Free points->4 random point
  Dans Symb on change le nom des points de façon que le quadrilatère A, B, C, D ne soit pas croisé.
  Puis on définit le quadrilatère A, B, C, D avec le menu Lines->Polygons->quadrilateral et ce sera le pavé de base : GE:=quadrilateral(GA,GB,GC,GD)

```
GA:=point()
GB:=point()
GC:=point()
GD:=point
GE:=quadrilateral(GA,GB,GC,GD)
```

```
GG:=segment(GA,GB)
GH:=segment(GB,GC)
GI:=segment(GC,GD)
GJ:=segment(GD,GA)
GK:=midpoint(GA,GB)
GL:=reflection(GK,GC)
GM:=reflection(GK,GD)
GN:=quadrilateral(GA,GB,GL,GM)
GO:=segment(GA,GB)
GP:=segment(GB,GL)
GQ:=segment(GL,GM)
GR:=segment(GM,GA)
translation(GB-GD,[GE,GN])
translation(GC-GA,[GE,GN])
```

– `L'inversion` L'inverseur de Peaucelier

```
GA:=element(-1.6..1.6,0.6)
GD:=circle(1,1)
GE:=point(1+EXP(i*GA)
GH:=circle(GE,2.)
GI:=circle(0,3.)
GJ:=inter(GH,GI)
GK:=reflection(GJ,GE)
GL:=locus(GK,GA)
GG:=quadrilateral(GE,GJ[0],GK,GJ[1])
GB:=segent(0,GJ[0])
GC:=segent(0,GJ[1])
```

24.10.2 Les lieux

Soit un triangle direct OAB rectangle en O avec $OA = a$ et $OB = b$.
Soit $D = At$ une demi droite variable telle que :
$(\overrightarrow{OA}, \overrightarrow{At}) = c, 0 \leq c \leq \frac{\pi}{2}$.
Soient $A1$ et $B1$ les projections respectives de A et B sur D.
Quelle est la valeur de c pour laquelle $A1$ et $B1$ sont confondus en un point que l'on nommera P ? Trouver les lieux de $A1$ et de $B1$ quand c varie.
Montrer que le triangle $PA1B1$ reste semblable au triangle OAB quand c varie.
Trouver le lieu de M milieu de $A1B1$ quand c varie.

```
GA:=point(0.,3.)
GB:=point(5.,0.)
GC:=arc(1.5*i,1.5)
GD:=(inter(GC,line(GA,GB)))[0]
GE:=arc(1.5*i,-pi/2.,pi/2)
GG:=element(0..1.57,0.25)
GH:=line(y=TAN(GG)*x)
GI:=projection(GH,GA)
GJ:=projection(GH,GB)
GK:=triangle(GD,GI,GJ)
GL:=triangle(GD,2.5,1.5*i)
```

```
GM:=midpoint(GI,GJ)
GN:=trace(GM)
```

24.11 Activités de géométrie

– Médiatrice de AB
Créer un segment AB.
Construire la médiatrice de AB, en utilisant la même construction qu'avec un compas.
Réponse
On sélectionne :
Lines->segment et on définit avec le curseur deux points A et B et le segment C est définit automatiquement dans SYMB (GC:=segment(GA,GB)).
perpen_bisector trace directement la médiatrice de AB.
Pour faire la même construction qu'avec un compas :
On sélectionne : Curves->Circles->circle
On pointe le centre B (ou on tape Alpha B), on valide avec ENTER ou avec , puis on pointe le point A (ou on tape Alpha A) .
Il s'inscrit en dessous de la figure :
circle(GB,GA-GB) (c'est le cercle D de centre B passant par A) que l'on valide avec ENTER (GD :=circle(GB,GA-GB) est définit automatiquement dans SYMB).
Puis, on pointe le centre A puis le point B.
Il s'inscrit en dessous de la figure :
circle(GA,GB-GA) (c'est le cercle E de centre A passant par B) que l'on valide avec ENTER (GE :=circle(GA,GB-GA) est définit automatiquement dans SYMB).
On peut taper dans SYMB :
GG:=line(inter(GD,GE)) pour tracer la droite joignant les deux points de l'intersection de GD et de GE (inter(GD,GE) est la liste des points de cette intersection).
Ou bien, on définit l'intersection avec Point->inter puis, en désignant le premier point d'intersection, puis le second point d'intersection.

– Milieu
Créer un segment [AB].
Construire le milieu de AB, soit en utilisant les coordonnées, soit en utilisant la même construction qu'avec un compas.
On tape : GI:=point(coordinates(GA)/2+coordinates(GB)/2)
ou bien on rajoute à la construction de la médiatrice (cf ci-dessus) :
GI:=single_inter(GC,GG)
On peut aussi comme exercice de programmation définir la fonction Milieu (il faut commencer la fonction par une majuscule car milieu est une commande de Xcas).
On tape :
Milieu(A,B):=point(coordinates(A)/2+coordinates(B)/2)
ou encore si on a défini la fonction Mediatrice :
Milieu(A,B):=single_inter(segment(A,B),Mediatrice(A,B))

– Isobarycentre
Créer 4 points A,B,C,D.
Définir l'isobarycentre de A,B,C,D, en utilisant les coordonnées.
Réponse
On tape dans SYMB :
```
GE:=point((coordinates(GA)+coordinates(GB)+coordinates(GC)+
coordinates(GD))/4)
```
On peut, comme exercice de programmation définir la fonction ISOBAR (**ATTENTION !** isobarycenter est une commande qui existe déjà).
On tape comme nom de programme ISOBAR et on coche CAS.
On tape (les variables doivent être des minuscules) :
```
(l)->BEGIN
LOCAL s,d;
d:=size(l);
s:=sum(l[k],k,1,d)/d;
RETURN s;
END;
```
On tape par exemple :
```
ISOBAR(0,1,1+i,i)
```
On obtient : (1+i)/2

– Barycentre
Créer 4 points A,B,C,D.
Définir le barycentre de [A,1],[B,−2],[C,1],[D,3], en utilisant les coordonnées.
Réponse
On tape :
```
GE:=point((coordinates(GA)-2*coordinates(GB)+coordinates(GC)+
3*coordinates(GD))/3)
```
On peut aussi comme exercice de programmation définir la fonction BARY (ATTENTION ! barycenter est une commande qui existe déjà) qui renvoie le barycentre des points $A, B...$ affectés des coefficients $\alpha, \beta....$
On tape comme nom de programme BARY et on coche CAS.
On tape (les variables doivent être des minuscules) et on suppose que l est la liste affix(GA),α,affixGB, $\beta...$:

```
(l)->BEGIN
LOCAL s,d;
d:=size(l);
s:=sum(l[k],k,2,d,2);
IF s==0 THEN RETURN "pas defini" END;
RETURN sum(L[k]*L[k+1],k,1,d,2)/s;
END;
```
On tape par exemple :
```
BARY(0,2,1,1
```
On obtient : 1/3

– Bissectrice d'un angle
Créer un triangle ABC.
Construire les bissectrices de l'angle A du triangle ABC, en utilisant la

même construction qu'avec un compas et en utilisant l'instruction `perpen_bissector` qui trace la médiatrice d'un segment.

Réponse

On sélectionne :
`Lines->Triangle->triangle` et on définit avec le curseur trois points `A`, `B` et `C` et le triangle `D` ainsi que ses cotés sont définis automatiquement dans `SYMB` (`GD:=triangle(GA,GB,GC)`), `GE:=segment([GA,GB])`,`GG:=segment([GB,GC])`, `GH:=segment([GC,GA])`.
On sélectionne : `Curves->Circles->circle`
On suppose que $AB < AC$ pour que le cercle de centre A qui passe par B coupe le segment `H` qui est le segment AC.
On pointe le centre `A` (ou on tape `Alpha A`), on valide avec `ENTER` ou avec `,` puis on pointe le point `B` (ou on tape `Alpha B`) .
Il s'inscrit en dessous de la figure :
`circle(GA,GB-GA)` qui)=est le cercle `I` de centre `A` passant par `B`.
On sélectionne `Points->Dep.points->inter` et on montre le cercle `I`, puis le segment `H` `GJ:=inter(GI,GH)` définit l'intersection du cercle `I` avec le segment `H` qui est le segment AC.
Puis on sélectionne `perpen_bissector` pour tracer la médiatrice de `BJ`
`GK:=perpen_bissector(GB,GJ)`
On peut aussi comme exercice de programmation définir les fonctions `Bissectrice` et `Exbissectrice` (Attention ! `bissector` et `exbissector` sont des commandes qui existent déjà).
On tape si on a défini la fonction `Mediatrice` :

```
Bissectrice(GA,GB,GC):=perpen_bissector(single_inter(half_line(GA,GB),
circle(A,2)),single_inter(half_line(GA,GC),circle(GA,2)))
Exbissectrice(GA,GB,GC):={local GC1:=GA+(GA-GC);Bissectrice(GA,GB,GC1)}
```

– Report d'une longueur
Étant donnés trois points `A` `B` et `C`, on veut construire un point `D` pour que $AD = BC$.
On utilise la commande `circle` et on tape :
`GD:= element(circle(GA,GB-GC))`

L'instruction `distance(GB,GC)` renvoie la longueur du segment BC (les unités étant définies sur la fenêtre graphique).
Si l'on veut reporter une longueur dans une direction donnée, on multiplie cette longueur par le vecteur unitaire de cette direction.
Exemple :
Étant donnés trois points `A` `B` et `C`, construire sur la demi-droite AB, un point D tel que $AD = AC$.
On tape :
`GD:=GA+distance(GA,GC)*(GB-GA)/distance(GA,GB)`
ou encore
`GD:=single_inter(circle(GA,GC-GA),half_line(GA,GB))`

– Report d'un angle
Étant donnés deux points `A` et `B`, on veut construire `C` pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ soit de mesure donnée par exemple 72 degrés ou $2 * \pi/5$ ra-

dians.
On tape, si on a coché `radian` dans la fenêtre de configuration du `cas` :
`GD:=rotation(GA,2*pi/5,line(GA,GB))`
ou, si on est en degré (on n'a pas coché `radian`) :
`GD:=rotation(GA,72,line(GA,GB))`
puis on tape :
`GC:=element(GD)`
L'instruction `angle(GA,GB,GC)` donne la mesure en radians (ou en degrés) de l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$, on peut donc vérifier la construction demandée.
Étant donné deux points `A` et `B`, on veut construire `C` pour que l'angle $(\overrightarrow{AB}, \overrightarrow{AC})$ soit égal à l'angle $(\overrightarrow{OM}, \overrightarrow{OP})$.
On tape :
`GD:=rotation(GA,angle(GO,GM,GP),line(GA,GB));`
`GC:=element(GD)`

– Perpendiculaire à la droite BC passant par A
Créer un point `A` et une droite `BC` ne passant pas par `A`.
Construire la perpendiculaire à la droite BC passant par A, en utilisant la même construction qu'avec un compas.
Réponse
On sélectionne :
`Points->point` et on définit avec le curseur le point `A`, puis on sélectionne `Lines->line` et on définit avec le curseur les points `B` et `C` et la droite `D` est définit automatiquement dans `SYMB` (`GD:=line(GB,GC)`).
On sélectionne : `Curves->Circles->circle`
On pointe le centre `B` (ou on tape `Alpha B`), on valide avec `ENTER` ou avec `,` puis on pointe le point `A` (ou on tape `Alpha A`) .
Il s'inscrit en dessous de la figure :
`circle(GB,GA-GB)` qui est le cercle `E` de centre `B` passant par `A`.
On pointe le centre `C` (ou on tape `Alpha C`), on valide avec `ENTER` ou avec `,` puis on pointe le point `A` (ou on tape `Alpha A`) .
Il s'inscrit en dessous de la figure :
`circle(GC,GA-GC)` qui est le cercle `G` de centre `C` passant par `A`.
On sélectionne `Points->Dep.points->inter` et on montre le premier point d'intersection des cercles `E` et `G`. Cela définit le point `H`, puis on montre leur second point d'intersection et cela définit le point `K`. On trace ensuite la droite `HK`.

– Perpendiculaire à la droite AB passant par A
Construire la perpendiculaire à la droite AB passant par A en utilisant la même construction qu'avec un compas.
Réponse
On sélectionne :
`Lines->line` et on définit avec le curseur les points `A` et `B` et la droite `C` est définie automatiquement dans `SYMB` (`GC:=line(GA,GB)`).
On sélectionne : `Curves->Circles->circle`
On pointe le centre `A` (ou on tape `Alpha A`), on valide avec `ENTER` ou avec `,` puis on pointe le point `B` (ou on tape `Alpha B`) .
Il s'inscrit en dessous de la figure :
`circle(GA,GB-GA)` qui est le cercle `D` de centre `A` passant par `B`.

On sélectionne `Points->Dep.points->inter` et on montre le point d'intersection (autre que `B`) du cercle `D` avec la droite `C`. Cela définit le point `E`, Puis on trace la médiatrice de `BE`. `perpen_bissector(GB,GE)` trace la mediatrice des deux points définit par `inter(C3,line(GA,GB))`, cette médiatrice passe par A et est perpendiculaire à AB.
On peut aussi comme exercice de programmation définir la fonction `PERP` (Attention `perpendicular` est une commande qui existe déjà).
On tape comme nom de programme `PERP` et on coche `CAS`.
On tape (les variables doivent être des minuscules ou des noms d'objets géométriques) et on suppose `GA` est un point et que `GD` est une droite :

```
(GA,GD)->BEGIN
LOCAL  GM,GE,GL;
GE:=element(GD);
IF est_element(GA,GD) THEN
GL:=inter(GD,circle(GA,1));
RETURN equation(perpen_bisector(GL));
END;
IF angle(GE,GA,GD)==pi/2 OR angle(GE,GA,GD)==-pi/2
THEN RETURN equation(line(GA,GE)); END;
GM:=midpoint(op(inter(GD,circle(GA,GE-GA))));
RETURN equation(line(GA,GM));
END;
```

On tape :
`PERP(point(i),line(0,1+i))`
On obtient :
`y=-3*x+1`
On tape :
`PERP(point(i),line(-2,2+2i))`
On obtient :
`y=-2*x+1`
On tape :
`PERP(point(1),line(0,1+i))`
On obtient :
`y=-x+1`
On tape :
`PERP(point(1),line(-2,1+i))`
On obtient :
`y=-3*x+3`

– Parallèle à une droite passant par A
Créer un point `A` et un segment `BC` ne passant pas par `A`. Construire la parallèle à la droite BC passant par A en utilisant l'instruction `perpendiculaire`.
Réponse
On clique avec la souris pour avoir un point `A` et deux points `B` et `C` et le segment `BC`.
On tape :
`GD:=perpendicular(GA,line(GB,GC))`
cela trace la perpendiculaire à BC passant par A

GP:=perpendicular(GA,GD)
cela trace la perpendiculaire à D passant par A.
On peut, comme exercice de programmation définir la fonction PARAL qui renvoie l'équation de la parallèle à D passant par A (il faut commencer le nom de la fonction par une majuscule car parallel est une commande de géométrie qui existe déjà).
On tape comme nom de programme PARAL et on coche CAS.
On tape (les variables doivent être des minuscules ou des noms d'objets géométriques) et on suppose GA est un point et que GD est une droite :

```
(GA,GD)->BEGIN
LOCAL  GP:=line(PERP(GA,GD));
RETURN PERP(GA,GP);
};
```

On tape :
PARAL(point(i),line(0,1+i))
On obtient :
y=x+1 On tape :
PARAL(point(1),line(-2,2+i))
On obtient :
y=(1/4)*x-1/4

– Parallèles à une droite situées à une distance d de cette droite Créer un point A et un segment BC ne passant pas par A.
Puis créer un point D pour définir d =sistance(A,D).
Construire les parallèles à la droite BC situées à une distance d =distance(A,D) de BC.
Réponse
On clique avec la souris pour avoir un point A et deux points B et C et le segment BC, puis on clique avec la souris pour avoir un point D ($d = distance(A, D)$).
On tape :

```
GD1:=perpendicular(GB,line(GB,GC));
GC1:=circle(GB,distance(GA,GD));
GI:=inter(GD1,GC1);
GE:=GI[0];
GF:=GI[1];
```

ou on utilise les nombres complexes pour définir GE et GF :
GE:=GB+i*(GC-GB)*distance(GA,GD)/distance(GB,GC) le point E est à une distance $d = distance(A, D)$ de la droite BC,
GF:=GB-i*(GC-GB)*distance(GA,GD)/distance(GB,GC) le point F est à une distance $d = distance(A, D)$ de la droite BC (E et F sont symètriques par rapport à BC),
puis,
parallel(GE,line(GB,GC)) trace une parallèle à la droite BC situées à une distance $d = distance(A, D)$ de BC.
parallel(GF,line(GB,GC)) trace l'autre parallèle à la droite BC situées à une distance $d = distance(A, D)$ de BC.

Chapitre 25

Le tableur

25.1 Généralités

Quand on a ouvert l'application `Tableur`, la touche `Menu` donne accés aux fonctions :
`SUM AVERAGE AMORT STAT1 REGRS PredY PredX HypZ1mean HypZ2mean`
Le tableur est une feuille de calculs ayant la forme d'un tableau composé de lignes et de colonnes qui déterminent des cases appelées cellules. Les cellules contiennent des valeurs ou des commandes ou encore des formules qui font références aux autres cellules.

25.2 Description de l'écran du tableur

Le tableur est un tableau composé de colonnes désignées par des lettres en majuscules `A,B,C,...` et quelquefois en minuscules `g,l,m, z` et de lignes numérotées par `1,2,...`.
Les cases du tableur sont appelées cellules.
Ainsi, `A1` désigne la première cellule du tableur.

25.2.1 Pour copier la formule d'une cellule

Par exemple :
On met `1` dans `A1` et on veut recopier vers le bas la formule `=A1+1` valable dans `A2` pour avoir 1,2,3,4,5 dans la colonne A.
Il y a 3 moyens de le faire : à vous de choisir celui qui vous convient !

- technique 1 : avec `select` du bandeau on sélectionne `A2` jusqu'à `A5`, et on tape : `=A1+1`,
- technique 2 : on entre `=A1+1` dans `A2`, on fait `copy`, puis avec `select` du bandeau on sélectionne `A3` jusqu'à `A5`, et on fait `paste` et on choisit la formule parmi les expressions qui sont dans le presse papier,
- technique 3 : on met `=A1+1` dans `A2`, puis on sélectionne `A2` jusqu'à `A5` puis `Edit` et `Enter`

25.2.2 Références absolues et relatives

Dans une cellule on peut mettre :
- une chaine de caractères,
- une expression algébrique,
- une formule faisant référence à d'autres cellules. Ces références peuvent être absolues ou relatives à la cellule qui contient la formule. Les références absolues sont obtenues en rajoutant `$` devant la lettre désignant la colonne ou devant le numéro de la ligne de la cellule référence.

Les références relatives permettent de désigner les cellules par rapport à une autre : ainsi `A0` mis dans la cellule `B1` désigne la cellule située dans la colonne précédente et à la ligne précédente et c'est cette information qui sera recopiée quand on recopiera la formule vers le bas ou vers la droite.
Exemples :
Dans `A1` il y a 1 et je tape dans `B2` la formule :

- `$A$1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, j'obtiens des 3 dans la colonne `B` car je recopie dans toutes les cases de la colonne `B` la formule `$A$1+2`. Si je recopie cette formule vers la droite, j'obtiens aussi des 3 dans la première ligne, car je recopie dans toutes les cases de la première ligne la formule `$A$1+2` puisque `$A$1` est la référence absolue de la case `A1`.
- `$A1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas cette formule deviendra `$A2+2` dans `B2`, `$A2+2` dans `B3`. La valeur de `B2` dépend donc de la valeur de `A1`, la valeur de `B3` dépend donc de la valeur de `A2` etc...
  Si je recopie cette formule vers la droite, cette formule deviendra `$A1+2` dans `C2`, `$A1+2` dans `D2` ...j'obtiens donc une ligne de 3. `$A0` fait toujours référence à la colonne `A` : `A` est une référence absolue mais `0` désigne ici la ligne précédente puisque `$A1` a été mis dans `B2`.
- `A$1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, j'obtiens des 3 dans la colonne `B` mais si je recopie cette formule vers la droite, cette formule deviendra `B$1+2` dans `C2`, `C$1+2` dans `D2` etc...
- `=A1+2` : dans `B2` il y aura 3, et si je recopie cette formule vers le bas, cette formule deviendra `=A2+2` dans `B3`, `=A3+2` dans `B4` etc...si je recopie cette formule vers la droite, cette formule deviendra `=B1+1` dans `C2`, `=C1+1` dans `D2` etc...

25.3 Les fonctions du tableur

25.3.1 `SUM`

`SUM` effectue la somme des cellules mises en argument.
Par exemple : `SUM(A1:B3)` effectue la somme `A1+A2+A3+B1+B2+B3`

25.3.2 `AVERAGE`

`AVERAGE`

25.3.3 AMORT

AMORT permet de calculer l'amortissement d'un prêt. La syntaxe est : AMORT(plage,n,i,pv,pmt,[ppyr
où :
plage fait référence aux cellules dans lesquelles on pourra lire les résultats,
n est le nombre de remboursements du prêt,
i est le taux d'intérêt, pv est la somme restant due pmt est le montant de chaque remboursement,
Par exemple :

Mais il me semble que c'est plus facile d'utiliser l'Application Finance.

25.3.4 STAT1

STAT1 permet de faire des statistiques à une variable.
La syntaxe est : STAT1(plage,[mode],["configuration"]).
plage est la source des données, par exemple : A1:B3
mode vaut :
1 si chaque colonne sont indépendantes,
2 si les colonnes sont utilisées par paires (données, fréquences),
3 si les colonnes sont utilisées par paires (données, poids),
4, Mais il me semble que c'est plus facile d'utiliser l'Application :
Stats-1Var.

25.3.5 REGRS

REGRS essaie de faire correspondre les données à une fonction (linéaire par défaut).
La syntaxe est : REGRS(plage,[mode],["configuration"]).
Par exemple : Mais il me semble que c'est plus facile d'utiliser les commandes linear_regression, exponential_regression, etc....

25.3.6 PredY PredX

Par exemple :

25.3.7 HypZ1mean HypZ2mean

Par exemple : Mais il me semble que c'est plus facile d'utiliser l'Application Inférence.

25.4 Utilisation du tableur sur des exemples

25.4.1 Exercice 1

La somme des nombres entiers impairs.
Écrire dans le tableur :

- les nombres entiers de 1 à 10 dans la colonne A,
- les carrés des nombres entiers de 1 à 10 dans la colonne B,

- les nombres entiers impairs $2k-1$ pour $k = 1..10$ dans la colonne `C`,
- la somme des entiers impairs $\sum_{j=1}^{k} 2j-1$ pour $k = 1..10$ dans la colonne `D`.

Calculer $\sum_{j=1}^{k} 2j-1$ lorsque $k \in \mathbb{N}$.
Une solution

- On tape `1` dans `A1`.
  On met `=A1+1` dans `A2`, puis, on recopie vers le bas la formule contenue dans `A2` : pour cela on sélectionne `A2...A10` (lorsque `select.` du bandeau est visible et on sélectionne `A2...A10` avec le curseur) et on tape `=A1+1`.
- On met `=A1^2` dans `B1`, puis, on recopie vers le bas la formule contenue dans `B1` : pour cela on sélectionne `B1...B10` (lorsque `select.` du bandeau est visible et on sélectionne `B1...B10` avec le curseur) et on tape `=A1^2`.
- On met `=2*A1-1` dans `C1`, puis, on recopie vers le bas la formule contenue dans `C1` : pour cela on sélectionne `C1...C10` (lorsque `select.` du bandeau est visible et on sélectionne `C1...C10` avec le curseur) et on tape `=2*A1-1`.
- On tape `1` dans `D1`.
  On met `=D1+C2` dans `D2`, puis, on recopie vers le bas la formule contenue dans `D2` : pour cela on sélectionne `D2...D10` (lorsque `select.` du bandeau est visible et on sélectionne `D2...D10` avec le curseur) et on tape `=D1+C2`.

On obtient :

	A	B	C	D
1	1	1	1	1
2	2	4	3	4
3	3	9	5	9
4	4	16	7	16
5	5	25	9	25
6	6	36	11	36
7	7	49	13	49
8	8	64	15	64
9	9	81	17	81
10	10	100	19	100

On va donc montrer par récurrence que :
$\sum_{j=1}^{k} 2j-1 = k^2$ lorsque $k \in \mathbb{N}$ La formule est vraie pour $k = 1..10$ (cf ci-dessus).
Supposons que pour $k = n$ $\sum_{j=1}^{n} 2j-1 = n^2$ alors pour $k = n+1$ on a $\sum_{j=1}^{n+1} 2j-1 = \sum_{j=1}^{n} +2(n+1)-1 = n^2+2n+1 = (n+1)^2$ Donc la formule est montrée.

25.4.2 Exercice 2

Le triangle de Pascal et la suite de Fibonacci.

La suite de Fibonacci

La suite de Fibonacci est la suite u définie par :

$u_0 = 1$
$u_1 = 1$
$u_n = u_{n-1} + u_{n-2}$ pour $n > 1$
On veut trouver les 11 premiers termes de cette suite.
On utilise le tableur.
On met :
`1` dans $A1$
`1` dans $A2$
On sélectionne la colonne A de 3 à 11 (on doit avoir dans le bandeau `Select.` et `Aller↓`) et on tape : `=A1+A2`
On obtient dans la colonne A :
`1,1,2,3,5,8,13,21,34,55,89`

Le triangle de Pascal

On veut utiliser le tableur pour créer le triangle de Pascal de 0 à 10.
Pour cela on utilise les relations pour $n \in \mathbb{N}$ et $p \in \mathbb{N}$:
$C_n^0 = 1$, $C_n^n = 1$, $C_n^p = 0$ si $p > n$ et $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$ pour $0 < p \leq n$.
On vérifie et on tape :
`simplify(comb(j-1,k-1)+comb(j-1,k)-comb(j,k))`
On obtient :
`0`
Pour remplir le tableur :
On met `1` dans $A1$ puis on copie $A1$ avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller↓` dans le bandeau et on sélectionne les cases de 1 à 11, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre des 1 dans la colonne A.
On met des `0` dans $B1$ puis on copie $B1$ avec les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller→` dans le bandeau et on selectionne les colonnes de B à K, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre des 0 dans la ligne `1` à partir de la colonne B.
On met `=A1+B1` dans $B2$ puis on copie $B2$ avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller→` dans le bandeau et on sélectionne les colonnes de B à K, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre 1 dans $B2$ et des 0 dans la ligne `2` à partir de la colonne C mais cela recopie aussi les formules `=B1+C1` dans $C2$, `=C1+D1` dans $D2$ etc.....
Puis on copie aussi la formule `=A1+B1` dans la colonne B (avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller↓` dans le bandeau et on sélectionne les cases de 2 à 11, puis on appuie sur les touches `Shift Paste` : cela a pour effet de mettre, dans la colonne B : `0,1,2,3,4..10` .
Puis on copie aussi la formule `=B1+C1` dans la colonne C (avec sur les touches `Shift Copy`, on appuie sur `Selection` du bandeau (qui devient `Select.`).
On choisit `Aller↓` dans le bandeau et on sélectionne les cases de 2 à 11, puis on appuie sur les touches `Shift Paste` (il faut recopier la formule et donc choisir `0→ Formule`) : cela a pour effet de mettre, dans la colonne C :
`0,0,1,3,6,10,15,21,28,36,45`.

On recopie de même la formule située dans $D2$ (resp $E2...K2$) dans la colonne D (resp $E...K$).
On obtient :

	A	B	C	D	E	F	g	H	I	J	K
1	1	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0
3	1	2	1	0	0	0	0	0	0	0	0
4	1	3	3	1	0	0	0	0	0	0	0
5	1	4	6	4	1	0	0	0	0	0	0
6	1	5	10	10	5	1	0	0	0	0	0
7	1	6	15	20	15	6	1	0	0	0	0
8	1	7	21	35	35	21	7	1	0	0	0
9	1	8	28	56	70	56	28	8	1	0	0
10	1	9	36	84	126	126	84	36	9	1	0
11	1	10	45	120	210	252	210	120	45	10	1

La suite de Fibonacci et le triangle de Pascal
Lorsqu'on fait la somme des diagonales montantes du triangle de Pascal on obtient la suite de Fibonacci.
Par exemple :

$$\begin{pmatrix} 1 & = \\ 1 & = \\ 2 & = \\ 3 & = \\ 5 & = \\ 8 & = \\ 13 & = \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \nearrow & \nearrow & \nearrow & \nearrow & \nearrow & \\ 1 & 1 & 0 & 0 & 0 & 0 \\ \nearrow & \nearrow & \nearrow & \nearrow & \nearrow & \\ 1 & 2 & 1 & 0 & 0 & 0 \\ \nearrow & \nearrow & \nearrow & \nearrow & & \\ 1 & 3 & 3 & 1 & 0 & 0 \\ \nearrow & \nearrow & \nearrow & & & \\ 1 & 4 & 6 & 4 & 1 & 0 \\ \nearrow & \nearrow & & & & \\ 1 & 5 & 10 & 10 & 5 & 1 \\ \nearrow & & & & & \\ 1 & 6 & 15 & 20 & 15 & 6 \end{pmatrix}$$

On tape :
```
A:=makemat((j,k)->comb(j,k),11,11)
L:=sum(A[j-k, k],k=1..j-1)$(j=2..12)
```
On obtient :
```
1,1,2,3,5,8,13,21,34,55,89
```
On tape :
```
L1:=0,sum(A[j-k-1,k-1],k=2..j-2)$(j=3..12)
```
On obtient :
```
0,0,1,1,2,3,5,8,13,21,34
```
On tape :
```
L2:=1,sum(A[j-k-1,k],k=1..j-2)$(j=3..12)
```
On obtient :
```
1,1,1,2,3,5,8,13,21,34,55
```

On tape :
`[L1]+[L2]`
On obtient :
`[1,1,2,3,5,8,13,21,34,55,89]`
Pour le montrer on utilise la relation pour $n \in \mathbb{N}$:
$C_n^0 = 1$, $C_n^n = 1$ et $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$ pour $0 < p < n$.
Soit a_n la suite égale à la somme des diagonales montantes du triangle de Pascal.
On a donc :
$a_0 = 1$, $a_1 = 1$ et pour $n > 1$:
$a_n = \sum_{p=0}^{\texttt{floor(n/2)}} \texttt{comb(n-p,p)} = 1 + \sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p,p)}$
$a_n = 1 + \sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p-1)} + \sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p)}$
On a pour tout n : $\texttt{floor(n/2)} - 1 = \texttt{floor((n-2)/2)}$.
On a si $n = 2k$, on a $\texttt{floor(n/2)} = \texttt{floor((n-1)/2)} + 1 = k$:
$1 + \sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p)} =$
$\sum_{p=0}^{\texttt{floor((n-1)/2)}} \texttt{comb(n-1-p,p)} + \texttt{comb(2k-1-k,k)} = a_{n-1}$
car $\texttt{comb(2k-1-k,k)} = 0$
$\sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p-1)} =$
$\sum_{p=0}^{\texttt{floor((n-2)/2)}} \texttt{comb(n-2-p,p)} = a_{n-2}$
car `floor(n/2)-1=floor((n-2)/2)`
On a si $n = 2k+1$ alors $\texttt{floor(n/2)} = \texttt{floor((n-1)/2)} = \texttt{k}$:
$1 + \sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p)} =$
$\sum_{p=0}^{\texttt{floor((n-1)/2)}} \texttt{comb(n-1-p,p)} = a_{n-1}$
$\sum_{p=1}^{\texttt{floor(n/2)}} \texttt{comb(n-p-1,p-1)} =$
$\sum_{p=0}^{\texttt{floor((n-2)/2)}} \texttt{comb(n-2-p,p)} = a_{n-2}$
car `floor(n/2)-1=floor((n-2)/2)`
Donc :
$a_0 = 1$, $a_1 = 1$ et pour $n > 1$ on a $a_n = a_{n-1} + a_{n-2}$
a_n est donc la suite de Fibonacci.

Chapitre 26

Les autres Applications

26.1 L'Application Fonction

On tape dans la vue symbolique de l'Application Fonction :
`F1(X)=SIN(2X)+X`
`F2(X)=∂(F1(A),A=X)`
On met `∂(F1(A),A=X)` en surbrillance et on appuie sur `EVAL` du bandeau.
On obtient si on est en `Radians` :
`F2(X)=COS(2*X)*2+1`
Dans `HOME`, on tape :
`F1(1)`
On obtient si on est en `Radians` :
`SIN(2)+1`
Dans `HOME`, on tape :
`F2(1)`
On obtient si on est en `Radians` :
`COS(2)*2+1`
Dans `HOME`, on tape :
`∂(F1(X),X=1)`
On obtient si on est en `Radians` :
`0.167706326906`

26.2 L'Application Suite

26.2.1 La suite de Fibonnacci

On tape :
`U1(1)=1`
`U1(2)=1`
`U1(N)=U1(N-1)+U1(N-2)`
On obtient en appuyant su `Num` :
`la suite de Fibonnacci`
On tape la suite des restes :
`U1(1)=1`
`U1(2)=1`
`U1(N)=U1(N-1)+U1(N-2)`

On obtient en appuyant sur `Num` :
`la suite de Fibonnacci`

26.2.2 Le PGCD

Voici une mise en en œuvre de l'algorithme d'Euclide avec la `HPrime`.
Voici la description de cet algorithme :
On effectue des divisions euclidiennes successives :

$$\begin{aligned} A = &\quad B \times Q_1 + R_1 &\quad 0 \leq R_1 < B \\ B = &\quad R_1 \times Q_2 + R_2 &\quad 0 \leq R_2 < R_1 \\ R_1 = &\quad R_2 \times Q_3 + R_3 &\quad 0 \leq R_3 < R_2 \\ &..... \\ R_{n-2} = &\quad R_{n-1} \times Q_n + R_n &\quad 0 \leq R_n < R_{n-1} \end{aligned}$$

Après un nombre fini d'étapes (au plus B), il existe un entier n tel que : $R_n = 0$.
on a alors :
$PGCD(A, B) = PGCD(B, R_1) = ...$
$PGCD(R_{n-1}, R_n) = PGCD(R_{n-1}, 0) = R_{n-1}$
À l'aide de l'Aplet Suite, on écrit la suite des restes.
On tape la suite des restes R :

```
U1(1)=76
U1(2)=56
U1(N)=irem(U1(N-2),U1(N-1))
```

On obtient en appuyant sur `Num` :
`76,56,20,16,4,0 donc ....le PGCD de 76 et 56 vaut 4`

26.2.3 L'identité de Bézout

L'algorithme d'Euclide permet de trouver un couple U, V vérifiant :

$$A \times U + B \times V = PGCD(A, B)$$

Avec l'Aplet Suite, on va définir "la suites des restes" R et deux suites U et V, de façon qu'á chaque étape on ait :
$R_n = U_n \times A + V_n \times B$.

Si Q est "la suites des quotients", $Q_n = iquo(R_{n-2}, R_{n-1})$ car Q_n est le quotient entier de R_{n-2} par R_{n-1} et on a :
$R_n = irem(R_{n-2}, R_{n-1}) = R_{n-2} - Q_n \times R_{n-1}$ car $R_{n-2} = R_{n-1}Q_n + R_n$ avec $0 \leq R_n < R_{n-1}$.
U_n et V_n vont donc vérifier la même relation de récurrence. On a au début :
$R_1 = A\ R_2 = B$
$U_1 = 1\ U_2 = 0$ puisque $A = 1 \times A + 0 \times B$
$V_1 = 0\ V_2 = 1$ puisque $B = 0 \times A + 1 \times B$

On tape dans `U1` la suite des restes R pour $A = 76$ et $B = 56$:

```
U1(1)=76
U1(2)=56
```

```
U1(N)=irem(U1(N-2),U1(N-1))
```

On tape la suite des quotients Q dans `U2` :

```
U2(1)=0
U2(2)=0
U2(N)=iquo(U1(N-2),U1(N-1))
```

0n va mettre U dans `U3` et dans V `U4` pour qu'à chaque étape on ait $76*U+56*V=R$.
Puisque $R(N)=R(N-2)-R(N-1)*Q(N)$ et $Q(N) iquo(R(N-2),R(N-1))$, si on a :
$76*U(N-2)+56*V(N-2)=R(N-2)$ (1) et
$76*U(N-1)+56*V(N-1)=R(N-1)$ (2)
En faisant (1)-(2)*Q(N), on a :
$76*(U(N-2)-U(N-1)*Q(N))+56*(V(N-2)-V(N-1)*Q(N))=R(N-2)-R(N-1)*Q(N)=R(N)$
Les relations de récurence sont donc :
$U(N)=U(N-2)-U(N-1)*Q(N)=U(N-2)-U(N-1)*iquo(R(N-2),R(N-1))$
$V(N)=V(N-2)-V(N-1)*Q(N)=V(N-2)-V(N-1)*iquo(R(N-2),R(N-1))$
On tape dans `U3` la suite des U :

```
U3(1)=1
U3(2)=0
U2(N)=U3(N-2)-U3(N-1)*iquo(U1(N-2),U1(N-1))
```

On tape dans `U4` la suite des V :

```
U4(1)=0
U4(2)=1
U4(N)=U4(N-2)-U4(N-1)*iquo(U1(N-2),U1(N-1))
```

Ainsi pour chaque `N` on a `76*U3(N)+56*U4(N)=U1(N)` On obtient en appuyant sur `Num` :

```
76,0,1,0 (76=76*1+56*0)
56,0,0,1 (56=76*0+56*1)
20,1,1,-1 (20=76*1+56*-1)
16,2,-2,3 (16=76*-2+56*3)
4,1,3,-4 (4=76*3+56*-4)
0,4,-14,19 (0=76*-14+56*19)
donc 76*3+56*-4=4=PGCD de 76 et 56
```

26.3 L'Aplet Paramétrique

On tape :

```
X1(T)=-COS(2T)
Y1(T)=SIN(3T)
```

On obtient :

```
le dessin d'une courbe ressemblant à un α
```

26.4 L'Aplet Polaire

On tape :
`R1(θ)=COS(3θ)`
On obtient :
`le dessin d'un trèfle`

26.5 L'Aplet Résoudre

On tape dans la vue symbolique de l'Aplet Résoudre :
`COS(X)-X`
en appuyant su `Num` :
On tape :
`1` Pour démarrer l'itération et on presse sur `SOLVE` du bandeau et on obtient :
`0.739085133215`
On tape maintenant dans `HOME` :
`X`
On obtient :
`0.739085133215`
la valeur obtenue a été stockée dans `X`.

26.6 L'Aplet Finance

On remplit les différents champs sauf un pour lequel on appuie sur `SOLVE` du bandeau.
Par exemple, on veut connaitre les mensualités d'un prêt à 4.6 % de 100 000 euros sur 10 ans, on tape :

```
NbPmt 120      IPYR 4.6
PV 100000      PPYR 12
PMT            CPYR 12
FV 0.00        Fin ▽
Taille du groupe 12
```

On met `PMT` en surbrillance et on appuie sur `SOLVE` du bandeau.
On obtient le montant du remboursement mensuel :
`-1041.21`
Les remboursements de ce prêt sont donc de 1041.21 euros par mois.
On appuie sur `AMORT` du bandeau.
On obtient le tableau d'amortissement par groupement de 12 mois :
`Princ` donne le montant des sommes remboursées
`Inter` donne le montant des interêts versés
`Balan` donne le montant des sommes dues
Par exemple au bout de la 2ième année on a :
`Princ=-16505.07` : on a donc remboursé 16505.07 euros
`Inter=-8484.0` : on a payé 8484 euros d'interêts
`Balan=83494.9` : il reste à rembourser 83494.93 euros (100000-16505.07)
En appuyant sur la touche `PLOT` on a le graphe de l'amortissement.
On peut lire :

1-12 Principal 8063.12 et Interêt 4431.41
et en utilisant la flèche vers la droite
13-24 Principal 8441.95 et Interêt 4052.59
On retrouve ainsi les valeurs précédentes :
8063.12+8441.95=16505.07 et 4431.41+4052.59=8484.0.
On retourne dans l'aplet Finance en appuyant sur la touche Num.
Si les remboursements sont trimestriels, on change :
NbPmt 40 et PPYR 4 et on laisse CPYR 12.
On met PMT en surbrillance et on appuie sur SOLVE du bandeau.
On obtient le montant du remboursement trimestriel :
-3133.03
Les remboursements de ce prêt sont donc de 3133.03 euros par trimestre.

26.7 L'Aplet Système Linéaire

Cette Aplet permet de résoudre des systèmes linéaires de 2 équations à 2 inconnues X, Y ou de 3 équations à 3 inconnues X, Y, Z.
Remarque
Pour résoudre un système linéaire de 2 équations à 2 inconnues X, Y, il faut cocher 2×2 du bandeau.
Le canevas du système s'affiche et il ne reste qu'a taper les coefficients.
La réponse s'inscrit en bas de l'écran. On tape :

```
1*X+1*Y+1*Z=3
2*X+(-2)*Y+1*Z=1
3*X+1*Y+1*Z=5
```

On obtient :

```
X:1,Y:1,Z:1
```

On tape :

```
1*X+1*Y+1*Z=3
2*X+2*Y+2*Z=1
3*X+1*Y+1*Z=5
```

On obtient :

```
No Solutions
```

26.8 Aplet Triangle Solver

Etant donné un triangle définit par 3 données : longueur de côté et/ou valeur d'angles (cf les cas d'égaliés des triangles).
L'aplet Triangle Solver permet de calculer la longueur des autres côtés et/ou la valeur des autres angles de ce triangle.
Soit le triangle de côtés A, B, C tel que $A = 4\ B = 3\ C = 5$.
On veut avoir la valeur des angles de ce triangle.
On tape :

```
A=4, B=3, C=5
```

On appuie sur SOLVE du bandeau. On obtient si on est ebn degrés :
α =5.31301E1 (c'est l'angle opposé au côté A)
β =3.68699E1 (c'est l'angle opposé au côté B)

δ =90 (c'est l'angle opposé au côté C)
Dans Home si on tape 5.31301E1 enter, puis on met 5.31301E1 en surbrillance et on appuie sur la touche SHIFT a<->b/c.
On obtient :
53°7'48.36"
Si on appuie à nouveau sur la touche SHIFT a<->b/c.
On obtient :
53.1301

26.9 Statistiques 1-Var

L'aplet Statistiques 1-Var permet de faire des statistiques à une variable avec ou sans effectif.
Exemple On mesure les tailles en cm de 10 personnes.
On obtient :
150,165,170,165,160,170,160,175,165,180.

1. Calculer la médiane, la moyenne et l'écart-type de cette série.

   On ouvre l'aplet Statistiques 1-Var et on utilise une seule colonne.
   On appuie sur START du bandeau et on tape dans D1 :
   150 ENTER 165 ENTER....
   On peut si on le désire trier les données avec TRIER du bandeau.
   On appuie sur Symb, on coche H1 :D1 et on vérifie que l'on a Plot1 : Histogramme
   On appuie sur la touche Num et sur STATS du bandeau.
   On obtient :
   Nb Item 10
   X min 150
   X Q1 160
   X med 165
   X Q3 170
   X max 180
   $\sum$X 1660
   $\sum$X^2 276200
   MoyX 166
   SX 8.432740
   σX 8
   SE X 2.6666667

2. Construire l'histogramme.
   On régle tout d'abord le Plot Setup (SHIFT Plot) : Largeur 5
   HRNG 150 180
   XRNG 150 180
   YRNG -1 10
   XTICK 1 YTICK 1
   Puis on appuie sur Plot.

On obtient :
`l'histogramme`

3. Dessiner la boite à moustache.
On appuie sur `Symb` et on met `Plot1 : Histogramme` en surbrillance.
Avec `CHOOS` du bandeau on choisit `Boite-à-moustache`, puis on appuie sur `Plot`.
On obtient :
`Le dessin de la boite à moustache`

4. Le même exercice en utilisant une colonne pour les effectifs.
On mesure les tailles en cm de 10 personnes.
On obtient :
```
150:1
160:2
165:3
170:2
175:1
180:1
```

On appuie sur `Num` et on tape dans `D2` :
`150 ENTER 160 ENTER 165 ENTER 170 ENTER 175 ENTER 180 ENTER`
et dans `D3` :
`1 ENTER 2 ENTER 3 ENTER 2 ENTER 1 ENTER 1 ENTER`
On appuie sur `Symb` et on décoche `H1 :D1`et on met :
`H2 :D2` et on tape `D3` lorsque `Freq` est en surbrillance. `H2` est coché ainsi que `Plot2 : Histogramme`.
On obtient la même chose que précédemment.
```
Nb Item 10
X min 150
X Q1 160
X med 167.5
X Q3 175
```

26.10 Statistiques 2-Var

L'aplet `Statistiques 2-Var` permet de faire des statistiques à deux variables.

Exemple
On mesure les tailles en cm et le poids en kg de 10 personnes.
On obtient :
```
150:41
165:53
170:70
165:63
160:52
170:68
160:62
```

175:72
165:58
180:75

1. Calculer la taille moyenne et le poids moyen de ces individus.
   Calculer le coefficient de corrélation.
   On ouvre l'aplet Statistiques 2-Var et on utilise deux colonnes.
   On appuie sur START du bandeau.
   On tape dans C1 :
   150 ENTER 165 ENTER....
   On tape dans C2 :
   41 ENTER 53 ENTER....
   On peut si on le désire trier les données avec TRIER du bandeau.
   On appuie sur Symb, on met S1 :C1 C2, on coche S1 et on vérifie que l'on a :
   Type1 :Lineaire
   Fit1 : m*X+b
   On appuie sur la touche Num et sur STATS du bandeau.
   On obtient :
   Nb Item 10
   Corr 0.91798
   CoefDet 0.84269
   SCov 81.7778
   PCov 73.6
   X max 180
   $\sum$XY 102660
   En appuyant sur X du bandeau on a :
   MoyX 166
   $\sum$X 1660
   $\sum$X^2 276200
   SX 8.432740
   σX 8
   SE X 2.6666667
   En appuyant sur Y du bandeau on a :
   MoyY 61.4
   $\sum$Y 614
   $\sum$Y^2 38704
   SY 10.5641
   σY 10.0220
   SE Y 3.340659

2. Dessiner le nuage de points
   On régle tout d'abord le Plot Setup (SHIFT Plot):XRNG 145 185
   YRNG 39 77
   XTICK 5 YTICK 2
   Puis on appuie sur Plot.
   On obtient :
   le nuage de points et le tracé de la droite de régression

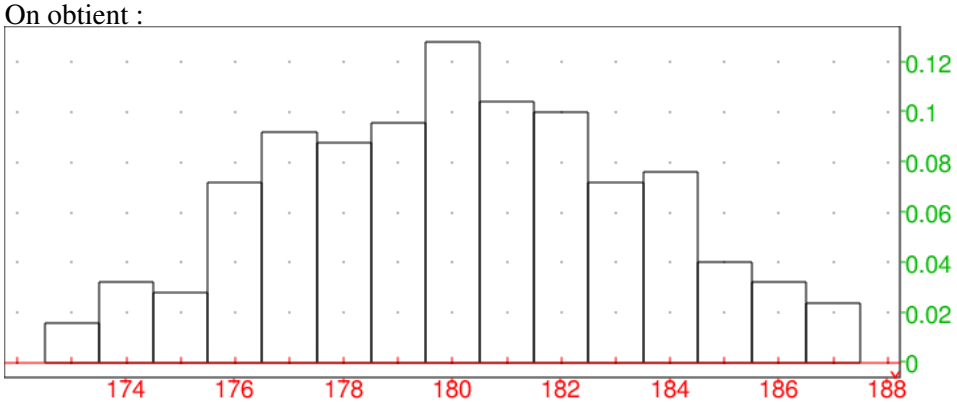
3. Déterminer une droite de régression linéaire par la méthode des moindres carrés.
   On obtient l'équation de la droite de régression linéaire en appuyant sur `Symb`.
   On obtient :
   `Fit1:1.15*X+-129.5`

26.10.1 Exercices

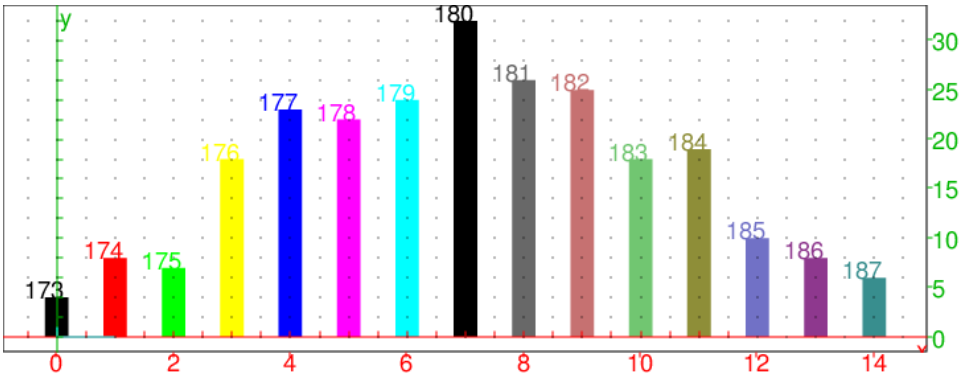
– Le but de l'activité est l'étude de la taille (en cm) portant sur 250 individus jouant au basket.

Taille	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187
Effectif	4	8	7	18	23	22	24	32	26	25	18	19	10	8	6

L'activité commence par un calcul des paramètres statistiques puis se poursuit avec des représentations graphiques : diagrammes en bâtons, diagramme en boîte et polygone des fréquences cumulées croissantes.
On tape :
```
Ts:=(173+j)$(j=0..14)
```
On obtient :
```
173,174,175,176,177,178,179,180,181,182,183,184,185,186,187
```
On tape :
```
Ef:=(4 ,8 ,7, 18, 23, 22, 24, 32, 26, 25, 18, 19, 10,
8, 6)
sum(Ef)
```
On obtient :
```
250
```
On tape :
```
histogram(tran([[Ts],[Ef]]))
```
On obtient :



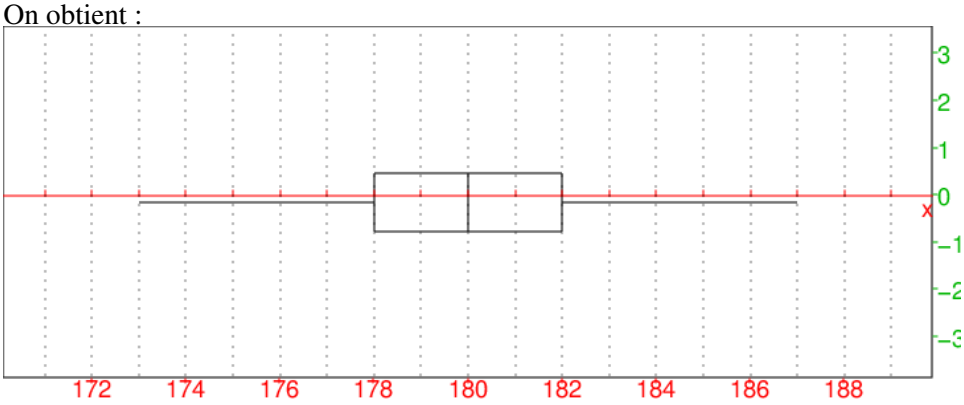
On tape :
```
bar_plo([[T],[Ef]])
```
On obtient :



On tape :

```
boxwhisker([T],[Ef])
```

On obtient :



On tape :

```
Efc:=(sum(Ef[j],j=1..k)/250.)$(k=1..size(Ef))
```

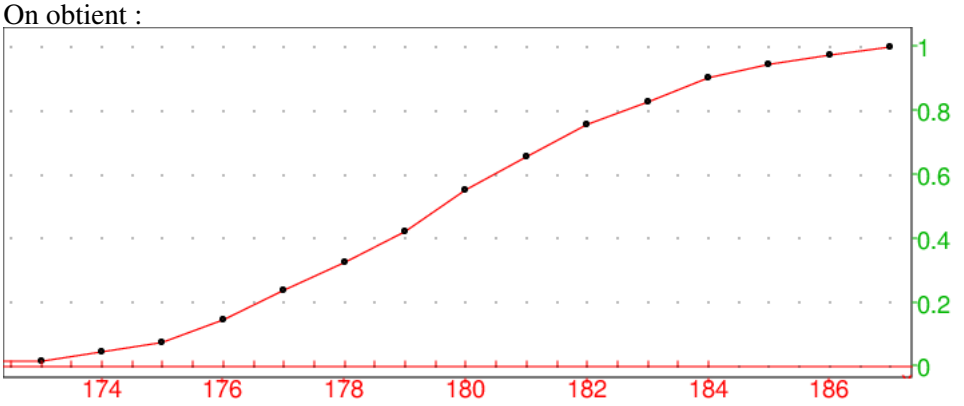
On obtient :

```
0.016,0.048,0.076,0.148,0.24,0.328,0.424,0.552,0.656,0.756,0.828,
```

On tape :

```
scatterplot([[T],[Efc]]),cumulated_frequencies(trn([[T],[Ef]]))
```

On obtient :



On tape :

```
evalf(mean([T],[Ef]))
```

On obtient :

```
180.104
```

On tape :

```
evalf(sttdev([T],[Ef]))
```

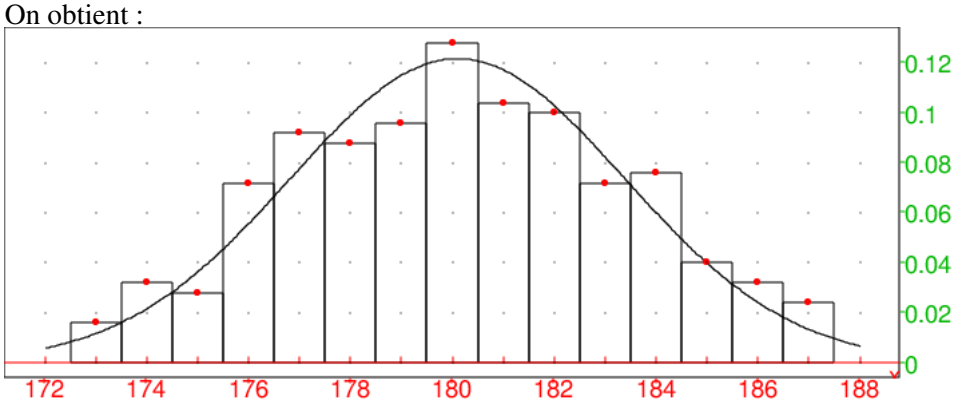
On obtient :

```
3.27859482096
```

On tape :

```
histogram(trn([[T],[Ef]])),
plotfunc(normald(180.104,3.27859482096,x),x=172..188),
scatterplot(trn([[T],[Ef]/250.]))
```

On obtient :



– Une urne contient 12 boules rouges et 3 boules vertes. On se propose de simuler le tirage d'une boule de l'urne puis d'observer la fluctuation d'échantillonnage sur des échantillons de taille 225. D'après le contenu de l'urne, la probabilité de tirer une boule verte est de $\frac{1}{5} = 0.2$.
Notre simulation est-elle convenable ? On tape la fonction `randmultinom` (cf 13.4.6) :

```
(n,p,c)->BEGIN
  local k,j,l,r,x,y;
  k:=size(p);
  if size(c)!=k then return "erreur"; end;
  x:=cumSum(p);
  if x[k]!=1 then return "erreur"; end;
    y:=MAKELIST([c[j],0],j,1,k);
   for j from 1 to n do
    r:=rand(0,1);
    l:=1;
    while r>x[l] do
      l:=l+1;
    end;
  y[l,2]:=y[l,2]+1
  end;
 return y;
END;
```

Attention Mettre `MAKELIST([c[j],0],j,1,k)` avec `MAKELIST` en majuscule ou
mettre `makelist(j->[c[j],0],1,k` ou
mettre `seq([c[j],0],j,1,k)`.
On tape :

```
L:=[randmultinom(225,[4/5,1/5],["R","V"]);
```

On obtient :

```
[["R",179],["V",46]]
```

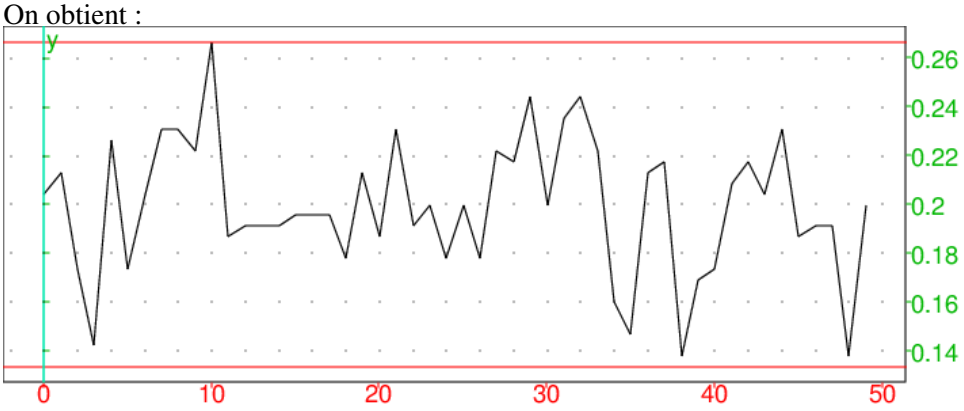
On obtient donc 46 fois la boule verte.
On analyse tout d'abord 50 échantillons de taille 225 pour voir la fluctuation.
On note `N` le nombre de fois que l'on fait une simulation (une simulation c'est 225 tirages).
`n` le nombre de fois que l'on a obtenu une boule verte,
`p` le pourcentage de boules vertes obtenues par cette simulation,
`Lp` la séquence des pourcentages obtenues.
On tape :

```
test0(N):={
  local L,p,n,k,Lp;
  Lp:=NULL;
pour k de 1 jusque N faire
  L:=randmultinom(225,[4/5,1/5],["R","V"]);
  n:=L[2,2])
  p:=n/225.;
  Lp:=Lp,p;
fpour;
 retourne Lp;
  }:;
```

Puis :

```
plotlist(test0(50)),line(y=2/15),line(y=4/15)
```

On obtient :



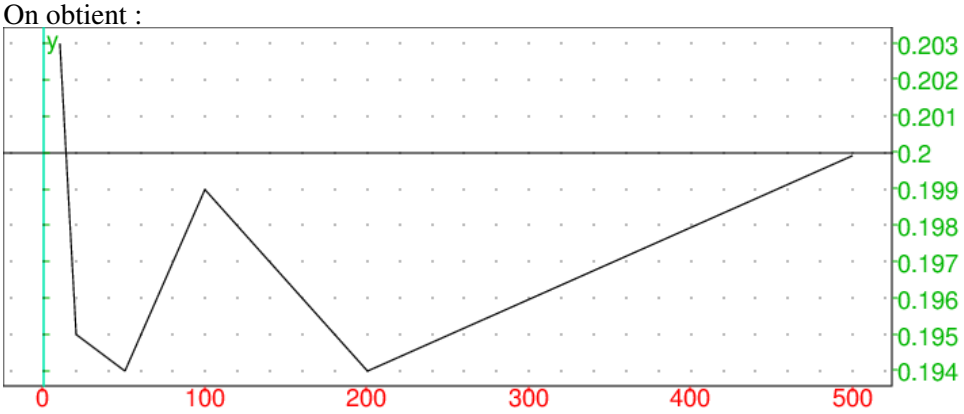
On analyse successivement t échantillons de taille $n = 225$ pour, $t \in 10, 20, 50, 100, 200, 500$. Pour notre problème, l'intervalle de fluctuation au seuil de 95% est : $p - \frac{1}{\sqrt{n}}p + \frac{1}{\sqrt{n}}$ avec $p = \frac{1}{5}$ et $n = 225$ c'est à dire $\frac{2}{15}, \frac{4}{15}$
Pour savoir si la simulation est corecte on fait un programme pour savoir si on a bien dans 95% des cas p dans l'intervalle $\frac{2}{15}, \frac{4}{15}$
On note `N` le nombre de fois que l'on fait une simulation (une simulation c'est 225 tirages).
Pour la `k` ième simulation, (`k=1..N`) on note :
`L` la liste des 225 tirages obtenus,
`n` le nombre de fois que l'on a obtenu une boule verte,
`p` le pourcentage de boules vertes obtenues par cette simulation,
l `s` le nombre de tirages tels que `2/15<p<4/15` lorsqu'on a fait `k` simulations,

`sn` le nombre de fois que l'on a obtenu une boule verte lorsqu'on a fait `k` simulations.
`pcn` le pourcentage de boules vertes obtenues par ces `N*225` tirages est donc `sn/(225*N)` Le nombre de fois où on a `2/15<p<4/15` est `s`. En pourcentage cela fait donc `pc=s/N`.
On vérifie alors si `pc>0.95`

```
test0(N):={
  local s,L,p,n,pc,sn,pcn,k,Le;
  s:=0;sn:=0;
Le:=NULL;
pour k de 1 jusque N faire
  L:=[randmultinom(225,[4/5,1/5],["R","V"])];
  n:=L[2,2]
  p:=n/225;
  Le:=Le,p;
fpour;
 retourne Le;
  }:;
test(N):={
  local s,L,p,n,pc,sn,pcn,k,Le;
  s:=0;sn:=0;
Le:=NULL;
pour k de 1 jusque N faire
  L:=[randmultinom(225,[4/5,1/5],["R","V"])];
  n:=L[2,2]
  p:=n/225;
  Le:=Le,p;
  si p>2/15 and p<4/15 alors s:=s+1; fsi;
  sn:=sn+n;
fpour;
  pc:=evalf(s/N);
  pcn:=evalf(sn/N/225);
  si pc>0.95 alors retourne pcn,pc,"correcte"; sinon retourne pcn,pc,"pas
  }:;
```

On tape :
`test(10)`
On obtient :
`0.203111111111,1.0,"correcte"`
On tape :
`test(20)`
On obtient :
`0.194888888889,0.95,"pas correcte"`
On tape :
`test(50)`
On obtient :
`0.194311111111,0.98,"correcte"`
On tape :
`test(100)`

On obtient :
`0.198888888889,0.97,"correcte"`
On tape :
`test(200)`
On obtient :
`0.193777777778,0.99,"correcte"`
`test(500)`
On obtient :
`0.19984,0.984,"correcte"`
On tape :
`plotlist([10,20,50,100,200,500],[0.203,0.195,0.194,0.199,0.1940.1 ), line(y=0.2)`
On obtient :



26.11 Aplet Inférence

L'aplet `Inférence` permet de faire des statistiques inférencielles.

Résultats à retenir

Si μ est la moyenne d'une population et m_α la moyenne des moyennes des valeurs d'un échantillon de taille n extrait de cette population, on a : $m_\alpha = \mu$,

Si σ l'écart-type d'une population et σ_α l'écart-type de la moyenne des moyennes des valeurs d'un échantillon de taille n extrait de cette population, on a : $\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$,

σ^2 la variance d'une population et s^2 la moyenne des variances des valeurs d'un échantillon de taille n extrait de cette population : $S^2 = \frac{n-1}{n}\sigma^2$,
Donc on a $\sigma_\alpha \frac{\sigma}{\sqrt{n}} = \frac{S}{\sqrt{n-1}}$.
Lorsqu'on a un échantillon de taille n de moyenne $\overline{x}$ et d'écart-type s on choisit faute de mieux :
$S = s$ et $m_\alpha = \overline{x}$ et donc la moyenne des valeurs des échantillons de taille n a pour moyenne $\overline{x}$ et pour écart-type $\frac{s}{\sqrt{n-1}}$.

En Résumé

– La moyenne et la variance d'une somme sont respectivement la somme des moyennes et la somme des variances.

- La moyenne et la variance d'une différence sont respectivement la différence des moyennes et la somme des variances.
- La moyenne de l'ensemble des moyennes des valeurs d'un échantillon de taille n extrait d'une population de moyenne m est m.
  La variance de l'ensemble des moyennes des valeurs d'un échantillon de taille n extrait d'une population de variance σ^2 est $\frac{\sigma^2}{n}$.
  L'écart-type de l'ensemble des moyennes des valeurs d'un échantillons de taille n extrait d'une population d'écart-type σ est $\frac{\sigma}{\sqrt{n}}$.
- La moyenne s des variances de l'ensemble des variances des valeurs d'un échantillon de taille n extrait d'une population de variance σ^2 est $\frac{n-1}{n}\sigma^2$.
- L'écart-type de l'ensemble des moyennes des valeurs d'un échantillon de taille n s'obtient en divisant la moyenne S des variances de l'ensemble des variances des valeurs d'un échantillon de taille n par $\sqrt{n-1}$:
  l'ensemble des moyennes des valeurs d'un échantillon de taille n a donc pour moyenne m et pour écart-type $\frac{S}{\sqrt{n-1}}$.

26.11.1 Fréquence d'un caractère et jugements sur échantillons

Comparaison d'une fréquence expérimentale et d'une fréquence théorique

Dans une population, la fréquence d'un caractère a comme fréquence p.
Soit un échantillon d'effectif n pour lequel la fréquence de ce caractère est f. La question est de savoir si cet échantillon a été extrait de cette population.
Pour que un échantillon d'effectif n, soit X la variable aléatoire égale au nombre de membres présentant le caractère considéré de probabilité p. X suit une loi binomiamle de moyenne np et d'écart-type $\sqrt{np(1-p)}$. Si n est grand ($n > 50$) la loi binomiamle peut être approchée par la loi normale.
La distribution des fréquences du caractère des échantillons de taille n a pour moyenne p et pour écart-type $\sqrt{\frac{p(1-p)}{n}}$ **Exemple**
On compte le nombre de fois qu'un joueur retourne un as quand il donne les cartes d'un jeu de 32 cartes : pour 800 donnes, ce joueur a retourné 180 fois un as. Ce joueur est-il un tricheur ?
On a $n = 800$, et la probabilité de retourner un as est :
$p = \frac{4}{32} = \frac{1}{8}$.
Donc le nombre X de retournements d'un as suit une loi binomiale de moyenne $n * p = 100$ et d'écart-type $\sqrt{np(1-p)} = 10\sqrt{\frac{7}{8}} \simeq 9.35414346693$.
Cherchons la probabilité pour que $70 < X < 130$, on tape dans `Home` :

```
normald_cdf(100,9.354,130)-normald_cdf(100,9.354,70)
```

On obtient :

```
0.998659588108
```

On peut donc affirmer que ce joueur est un tricheur : la probabilité de ce tromper étant inférieure à $\frac{2}{1000}$.
Avec la calculatrice :
On appuie sur `Symb` et on choisit :

```
Test d'Hypo
Type Z-Test: 1 π
Hypoth Alt π ≠ π0
```

On appuie sur `Num` et on tape :

x=180
n=800
μ_0=0.125
σ=50.25
α=0.02
On appuie sur CALC du bandeau :
On obtient :
Résultat 0 (Rejet de l'hypothèse à α=0.02)
Test Z 8.5523597412
Test $\widehat{p}$ 0.225
Prob. 1.2059722286E-17
Crit Z ±2.32634787404
Minimum critique 0.0906543...
Maximum critique 0.15934...
On tape dans Home :
(180-100)/9.35414346693
On obtient :
8.5523597412
On tape dans Home :
normald_icdf(100,9.35414346693,0.99)
On obtient :
121.760991768
On tape dans Home :
(121.760991768-100)/9.35414346693
On obtient :
2.32634787407
On tape dans Home pour $\alpha = 0.002$:
normald_icdf(100,9.354,0.001),
normald_icdf(100,9.354,0.999)
On obtient :
71.0939670081,128.906032992

Intervalle de confiance d'une proportion ou fréquence

Comment évaluée la fréquence p d'un caractère dans une population au vue d'un échantillon de taille n.
Exemple
On jette un dé 3000 fois et on a obtenu 490 fois le six. Peut-on dire que le dé est régulier ? On a :
$n = 3000$, $f = 49/300 \simeq 0.163333333333$, $1-f = 251/300 \simeq 0.836666666667$ donc $m = nf = 490$ et $s = \sqrt{nf(1-f)} \simeq 20.2476336066$ Au seuil de $\alpha = 0.002$, on tape dans Home :
normald_cdf(3000*49./300,sqrt(3000*49./300*251/300),500)-normald_cdf(
On obtient :
0.378612513321
Donc la probabilité que p soit dans l'intervalle $[480, 500]$ est 0.378612513321. Il est donc tout à fait raisonnable d'admettre que $p = \frac{1}{6}$ c'est à dire que le dé est régulier car sinon la probabilité de se tromper serait de $1 - 0.378612513321 \simeq$

0.621387486679 **Avec la calculatrice :**
On appuie sur `Symb` et on choisit :
`Inter de Confiance`
`Type Z-Test: 1 π`
On appuie sur `Num` et on tape :
`x=490`
`n=3000`
μ_0`=0.166666666667`
α`=0.05`
On appuie sur `CALC` du bandeau :
On obtient :
`C 0.95`
`Crit Z ± 1.95996398454`
`Minimum critique 0.150105122453`
`Maximum critique 0.176561544214...`
Avec :
`C=0.38`
On appuie sur `CALC` du bandeau :
On obtient :
`C 0.38`
`Crit Z ± 0.495850347347`
`Minimum critique 0.159986734614`
`Maximum critique 0.166679932052`

Comparaison des fréquences de 2 échantillons

On a 2 échantillons d'effectifs n_1 et n_2 pour lesquels les fréquences d'un caractère sont f_1 et f_2. Si f_1 et f_2 sont différents comment savoir si les 2 échantillons proviennent d'une même population ou pas ?
Si on fait l'hypothèse que les 2 échantillons proviennent d'une même population on peut estimer la fréquence p de la population entière en reunissant les 2 échantillons avec $p = \frac{n_1 f_1 + n_2 f_2}{n_1 + n_2}$.
L'écart-type de la distribution des fréquences du caractère de l'échantillon 1 est $\sqrt{\frac{f_1(1-f_1)}{n_1}}$ et celui de l'échantillon 2 est $\sqrt{\frac{f_2(1-f_2)}{n_2}}$.
On considère la distibution des valeurs de $f_1 - f_2$, si les 2 échantillons proviennent d'une même population :
la moyenne de $f_1 - f_2$ est 0 et
l'écart-type s de $f_1 - f_2$ est $\sqrt{\frac{p(1-p)}{n_1} + \frac{p(1-p)}{n_2}}$
Exemple
Dans la classe A de 40 élèves il y a eu 23 reçus et
dans la classe B de 40 élèves il y a eu 17 reçus. Les élèves de la classe A et de la classe B appartiennent-ils à la même population ?
Si ils appartiennent à la même population, la distribution des fréquences $f_1 - f_2$ suit une loi normale :
de moyenne $p = \frac{40}{80} = 0.5$ et
d'écart-type $s = \sqrt{\frac{2*0.5*0.5}{40}} = 0.111803398875$
On sait que $f_1 = \frac{23}{40} \simeq 0.575$ et $f_2 = \frac{17}{40} \simeq 0.425$ donc $f_1 - f_2 = 0.15$ et

$\frac{f_1-f_2}{s} \simeq 1.3416407865$
On tape :
`normal_cdf(0,0.111803398875,0.15)-normal_cdf(0,0.111803398875,-0.15)`
ou on tape :
`normal_cdf(0,1,1.3416407865)-normal_cdf(0,1,-1.3416407865)`
On obtient :
`0.820287505121`
On tape :
`1-0.820287505121`
On obtient :
`0.179712494879`
Cela veut dire que l'on peut dire que les 2 classes proviennent de la même population car si on affirmait le contraire on aurait une probabilité de 17.97 % de se tromper.
On tape si on veut un résultat au seuil de 5% :
`normal_icdf(0,0.111803398875,0.025),normal_icdf(0,0.111803398875,0.975`
On obtient :
`-0.219130635144,0.219130635144`

On tape si on veut un résultat au seuil de 5% :
`normal_icdf(0,1,0.025),normal_icdf(0,1,0.975)`
On obtient :
`-1.95996398454,1.95996398454`
Comme 0.15 se trouve dans l'intervalle [-0.219130635144,0.219130635144], on peut dire qu'au seuil de 5% les deux classes proviennent de la même population.
Avec l'Aplet Inférence :
On appuie sur `Symb` et on choisit :
`Test d'Hypo`
`Type Z-Test:` 1 $\pi_1 - \pi_2$
`Hypoth Alt` $\pi_1 \neq \pi_2$
On appuie sur `Num` et on tape :
x_1=23
x_2=17
n_1=40
n_2=40
α=0.05
On appuie sur `CALC` du bandeau :
On obtient :
`Résultat 1` (Non Rejet de H_0 à α=0.05)
`Test Z 1.3416407865`
`Test` $\Delta\widehat{p}$ `0.15`
`Prob. 0.179712494879`
`Crit Z` $\pm$`1.95996398454`
`Minimum critique -0.0666513904072`
`Maximum critique 0.366651390407`
On tape dans `Home` :
`normal_icdf(0.15,0.111803398875,0.025),normal_icdf(0.15,0.11180339887`
On obtient :

```
-0.0691306351442,0.369130635144
```

L'échantillon 1 a comme moyenne :
$f_1 = \frac{23}{40} = 0.575$
et comme variance :
$\frac{f_1(1-f_1)}{n_1} = \frac{0.575*0.425}{40}$
L'échantillon 2 a comme moyenne :
$f_1 = \frac{17}{40} = 0.425$
et comme variance :
$\frac{f_2(1-f_2)}{n_2} = \frac{0.425*0.575}{40}$
On considère la distribution des fréquences $f_1 - f_2$: on peut dire qu'elle a comme moyenne la différence des moyennes soit :
$0.575 - 0.425 = 0.15$. et comme variance la somme de ces 2 variances i.e :
$\frac{2*0.425*0.575}{40}$
On tape dans `Home` pour avoir l'écart-type de la distribution des fréquences $f_1 - f_2$:

```
sqrt(2*0.575*0.425/40.)
```

On obtient :

```
0.1105384548472
normal_icdf(0.15,0.1105384548472,0.025),normal_icdf(0.15,0.1105384548472,0.97
```

On obtient le minimum critique et le maximum critique :

```
-0.0666513904072,0.366651390407
```

26.11.2 Échantillons extraits d'une distribution normale

Comparaison d'une moyenne expérimentale et d'une moyenne théorique

Si la serie statistique suit une distribution normale de moyenne m et d'écart-type σ, la distribution des moyennes m_α des échantillons d'effectif n suit une loi normale de moyenne m et d'écart-type $\sigma_\alpha = \frac{\sigma}{\sqrt{n}}$.
Si S est la moyenne des écarts-types des échantillons d'effectif n alors on a $\frac{S}{sqrtn} = \frac{\sigma}{sqrtn-1}$ et donc $\sigma_\alpha = \frac{S}{sqrtn-1}$.
Exemple
On pèse 100 pains de 400 gr pris au hasard dans une boulangerie et on obtient une moyenne m_1 de 390 gr avec un écart-type s_1 de 50 gr.
Le boulanger respecte-t-il le poids de ses pains de 400 gr pour le seuil de confiance de 95 % ?
Estimer la moyenne de la population à partir de $\overline{x}$ et d'écart-type s de cet échantillon.

- Il s'agit de comparer une moyenne expérimentale $m_\alpha = \overline{x}$ à une moyenne théorique μ.
  L'hypothèse (H_0) est que la différence entre ces 2 moyennes n'est pas significative ie (H_0) : $\overline{x} = \mu$. L'hypothèse alyternative (H_1) est donc (H_1) : $\overline{x} \neq \mu$.
  L'écart-type s de l'échantillon de 100 éléments est 50, on estime donc l'écart-type de la population à :
  $50 * \sqrt{100/99} \simeq 50.25$

On sait que :

$$U = \frac{\overline{x} - \mu}{\frac{\sigma}{\sqrt{(n)}}}$$

suit sensiblement une loi centrée réduite car $n = 100 > 30$.
On pose donc :

$$U = \frac{390 - 400}{\frac{50}{\sqrt{(99)}}} = -1.98997487421$$

Etant donné α, on cherche u_α tel que :
Prob(-u_α<U<u_α)=$1 - \alpha$.
Si $\alpha = 0.05$,
0n tape : `normald_icdf(0.975)`
On obtient : `1.95996398454` $\simeq$ `1.96`
0n tape : `normald_icdf(0.025)`
On obtient : `-1.95996398454` $\simeq$ `-1.96`
Donc Prob(-1.96<U<1.96)=0.975-0.025=0.95=1-0.05 et on a $u_\alpha = 1.96$.
Comme $U \notin]-u_\alpha, u_\alpha[$, l'hypothèse (H_0) est écartée avec un risque d'erreur de moins de 5%. Donc on peut dire, avec un risque d'erreur moins de 5%, que le boulanger ne respecte pas le poids de ses pains de 400 gr.
Avec la calculatrice :
On appuie sur `Symb` et on choisit :
`Test d'Hypo`
`Type Z-Test: 1` μ
`Hypoth Alt` $\mu \neq \mu_0$
On appuie sur `Num` et on tape :
`N=100`
μ`=400`
σ`=50.25`
α`=0.05`
On appuie sur `CALC` du bandeau :
On obtient :
`Résultat 0` (Rejet de l'hypothèse à α=0.05)
`Test Z -1.99`
`Test` $\overline{x}$ `390`
`Prob Z 0.0466`
`Crit Z` $\pm$`1.9599`
`Minimum critique 390.151`
`Maximum critique 409.849`
On tape dans `Home` :
`normald_icdf(400,5.025,0.025),`
`normald_icdf(400,5.025,0.975)`
On obtient :
`390.151180978,409.848819022`

– Il s'agit de trouver un intervalle au seuil de 5% de centre $\overline{x}$ dans lequel m se trouve avec une probabilité de 95 %.
On choisit faute de mieux :
$\overline{x} = m_\alpha$

$s = s_\alpha = \sigma\frac{\sqrt{n-1}}{\sqrt{n}}$ et
$s = sigma_\alpha$ donc La moyenne des moyennes des échantillons d'effectif n suit une loi normale de moyenne $\overline{x}$ et d'écart-type $\frac{\sigma}{\sqrt{n}} = \frac{s}{\sqrt{n-1}}$ Donc $\sigma = \frac{500}{\sqrt{99}} \simeq 50.25$.

Avec la calculatrice :
On appuie sur `Symb` et on choisit :
`Inter de Confiance`
`.Type Z-int: 1` μ
On appuie sur `Num` et on tape :
$\overline{x}$ `390`
`N=100`
σ`=50.25`
`C=0.95`
On appuie sur `CALC` du bandeau :
On obtient :
`C 0.95`
`Crit.Z` $\pm$ `1.96`
`Minimum critique 380.1511...`
`Maximum critique 399.8488...`
On tape dans `Home` :
`normald_icdf(390,5.025,0.025),`
`normald_icdf(390,5.025,0.975)`
On obtient :
`380.151180978,399.848819022`

26.11.3 Échantillons extraits d'une distribution de Student

On utilise la loi de Student (valable plutôt pour les petits échantillons (n<30)) : La variable $T = \frac{|m_\alpha - m|}{s}\sqrt{n}$ suit une loi de Student de $n-1$ degés de liberté.

Comparaison d'une moyenne expérimentale à une moyenne théorique

On peut faire l'exercice précédent avec la loi de Student :
– On appuie sur `Symb` et on choisit :
`Test d'Hypo`
`Type T-Test: 1` μ
`Hypoth Alt` $\mu \neq \mu_0$
On appuie sur `Num` et on tape :
$\overline{x}$`=390`
`S=50`
`N=100`
μ_0`=400`
α`=0.05`
On appuie sur `CALC` du bandeau :
On obtient :
`Résultat 0` (Rejet de l'hypothèse à α=0.05)

Test T -2
Test $\overline{x}$ 390
Prob T 0.048239...
DF=99
Crit T ±1.9842...
Minimum critique 390.0789
Maximum critique 409.921
On tape dans Home :
student_icdf(99,0.025),student_icdf(99,0.975)
On obtient :
-1.98421695159,1.98421695159
On tape dans Home :
student_icdf(99,0.025)*50/sqrt(100)+400,
student_icdf(99,0.975)*50/sqrt(100)+400
On obtient :
390.078915242,409.921084758

– On appuie sur Symb et on choisit :
Inter de Confiance
Type T-int: 1 μ
On appuie sur Num et on tape :
$\overline{x}$: 390
S:=50
N=100
C=0.95
On appuie sur CALC du bandeau :
On obtient :
C 0.95
DF=99
Crit T ±1.9842
Min Crit $\overline{x}$ 380.0789
Maxi Crit $\overline{x}$ 399.9210
On tape dans Home :
student_icdf(99,0.025)*5+390,
student_icdf(99,0.975)*5+390
On obtient :
380.078915242,399.921084758

26.12 L'Aplet Linear Explorer

On tape :

On obtient :

26.13 L'Aplet Quadratic Explorer

On tape :

On obtient :

26.13.1 L'Aplet Trig Explorer

On tape :

On obtient :

Cinquième partie

La programmation

Chapitre 27

Généralités

27.1 La forme des programmes HOME et celle des programmes CAS

Les exemples de programmes qui suivent sont soit des programmes qui utilisent seulement les commandes de HOME, soit des programmes qui utilisent les commandes de HOME et du CAS.
Pour cela on tape :
`Shift Program` puis,
`Nouv.` du bandeau.
On obtient une fenêtre avec `Nom` et `CAS`.
On coche `CAS` pour écrire un programme CAS et on met comme `Nom` par exemple : `ABC`, puis `OK` et on tape le programme.
La forme d'un programme est :

- Pour un programme HOME ayant pour nom `ABC`
  ```
  EXPORT ABC(<nom des parametres>)
  BEGIN
  LOCAL <nom des var locales> ;
  <instructions>;
  RETURN <resultat>;
  END;
  ```
  Avec `Shift Program` on a la liste des programmes valides et parmi cette liste on trouvera `ABC`.
- Pour un programme ayant pour nom `ABC`. On remarquera que `ABC` ne figure pas dans le programme.
  ```
  (<nom des parametres>)->BEGIN
  LOCAL <nom des var locales> ;
  <instructions>;
  RETURN  <resultat>;
  END;
  ```
  Avec `Shift Program` on a la liste des programmes valides et parmi cette liste on trouvera `ABC(CAS)`.

27.2 Pour avoir un programme peu différent d'un programme déjà fait

Vous avez taper un programme et vous voulez faire un programme peu différent sans avoir tout à retaper. Par exemple vous avez tapé le programme CAS `PERP` qui renvoie l'équation de la perpendicuaire à une droite passant par un point :

```
(GA,GD)->BEGIN
LOCAL  GM,GE,GL;
GE:=element(GD);
IF est_element(GA,GD) THEN
GL:=inter(GD,circle(GA,1));
RETURN equation(perpen_bisector(GL));
END;
IF angle(GE,GA,GD)==pi/2 OR angle(GE,GA,GD)==-pi/2
THEN RETURN equation(line(GA,GE)); END;
GM:=midpoint(op(inter(GD,circle(GA,GE-GA))));
RETURN equation(line(GA,GM));
END;
```

Vous voulez avoir le programme `PIEDP` qui renvoie l'affixe du pied de cette perpendiculaire.
On tape dans CAS :
`PIEDP:=PERP`
Puis `Shift Program`
On voit que `PIEDP` est dans la liste des programmes. Il suuffit alors de l'éditer et de le modifier en :

```
(GA,GD)->BEGIN
LOCAL  GM,GE,GL;
GE:=element(GD);
IF est_element(GA,GD) THEN
GL:=inter(GD,circle(GA,1));
RETURN affix(GA));
END;
IF angle(GE,GA,GD)==pi/2 OR angle(GE,GA,GD)==-pi/2
THEN RETURN affix(GE); END;
GM:=midpoint(op(inter(GD,circle(GA,GE-GA))));
RETURN affix(GM);
END;
```

On tape :
`PIEDP(point(1),line(-2,2+2i))`
On obtient :
`2/5+6i/5`

Chapitre 28

Les instructions de programmation

28.1 Les variables

28.1.1 Le nom des variables

Dans HOME il n'existe pas de variables symboliques et le nom des variables est imposé par exemple :
A..Z pour les variables rèelles (qui valent 0 au départ),
L0..L9 pour les listes (qui valent { } au départ),
M0..M9 pour les matrices (qui valent [[0]] au départ).
Dans CAS, à part les noms des variables de HOME, le nom d'une variable est une suite de lettres ou chiffres commençant par une lettre.
Attention ! ! ! il y a des noms qui sont déjà employés pas le système.
Dans CAS, toutes les variables sont symboliques tant qu'elles ne sont pas affectées.
Les variables définies dans HOME sont valables dans le CAS et les variables affectées définies dans le CAS sont valables dans HOME.

28.1.2 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]
```

28.1.3 Les entrées : `INPUT input InputStr`

`input INPUT` permettent de saisir des expressions et `InputStr` permettent de saisir des chaines de caractères. `input, INPUT` et `InputStr` 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, INPUT` et InputStr 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` on peut entrer des nombres ou des chaînes de caractères (il faut alors mettre "...") ou des noms de variables (sans mettre "...").
Avec `InputStr` 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"`.

28.1.4 Les sorties : `print`

`print` a comme argument une séquence de chaîne de caractères ou de noms de variables.
`print` écrit 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 s'écrit et la réponse est 1
```

28.1.5 L'instruction d'affectation : => := ▶

Pour stocker un objet dans une variable on utilise dans HOME `Sto▶` du bandeau.
Dans HOME on tape par exemple :
`12 Sto▶ A`
cela a pour effet de mettre `12` dans la variable `A`.
Dans CAS ou dans un programme CAS on tape par exemple :
`12 => a`
Ou on tape
`a:=12`
cela a pour effet de mettre `12` dans la variable `a`.
Dans CAS ou dans un programme CAS on peut mettre : `a,b:=12,34` ou `a,b:=[12,34]`
cela a pour effet de mettre `12` dans la variable `a` et `34` dans la variable `b`.
Par exemple,on tape pour mettre le quotient et le reste de la division euclidienne respectivement dans `q` et`r` en utilisant la commande `iquorem` du CAS :

`q,r:=iquorem(365,12)`
cela a pour effet de mettre `30` dans la variable `q` et `5` dans la variable `r` car $365 = 30 * 12 + 5$.
Attention il faut utiliser cela avec prudence car ces 2 affectations se font simultanément par exemple :
`a:=1;b:=2; a:=a+b;a:=1;b:=a-b;` est équivalent à :
`a:=1;b:=2;c:=a;a:=a+b;b:=c-b;`
donc après `a` vaut `3` et `b` vaut -1 (et non `1`).
MAIS
`purge(a);a,b:=2,a+1` a pour effet de mettre `2` dans `a` et `3` dans `b`.

28.1.6 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`.

28.1.7 Fonction testant le type de son argument : `TYPE type`

`TYPE` est une fonction de HOME qui a un argument et qui renvoie le type de cet argument par exemple :
`TYPE(1)=TYPE(pi)=0, TYPE(i)=3, TYPE([1,2])=6, TYPE({1,2})=6`.
`type` est une fonction du CAS 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
```

28.1.8 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`

28.1.9 Faire une hypothèse sur une variable : `assume`

`assume` permet de faire des hypothèses sur une variable.
`assume` 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 directement :

```
assume(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,line(x=1),line(x+(m-2)*y-1))`
puis dans un niveau de géométrie, on tape :
`assume(a=[2.0,-5,5,0.1])`
`dr(evalf(a))`
qui renvoie `line(x=2)` lorsque `a:=2` et `line(y=(-5*x+5))` lorsque `a:=2.2` alors que
`dr(a)` renvoie `line(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)
```

28.1.10 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
```

puis

```
b
```

On tape :

```
assume(b)
```

On obtient :

```
[DOM_INT]
```

28.1.11 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.

28.1.12 Effacer le contenu d'une variable : `purge`

`purge` 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`.

28.1.13 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.

28.1.14 Accès aux réponses : `Ans ans(n)`

`Ans` (accessible avec les touches (Shift +) ou `Ans()` désigne la réponse précédente,
`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(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)`.

28.2 Les instructions conditionnelles

`IF` $< cond >$`THEN` $< inst_1 >$ `END`
Si la condition $< cond >$ est vraie on exécute les instructions $< inst_1 >$ et sinon on ne fait rien.
On tape :

```
3=>X
IF X>0 THEN X+1 END
```

ou

```
IF X>0 THEN X+1;END
```

On obtient :

```
3
```

On tape :

```
-3=>X
IF X>0 THEN X+1 END
```

ou

```
IF X>0 THEN X+1;END
```

On obtient :
```
-3
```

IFTE
Si la condition donnée en premier argument est vraie on exécute le deuxième argument et sinon on exécute le troisième argument.
On tape :
```
3=>X
IFTE(X>0,X+1,X-1)
```
On obtient :
```
4
```
On tape pour définir la valeur absolue :
```
-3=>X
IFTE(X>0,X,-X)
```
On obtient :
```
3
```
On tape :

```
EXPORT ESSAI0(X,A)
BEGIN
RETURN IFTE(X<-ABS(A),-1,IFTE(X<ABS(A),0,1));
END;
```

Puis on tape :
```
ESSAI0(-5,3)
```
On obtient :
```
-1
```
On tape :
```
ESSAI0(-2,3)
```
On obtient :
```
0
```
On tape :
```
ESSAI0(5,3)
```
On obtient :
```
1
```

IF $<cond>$THEN $<inst_1>$ ELSE $<inst_2>$ END
Si la condition $<cond>$ est vraie on exécute les instructions $<inst_1>$ et sinon on exécute les instructions $<inst_2>$.
On tape :
```
3=>X
IF X>0 THEN X+1 ELSE X-1 END
```
ou
```
IF X>0 THEN X+1; ELSE X-1; END
```
On obtient :
```
3
```
On tape :
```
-3=>X
IF X>0 THEN X+1 ELSE X-1 END
```

ou
`IF X>0 THEN X+1; ELSE X-1; END`
On obtient :
`-4`
On tape :

```
EXPORT ESSAI(X,A)
BEGIN
IF X<-ABS(A) THEN RETURN -1; END;
IF X<ABS(A) THEN RETURN 0; END;
RETURN 1;
END;
```

Puis on tape :
`ESSAI(-5,3)`
On obtient :
`-1`
On tape :
`ESSAI(-2,3)`
On obtient :
`0`
On tape :
`ESSAI(5,3)`
On obtient :
`1`

CASE ...END

CASE
IF $< cond_1 >$ THEN $< inst_1 >$ END;
IF $< cond_2 >$ THEN $< inst_2 >$ END;
IF $< cond_3 >$ THEN $< inst_3 >$ END;
DEFAULT $< inst_4 >$;
END
On utilise `CASE` pour éviter de faire des `IF` imbriqués.
$< cond_1 >$ est évaluée :

- si $< cond_1 >$ est vraie on exécute les instructions $< inst_1 >$ et on termine le `CASE` en faisant les instructions qui suivent le `END` du `CASE`.
- si $< cond_1 >$ est fausse, alors $< cond_2 >$ est évaluée, si elle est vraie on exécute les instructions $< inst_2 >$ et on termine le `CASE` en faisant les instructions qui suivent `END` du `CASE` etc... L'instruction $< inst_4 >$ se fait si les 3 conditions $< cond_1 >, < cond_2 >, < cond_3 >$ sont fausses.

```
CASE
IF X<-1 THEN -1=>R; END;
IF X<1; THEN 0=>R; END;
IF X>=1 THEN 1=>R; END;
END;
R;
```

ou bien :

```
CASE
IF X<-1 THEN -1=>R; END;
IF X<1 THEN 0=>R; END;
DEFAULT 1=>R;
END;
R;
```

IFERR

La syntaxe est :
`IFERR <inst0> THEN <inst1> ELSE <inst2> END`
Si une erreur est dédectée dans les instructions `<inst0>`, on effectue les instructions `<inst1>` et sinon on effectue les instructions `<inst2>`.
On tape (par exemple si on ne sait pas l'ordre des arguments de la commande `POS`) :

```
IFERR(A:=POS(5,[1,3,5,2,4]); THEN
A:=POS([1,3,5,2,4],5);  ELSE
A:=POS(5,[1,3,5,2,4]);
 END
```

On obtient :
`4`

CONTINUE

Lorsque `CONTINUE;` se trouve parmi les instructions d'une itération, elle a pour effet de sauter les instructions qui la suivent pour aller à la prochaine itération.
On tape pour calculer 1+2+4+5=$\sum_{j\neq 3 et j=1}^{5} j$:

```
A:=0;
FOR J FROM 1 TO 5 DO
IF J==3 THEN CONTINUE; END;
A:=A+J;
END;
```

On obtient :
`12`

28.3 Les boucles

28.3.1 Les instructions `FOR FROM TO DO END` et `FOR FROM TO STEP DO END`

On tape :
`S:=0; FOR J FROM 1 TO 5 DO S:=S+J;END`
On obtient :
`15`

car 1+2+3+4+5=15
On tape :
```
S:=0; FOR J FROM 2 TO 10 STEP 2 DO S:=S+J;END
```
On obtient :
```
30
```
car 2+4+6+8+10=30

28.3.2 Pour créer des boucles itératives : `ITERATE`

Pour faire une itération. On tape :
```
ITERATE(X^2,X,2,3)
```
cela veut dire que `X:=2; FOR J FROM 1 TO 3 DO X^2=>X;END;`
On obtient :
```
256
```
car X vaut 2 puis $2^2 = 4$ puis $4^2 = 16$ puis $16^2 = 256$

28.3.3 L'instruction `WHILE DO END`

On tape :
```
A:=1; WHILE A<=1 DO A:=A+1; END;A;
```
On obtient :
```
2
```
On tape :
```
S:=0;J:=1;WHILE J<=5 DO S:=S+J;J:=J+1; END;S
```
On obtient comme valeur de `S` :
```
15
```

28.3.4 L'instruction `REPEAT UNTIL`

On tape :
```
A:=1; REPEAT A:=A+1 UNTIL A>1;A;
```
On obtient comme valeur de `A` :
```
2
```
On tape :
```
A:=1; REPEAT A:=A+1 UNTIL A>4;A;
```
On obtient comme valeur de `A` :
```
5
```

28.3.5 L'instruction `BREAK`

On tape :
```
BREAK
```
On obtient :
```
La sortie d'une boucle
```
Par exemple pour avoir la valeur approchée de la somme $6\sum_{j=1}^{\infty} 1/j^2$, on décide de ne pas additionner les termes inférieurs à P et de ne pas faire plus de 100 additions.
On tape

```
EXPORT PI2S6(P)
```

```
BEGIN
LOCAL J,S,U;
FOR J FROM 1 TO 100 DO
U:=1/J^2;
IF U<P THEN
BREAK;
END;
S:=S+U;
END;
RETURN S;
END;
```

On tape :
`PI2S6(0.001)`
On obtient :
`1.61319070033`
On tape :
`PI2S6(0.0001)`
On obtient :
`1.63498390018`
On tape :
`PI^2/6`
On obtient :
`1.64493406685`

28.3.6 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 l'aplet 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 :

```
seulement le dernier point
```

Mais si 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
```

28.4 Les commentaires

`//`

`//` commence une ligne destinée à faire un commentaire.

28.5 Les variables

En programmation les variables ont comme noms une suite de lettres ou chiffres commencant par une lettre.
Les variables qui sont locales au programme seront déclarées en utilisant le mot clé `LOCAL`, par exemple : `LOCAL A,B,AB,x;`.
Dans ce cas les variables sont initialisées à 0.
Pour avoir une variable formelle on écrira : `x:='x'`.

28.6 Les opérateurs booléens

`<,<=,==,!=,>,>=`

<, <=, >, >= sont des opérateurs booléens infixés testant l'inégalité.
== est un opérateur booléen infixé testant l'égalité.
<> ou != ou $\neq$ est un opérateur booléen infixé testant la non égalité.

`AND and`

`AND` ou `and` est l'opérateur booléen infixé et.
On tape :

```
1 AND 0
```

On obtient :

```
0
```

On tape :

```
1 AND 1
```

On obtient :

```
1
```

On tape :

```
0 AND 0
```

On obtient :

```
0
```

`NOT`

`NOT` renvoie l'inverse logique de l'argument.
On tape :
```
NOT 1
```
On obtient :
```
0
```
On tape :
```
NOT 0
```
On obtient :
```
1
```

`OR or`

`OR` ou `or` est l'opérateur booléen infixé *ou*.
On tape :
```
1 OR 0
```
On obtient :
```
1
```
On tape :
```
1 OR 1
```
On obtient :
```
1
```
On tape :
```
0 OR 0
```
On obtient :
```
0
```

`XOR`

`XOR` est l'opérateur booléen infixé *ou* exclusif.
On tape :
```
1 XOR 0
```
On obtient :
```
1
```
On tape :
```
1 XOR 1
```
On obtient :
```
0
```
On tape :
```
0 XOR 0
```
On obtient :
```
0
```
sectionLes commandes d'invite `CHOOSE`
On tape pour choisir la valeur de `A` parmi les 3 valeurs, (1,2,3) :
```
CHOOSE(A,"TITRE:A=","UN","DEUX","TROIS")
```
On obtient :
```
L'ouverture d'une boite de dialogue comportant 3 items :
si on clique sur le 1ier (resp 2nd, 3ième) item, cela stocke
1 (resp 2, 3) dans A
```

FREEZE
On tape :
```
FREEZE
```
On obtient :
```
un gel de l'écran, on appuie sur une touche pour degeler
```

GETKEY
On tape :
```
A:=GETKEY
```
On obtient, si on a appuyer sur . :
```
48
```
On tape :
```
A:=GETKEY
```
On obtient, si on n'a pas pressé de touche :
```
-1
```

ISKEYDOWN
On tape :
```
ISKEYDOWN(48)
```
On obtient, si on n'a pas pressé la touche . :
```
0
```
On tape :
```
ISKEYDOWN(48)
```
On obtient si on a pressé la touche . :
```
1
```

INPUT
On tape :
```
INPUT(C,"TITRE:C=")
```
On obtient :
```
Un écran permettant d'entrer une valeur qui sera stockée
dans la variable C
```

MSGBOX
On tape :
```
A:=3
MSGBOX(2*A)
```
ou
```
MSGBOX(2*A,0)
```
On obtient :
```
6 et OK dans le bandeau
```
Si on appuie sur `OK` alors `MSGBOX(2*A)` ou `MSGBOX(2*A,0)` renvoie 1.
On tape :
```
A:=3
MSGBOX(2*A,1)
```
On obtient :
```
6 et CANCEL et OK dans le bandeau
```
On tape :

Si on appuie sur `CANCEL` alors `MSGBOX(2*A,1)` renvoie 0.
Si on appuie sur `OK` alors `MSGBOX(2*A,1)` renvoie 1.
On tape :
```
A:=3
MSGBOX("A= "+A)
```
On obtient :
```
"A= 3"
```

`PRINT`
On tape :
```
A:=3
PRINT(A)
```
On obtient :
```
A:3
```

`WAIT`
On tape :
```
WAIT(5)
```
On obtient :
```
Une interruption du programme pendant 5 secondes
```

`EDITMAT`
On tape :
```
EDITMAT(M)
```
On obtient :
```
L'ouverture d'un éditeur de matrice permettant d'entrer
la matrice M
```

Une utilisation de `GETKEY` et de `ISKEYDOWN`.
Le programme suivant permet de connaitre le code des touches sur lesquelles on appuie et se termine lorsqu'on appuie sur la touche `..`

```
EXPORT AA()
BEGIN
LOCAL A,L;
L:=[];
REPEAT
REPEAT
A:=GETKEY;
UNTIL A!=-1;
L:=CONCAT(L,A);
UNTIL ISKEYDOWN(48);
RETURN L;
END;
```

On tape :
`AA()` puis `ENTER 1230.`
On obtient :

```
[42,43,44,47,48]
```

On peut aussi écrire le programme suivant qui donnera le même résultat :

```
EXPORT AAA()
BEGIN
LOCAL A,L,N;
L:=[];
N:=0
WHILE N==0 DO
REPEAT
A:=GETKEY;
UNTIL A!=-1;
L:=CONCAT(L,A);
N:=ISKEYDOWN(48);
END;
RETURN L;
END;
```

On tape :
`AAA()` puis `ENTER 1230.`
On obtient :
`[42,43,44,47,48]`

Un exemple un lancer de dés : 2 joueurs `A` et `B` lancent à tour de rôle 2 dés et comptent leur scores `SA` et `SB` en éliminant du score certaines parties (par exemple les parties qui contiennt un 6). Ils décident de s'arreter de jouer au bout de 2mn.
Le programme va afficher le résultat du jet des 2 dés pour chacun des 2 joueurs. On remarquera que l'affichage du jet du joueur `B` : `MSGBOX(N+1+":B="+B,1))` a comme deuxième paramètre `1` et donc on a dans le bandeau `CANCEL` et `OK`. Si on appuie sur `CANCEL`, `MSGBOX(N+1+":B="+B,1)` renvoie `0` et si on n'appuie pas sur `CANCEL`, `MSGBOX(N+1+":B="+B,1)` renvoie `1`. Donc si on appuie sur `CANCEL` on annule la partie.
Pour arrêter de jouer il suffit d'appuyer sur `OK` (touche 5).
Le programme affiche alors le score et la liste des différents coups.

```
EXPORT DEUXDES()
BEGIN
LOCAL SA,SB,A,B,C,N,L;
SA:=0;
SB:=0;
N:=0;
L:=[];
RANDSEED
WHILE ISKEYDOWN(5)==0 DO
A:=(RANDOM 6+RANDOM 6);
MSGBOX(N+1+":A="+A,0);
B:=(RANDOM 6+RANDOM 6);
C:=MSGBOX(N+1+":B="+B,1);
```

```
IF C==-1 THEN
L[N]:=[A,B];
N:=N+1;
IF A>B THEN
SA:=SA+1;
ELSE
SB:=SB+1;
END;
END;
END;
RETURN SA,SB,L;
END;
```

28.7 Les commandes des Applications

CHECK

On tape si l'aplet courante est `Fonction` :

```
F2(X):=COS(X)+X
CHECK(2)
```

On obtient :

```
La définition de F2 dans l'aplet Fonction et la fonction
F2 est cochée
```

UNCHECK

On tape si l'aplet courante est `Fonction` :

```
UNCHECK(2)
```

On obtient :

```
La fonction F2 est décochée
```

SELECT

On tape :

```
SELECT
```

On obtient :

STARTAPLET

On tape :

```
STARTAPLET
```

On obtient :

STARTVIEW

On tape :

```
STARTVIEW(1)
```

On obtient :

noir :0 gris fonce :1 gris clair :2 blanc :3 Symbolic : 0 Plot : 1 Numeric : 2 Sym-

bolic Setup : 3 Plot Setup : 4 Numeric Setup : 5 First special view (Split Screen Plot Detail) : 6 Second special view (Split Screen Plot Table) : 7 Third special view (Autoscale) : 8 Fourth special view (Decimal) : 9 Fifth special view (Integer) : 10 Sixth special view (Trig) : 11

HomeScreen : -1 Home Modes : =-2 Memory Manager : -3 APP Library : -4 Aplet Note Editor : -5 MatrixCatalog : -6 ListsCatalog : =-8 ProgramCatalog : =-10 NoteCatalog : =-12

Chapitre 29

Pour apprendre à programmer

29.1 L'instruction conditionnelle IF

Trois magasins vendant les mêmes laines au même prix unitaire de p euros, font une promotion.
Dans le magasin1 la promotion précise :

- réduction de 10 % si vous achetez plus de 5 pelotes mais moins que 10,
- réduction de 20 % si vous achetez au moins 10 pelotes.

Dans le magasin2 la promotion précise :

- 1 gratuite pour 8 pelotes achetèes
- 2 pelotes gratuites pour 13 pelotes achetèes.

Dans le magasin3, la promotion précise :

- réduction de 10 % si vous achetez 5 pelotes,
- réduction de 20 % si vous achetez 10 pelotes,
- par exemple si vous achetez 7 pelotes à p euros l'unité, vous avez la réduction seulement sur 5 pelotes et vous payez $2 * p + 5 * 0.9 * p = 6.5 * p$ et si vous achetez 17 pelotes vous avez la réduction seulement sur 10 pelotes et sur 5 pelotes et vous payez $2 * p + 5 * 0.9 * p + 10 * 0.8 * p = 14.5 * p$.

Vous avez besoin de 9 pelotes, quel magasin choisissez-vous ?
Vous avez besoin de 15 pelotes, quel magasin choisissez-vous ?

Faire le programme `prix(n,p)` pour chaque magasin qui donne le prix que vous allez payer, le nombre de pelotes et l'économie réalisée grâce à la promotion, lors de l'achat de `n` pelotes au prix unitaire de `p`. **Les programmes** prix1

```
(n,p)->BEGIN
  LOCAL p1,p2,n1,n2,r1;
  p1:=0.8*p;
  p2:=0.9*p;
  r1:=irem(n,10);
  IF n<5 THEN return n*p,n,0;END;
  IF 5<=n and n<10 THEN  return n*p2,n,n*(p-p2);END;
  IF n>10 THEN return n*p1,n, n*(p-p1);END;
END;
```

prix2

```
(n,p)->BEGIN:={
  LOCAL  r1,q1;
  r1:= irem(n,13);
  q1:=iquo(n,13);
  IF r1>=8 THEN return n*p, n+2*q1+1, 2*q1*p+p; END;
  IF r1<8 THEN return n*p, n+2*q1, 2*q1*p; END;
END;
```

prix3

```
(n,p)->BEGIN
  LOCAL p1,p2,n1,n2,r1;
  p1:=0.8*p;
  p2:=0.9*p;
  r1:=irem(n,10);
  IF r1<5 THEN n1:=n-r1; return n1*p1+r1*p,n,(n-r1)*p-n1*p1;END;
  IF r1>=5 THEN n1:=n-r1; return n1*p1+5*p2+(r1-5)*p,n,(n-r1+5)*p-n1*p
  END;
```

29.2 Les boucles FOR et WHILE

29.2.1 Faire compter la calculatrice de 1 en 1 en utilisant un affichage

On veut que la calculatrice affiche : `0` puis `1` puis `2` etc..

Sans programme

On tape :
`n:=0;`
Puis `Enter`
On tape ensuite :
`n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait afficher le nombre suivant) :
`1`, puis `2` etc ...

Avec un programme du CAS

La boucle `FOR`
On donne comme nom de programme `comptef` et on coche `CAS`.
On tape, pour que s'affiche la suite `0,1,2..p`, le programme qui utilise une boucle `FOR` :

```
(p)->BEGIN
 LOCAL n;
 FOR n FROM 0 TO p DO
     PRINT(n);
```

```
  END;
  RETURN n;
  END;
```

On remarquera que :
n est initialisé par la valeur qui suit FROM et la valeur qui suit TO sert à faire le test d'arrêt de la boucle,
l'instruction n:=n+1, puis le test n<=p se fait automatiquement dans une boucle FOR. La boucle s'arrête au premier entier n strictement supérieur à p.
On tape dans le CAS :
comptef(-1)
On obtient :
0 car le test n<=p se fait en début de boucle.
On tape dans le CAS :
comptef(4)
On obtient :

```
0
1
2
3
4
```

puis 5 car 5 est le premier entier strictement supérieur à p=4
En résumé
FOR initialise la variable du FOR fait le test si le test est vrai il exécute le corps de la boucle (i.e. toutes les instructions jusqu'au END du FOR) puis la variable du FOR est incrémentée, puis il fait le test : si le test est vrai il exécute le corps de la boucle etc.. ; et si le test est faux il exécute les instructions qui suivent le END du FOR.
La boucle WHILE
On donne comme nom de programme comptew et on coche CAS.
On tape, pour que s'affiche la suite 0,1,2..p, le programme qui utilise une boucle WHILE :

```
(p)->BEGIN
 LOCAL n;
  n:=0;
  WHILE n <=p DO
     PRINT(n);
     n:=n+1;
 END
 RETURN n;
 END;
```

On remarquera que :
n doit être initialisé avant le commencement de la boucle,
WHILE effectue le test n<=p : si le test est vrai il exécute le corps de la boucle (i.e. toutes les instructions jusqu'au END du WHILE) mais attention il faut changer

dans le corps de la boucle la valeur d'au moins une variable du test pour que le test devienne faux à un moment afin de ne pas avoir une boucle infinie, c'est ici l'instruction `n:=n+1`. Puis `WHILE` effectue le test `n<=p` si le test estvrai il exécute le corps de la boucle etc..., et si le test est faux il exécute les instructions qui suivent le `END` du `WHILE`.
La boucle s'arrête au premier entier `n` strictement supérieur à `p`.
On tape dans le CAS :
`comptew(-1)`
On obtient :
`0` car le test `n<=p` se fait en début de boucle.
On tape dans le CAS :
`comptew(4)`
On obtient :

```
0
1
2
3
4
```

puis
`5` car 5 est le premier entier strictement supérieur à `p=4`

29.2.2 Faire compter la calculatrice de 1 en 1 en utilisant une liste ou une séquence

Plutôt qu'afficher les nombres avec la commande `PRINT` on va mettre ces nombres dans un liste ou une séquence.

Avec une liste

La liste vide est `[]` et la commande `l:=append(l,a)` met l'élément `a` à la fin de la liste `l`. **Avec une boucle** `FOR`
On donne comme nom de programme `comptelf` et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `FOR` :

```
(p)->BEGIN
 LOCAL n,l;
 l:=[];
 FOR n FROM 0 TO p DO
    l:=append(l,n);
 END;
 RETURN l;
 END;
```

Avec une boucle `WHILE`
On donne comme nom de programme `comptelw` et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `WHILE` :

```
(p)->BEGIN
 LOCAL n,l;
 l:=[];
 n:=0;
  WHILE n <=p DO
    l:=append(l,n);
    n:=n+1;
END
 RETURN l;
 END;
```

On tape :
`comptelf(4)` ou `comptelw(4)`
On obtient :
`[0,1,2,3,4]`

Avec une séquence

La séquence vide est `NULL` et la commande `l:=l,a` met l'élément `a` à la fin de la séquence `l`. **Avec une boucle** `FOR`
On donne comme nom de programme `comptesf` (ou on modifie le programme précédent) et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `FOR` :

```
(p)->BEGIN
 LOCAL n,l;
 l:=NULL;
 FOR n FROM 0 TO p DO
    l:=l,n;
 END;
 RETURN l;
 END;
```

Avec une boucle `WHILE`
On donne comme nom de programme `comptesw` (ou on modifie le programme précédent) et on coche `CAS`.
On tape, pour que s'affiche la liste `[0,1,2..p]`, le programme qui utilise une boucle `WHILE` :

```
(p)->BEGIN
 LOCAL n,l;
 l:=NULL;
 n:=0;
  WHILE n <=p DO
    l:=l,n;
    n:=n+1;
END;
 RETURN l;
 END;
```

On tape :
`comptesf(4)` ou `comptesw(4)`
On obtient :
`0,1,2,3,4`

29.3 Valeur approchée de la somme d'une série

29.3.1 Série de terme général $u_n = 1/n^2$

Sans programme

On tape dans le CAS :
`s:=0;n:=1;`
Puis `Enter`
On tape ensuite :
`s:=s+1/n^2;n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une addition de plus) :
`[1,2]`
`[5/4,3]`
`[49/36,4]`
`[205/144,5]`
`[5269/3600,6]`
`[5369/3600,7]`
`[266681/176400,8]` etc...

Avec un programme du CAS

On donne comme nom de programme `sommeu` et on coche `CAS`.
On tape, pour avoir la somme des $1 + 1/2^2 + ... + 1/1/p^2$, le programme :

```
(p)->BEGIN
 LOCAL s,n;
 s:=0;
 FOR n FROM 1 TO p DO
     s:=s+1/n^2;
     PRINT([s,n]);
 END;
 RETURN [s,n];
 END;
```

On tape dans le CAS :
`sommeu(7)`
On obtient :

```
[1,1]
[5/4,2]
[49/36,3]
[205/144,4]
```

```
[5269/3600,5]
[5369/3600,6]
[266681/176400,7]

[266681/176400,8]
```

On remarquera qu'il faudrait mettre `PRINT([s,n+1]);` pour avoir les mêmes résultats car dans le `FOR` l'incrémentation de `n` se fait quand le corps de la boucle a été exécuté. Mais le résultat final est le même car l'incrémentation de `n` a été faite et comme 8>7 le `FOR` s'arrête.

29.3.2 Série de terme général $v_n = (-1)^(n+1)/n$

Sans programme

On tape dans le CAS :
`s:=0;n:=1;`
Puis `Enter`
On tape ensuite :
`s:=s+(-1)^(n+1)/n;n:=n+1`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une opération de plus) :
`[1,2]`
`[1/2,3]`
`[5/6,4]`
`[7/12,5]`
`[47/60,6]`
`[37/60,7]`
`[319/420,8]` etc

Avec un programme du CAS

On donne comme nom de programme `sommev` et on coche `CAS`.
On tape, pour avoir la somme des $1-1/2+1/3...+(-1)^(p+1)/p$, le programme :

```
(p)->BEGIN
 LOCAL s,n;
 s:=0;
 FOR n FROM 1 TO p DO
     s:=s+(-1)^(n+1)/n;
     PRINT([s,n]);
 END
 RETURN [s,n];
 END;
```

On tape dans le CAS :
`sommeu(7)`
On obtient :

```
[1,1]
[1/2,2]
```

```
[5/6,3]
[7/12,4]
[47/60,5]
[37/60,6]
[319/420,7]

[319/420,8]
```

29.3.3 La série de terme général $w_n = 1/n$ est divergente

Pour montrer cela on montre que pour $p > 1$ on a $\sum_{n=2^p-1}^{2^p} 1/k \geq 1/2$

Sans programme

On tape dans le CAS :
`s:=1;n:=1;k:=0;`
Puis `Enter`
On tape ensuite :
`s:=s+sum(1/k,k=n+1..2*n);n:=2*n;k:=1+k;1+k/2<=s`
Puis `Enter`,`Enter` etc ...
On obtient (chaque `Enter` fait une opération de plus) :
`[3/2,2,1]`
`[25/12,4,1]`
`[761/280,5,1]`
`[2436559/720720,6,1]` etc

Avec un programme du CAS

On donne comme nom de programme `sommediv` et on coche `CAS`.
On tape, dans le CAS, pour avoir la somme des `n` :

```
sommediv(p)->BEGIN
 LOCAL s,n,k;
 s:=1;
 n:=1;
 FOR k FROM 1 TO p DO
   s:=s+sum(1/k,k=n+1..2*n);
   n:=2*n;
   1+k/2<=s;
   PRINT([s,n,k,1+k/2,1+k/2<=s]);
 END
 RETURN [s,n,1+(k-1)/2<=s];
 END;
```

On tape dans le CAS :
`sommediv(7)`
On obtient :

```
3/2,2,1,vrai]
[25/12,4,2,vrai]
[761/280,8,3,vrai]
[2436559/720720,16,4,vrai]
```

puis, `[2436559/720720,16,5,vrai]` **Exercice** Pour quelle valeur de n $\sum k = 1^n 1/k > p$? On tape, dans le CAS, pour avoir la somme des `n` :

```
sommesup(p)->BEGIN
 LOCAL s,n;
 s:=0;
 n:=0;
 WHILE s<p DO
   n:=n+1;
   s:=s+1/n;
 END;
 RETURN evalf(s),n;
 END;
```

On tape dans le CAS :
`sommesup(4)`
On obtient :
`4.02724519544,31`

29.4 Écriture décimale d'une fraction

29.4.1 Sans programme

Par exemple on veut trouver le premières décimale de $f = \frac{355}{113}$ (f est une fraction qui donne π avec 6 décimales exactes !) On tape dans le CAS :
`f:=355/113;f1:=floor(f);l:=f1;n:=numer(f-f1);d:=denom(f-f1);`
Puis `Enter`
On tape ensuite :
`ds,n:=iquorem(10*n,d); L:=L,ds;`
Puis `Enter`,`Enter` etc ... On obtient (chaque `Enter` donne une décimale de plus) :

```
[[1,47],3,1]
[[1,47],3,1,4]
[[1,47],3,1,4,1]
[[1,47],3,1,4,1,5]
[[1,47],3,1,4,1,5,9]
[[1,47],3,1,4,1,5,9,2]
```

Les décimales obtenus sont dans la liste `l` qui commence par la partie entière de la fraction `f`.
Ou bien on tape :
`f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);`
Puis `Enter`
On tape ensuite :
`ds,n:=iquorem(10*n,d):; L:=L,ds:;`

Puis Enter,Enter etc ... On obtient :
["Done","Done"],["Done","Done"] etc
Puis on tape :
l
On obtient après 30 Enter :
[3,1,4,1,5,9,2,9,2,0,3,5,3,9,8,2,3,0,0,8,8,4,9,5,5,7,5,2,2,1,2]
Comme c'est difficile de compter le nombre de Enter, on peut faire afficher par exemple 5 décimales supplémentaire chaque fois que l'on appuie sur Enter. Le premmier Enter affichera la partie entière suivie de 5 décimales. On peut aussi vouloir ou ne pas vouloir afficher ds et n.
On tape :
f:=355/113;f1:=floor(f);L:=f1;n:=numer(f-f1);d:=denom(f-f1);
Puis Enter puis,
ds,n:=iquorem(105*n,d); l:=l*10^5+ds;
ou bien si on ne veut pas les valeurs de ds et n :
ds,n:=iquorem(10^5*n,d):; l:=l*10^5+ds;
On obtient après le premier Enter :
[14159 33],314159
(ou ["Done","Done"],314159)
On remarquera que : $113 * 314159 + 33 = 355 * 10^5$
Si on appuie 8 fois sur enter on obtient un nombre ayant 41 chiffres : la partie entière 3 suivi des 40 décimales de 355/113 :
[5309 83],3141592920353982300884955752212389380 5309 Si on tape :
l
On obtient :
31415929203539823008849557522123893805309
Remarquez que 5309 n'a que 4 chiffres c'est donc que les dernières décimales sont : 05309 et on a :
$113 * l + 83 = 355 * 10^{40}$

29.4.2 Avec un programme du CAS

On donne comme nom de programme decimales et on coche CAS.
Le programme decimales renvoie une séquence l qui donne la partie entière (f1) et les p premières decimales (ds) d'une fraction f. On utilise les fonctions :
floor qui donne la partie entière s'un nombre,
numer qui donne le numérateur d'une fraction simplifiée,
denom qui donne le dénominateur d'une fraction simplifiée,
iquorem qui donne le quotient et le reste de la division euclidienne ds,n:=iquorem(10*n,d); est équivalent à :
ds:=iquo(10*n,d); (pour avoir le quotient de 10*n par d) et n:=irem(10*n,d); (pour avoir le reste de 10*n par d).
On tape, Shift Program puis Nouv. du bandeau.
On obtient une fenêtre avec Nom et CAS. On coche CAS et met comme Nom : decimales puis OK et on tape le programme qui donne la partie entière et les p décimales une à une du nombre rationnel f :

```
(f,p)->BEGIN
```

```
LOCAL n,d,l,f1,j,ds;
f1:=floor(f);
l:=f1;
n:=numer(f-f1);
d:=denom(f-f1);
FOR j FROM 1 TO p DO
    ds,n:=iquorem(10*n,d);
    l:=l,ds;
END;
RETURN l;
END;
```

On tape dans le CAS :
`decimales(355/311,20)`
On obtient :
`1,1,4,1,4,7,9,0,9,9,6,7,8,4,5,6,5,9,1,6,3`
On tape dans le CAS :
`decimales(355/113,20)`
On obtient une approximation de π à $3 * 10^{-7}$ pès :
`[3,1,4,1,5,9,2,9,2,0,3,5,3,9,8,2,3,0,0,8,8]`
On a en effet : `evalf(pi)` renvoie 3.1415926536 On peut déterminer les décimales par p groupes de g décimales et renvoyer un nombre entier l. L'écriture décimale de la fraction est alors $l * 10^(- p * g)$. On donne comme nom de programme `decimalesg` et on coche `CAS` (ou on modifie le programme `decimales`) :

```
(f,p,g)->BEGIN
 LOCAL n,d,l,f1,j,ds;
 f1:=floor(f);
 l:=f1;
 n:=numer(f-f1);
 d:=denom(f-f1);
 FOR j FROM 1 TO p DO
     ds,n:=iquorem(10^g*n,d);
     l:=l*10^g+ds;
 END;
 RETURN l;
END;
```

On tape dans le CAS :
`decimalesg(355/311,8,5)`
On obtient :
`11414790996784565916398713826366559485530` On tape dans le CAS :
`decimalesg(355/113,8,5)`
On obtient :
`31415929203539823008849557522123893805309`

29.5 La mèthode de Newton et l'algorithme de Héron

29.5.1 La méthode de Newton

Soit f deux fois dérivable ayant un zéro et un seul r dans l'intervalle $[a\ ;\ b]$. Supposons de plus que f' et f'' ont un signe constant sur $[a\ ;\ b]$. La méthode de Newton consiste à approcher r par l'abscisse x_1 du point commun à Ox et à la tangente en un point M_0 du graphe de f. Si M_0 a pour coordonées $(x_0, f(x_0))$ ($x_0 \in [a\ ;\ b]$), la tangente en M_0 a pour équation :
$y = f(x_0) + f'(x_0) * (x - x_0)$ et donc on a :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

On peut alors réitérer le processus, et on obtient une suite x_n qui converge vers r soit par valeurs supérieures, si $f' * f'' > 0$ sur $[a\ ;\ b]$ (i.e. si $f'(r) > 0$ et si f est convexe ($f'' > 0$ sur $[a\ ;\ b]$) ou si $f'(r) < 0$ et si f est concave ($f'' < 0$ sur $[a\ ;\ b]$)) soit par valeurs inférieures, si $f' * f'' < 0$ sur $[a\ ;\ b]$ (i.e. si $f'(r) < 0$ et si f est convexe ($f'' > 0$ sur $[a\ ;\ b]$) ou si $f'(r) > 0$ et si f est concave ($f'' < 0$ sur $[a\ ;\ b]$)). **L'algorithme de Héron** est un cas particulier de l'application de la méthode de Newton pour chercher les valeurs approchées de $\sqrt{a}$ pour a entier.
Dans ce cas $\sqrt{a}$ est un zéro de $f(x) = x^2 - a$ et $g(x) = f'(x) = 2x$ donc la suite des itérés est donné par :

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}(x_n - \frac{a}{x_n})$$

29.5.2 Traduction de l'algorithme de Newton

La fonction `newton_rac` renvoie la valeur approchée à p près de la racine de $f(x) = 0$ on commençant l'itération avec $x0$.
On remarquera que le paramètre f est une fonction et donc, que sa dérivée est la fonction `g:=function_diff(f)`.
On cherche une valeur approchée donc il faut écrire :
`x0:=evalf(x0)` car si on ne met pas `evalf`, les calculs de l'itération se feront excactement et seront vite compliqués.
On regarde au début si la suite des x_i ($i = 0..n$) est croissante ou décroissante à partir de $n = 1$ en comparant au début x_1 et x_2. On donne comme nom de programme `newtonrac` et on coche `CAS`.
Le programme `newtonrac` donne un zéro de `f` proche de `x0` à `p` près.
On tape le programme :

```
(f,x0,p)->BEGIN
LOCAL x1,h,g;
g:=function_diff(f)
x0:=evalf(x0);
x0:=x0-f(x0)/g(x0);
x1:=x0-f(x0)/g(x0);
IF (x1>x0) THEN
  h:=p;
ELSE
```

```
  h:=-p;
END;
WHILE (f(x1)*f(x1+h)>0) DO
   x1:=x1-f(x1)/g(x1);
END;
RETURN x1;
END;
```

On tape dans le CAS :
`f(x):=cos(x)-x`
`newtonrac(f,0.4,1e-10)`
On obtient :
`0.739085133215` On tape dans le CAS :
`cos(0.739085133215)-0.739085133215`
On obtient :
`2.70006239589e-13`

29.5.3 Traduction de l'algorithme de Héron

On donne comme nom de programme `heron` et on coche `CAS`.
Le programme `heron` donne une fraction qui approche `sqrt(a)` à `p` près lorsque `x0` est proche de `sqrt(a)`.
On tape

```
(a,x0,p)->BEGIN
LOCAL b;
b:=x0-p;
WHILE b^2>a DO
x0:=(x0+a/x0)/2;
b:=x0-p;
END;
RETURN x0;
END;
```

On tape dans le CAS :
`heron(2,3/2,10^-10)`
On obtient :
`66587/470832`
On tape dans le CAS :
`decimalesg(66587/470832,2,5)` On obtient :
`14142135623`
On tape dans le CAS :
`f:=heron(2,2,10^-40)`
On obtient :
`157258404803291863335321 7/111198484434986813793811 2`
On tape dans le CAS :
`r2:=decimalesg(f,8,5)` On obtient :
`1414213562373095048801688724209698078569 6`
et $sqrt2 \simeq r2 * 10^-40$ La librairie des flottants longs n'est pasimplémentés dans la HPPrime.

On utilise le logiciel `Xcas` pour vérifié.
On tape dans `Xcas` :
```
evalf(r2*10^-40,41)
```
On obtient :
```
1.41421356237309504880168872420969807856967
```
On tape dans `Xcas` :
```
evalf(sqrt(2),41)
```
On obtient car `Xcas` arrondit la dernière décimale :
```
1.41421356237309504880168872420969807856967
```

Chapitre 30

Exemple de programmes

30.1 Le PGCD et l'identité de Bézout depuis Home

30.1.1 Le PGCD

On utilise l'algorithme d'Euclide : On tape :

```
EXPORT PGCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
R:=A MOD B;
A:=B;
B:=R;
END;
RETURN A;
END;
```

Ou bien on utilise la fonction `irem` qui renvoie le reste de la division euclidienne :

```
EXPORT PGCD(A,B)
BEGIN
LOCAL R;
WHILE B<>0 DO
R:=CAS.irem(A,B);
A:=B;
B:=R;
END;
RETURN A;
END;
```

On tape :
`PGCD(45,25)`
On obtient : `5`.

30.1.2 L'identité de Bézout pour A et B

On utilise l'algorithme d'Euclide et les variables `U,V,R` qui vont évoluer de façon qu'à l'ètape k on ait $A*U_k+B*V_k=R_k$.
Ainsi quand R_p est le pgcd de A et B, on aura :
$A*U_p+B*V_p$ =pgcd(A,B).
Au début on a :
(1) $A=U_1*A+V_1*B$ ($R_1=A, U_1=1, V_1=0$)
(2) $B=U_2*A+V_2*B$ ($R_2=B, U_2=0, V_2=1$)
On veut avoir :
$R_3=U_3*A+V_3*B$
comme $R_3=A-B*Q_3$ (avec Q_3 quotient entier de $A=R_1$ par $B=R_2$) on trouve en faisant $(1)-Q_3*(2)$:
$U_3=U_1-Q_3*U_2$ et $V_3=V_1-Q_3*V_2$
et ainsi $R_3=U_3*A+V_3*B$. et à chaque étape on aura $R_k=U_k*A+V_k*B$ avec les relations :
$U_k=U_{k-2}1-Q_k*U_{k-1}$ et $V_k=V_{k-2}-Q_k*V_{k-1}$.
Pour écrire le programme on a besoin de 3 listes L_1, L_2, L_3 qui seront 3 étapes successives de $[U_k, V_k, R_k]$.
Au début :
$L1=1,0,R1$ ($R1=A$)
$L2:=0,1,R2$ ($R2=B$)
On calcule $L3$:
$L3$ est obtenu à partir de $L1$ et de $L2$ et si $Q3, R3$:=iquorem$(R1,R2)$, on $R3=R1-R2*Q3$ et donc $L3=L1-Q3*L2$.
Puis, $R1$ prend la valeur de $R2$ et $L1$ la valeur de $L2$ et
$R2$ prend la valeur de $R3$ et $L2$ la valeur de $L3$ ...etc
On s'arrete quand $R2=0$ et alors $R1$ =pgcd(A,B).
On tape :

```
EXPORT BEZOUT(A,B)
BEGIN
LOCAL L1,L2,L3,Q3,R1,R2,R3;
R1:=A;
R2:=B;
L1:={1,0,R1};
L2:={0,1,R2};
WHILE B<>0 DO
Q3,R3:=CAS.iquorem(R1,R2);
R1:=R2;
R2:=R3;
L3:=L1-Q3*L2;
L1:=L2;
L2:=L3;
END;
RETURN L1;
END;
```

On peut alléger le nombre de variables :

```
EXPORT BEZOUT(A,B)
BEGIN
LOCAL L1,L2,L3,Q;
L1:={1,0,A};
L2:={0,1,B};
WHILE L2(3)<>0 DO
//Q:=iquo(L1(3),L2(3));
Q:=(L1(3)-L1(3) MOD L2(3))/L2(3)
L3:=L1-Q*L2;
L1:=L2;
L2:=L3;
END;
RETURN L1;
END;
```

On tape :
`BEZOUT(45,10)`
On obtient : `1,-4,5`.
Ce qui veut dire que `1*45-4*10=5=PGCD(45,25)` On tape :
`BEZOUT(45,25)`
On obtient : `-1,2,5`.
Ce qui veut dire que `-1*45+2*25=5=PGCD(45,25)`

30.2 Le PGCD et l'identité de Bézout depuis CAS

30.2.1 Le PGCD avec le CAS sans faire de programme

On peut mettre en œuvre l'algorithme d'Euclide en utilisant la touche `Enter` de la calculatrice. On tape sur une ligne :
`a:=72;b:=33;` puis sur la touche `Enter`
Puis, on tape sur une ligne :
`r:=irem(a,b);a:=b;b:=r;` puis sur la touche `Enter` plusieurs fois jusqu'a ce que la dernière valeur soit nulle.
Lorsque la dernière valeur est nulle, le pgcd de `a` et `b` est la valeur `3` situé au dessus du 0.
On peut vérifier car la commande `gcd` donnant le pgcd existe :
`gcd(72,33)` renvoie bien `3`.

30.2.2 Le PGCD avec un programme CAS

On utilise la fonction `irem` pour écrire l'algorithme d'Euclide.
On coche `CAS` et on tape le nom du programme : `pgcd` et on tape :

```
(a,b)->
BEGIN
LOCAL r;
WHILE b<>0 DO
r:=irem(a,b);
a:=b;
```

```
b:=r;
END;
RETURN(a);
END;
```

On tape :
`pgcd(45,25)`
On obtient : `5`.

30.2.3 L'identité de Bézout avec le CAS sans faire de programme

On peut mettre en œuvre l'algorithme donnant les coefficients de l'identité de Bézout en utilisant la touche `Enter` de la calculatrice. On tape sur une ligne :
`a:=72;b:=33;l1:=[1,0,a];l2:=[0,1,b]` puis sur la touche `Enter`
Puis, on tape sur une ligne :
`q:=iquo(l1(3),l2(3));l3:=l1-q*l2;l1:=l2;l2:=l3;` puis sur la touche `Enter` plusieurs fois jusqu'a ce que la dernière valeur de la dernière liste soit nulle.
Lorsque cette dernière valeur est nulle, l'identidé de Bezout `[-5,11,3]` est la dernière liste située au dessus du 0 : cela veut dire que `-5*72+11*33=3]`.
On peut vérifier car la commande `iegcd` donnant l'identidé de Bezout existe :
`iegcd(72,33)` renvoie bien `[-5,11,3]`.

30.2.4 L'identité de Bézout avec un programme CAS

On utilise l'algorithme d'Euclide et les variables `u,v,r` qui vont évoluer de façon qu'à l'ètape k on ait $a * u_k + b * v_k = r_k$.
Ainsi quand r_p est le pgcd de a et b, on aura :
$a * u_p + b * v_p =$pgcd(a, b).
Pour écrire le programme on a besoin de 3 listes L_1, L_2, L_3 qui seront 3 étapes successives de $[u_k, v_k, r_k]$.
Au début :
$L1 = 1, 0, r1$ ($r1 = a$)
$L2 := 0, 1, r2$ ($r2 = b$)
On calcule $L3$:
$L3$ est obtenu à partir de $L1$ et de $L2$ et si $q3, r3 :=$iquorem$(r1, r2)$, on a : $r3 = r1 - r2 * q3$ et donc $L3 = L1 - q3 * L2$.
Puis, $r1$ prend la valeur de $r2$ et $L1$ la valeur de $L2$ et
$r2$ prend la valeur de $r3$ et $L2$ la valeur de $L3$ ...etc
On s'arrete quand $e2 = 0$ et alors $r1 =$pgcd(a, b).
On tape :

```
(a,b)->BEGIN
LOCAL l1,l2,l3,q;
l1:=[1,0,a];
l2:=[0,1,b];
WHILE l2(3)<>0 DO
q:=iquo(l1(3),l2(3));
l3:=l1-q*l2;
```

```
l1:=l2;
l2:=l3;
END;
RETURN l1;
END;
```

On tape :
`BEZOUT(45,10)`
On obtient : `1,-4,5`.
Ce qui veut dire que `1*45-4*10=5=PGCD(45,25)` On tape :
`BEZOUT(45,25)`
On obtient : `-1,2,5`.
Ce qui veut dire que `-1*45+2*25=5=PGCD(45,25)`