Démarrer en calcul formel

R. De Graeve, B. Parisse, B. Ycart

Université Joseph Fourier, Grenoble

`Xcas` est un logiciel libre de calcul formel. Il est téléchargeable à partir de :

`http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`

C'est un équivalent de Maple et Mupad, avec lesquels il est largement compatible. Il est possible de paramétrer `Xcas` pour qu'il accepte les syntaxes de Maple, Mupad ou des calculatrices TI (89, Voyage 200, Nspire CAS). Nous nous limiterons à la syntaxe propre à `Xcas`.

Ce cours d'introduction est destiné à faciliter la prise en main de `Xcas` par un utilisateur connaissant un peu de mathématiques (niveau terminale S, première année d'université scientifique), et ayant une pratique minimale de l'outil informatique.

Il est hors de question d'illustrer ici toutes les possibilités de `Xcas`. En particulier, nous ne parlerons ni de géométrie interactive, ni de la tortue logo, ni du tableur. Pour une pratique plus avancée, on se reportera à l'aide en ligne et aux différents documents disponibles à partir de la page d'accueil du logiciel.

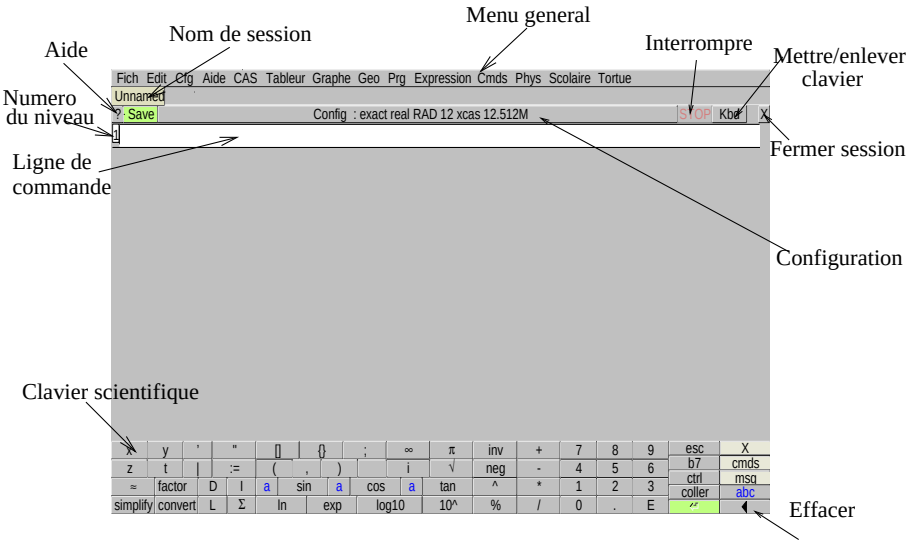
Le but de ce qui suit est d'aider le débutant en introduisant quelques unes des commandes les plus courantes. Il est conseillé de lire ce document après avoir lancé `Xcas` (sous Windows, cliquer sur le raccourci `xcasfr.bat`, sous linux Gnome, cliquer sur Xcas dans le menu Education ou en tapant depuis un Terminal `xcas &` puis la touche `enter`, sur Mac en cliquant sur xcas dans le menu Applications), en exécutant les commandes proposées une par une pour en comprendre l'effet.

La table des matières et l'index sont à la fin de ce document, cf. l'appendice A

1 Pour commencer

1.1 Interface

Pour l'instant, vous allez simplement saisir vos premières commandes. L'interface offre bien d'autres possibilités que vous découvrirez ensuite. Elle apparaît comme suit au lancement de `Xcas`.



Vous pouvez la redimensionner. De haut en bas cette interface fait apparaître :

- une barre de menu cliquable : `Fich`, `Edit`, `Cfg`, `Aide`, `CAS`,`Expression`, `Cmd`,`Prg`, `Graphic`, `Geo`,`Tableur`,...
- un onglet indiquant le nom de la session, ou `Unnamed` si la session n'a pas encore été sauvegardée (on peut ouvrir plusieurs sessions en parallèle et donc avoir plusieurs onglets représentant ces sessions),
- une zone de gestion de la session avec :
  - un bouton `?` pour ouvrir l'index de l'aide,
  - une barre-bouton `Save` pour sauvegarder la session,
  - un bouton affichant la configuration du CAS `Config: exact real ...` et permettant de la modifier,
  - un bouton rouge `STOP` permettant d'interrompre un calcul trop long,
  - un bouton `Kbd` pour faire apparaitre un clavier ressemblant à celui d'une calculatrice (on peut le voir ci-dessus). Il peut faciliter vos saisies, peut faire afficher une fenêtre de messages avec touche `Kbd->msg` (ou avec le menu `Cfg->Montrer->msg`) et afficher le bandeau des commandes avec la touche `Kbd->cmds` (ou avec le menu `Cfg->Montrer->bandeau`)
  - un bouton `x` pour fermer la session,
- une zone rectangulaire blanche numérotée 1 (première ligne de commande) dans laquelle vous pouvez taper votre première commande (cliquez si nécessaire pour faire apparaitre le curseur dans cette ligne de commande) : `1+1`, suivi de la touche "Entrée" ("Enter" ou "Return" selon les claviers). Le résultat apparaît au-dessous, et une nouvelle ligne de commande s'ouvre, numérotée 2.

Vous pouvez modifier l'aspect de l'interface et sauvegarder vos modifications pour les utilisations futures (menu `Cfg`).

Vous n'avez pour l'instant qu'à entrer des commandes dans les lignes de commandes successives. Si vous utilisez la version html de ce cours, vous pouvez copier-coller les commandes proposées depuis votre navigateur. Chaque ligne de commande saisie

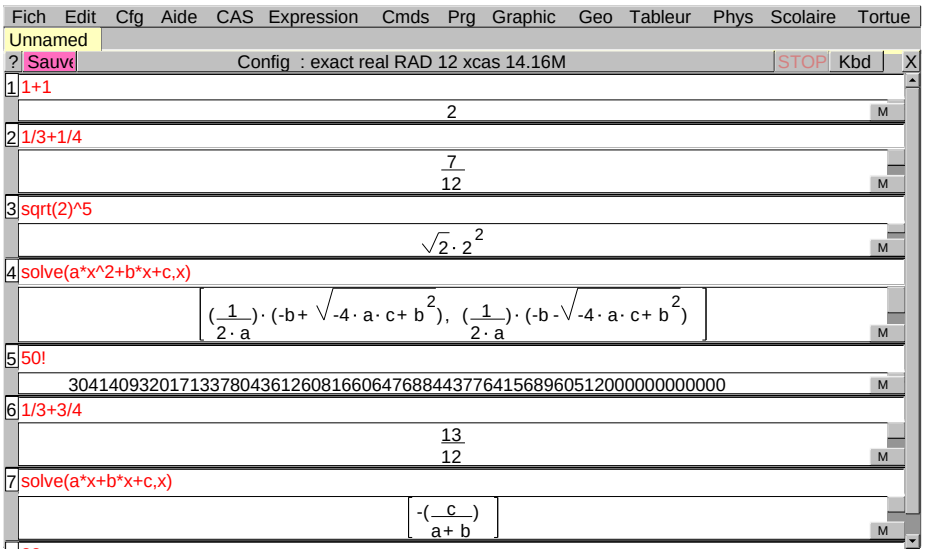
est exécutée par la touche "Entrée". Essayez par exemple d'exécuter les commandes suivantes :

```
1/3+1/4
sqrt(2)^5
solve(a*x^2+b*x+c,x)
50!
```

Toutes les commandes sont gardées en mémoire. Vous pouvez donc remonter dans l'historique de votre session pour faire afficher à nouveau des commandes antérieures avec `Ctrl+↑` pour par exemple les modifier. Essayez, par exemple, en modifiant les commandes précédentes d'exécuter aussi :

```
1/3+3/4
solve(a*x+b*x+c,x)
```

On obtient alors



On peut alors voir apparaitre, sur la droite, une barre de scroll permettant de se déplacer dans les niveaux de la session et ici par exemple d'avoir accès au niveau 8. Le menu `Edit` vous permet de préparer des sessions plus élaborées qu'une simple succession de commandes. Vous pouvez créer des groupes de lignes de commandes (sections), ajouter des commentaires ou fusionner des niveaux en un seul niveau.

1.2 Les commandes et l'aide en ligne

Les commandes sont regroupées par thèmes dans les menus du bandeau supérieur : `CAS`, `Expression`,`Cmds`, `Prg`, `Graphic`, `Geo`, `Tableur`, `Phys`, `Scolaire`, `Tortue`. Certains menus sont des menus dit menus "Assistant" parce que les commandes sont classées par thème et sont explicitées (menu `CAS`) ou parce qu'une boite de dialogue vous demande de préciser les paramètres de la commande choisie (menu

Tableur▶Maths ou menu Graphic).
Les autres menus contiennent les noms des commandes : le menu Cmds contient toutes les commandes de calcul formel, le menu Prg contient toutes les commandes que l'on utilise en programmation, le menu Geo contient toutes les commandes de géométrie... Lorsqu'on sélectionne une commande dans un menu,

- soit une boite de dialogue s'ouvre vous permettant de spécifier les arguments de la commande (par exemple pour tracer une courbe depuis le menu Graphic ou pour faire des statistiques depuis le menu Tableur▶Maths,
- soit la commande est recopiée dans la ligne de commande. Pour connaitre la syntaxe de cette commande, appuyez sur le bouton ? en haut à gauche, ou faites afficher la zone de Messages (en utilisant le menu Cfg->Montrer->msg). Vous pouvez aussi :
  - ouvrir l'index de l'aide à la commande sélectionnée (cela est automatique si on a cocher la case Aide index automatique dans le menu de configuration générale : Cfg->Configuration generale). Il faut alors cliquer sur le bouton OK pour que la commande soit recopiée dans la ligne de commande à condition que le curseur soit dans une ligne de commande.
    utoriel.tex Vous pouvez aussi cliquer sur le bouton Details pour afficher la page du manuel correspondant à la commande dans votre navigateur.
  - ouvrir automatiquement la page correspondante du manuel dans votre navigateur, en cochant la case Aide HTML automatique dans le menu de configuration générale (Cfg-> Configuration generale).

Le menu Aide contient les différentes formes d'aide possible : un guide de l'utilisateur (interface), un guide de référence (Manuels->Calcul formel, aide detaillée sur chaque commande), un Index (liste des commandes classées par ordre alphabétique avec une ligne d'entrée permettant de se déplacer facilement) et une recherche par mots clefs.

Si vous connaissez déjà le nom d'une commande et que vous désirez vérifier sa syntaxe (par exemple factor), vous pouvez saisir le début du nom de commande (disons fact) puis taper sur la touche de tabulation (située à gauche de la touche A sur un clavier français) ou cliquer sur le bouton ? en haut à gauche. L'index des commandes apparaît alors dans une fenêtre, positionné à la première complétion possible, avec une aide succinte sur chaque commande. Par exemple, vous voulez factoriser un polynôme, vous supposez que le nom de commande commence par fact, vous tapez donc fact puis la touche de tabulation, vous sélectionnez à la souris factor (ou un des exemples) puis vous cliquez sur OK.

Vous pouvez aussi saisir ?factor pour avoir l'aide succinte en réponse. Si le nom que vous avez saisi n'est pas reconnu, des commandes proches vous sont suggérées.

1.3 Entrer des commandes

L'exécution d'une ligne se fait simplement par la touche "Entrée". Si on ne souhaite pas afficher le résultat, on termine la ligne de commande par :; et on valide avec "Entrée". On peut éditer plusieurs commandes à la file avant leur exécution à condition de les séparer par un point-virgule.

Au début, de nombreuses erreurs proviennent d'une mauvaise traduction des mathématiques : `Xcas` ne peut pas les comprendre telles que vous les écrivez. Votre clavier vous permet de taper ax^2+bx+c, mais votre ordinateur ne peut pas comprendre que vous souhaitez élever x au carré, le multiplier par a, etc... Vous devez spécifier chaque opération, et la syntaxe correcte est `a*x^2+b*x+c`. La multiplication doit être notée par une étoile dans les commandes, mais est notée par un point dans les réponses. Nous insistons sur le fait que pour `Xcas`, `ax` est une variable dont le nom comporte deux lettres, et pas le produit de `a` par `x`.

Opérations	
`+`	addition
`-`	soustraction
`*`	mutiplication
`/`	division
`^`	puissance

Modulo quelques précautions, l'écriture des formules est assez directe. Les parenthèses ont le sens usuel pour spécifier l'ordre des opérations. Les crochets sont réservés aux listes et aux indices. Les priorités entre opérations sont standard (la multiplication est prioritaire sur l'addition, la puissance sur la multiplication). Par exemple :

- `2*a+b` retourne $2 \cdot a + b$
- `a/2*b` retourne $\frac{a \cdot b}{2}$
- `a/2/b` retourne $\frac{\frac{a}{2}}{b}$
- `normal(a/2/b)` retourne $\frac{a}{2 \cdot b}$
- `a^2*b` retourne $a^2 \cdot b$

Dans le doute, il est toujours prudent de mettre des parenthèses pour s'assurer que l'ordre des opérations est celui souhaité.

Les commandes sont numérotées, ainsi que les réponses, mais, si vous avez modifié une ligne de commande, celle-ci garde le numéro qu'elle avait. On peut rappeler par `ans()` (answer) la réponse précédente c'est à dire la réponse de la dernière commande évaluée.

2 Les objets du calcul formel

2.1 Les nombres

Les nombres peuvent être exacts ou approchés. Les nombres exacts sont les constantes prédéfinies, les entiers, les fractions d'entiers et plus généralement toute expression ne contenant que des entiers et des constantes, comme `sqrt(2)*e^(i*pi/3)`. Les nombres approchés sont notés avec la notation scientifique standard : partie entière suivie du point de séparation et partie fractionnaire (éventuellement suivie de `e` et d'un exposant). Par exemple, `2` est un entier exact, `2.0` est la version approchée du

même entier ; `1/2` est un rationnel, `0.5` est la version approchée du même rationnel. `Xcas` peut gérer des nombres entiers en précision arbitraire : essayez de taper `500!` et comptez le nombre de chiffres de la réponse.

On passe d'une valeur exacte à une valeur approchée par `evalf`, on transforme une valeur approchée en un rationnel exact par `exact` Les calculs sont effectués en mode exact si tous les nombres qui interviennent sont exacts. Ils sont effectués en mode approché si un des nombres est approché. Ainsi `1.5+1` renvoie un nombre approché alors que `3/2+1` renvoie un nombre exact.

```
sqrt(2)
evalf(sqrt(2))
sqrt(2)-evalf(sqrt(2))
exact(evalf(sqrt(2)))*10^9
exact(evalf(sqrt(2)*10^9))
```

Pour les nombres réels approchés, la précision par défaut est proche de 14 chiffres significatifs (la précision relative est de 53 ou 45 bits pour les réels flottants normalisés selon les versions de Xcas). Elle peut être augmentée, en donnant le nombre de décimales désiré comme second argument de `evalf`.

```
evalf(sqrt(2),50)
evalf(pi,100)
```

On peut aussi changer la précision par défaut pour tous les calculs en modifiant la variable `Digits`.

```
Digits:=50
evalf(pi)
evalf(exp(pi*sqrt(163)))
```

La lettre `i` est réservée à $\sqrt{-1}$ et ne peut être réaffectée ; en particulier on ne peut pas l'utiliser comme indice de boucle.

```
(1+2*i)^2
(1+2*i)/(1-2*i)
e^(i*pi/3)
```

`Xcas` distingue l'infini non signé `infinity` (∞), de `+infinity` ou `inf` ($+\infty$) et de `-infinity` ou `-inf`($-\infty$).

```
1/0; (1/0)^2; -(1/0)^2
```

Constantes prédéfinies	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i = \sqrt{-1}$
`infinity`	∞
`+infinity` ou `inf`	$+\infty$
`-infinity` ou `-inf`	$-\infty$

2.2 Les caractères et les chaînes

Une chaîne est parenthésée par des guillemets (`"`). Un caractère est une chaîne ayant un seul élément.

```
s:="azertyuiop"
size(s)
s[0]+s[3]+s[size(s)-1]
concat(s[0],concat(s[3],s[size(s)-1]))
head(s)
tail(s)
mid(s,3,2)
l:=asc(s)
ss:=char(l)
string(123)
expr(123)
expr(0123)
```

Chaînes	
`asc`	chaîne->liste des codes ASCII
`char`	liste des codes ASCII->chaîne
`size`	nombre de caractères
`concat` ou `+`	concaténation
`mid`	morceau de chaîne
`head`	premier caractère
`tail`	chaîne sans le 1ier caractère
`string`	nombre ou expression->chaîne
`expr`	chaîne->nombre (base 10 ou 8) ou expression

2.3 Les variables

On dit qu'une variable est formelle si elle ne contient aucune valeur : toutes les variables sont formelles tant qu'elles n'ont pas été affectées (à une valeur). L'affectation est notée `:=`. Au début de la session `a` est formelle, elle devient affectée après l'instruction `a:=3`, `a` sera alors remplacé par 3 dans tous les calculs qui suivent, et `a+1` renverra 4. `Xcas` conserve tout le contenu de votre session. Si vous voulez que la variable `a` après l'avoir affectée, soit à nouveau une variable formelle, il faut la "vider" par `purge(a)`. Dans les exemples qui suivent, les variables utilisées sont supposées avoir été purgées avant chaque suite de commandes.

Il ne faut pas confondre

- le signe `:=` qui désigne l'affectation
- le signe `==` qui désigne une égalité booléenne : c'est une opération binaire qui retourne 1 ou 0 (1 pour true qui veut dire Vrai et 0 pour false qui veut dire Faux)
- le signe `=` utilisé pour définir une équation.

```
a==b
```

```
a:=b
a==b
solve(a=b,a)
solve(2*a=b+1,a)
```

On peut faire certains types d'hypothèses sur une variable avec la commande `assume`, par exemple `assume(a>2)`. Une hypothèse est une forme spéciale d'affectation, elle efface une éventuelle valeur précédemment affectée à la variable. Lors d'un calcul, la variable n'est pas remplacée mais l'hypothèse sera utilisée dans la mesure du possible, par exemple `abs(a)` renverra `a` si on fait l'hypothèse `a>2`.

```
sqrt(a^2)
assume(a<0)
sqrt(a^2)
assume(n,integer)
sin(n*pi)
```

La fonction `subst` permet de remplacer une variable dans une expression par un nombre ou une autre expression, sans affecter cette variable.

```
subst(a^2+1,a=1)
subst(a^2+1,a=sqrt(b-1))
a^2+1
```

Remarque : pour stocker une valeur dans une variable par référence, par exemple pour modifier une valeur dans une liste (un vecteur, une matrice), sans recréer une nouvelle liste mais en modifiant en place la liste existante, on utilise l'instruction `=<` au lieu de `:=`. Cette instruction est plus rapide que l'instruction `:=`, car elle économise le temps de copie de la liste.

2.4 Les expressions

Une expression est une combinaison de nombres et de variables reliés entre eux par des opérations : par exemple `x^2+2*x+c`.

Lorsqu'on valide une commande, `Xcas` remplace les variables par leur valeur si elles en ont une, et exécute les opérations.

```
(a-2)*x^2+a*x+1
a:=2
(a-2)*x^2+a*x+1
```

Certaines opérations de simplification sont exécutées automatiquement lors d'une évaluation :

- les opérations sur les entiers et sur les fractions, y compris la mise sous forme irréductible
- les simplifications triviales comme $x+0=x$, $x-x=0$, $x^1=x$...
- quelques formes trigonométriques comme $\cos(-x)=\cos(x)$, $\cos(\pi/4)=\sqrt{2}/2$, $\tan(\pi/4)=1$...

Nous verrons dans la section suivante comment obtenir plus de simplifications.

2.5 Développer et simplifier

En-dehors des règles de la section précédente, il n'y a pas de simplification systématique. Il y a deux raisons à cela. La première est que les simplifications non triviales sont parfois coûteuses en temps, et le choix d'en faire ou non est laissé à l'utilisateur ; la deuxième est qu'il y a en général plusieurs manières de simplifier une même expression, selon l'usage que l'on veut en faire. Les principales commandes pour transformer une expression sont les suivantes :

- `expand` : développe une expression en tenant compte uniquement de la distributivité de la multiplication sur l'addition et du développement des puissances entières.
- `normal` et `ratnormal` : d'un bon rapport temps d'exécution-simplification, elles écrivent une fraction rationnelle (rapport de deux polynômes) sous forme de fraction irréductible développée ; `normal` tient compte des nombres algébriques (par exemple comme `sqrt(2)`) mais pas `ratnormal`. Les deux ne tiennent pas compte des relations entre fonctions transcendantes (par exemple comme `sin` et `cos`).
- `factor` : un peu plus lente que les précédentes, elle écrit une fraction sous forme irréductible factorisée.
- `simplify` : elle essaie de se ramener à des variables algébriquement indépendantes avant d'appliquer `normal`. Ceci est plus coûteux en temps et "aveugle" (on ne contrôle pas les réécritures intermédiaires). Les simplifications faisant intervenir des extensions algébriques (par exemple des racines carrées) nécessitent parfois deux appels et/ou des hypothèses (`assume`) pour enlever des valeurs absolues avant d'obtenir la simplification souhaitée.
- `tsimplify` essaie de se ramener à des variables algébriquement indépendantes mais sans appliquer `normal` ensuite.

Dans le menu `Expression` du bandeau supérieur, les sous-menus sont des menus de réécriture et contiennent d'autres fonctions, pour des transformations plus ou moins spécialisées.

```
b:=sqrt(1-a^2)/sqrt(1-a)
ratnormal(b)
normal(b)
tsimplify(b)
simplify(b)
simplify(simplify(b))
assume(a<1)
simplify(b)
simplify(simplify(b))
```

La fonction `convert` permet de passer d'une expression à une autre équivalente, sous un format qui est spécifié par le deuxième argument.

```
convert(exp(i*x),sincos)
convert(1/(x^4-1),partfrac)
convert(series(sin(x),x=0,6),polynom)
```

Transformations	
`simplify`	simplifier
`tsimplify`	simplifier (moins puissant)
`normal`	forme normale
`ratnormal`	forme normale (moins puissant)
`expand`	développer
`factor`	factoriser
`assume`	rajout d'hypothèses
`convert`	transformer en un format spécifié

2.6 Les fonctions

De nombreuses fonctions sont déjà définies dans `Xcas`, en particulier les fonctions classiques. Les plus courantes figurent dans le tableau ci-après ; pour les autres, voir le menu `Cmds->Reel->Special`.

Sinon l'utilisateur peut définir ses propre fonctions, par exemple :

- Définition d'une fonction d'une variable :
  `f(x) :=x*exp(x)` ou
  `f :=x->x*exp(x)` ou
  `f :=unapply(x*exp(x),x)`
- Définition de sa dérivée :
  `g :=function_diff(f)` ou
  `g :=unapply(diff(f(x),x),x))`
  ATTENTION `g(x) :=diff(f(x),x)` N'EST PAS VALABLE ! car ce qui est à droite de `:=` n'est pas évalué lors de la définition....il faut utiliser `unapply`.
- Définition d'une fonction de 2 variables :
  `h(r,t) :=(r*exp(t),r*t)` ou
  `h :=(r,t)->(r*exp(t),r*t);`
- Définition à partir d'une fonction de 2 variables, d'une fonction qui à 1 variable fait correspondre une fonction :
  `k(t) :=unapply(h(r,t),r)`
  ATTENTION Ici `k(t)` est une fonction de la variable `r` qui à `r` fait correspondre `h(r,t)`. On a par exemple : `k(1)(2)=(2*exp(1),2)`. Là aussi, il faut utiliser `unapply`.

Fonctions classiques	
`abs`	valeur absolue
`sign`	signe (-1,0,+1)
`max`	maximum
`min`	minimum
`round`	arrondi
`floor`	partie entière (plus grand entier $\leq$)
`frac`	partie fractionnaire
`ceil`	plus petit entier $\geq$
`re`	partie réelle
`im`	partie imaginaire
`abs`	module
`arg`	argument
`conj`	conjugué
`affixe`	affixe
`coordonees`	coordonnées
`factorial ou !`	factorielle
`sqrt`	racine carrée
`exp`	exponentielle
`log`	logarithme naturel
`ln`	logarithme naturel
`log10`	logarithme en base 10
`sin`	sinus
`cos`	cosinus
`tan`	tangente
`cot`	cotangente
`asin`	arc sinus
`acos`	arc cosinus
`atan`	arc tangente
`sinh`	sinus hyperbolique
`cosh`	cosinus hyperbolique
`tanh`	tangente hyperbolique
`asinh`	argument sinus hyperbolique
`acosh`	argument cosinus hyperbolique
`atanh`	argument tangente hyperbolique

Pour créer une nouvelle fonction, il faut la déclarer à l'aide d'une expression contenant la variable. Par exemple l'expression x^2-1 est définie par `x^2-1`. Pour la transformer en la fonction f qui à x associe x^2-1, trois possibilités existent :

```
f(x):= x^2-1
f:=x->x^2-1
f:=unapply(x^2-1,x)

f(2);
f(a^2);
```

Si `f` est une fonction d'une variable et `E` est une expression, `f(E)` est une autre expression. Il est essentiel de ne pas confondre fonction et expression. Si on définit : `E:=x^2-1`, alors la variable `E` contient l'expression x^2-1. Pour avoir la valeur de cette expression en $x=2$ il faut écrire `subst(E,x=2)` et non `E(2)` car `E` n'est pas une fonction. Lorsqu'on définit une fonction, le membre de droite de l'affectation n'est pas évalué. Ainsi l'écriture `E:=x^2-1; f(x):=E` définit la fonction $f : x \mapsto E$ car `E` n'est pas évalué. Par contre `E:= x^2-1; f:=unapply(E,x)` définit bien la fonction $f : x \mapsto x^2-1$ car `E` est évalué.

On peut ajouter et multiplier des fonctions, par exemple `f:=sin*exp`. Pour composer des fonctions, on utilise l'opérateur `@` et pour composer plusieurs fois une fonction avec elle-même, on utilise l'opérateur `@@`.

```
f:=x->x^2-1;
(f@f)(2);
(f@sqrt)(a);
f1:=f@sin
f2:=f@f
f3:=f@@3
f1(a)
f2(a)
f3(a)
```

On peut définir des fonctions de plusieurs variables à valeurs dans $\mathbb{R}$ comme :
`f(x,y):=x+2*y`
et des fonctions de plusieurs variables à valeurs dans $\mathbb{R}^p$ par exemple :
`f(x,y):=(x+2*y,x-y)`

2.7 Listes, séquences, ensembles

`Xcas` distingue plusieurs sortes de collections d'objets, séparés par des virgules :

- les listes (entre crochets)
- les séquences (entre parenthèses)
- les ensembles (entre pourcentage-accolades)

```
liste:=[1,2,4,2]
sequence:=(1,2,4,2)
ensemble:=%{1,2,4,2%}
```

Les listes peuvent contenir des listes (c'est le cas des matrices), alors que les séquences sont plates (un élément d'une séquence ne peut pas être une séquence). Dans un ensemble, l'ordre n'a pas d'importance et chaque objet est unique. Il existe une autre structure, appelée table, dont nous reparlerons plus loin.

Il suffit de mettre une séquence entre crochets pour en faire une liste ou entre accolades précédées de `%` pour en faire un ensemble. On passe d'une liste à sa séquence associée par `op`, d'une séquence à sa liste associée en la mettant entre crochets ou avec la fonction `nop`. Le nombre d'éléments d'une liste est donné par `size` (ou `nops`).

```
se:=(1,2,4,2)
li:=[se]
op(li)
nop(se)
nops(se)
%{se%}
size([se])
size(%{se%})
```

Pour fabriquer une liste ou une séquence, on utilise des commandes d'itération comme `$` ou `seq` (qui itèrent une expression) ou `makelist` (qui définit une liste à l'aide d'une fonction).

```
1$5
k^2 $ (k=-2..2)
seq(k^2,k=-2..2)
seq(k^2,k,-2..2)
[k^2$(k=-2..2)]
seq(k^2,k,-2,2)
seq(k^2,k,-2,2,2)
makelist(x->x^2,-2,2)
seq(k^2,k,-2,2,2)
makelist(x->x^2,-2,2,2)
```

La séquence vide est notée `NULL`, la liste vide `[]`. Pour ajouter un élément à une séquence il suffit d'écrire la séquence et l'élément séparés par une virgule. Pour ajouter un élément à une liste on utilise `append`. On accède à un élément d'une liste ou d'une séquence grâce à son indice mis entre crochets, le premier élément étant d'indice 0.

```
se:=NULL; se:=se,k^2$(k=-2..2); se:=se,1
li:=[1,2]; (li:=append(li,k^2))$(k=-2..2)
li[0],li[1],li[2]
```

Les polynômes sont souvent définis par une expression, mais ils peuvent aussi être représentés par la liste de leurs coefficients par ordre de degré décroissant, avec comme délimiteurs `poly1[` et `]`. Il existe aussi une représentation pour les polynômes à plusieurs variables. Les fonctions `symb2poly` et `poly2symb` permettent de passer de la représentation expression à la représentation par liste et inversement, le deuxième argument détermine s'il s'agit de polynômes en une variable (on met le nom de la variable) ou de polynômes à plusieurs variables (on met la liste des variables).

Séquences et listes	
`E$(k=n..m)`	créer une séquence
`seq(E,k=n..m)`	créer une séquence
`[E$(k=n..m)]`	créer une liste
`makelist(f,k,n,m,p)`	créer une liste
`op(li)`	passer de liste à séquence
`nop(se)`	passer de séquence à liste
`nops(li)`	nombre d'éléments
`size(li)`	nombre d'éléments
`sum`	somme des éléments
`product`	produit des éléments
`cumSum`	sommes cumulées des éléments
`apply(f,li)`	appliquer une fonction aux éléments d'une liste
`map(li,f)`	appliquer une fonction aux éléments d'une liste
`map(li,f,matrix)`	appliquer une fonction aux éléments d'une matrice
`poly2symb`	polynôme associé à une liste
`symb2poly`	coefficients d'un polynôme

2.8 Temps de calcul, place mémoire

Le principal problème du calcul formel est la complexité des calculs intermédiaires. Elle se traduit à la fois par le temps nécessaire à l'exécution des commandes et par la place mémoire requise. Les algorithmes implémentés dans les fonctions de `Xcas` sont performants, mais ils ne peuvent pas être optimaux dans tous les cas. La fonction `time` permet de connaître le temps d'exécution d'une commande (si ce temps est très court, `Xcas` exécute plusieurs fois la commande pour afficher un résultat plus précis). La mémoire utilisée apparaît dans les versions Unix dans la ligne d'état (la barre-bouton). Si le temps d'exécution d'une commande dépasse quelques secondes, il est possible que vous ayez commis une erreur de saisie. N'hésitez pas à interrompre l'exécution (bouton rouge `STOP` en haut à droite, il est conseillé de faire une sauvegarde de votre session auparavant).

3 Outils pour l'Analyse

3.1 Dérivées

La fonction `diff` permet de calculer la dérivée d'une expression par rapport à une ou plusieurs de ses variables. Pour dériver une fonction f, on peut appliquer `diff` à l'expression $f(x)$, mais alors le résultat est une expression. Si on souhaite définir la fonction dérivée, il faut utiliser `function_diff`.

```
E:=x^2-1; diff(E);
f:=unapply(E,x); diff(f(x));
f1:=function_diff(f);f1(x);
```

Il ne **faut pas** définir la fonction dérivée par `f1(x):=diff(f(x))`, car dans cette définition, `x` aurait deux sens incompatibles : c'est d'une part la variable formelle de dérivation et d'autre part l'argument de la fonction `f1`. D'autre part, cette définition évaluerait `diff` à chaque appel de la fonction (car le membre de droite d'une affectation n'est jamais évalué), ce qui serait inefficace.
Il faut utiliser `f1:=function_diff(f)`, ou `f1:=unapply(diff(f(x)),x)`.

La fonction `diff` s'applique à n'importe quelle combinaison de variables, et permet de calculer des dérivées partielles successives.

```
E:=sin(x*y)
diff(E,x)
diff(E,y)
diff(E,x,y)-diff(E,y,x)
simplify(ans())
diff(E,x$2,y$3)
```

Si le deuxième argument de `diff` est une liste, une liste de dérivées est retournée. Par exemple pour calculer le gradient de $\sin(xy)$: `diff(sin(x*y),[x,y])` (on peut aussi utiliser `grad`). Des commandes particulières permettent de calculer les combinaisons classiques de dérivées partielles.

Dérivées	
`diff(ex,t)`	dérivée d'une expression par rapport à t
`function_diff(f)`	fonction dérivée d'une fonction
`diff(ex,x$n,y$m)`	dérivées partielles
`grad`	gradient
`divergence`	divergence
`curl`	rotationnel
`laplacian`	laplacien
`hessian`	matrice hessienne

3.2 Limites et développements limités

La fonction `limit` calcule les limites finies ou infinies, quand elles existent. On peut demander une limite à gauche ou à droite à l'aide d'un quatrième argument (+1 ou -1). Quand la fonction dépend d'un paramètre, la limite obtenue peut dépendre des hypothèses faites, avec la fonction `assume`, sur ce paramètre.

```
limit(1/x,x,0)
limit(1/x,x,0,1)
limit(1/x,x,0,-1)
limit(a/x,x,0,1)
assume(a>0)
limit(a/x,x,0,1)
```

Pour les développements limités, deux fonctions sont disponibles, `series` et `taylor`. La différence est que l'ordre du développement doit être spécifié pour `series`, il est égal à 6 par défaut pour `taylor`.

L'ordre du développement limité demandé est utilisé par `Xcas` en interne pour faire ses développements. En cas de simplifications, l'ordre du développement obtenu pourra être inférieur, il faudra alors recommencer le calcul avec un ordre plus grand. L'expression retournée est constituée du polynôme de Taylor, plus un reste dans lequel apparaît une fonction `order_size` qui est telle que pour tout $a > 0$, x^a`order_size`(x) tend vers 0 quand x tend vers 0. Pour supprimer le reste et ne garder que le polynôme de Taylor, on peut utiliser `convert` avec l'option `polynom`.

```
taylor(1/(x^2+1),0)
taylor(1/(x^2+a^2),x=0)
series(1/(x^2+1),0,11)
series(1/(x^2+1),+infinity,11)
series(tan(x),pi/4,3)
series(sin(x)^3/((1-cos(x))*tan(x)),0,4)
series(sin(x)^3/((1-cos(x))*tan(x)),0,6)
series(tan(sin(x))-sin(tan(x)),0,13)
convert(ans(),polynom)
series(f(x),0,3)
g:=f@f; series(g(x),0,2)
```

Limites et développements limités	
`limit(ex,x,a)`	limite en a
`limit(ex,x,a,1)`	limite à droite en a
`limit(ex,x,a,-1)`	limite à gauche en a
`taylor(ex,a)`	développement limité en a ordre 6
`series(ex,a,n)`	développement limité en a ordre n

3.3 Primitives et intégrales

La fonction `int` calcule une primitive d'une expressionpar rapport à x ou par rapport à la variable donnée en argument. Si l'expression comporte plusieurs variables, il vaut préciser la variable d'intégration. Si on ajoute deux arguments a et b après la variable d'intégration, on calcule l'intégrale sur l'intervalle $[a,b]$. Eventuellement les bornes de l'intégrale peuvent être des expressions, ce qui permet de calculer des intégrales multiples.

```
int(x^2-1)
int(x^2-1,x,-1,1)
int(x*y,x)
int(x*y,y,0,x)
int(int(x*y,y,0,x),x,0,1)
```

Pour calculer une intégrale, un logiciel de calcul formel recherche une primitive puis l'évalue entre les bornes, afin d'obtenir une valeur exacte. Dans certains cas, il est inutile de calculer une primitive, soit parce qu'il n'en existe pas qui s'exprime avec les fonctions élémentaires, soit parce qu'un calcul numérique est plus adapté (par exemple

si le temps de calcul de la primitive est trop long, si la fonction présente des singularités dans l'intervalle d'intégration, etc...). Dans ce cas, on demande une valeur approchée en utilisant `evalf`, ou bien on utilise directement la fonction `romberg`, qui est appelée par `evalf`.

```
int(exp(-x^2))
int(exp(-x^2),x,0,10)
evalf(int(exp(-x^2),x,0,10))
romberg(exp(-x^2),x,0,10)
ans()/sqrt(pi))
```

Intégrales	
`int(E)`	primitive d'une expression
`int(E,x,a,b)`	intégrale exacte
`romberg(E,x,a,b)`	intégrale approchée

3.4 Résolution d'équations

Comme pour les intégrales on distingue :

- la résolution exacte qui renvoie toutes les solutions lorsque c'est possible (par exemple pour certaines équations polynomiales ou s'y ramenant)
- la résolution approchée qui calcule par un algorithme itératif une valeur proche d'une des solutions.

La résolution exacte s'effectue à l'aide de `solve`, dont le premier argument est une équation. Le membre de droite est supposé nul s'il n'est pas précisé. Par défaut `solve` ne retourne pas les solutions complexes. Pour les obtenir, il faut cocher `Complexe` dans la configuration du CAS (`Cfg->Configuration de CAS` ou sur la barre-bouton `Config :exact...`)
Exécutez les commandes suivantes avant et après avoir activé l'option `Complex`.

```
solve(x^2-a*x+2,x)
solve(x^2+2,x)
solve(x^3=1,x)
```

Les racines exactes sont calculées pour les polynômes de degré 1 et 2 (les formules de Cardan et Ferrari pour les degrés 3 et 4 ne sont pas utilisées, car les solutions obtenues ne sont pas facilement maniables). En degré supérieur, la fonction `solve` affiche un message d'erreur et renvoie une liste vide.

Pour les équations trigonométriques, les solutions principales sont renvoyées. Pour obtenir toutes les solutions, il faut activer l'option `All_trig_sol`. Comparer les commandes suivantes avec et sans cette option.

```
solve(cos(x),x)
solve(cos(x)+sin(x),x)
```

La fonction `solve` peut aussi résoudre des systèmes d'équations. Le premier argument est la liste des équations, le second est la liste des variables.

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

La fonction de résolution approchée est `fsolve`. Elle propose en option différents algorithmes (menus `Calc->Num_solve_eq` et `Calc->Num_solve_syst`). Le plus célèbre est l'algorithme de Newton, qui a de multiples variantes. Le principe général de tous ces algorithmes est de calculer les termes successifs d'une suite qui converge vers une solution de l'équation ou du système proposé. Il faut pour cela choisir selon les cas un point de départ, ou un intervalle de recherche.

```
fsolve((x^5+2*x+1)=0,x,1,newton_solver)
newton(x^5+2*x+1,x,1.0)
newton(x^5+2*x+1,x,1+i)
newton(x^5+2*x+1,x,-1+i)
```

Equations	
`solve(eq,x)`	résolution exacte d'une équation
`solve([eq1,eq2],[x,y])`	résolution exacte d'un système
`fsolve(eq,x)`	résolution approchée d'une équation
`fsolve([eq1,eq2],[x,y])`	résolution approchée d'un système
`newton`	méthode de Newton
`linsolve`	système linéaire
`proot`	racines approchées d'un polynôme

3.5 Equations différentielles

Comme dans les deux sections précédentes, on distingue le calcul exact, qui n'est pas toujours possible, du calcul approché. La résolution exacte s'effectue par `desolve`. Les dérivées de la fonction inconnue y peuvent s'écrire y', y'', qui sont traduits en `diff(y)`, `diff(diff(y))`. Si on ne spécifie pas de condition initiale, le résultat est donné en fonction de constantes arbitraires.

```
desolve(y'=y,y)
desolve(y''+2*y'+y=0,y)
desolve((x^2-1)*y'+2*y=0,y)
```

Les conditions initiales sont vues comme des équations supplémentaires, qui forment une liste avec l'équation différentielle.

```
desolve([y'=y,y(0)=1],y)
desolve([y"+2*y'+y=0,y(0)=1],y)
desolve([y"+2*y'+y=0,y(0)=1,y'(0)=1],y)
desolve([y"+2*y'+y=0,y(0)=1,y(1)=0],y)
desolve([(x^2-1)*y'+2*y=0,y(0)=1],y)
desolve((t^2-1)*diff(y(t),t)+2*y(t)=0,y(t))
```

La fonction `odesolve` permet de résoudre par des méthodes numériques une équation différentielle $y' = f(x,y)$ passant par un point (x_0,y_0). Par exemple

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

permet de calculer $y(2)$ où $y(x)$ est la solution de $y'(x) = \sin(xy)$, telle que $y(0) = 1$. La fonction `plotode` représente graphiquement la solution d'une équation différentielle, `plotfield` représente le champ des tangentes. La fonction `interactive_odeplot` représente le champ des tangentes et permet de cliquer sur le graphique pour tracer les solutions passant par les points cliqués.

```
plotfield(sin(x*y),[x,y])
plotode(sin(x*y),[x,y],[0,1])
erase()
interactive_plotode(sin(x*y),[x,y])
```

Equations différentielles	
`desolve`	résolution exacte
`odesolve`	résolution approchée
`plotode`	tracé de trajectoire
`plotfield`	tracé d'un champ de vecteurs
`interactive_plotode`	interface cliquable

4 Outils pour l'Algèbre

4.1 Arithmétique des entiers

Les opérations sur les entiers figurent dans le menu `Cmds->Entier`. Les calculs modulo p se font en utilisant `% p`. Une fois défini un entier modulo p, disons `a:=3%5`, tous les calculs seront efffectués dans $\mathbb{Z}/p\mathbb{Z}$: `a*2` renvoie `1%5` (6 modulo 5), `1/a` renvoie `2%5`, ... Pour calculer efficacement les puissances modulo p, on peut utiliser ce qui précède, ou la fonction `powermod` ou `powmod`.

```
a:=3%5
a+12
a^4
powermod(3,4,5)
```

Nombres entiers	
a%p	a modulo p
powmod(a,n,p)	a^n modulo p
irem	reste de la division euclidienne
iquo	quotient de la division euclidienne
iquorem	quotient et reste
ifactor	décomposition en facteurs premiers
ifactors	liste des facteurs premiers
idivis	liste des diviseurs
gcd	plus grand diviseur commun
lcm	plus petit multiple commun
iegcd	identité de Bezout
iabcuv	renvoie $[u,v]$ tels que $au+bv=c$
is_prime	l'entier est-il premier
nextprime	prochain entier premier
previousprime	entier premier précédent

4.2 Polynômes et fractions rationnelles

Les fonctions de traitement des polynômes sont dans le menu `Cmds->Polyn\^omes`.

On utilise `normal` ou `expand` pour développer, ou plus généralement mettre une fraction sous forme irréductible, et `factor` pour factoriser. Le résultat dépend du corps de nombres dans lequel on se place. Par défaut il s'agit des rationnels si les coefficients sont exacts ou des réels sinon. Pour les complexes (exacts ou approchées), il faut activer l'option `Complexe` à partir du bouton rappelant la configuration `Config:...`. On peut aussi déclarer les coefficients comme des entiers modulo p pour travailler dans $\mathbb{Z}/p\mathbb{Z}$ (commande `%`) ou dans un corps fini (défini par la commande `GF`). Exécutez les commandes suivantes avant et après avoir activé l'option `Complexe`.

```
P:=x^4-1
factor(P)
gcd(P,x^3-1)
divis(P)
propfrac(x^4/P)
partfrac(4/P)
Q:=(x^4+1)%3
factor(Q)
G:=GF(2,8,['a','G'])
factor(G(a^3)*x^2+1)
genpoly(5,3,x)
genpoly(2,3,x)
genpoly(2*y+5,3,x)
```

Polynômes	
`normal`	forme normale (développée et réduite)
`expand`	forme développée
`ptayl`	forme de Taylor
`peval` ou `horner`	évaluation en un point par l'algorithme de Horner
`genpoly`	polynôme défini par sa valeur en un point
`canonical_form`	trinôme sous forme canonique
`coeff`	(liste des) coefficient(s)
`poly2symb`	de l'expression algébrique à la forme symbolique
`symb2poly`	de la forme symbolique à l'expression algébrique
`pcoeff`	polynôme décrit par ses racines
`degree`	degré
`lcoeff`	coefficient du terme de plus haut degré
`valuation`	degré du monôme de plus bas degré
`tcoeff`	coefficient du terme de plus bas degré
`factor`	décomposition en facteurs premiers
`factors`	liste des facteurs premiers
`divis`	liste des diviseurs
`collect`	factorisation sur les entiers
`froot`	racines avec leurs multiplicités
`proot`	valeurs approchées des racines
`sturmab`	nombre de racines dans un intervalle
`getNum`	numérateur d'une fraction rationnelle
`getDenom`	dénominateur d'une fraction rationnelle
`propfrac`	isole partie entière et fraction propre
`partfrac`	décomposition en éléments simples
`quo`	quotient de la division euclidienne
`rem`	reste de la division euclidienne
`gcd`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`egcd`	identité de Bezout
`divpc`	division suivant les puissances croissantes
`randpoly`	polynôme aléatoire
`cyclotomic`	polynômes cyclotomiques
`lagrange`	polynômes de Lagrange
`hermite`	polynômes de Hermite
`laguerre`	polynômes de Laguerre
`tchebyshev1`	polynômes de Tchebyshev
`tchebyshev2`	polynômes de Tchebyshev

4.3 Trigonométrie

Le menu `Cmds->R\'eel->Transcendental` contient les fonctions circulaires et hyperboliques ainsi que leurs inverses. Pour linéariser et développer on utilise `tlin` et `texpand`. Beaucoup d'autres réécritures sont accessibles à partir des menus

- `Expression->Trigo` : transformations en $\tan(x/2)$ (`halftan`), transformation des tangentes en sinus et cosinus (`tan2sincos`), ...
- `Expression->Trigo exp` : transformation des fonctions trigonométriques en exponentielles par les formules d'Euler (`trig2exp`), transformation des exponentielles en fonctions trigonométriques (`exp2trig`), transformation des exponentielles en puissances (`exp2pow`)...,
- `Expression->Trigo inv` : transformation des fonctions inverses

```
exp2pow(exp(3*ln(x)))
exp2trig(exp(i*x))
trig2exp(cos(x))
E:=sin(x)^4+sin(x)^3
El:=tlin(E)
texpand(El)
tsimplify(E)
tsimplify(El)
tsimplify(E-El)
halftan(E)
trig2exp(El)
Et:=trigtan(E)
tan2sincos(Et)
tan2sincos2(Et)
tan2cossin2(Et)
```

Trigonométrie	
`tlin`	linéariser
`tcollect`	linéariser et regrouper
`texpand`	forme polynomiale
`trig2exp`	trigonométrique vers exponentielle
`exp2trig`	exponentielle vers trigonométrique
`hyp2exp`	hyperbolique vers exponentielle

4.4 Vecteurs et matrices

Un vecteur est une liste de nombres, une matrice est la liste de ses vecteurs lignes. Le produit matriciel est noté comme le produit ordinaire `*`. Les vecteurs sont a priori des vecteurs lignes, mais le produit à droite par un vecteur ligne est effectué comme si c'était une colonne. En particulier, si `v` et `w` sont deux vecteurs de même taille, `v*w` retourne leur produit scalaire.

```
A:=[[1,2,3],[4,5,6],[7,8,9]]
v:=[1,1,1]
v*v
A*v
v*A
```

```
B:=[[1,1,1],[2,2,2]]
A*B
B*A
A*tran(B)
```

A partir d'une fonction qui à deux indices (j,k) associe un réel $a(j,k)$, on peut constituer une matrice avec `makemat` ou `matrix`. Pour `makemat` les indices commencent à 0, pour `matrix` il commencent à 1.

```
makemat((j,k)->j+2*k,3,2)
matrix(3,2,(j,k)->j+2*k)
```

On peut aussi créer des matrices par blocs avec la commande `blockmatrix`.

```
A:=makemat((j,k)->j+2*k,3,2)
B:=idn(3)
blockmatrix(1,2,[A,B])
blockmatrix(2,2,[A,B,B,A])
```

On accède à un élément d'une matrice grâce à deux indices séparés par une virgule et mis entre crochets. Le premier indice est l'indice de la ligne et le deuxième celui de la colonne. Les indices commencent à 0. Par exemple, si `A:=[[0,2],[1,3],[2,4]]` alors `A[2,1]` renvoie `4`. Pour extraire un bloc de la matrice, on utilise des intervalles comme indices : `A[1..2,0..1]` renvoie le bloc constitué des lignes 1 à 2 et des colonnes 0 à 1.

Notez que les matrices de `Xcas` sont recopiées entièrement à chaque modification d'un coefficient. Ceci est pénalisant si on modifie successivement dans un programme beaucoup de coefficients d'une même (grande) matrice.

Vecteurs et matrices	
`v*w`	produit scalaire
`cross(v,w)`	produit vectoriel
`A*B`	produit matriciel
`A.*B`	produit terme à terme
`1/A`	inverse
`tran`	transposée
`rank`	rang
`det`	déterminant
`ker`	base du noyau
`image`	base de l'image
`idn`	matrice identité
`ranm`	matrice à coefficients aléatoires

4.5 Systèmes linéaires

La fonction `linsolve` résout une liste d'équations linéaires, avec la même syntaxe que solve. On peut aussi utiliser `simult` pour résoudre plusieurs systèmes d'équations linéaires qui ne diffèrent que par leur second membre, en mettant comme premier

argument la matrice du système et comme second argument la matrice dont la (ou les) colonnes sont le (ou les) second membre(s) des systèmes, ou bien `rref` d'argument une matrice obtenue en bordant la matrice du système avec lesecond membre (`border(A,tran(b))` si `b` est une matrice colonne). Quand le système est impossible, `linsolve` retourne la liste vide, `simult` retourne un message d'erreur, `rref` retourne une matrice dont une des lignes est nulle, sauf le dernier coefficient. Quand le système est indéterminé, `linsolve` retourne la solution fonction de certaines variables, `simult` retourne seulement une solution, `rref` retourne une matrice dont une ou plusieurs lignes sont nulles. L'exemple ci-dessous concerne le système

$$\begin{cases} x & + & y & + & az & = & 1 \\ x & + & ay & + & z & = & 1 \\ ax & + & y & + & z & = & -2 \end{cases}$$

Il a une solution unique pour $a \neq 1$ et $a \neq -2$, il est impossible pour $a = 1$ et il est indéterminé pour $a = -2$.

```
linsolve([x+y+a*z=1,x+a*y+z=1,x+a*y+z=-2],[x,y,z])
a:=1
linsolve([x+y+a*z=1,x+a*y+z=1,x+a*y+z=-2],[x,y,z])
a:=-2
linsolve([x+y+a*z=1,x+a*y+z=1,x+a*y+z=-2],[x,y,z])
purge(a)
A:=[[1,1,a],[1,a,1],[a,1,1]]
solve(det(A),a)
A1:=subst(A,a=1)
rank(A1)
image(A1)
ker(A1)
A2:=subst(A,a=-2)
rank(A2)
image(A2)
ker(A2)
b:= [1,1,-2]
B:=tran(b)
simult(A,B)
simult(A1,B)
simult(A2,B)
M:=blockmatrix(1,2,[A,B])
rref(M)
rref(border(A,b))
rref(border(A1,b))
rref(border(A2,b))
```

Systèmes linéaires	
`linsolve`	résolution d'un système
`simult`	résolution simultanée de plusieurs systèmes
`rref`	réduction de Gauss-Jordan
`rank`	rang
`det`	déterminant du système

4.6 Réduction des matrices

La fonction `jordan` prend en entrée une matrice A et retourne en sortie une matrice de passage P et une forme réduite de Jordan J telles que $P^{-1}AP = J$. Soit A est diagonalisable auquel cas J est diagonale et contient les valeurs propres de A sur la diagonale, soit A n'est pas diagonalisable et J comporte des "1" ou des "0" au-dessus de la diagonale. Pour les matrices exactes et symboliques, seules les valeurs propres calculables par `solve` sont accessibles. Pour des matrices de nombres approchés, un algorithme numérique est utilisé, et il risque d'échouer en cas de valeurs propres multiples ou très proches. La matrice A de l'exemple qui suit a pour valeurs propres doubles 1 et 2. Elle est diagonalisable pour $a = 0$, non diagonalisable pour $a \neq 0$.

```
A:=[[1,1,-1,0],[0,1,0,a],[0,-1,2,0],[1,0,1,2]
factor(poly2symb(simplify(pcar(A))))
jordan(A)
eigenvals(A)
eigenvects(A)
jordan(subs(A,a=0))
eigenvects(subs(A,a=1))
jordan(evalf(subs(A,a=0)))
jordan(evalf(subs(A,a=1)))
```

Certaines fonctions, définies par des séries entières, s'étendent aux matrices dès lors que l'on sait calculer leur forme de Jordan. La plus utile est l'exponentielle.

```
A:=[[0,1,0],[0,0,1],[-2,1,2]]
jordan(A)
exp(A)
ln(A)
sin(A)
```

Réduction des matrices	
`jordan`	diagonalisation ou réduction de Jordan
`pcar`	coefficients du polynôme caractéristique
`pmin`	coefficients du polynôme minimal
`eigenvals`	valeurs propres
`eigenvects`	vecteurs propres

5 Représentations graphiques

Pour obtenir une représentation graphique dans Xcas, il faut saisir une commande dont la sortie est un objet graphique. Pour vous aider à effectuer les représentations graphiques les plus courantes, le menu `Graphic` vous propose des boites de dialogue qui se chargent de créer une ligne de commande et de le valider pour afficher le graphique souhaité.

Si vous souhaitez représenter plusieurs objets graphiques dans une même représentation, vous devez séparer les commandes les créant par un `;`. Vous pouvez aussi créer une figure (menu `Geo`, `Nouvelle figure`) et utiliser le menu `Geo` pour créer des objets géométriques.

Chaque commande graphique crée un objet graphique 2-d ou 3-d, qui est traduit en réponse par une image dans la fenêtre `Xcas`. A droite de cette image, des boutons de zoom `in` et `out` permettent d'agrandir ou de rapetisser la représentation, des flèches permettent de la déplacer.

Les paramètres par défaut (en particulier les intervalles de représentation en abscisse et ordonnée) peuvent être changés dans la fenêtre de configuration graphique accessible depuis le menu `Cfg->Configuration graphique`.

Notez enfin que les objets graphiques 2-d sont aussi affichés dans une fenêtre appelée `DispG` (Display Graphics) que vous pouvez faire apparaître par le menu `Cfg->Montrer->DispG` ou avec la commande `DispG`. La différence est que les graphiques successifs sont tracés individuellement dans chaque fenêtre de réponse, alors qu'ils sont superposés dans la fenêtre `DispG`. Vous pouvez effacer la fenêtre `DispG` par la commande `ClrGraph`.

5.1 Tracés de courbes

Pour créer rapidement des tracés de courbes simples, il est conseillé d'utiliser le menu `Graphic`.

L'instruction de tracé d'une courbe représentative de fonction est `plot` avec en paramètres une expression ou une liste d'expressions dont on veut la représentation graphique, puis la variable (éventuellement on indique l'intervalle de valeurs de la variable). Pour distinguer plusieurs courbes, on peut utiliser un troisième argument par exemple `couleur=` suivi de la liste des couleurs à utiliser. Les couleurs peuvent être codées par leur nom français, leur nom anglais ou leur numéro. La fonction `couleur` change la couleur de base pour toutes les fonctions graphiques qui suivent. La fonction `tangent` permet d'obtenir la tangente à une courbe en un point.

```
E:=(2*x+1)/(x^2+1);plot(E)
plot(E,x=-2..2,color=red)
couleur(vert);plot(E,couleur=rouge);tangent(plot(E),0)
DispG()
plot([sin(x),x,x-x^3/6],x=-2..2,couleur=[rouge,bleu,vert])
erase
li:=[(x+k*0.5)^2$(k=-5..5)];;
plot(li,x=-8..8,couleur=[k$(k=0..10)])
```

La fonction `plotparam` permet d'effectuer le tracé de $(x(t),y(t))$. Il faut définir les deux coordonnées comme une seule expression complexe dont $x(t)$ est la partie réelle et $y(t)$ la partie imaginaire. La fonction `plotpolar` trace les courbes en coordonnées polaires. La commande `plotimplicit(f(x,y),x,y)` trace l'ensemble des solutions de $f(x,y)=0$.

```
plotparam(sin(t)^3+i*cos(t)^3,t,0,2*pi)
plotparam(t^2+i*t^3,t,-1,1)
plotpolar(1/(1-2sin(t/2)),t,0,4*pi)
plotpolar(tan(t)+tan(t/2),t,0,2*pi)
plotimplicit(x^2+4*y^2-4,x,y)
```

Tracés de courbes	
`plot`	graphe d'une expression d'une variable
`plotfunc`	graphe d'une expression d'1 ou 2 variable(s)
`couleur`	choisir la couleur d'un tracé
`tangent`	tangente à une courbe
`plotparam`	courbe paramétrique
`plotpolar`	courbe en polaires
`plotimplicit`	courbe implicite

5.2 Objets graphiques 2D

`Xcas` étant aussi un logiciel de géométrie plane, de nombreuses figures peuvent être tracées (dans un écran que l'on obtient avec `Alt+g`) par des commandes du menu `Geo`, par exemple des polygones, des coniques... Les arguments de ces commandes sont souvent des points (commande `point`) qui peuvent en général être saisis directement par leur affixe complexe. Par exemple `cercle(2+3*i,2)` trace le cercle centré au point $(2,3)$, de rayon 2. La commande `legende` permet de placer un texte à un endroit, lui aussi spécifié par un nombre complexe. Les fonctions `polygonplot` et `scatterplot` prennent en entrée une liste d'abscisses et une liste d'ordonnées.

```
lx:=[k$(k=1..10)]
ly:=apply(sin,lx)
polygonplot(lx,ly)
erase
scatterplot(lx,ly)
polygone_ouvert(lx+i*ly)
```

Objets graphiques 2D	
`legend`	met du texte à partir d'un point donné
`point`	point donné par son affixe ou 2 coordonnées
`segment`	segment donnée par 2 points
`droite`	droite donnée par son équation ou 2 points
`cercle`	cercle donnée par centre et rayon
`inter`	intersection de courbes
`equation`	équation cartésienne
`parameq`	équation paramétrique
`polygonplot`	ligne polygonale
`scatterplot`	nuage de points
`polygone`	polygone fermé
`polygone_ouvert`	polygone ouvert

5.3 Objets graphiques 3D

- Pour tracer une surface définie par l'équation $z = f(x,y)$, on utilise la commande `plotfunc`, avec en arguments l'équation de la surface et la liste des deux variables. On peut aussi indiquer la plage de variable, et la discrétisation.
  ```
  plotfunc(x^2-y^2,[x,y])
  plotfunc(x+y^2,[x=-5..5,y=-2..2],xstep=0.5,ystep=0.1,
  affichage=vert+rempli)
  ```
  On obtient une surface en dimension 3. Pour modifier le point de vue, utilisez la souris en-dehors du cube de visualisation ou cliquez dans la figure 3-d, puis utilisez les touches x,X,y,Y,z,Z (rotations par rapport aux axes), + et - pour les zooms, les flèches de direction et page up/down pour changer la fenêtre de visualisation (la fenêtre de calcul par défaut est définie par la configuration graphique si on ne l'a pas indiquée en paramètre dans `plotfunc`)
- On peut aussi tracer une surface paramétrée avec `plotparam` dont le premier argument est une liste de taille 3 contenant les coordonnées du point et les 2 arguments suivants les paramètres :
  ```
  plotparam([u,v,u+v],u=-1..1,v=-2..2)
  ```
- Pour tracer des courbes paramétrées dans l'espace, on utilise aussi la commande `plotparam` mais avec un seul paramètre :
  ```
  plotparam([u,u^2,u^3],u=-1..1)
  ```
- On peut aussi tracer des objets géométriques 3D dans une figure 3-d que l'on obtient avec `Alt+h`, puis en utilisant des commandes du menu `Geo`, par exemple : point, droite ,plan, polygone, sphere…
  ```
  plan(z=x+y);
  droite(x=y,z=y);
  A:=point(1,2,3); B:=point(2,-1,1); C:=point(1,0,0);
  plan(A,B,C,couleur=cyan);
  droite(A,B,affichage=line_width_3)
  ```

Objets graphiques 3D	
`plotfunc`	surface par équation
`plotparam`	surface ou courbe paramétrique
`point`	point donné par 3 coordonnées
`droite`	droite donnée par 2 équations ou 2 points
`plan`	plan donné par 1 équation ou 3 points
`sphere`	sphère donnée par centre et rayon
`cone`	cone donné par centre, axe, angle d'ouverture
`inter`	intersection
`polygone`	polygone
`polygone_ouvert`	polygone ouvert

6 Programmation

6.1 Le langage

`Xcas` permet d'écrire des programmes, comme n'importe quel langage de programmation. Voici ses principales caractéristiques.

- C'est un langage fonctionnel. L'argument d'une fonction peut être une autre fonction. Si c'est le cas, on peut soit donner le nom de la fonction argument dans la commande, soit sa définition : par exemple `function_diff(f)` ou bien `function_diff(x->x^2)`.
- Il n'y a pas de distinction entre programme et fonction : une fonction renvoie la valeur de la dernière instruction évaluée ou ce qui suit le mot réservé `return`. Comme pour tous les environnements de calcul, programmer consiste à étendre `Xcas` en lui rajoutant les fonctions souhaitées. Structurer la programmation consiste à hiérarchiser les différentes fonctions qui s'appellent entre elles.
- Le langage est non typé. On distingue seulement les variables globales, qui ne sont pas déclarées, et les variables locales, déclarées en début de fonction.

Dans un programme, lorsqu'on appelle une variable munie d'un indice qui n'est pas affectée à une liste, séquence ou matrice, c'est une table qui est créée, et non une liste. Une table est un conteneur d'objets analogue aux listes et aux séquences. La différence est qu'elle peut être indicée par autre chose qu'un entier, par exemple une chaîne de caractères... Si `a` est une variable formelle, la commande `a[4]:=2` crée une table `a`. Pour que `a` soit une liste, il faut d'abord affecter `a` à une liste par exemple `a:=[0$10]` (si la taille de la liste est connue) ou `a:=[]` puis `a[4]:=2`. Même si le langage est non typé, il est donc recommandé d'initialiser les variables avant de les utiliser.

La syntaxe de déclaration d'une fonction est la suivante.

```
nom_fonction(var1,var2,...):={
local var_loc1, var_loc2,... ;
  instruction1;
  instruction2;
  ...
}
```

La syntaxe est soit avec des mots clef en français soit celle du langage C++.

Instructions en fançais	
affectation	`a:=2;`
entrée expression	`saisir("a=",a);`
entrée chaine	`saisir_chaine("a=",a);`
sortie	`afficher("a=",a);`
valeur retournée	`retourne(a);`
arrêt dans boucle	`break;`
alternative	`si <condition> alors <inst> fsi;` `si <condition> alors <inst1> sinon <inst2> fsi;`
boucle pour	`pour j de a jusque b faire <inst> fpour;` `pour j de a jusque b pas p faire <inst> fpour;`
boucle répéter	`repeter <inst> jusqua <condition>;`
boucle tantque	`tantque <condition> faire <inst> ftantque;`
boucle faire	`faire <inst1> si <condition> break;<inst2> ffaire;`

Instructions comme en C++	
affectation	`a:=2;`
entrée expression	`input("a=",a);`
entrée chaine	`textinput("a=",a);`
sortie	`print("a=",a);`
valeur retournée	`return(a);`
arrêt dans boucle	`break;`
alternative	`if (<condition>) {<inst>};` `if (<condition>) {<inst1>} else {<inst2>};`
boucle pour	`for (j:= a;j<=b;j++) {<inst>};` `for (j:= a;j<=b;j:=j+p) {<inst>};`
boucle répéter	`repeat <inst> until <condition>;`
boucle tantque	`while (<condition>) {<inst>};`
boucle faire	`do <inst1> if (<condition>) break;<inst2> od;`

Pour les tests, une condition est un booléen, résultat d'une expression logique, utilisant les opérateurs habituels.

Opérateurs logiques			
`==`	teste l'égalité	`!=`	teste la différence
`<`	teste la stricte infériorité	`>`	teste la stricte supériorité
`<=`	teste l'infériorité ou l'égalité	`>=`	teste la supériorité ou l'égalité
`&&, et`	opérateur booléen infixé et	`||, ou`	opérateur booléen infixé ou
`vrai`	est le booléen true ou 1	`faux`	est le booléen false ou 0
`non, !`	inverse logique		

Attention, `i` désigne $\sqrt{-1}$ et ne peut pas être utilisé comme variable de boucle. L'instruction `break;` permet de sortir d'une boucle et `continue;` de passer immédiatement à l'itération suivante. De nombreuses variantes sont reconnues en particulier en mode de compatibilité avec Maple, Mupad et les TI89/Voyage 200. On peut capturer des erreurs d'exécution par

```
try {bloc_erreurs_capturees}
catch (variable)
    {bloc_execute_si_erreur}
```

Par exemple :

```
try{A:=idn(2)*idn(3)}
catch(erreur)
{print("l'erreur est "+erreur)}
```

6.2 Quelques exemples

Pour écrire un programme, il est conseillé d'ouvrir un éditeur de programme avec le menu `Prg->Nouveau programme`. Le menu `Prg` de l'éditeur permet d'entrer facilement les structures de programmation. On peut ensuite sauvegarder le texte du programme indépendamment de la session de travail pour l'utiliser ensuite dans une autre session de travail.

Voici un programme qui donne le quotient et le reste de la division euclidienne de 2 entiers en utilisant les fonctions `iquo` qui renvoie le quotient et `irem` qui renvoie le reste (c'est la fonction `iquorem` de `Xcas`).

idiv2	

`idiv2(a,b):={`
`  local q,r;`
`  if (b!=0) {`
`  q:=iquo(a,b);`
`  r:=irem(a,b);}`
`  else {`
`  q:=0;`
`  r:=a;`
`  }`
`  return [q,r];`
`}`
	`idiv2(a,b):={`
	`  local q,r;`
	`  si b!=0 alors`
	`  q:=iquo(a,b);`
	`  r:=irem(a,b);`
	`  sinon`
	`  q:=0;`
	`  r:=a;`
	`  fsi`
	`  retourne [q,r];`
	`}`

Saisissez cette fonction `idiv2` dans un éditeur de programme, testez-la (bouton `OK`) puis sauvegardez par exemple sous le nom `idiv2.cxx`. Vous pouvez utiliser cette fonction dans une ligne de commande, en tapant par exemple `idiv2(25,15)`. Vous pourrez utiliser cette fonction dans une autre session `Xcas`, en utilisant la commande `read("idiv2.cxx")` ou en l'ouvrant depuis un éditeur de programme (et en le validant par OK).

Voici maintenant deux versions du calcul du PGCD de deux entiers, une version itérative, puis une version récursive.

pgcd_iteratif	

pgcdi(a,b):={
local r;
while (b!=0) {
r:=irem(a,b);
a:=b;
b:=r;
}
return a;
}:;
	pgcdi(a,b):={
	local r;
	tantque b!=0 faire
	r:=irem(a,b);
	a:=b;
	b:=r;
	ftantque
	retourne a;
	}:;

pgcd_recursif	

pgcdr(a,b):={
if (b!=0) return a;
return pgcdr(b,irem(a,b));
}:;
	pgcdr(a,b):={
	si b!=0 alors retourne a;fsi
	retourne pgcdr(b,irem(a,b));
	}:;

Il arrive parfois qu'un programme ne fonctionne pas du premier coup comme prévu (!) Il est alors possible de l'exécuter en mode pas-à-pas pour le mettre au point, avec la commande `debug`. Pour plus de détails consulter le menu `Aide->Interface`. Par exemple, pour le programme `idiv2`, on lance la mise au point en tapant :
`debug(idiv2(25,15))`
Le débuggueur affiche automatiquement la valeur des paramètres `a,b` puis des variables locales `q,r` lors de l'exécution instruction par instruction avec le bouton `sst`.

6.3 Style de programmation

`Xcas` est interprété et non compilé. Plus que le nombre de lignes du programme, c'est le nombre d'instructions réellement exécutées qui influence le temps de calcul. En règle générale, il est plus rapide de créer des listes ou des séquences que de programmer des boucles. Voici quelques manières de calculer 5000! : comparez leurs temps d'exécution.

```
5000!
product([n$(n=1..5000)])
product(cumSum([1$5000]))
f:=1; (f:=f*n)$(n=2..5000):;f
f:=1; for(n:=1;n<=5000;n++) {f:=f*n}
f:=1;n:=1; while(n<5000) {n:=n+1; f:=f*n}
f:=1; (f:=f*n)$(n=2..5000)
```

La rapidité d'exécution est parfois contradictoire avec la clarté du programme, et on doit accepter des compromis. Dans une utilisation courante, le temps de calcul n'est pas réellement un enjeu : on utilise en général les langages interprétés comme `Xcas` pour tester des algorithmes et réaliser des maquettes. Les applications en vraie grandeur sont codées dans des langages compilés comme C++ (en utilisant par exemple la librarie `giac` pour les fonctions de calcul formel).

7 Des exercices corrigés avec Xcas

7.1 Fonction et représentation graphique (niveau terminale S)

7.1.1 Exercice 1

On considère la fonction f de $\mathbb{R}$-{3} dans $\mathbb{R}$ définie par :

$$f(x) = (x+1)\ln|x-3|$$

1. Calculer la derivée première f' et seconde f'' de f.
   En déduire les variations de f'.
2. Calculer les limites de f' en $-\infty$ et en 3 à gauche.
3. Montrer que f' s'annule une seule fois en α sur $]-\infty;3[$. Donner un encadrement de α d'amplitude 0.1.
4. Étudier le signe de $f'(x)$ sur $\mathbb{R}$-{3} et en déduire les variations de f.
5. Tracer la courbe C de f dans un repère orthonormé (unité $1cm$).
6. Calculer l'aire en cm^2 de la région comprise entre C, l'axe des abscisses et les droites d'équations $x=-1$ et $x=2$.

Réponses

1. On tape pour définir la fonction f :

   ```
   f(x):=(x+1)*ln(abs(x-3))
   ```

   On tape pour définir la fonction f' :

   ```
   f1:=function_diff(f):;
   ```

   puis,

   ```
   f1(x)
   ```

   On obtient :

   ```
   ln(abs(x-3))+(x+1)/(x-3)
   ```

   Donc $f'(x) = \ln(|x-3|) + \dfrac{x+1}{x-3}$.
   On tape pour définir la fonction f'' :

   ```
   f2:=function_diff(f1):;
   ```

   puis,

   ```
   f2(x)
   ```

   On obtient :

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

   puis pour simplifier l'écriture, on tape :

   ```
   normal(f2(x))
   ```

   On obtient :

```
(x-7)/(x^2-6*x+9)
```

ou bien pour factoriser, on tape :

```
factor(f2(x))
```

On obtient :

```
(x-7)/((x-3)^2)
```

Donc $f''(x) = \dfrac{x-7}{(x-3)^2}$

Autre façon

On tape pour définir la fonction f :

```
f(x):=(x+1)*ln(abs(x-3))
```

On tape pour calculer $f'(x)$:

```
dfx:=diff(f(x)
```

On obtient :

```
ln(abs(x-3))+(x+1)/(x-3)
```

Donc $f'(x) = \ln(|x-3|) + \dfrac{x+1}{x-3}$.

Et on tape pour définir la fonction f' à partir de `dfx` :

```
f1:=unapply(dfx,x);
```

On tape pour calculer $f''(x)$:

```
ddfx:=diff(dfx)
```

On obtient :

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

ou pour avoir une écriture factorisée, on tape directement :

```
ddfx:=factor(diff(dfx))
```

On obtient :

```
(x-7)/((x-3)^2)
```

Donc $f''(x) = \dfrac{x-7}{(x-3)^2}$

Et on tape pour définir la fonction f'' à partir de `ddfx` :

```
f2:=unapply(ddfx,x);
```

Cette façon de faire à l'avantage de définir la fonction $f2 = f''$ à partir d'une expression simplifiée ou factorisée.

Attention!!! On ne peut pas écrire par exemple :
`g(x) :=normal(diff(f(x)))` pour définir la fonction $g = f'$ mais on doit écrire `g :=unapply(normal(diff(f(x))),x)` car sinon il y a confusion entre x variable de dérivation et x variable de la fonction g.

2. On tape pour avoir la limite de f' en $-\infty$:

```
limit(f1(x),x,-infinity)
```

On obtient :

```
+infinity
```

On tape pour avoir la limite de f' en 3^- :

```
limit(f1(x),x,3,-1)
```

On obtient :

```
-infinity
```

3. f' est continue et décroissante de $+\infty$ à $-\infty$ sur $]-\infty;3[$ puisque $f''(x)<0$ sur $]-\infty;3[$. Il existe donc α unique dans $]-\infty;3[$ tel que $f'(\alpha)=0$.
On tape pour avoir une valeur approchée de α :

```
assume(x<3);fsolve(f1(x),x)
```

On obtient :

```
x,0.776592890991
```

Puis, on tape pour enlever l'hypothèse sur x :

```
purge(x)
```

On tape :

```
f1(0.7)
```

On obtient :

```
0.0937786881525
```

On tape :

```
f1(0.8)
```

On obtient :

```
-0.0297244578175
```

On a $f1(0.7)>0$ et $f1(0.8)<0$ donc $0.7<\alpha<0;8$.

4. Puisque $f''(7)=0$, on tape pour avoir le minimum de f' sur $]3;+\infty[$:

```
f1(7)
```

On obtient :

```
ln(4)+2
```

Le minimum de f' sur $]3;+\infty[$ est donc positif.
Donc $f'(x)>0$ si $x\in]-\infty;\alpha[\cup]3;+\infty[$ et $f'(x)<0$ si $x\in]\alpha;3[$.
Donc f est croissante sur $]-\infty;\alpha[\cup]3;+\infty[$ et est décroissante sur $]\alpha;3[$.

5. On cherche les limites de f en $-\infty$, $+\infty$, et en 3.
On tape :

```
limit(f(x),x,-infinity)
```

On obtient :

```
-infinity
```

On tape :

```
limit(f(x),x,+infinity)
```

On obtient :

```
+infinity
```

On tape :

```
limit(f(x),x,3)
```

On obtient :

```
-infinity
```

On trace les graphes de f et des deux droites $x=-1$ et $x=2$, on tape :

```
plofunc(f(x),x);droite(x=1);droite(x=2)
```

On obtient le tracé du graphe de f et le tracé des droites $x=-1$ et $x=2$.

6. On tape pour trouver l'aire en cm^2 :

```
integrate(f(x),x,-1,2)
```

On obtient :

```
8*ln(4)-12+15/4
```

On tape :

```
normal(8*ln(4)-12+15/4))
```

On obtient :

```
8*ln(4)-33/4
```

On tape si on veut faire l'intégration par parties :

```
ibpu((x+1)*ln(abs(x-3)),ln(abs(x-3)))
```

On obtient :

```
[((x^2)/2+x)*ln(abs(x-3)),(-x^2-2*x)/(2*x-6)]
```

On tape pour terminer l'intégration :

```
A:=ibpu([((x^2)/2+x)*ln(abs(x-3)),(-x^2-2*x)/(2*x-6)],0)
```

On obtient :

```
(-x^2-10*x)/4-15*1/2*ln(abs(x-3))+((x^2)/2+x)*ln(abs(x-3))
```

On tape :

```
preval(A,-1,2)
```

On obtient :

```
8*ln(4)-9/4-6)
```

On tape :

```
normal(8*ln(4)-9/4-6))
```

On obtient :

```
8*ln(4)-33/4
```

Donc l'aire cherchée vaut $(8*\ln(4)-33/4)cm^2$;

7.1.2 Exercice 2

On considère la fonction f de $\mathbb{R}$ dans $\mathbb{R}$ définie par :

$$f(x)=\frac{\exp(x)^2-\exp(x)+1}{\exp(x)^3+\exp(x)}$$

1. Montrer que pour tout $x\in\mathbb{R}$, $P(x)=x^4-2x^3+2x^2+1\geq 1$
2. Étudier les variations de f et tracer son graphe.
3. Trouver l'équation de la tangente au graphe au point d'abscisse $x=0$
4. Calculer $\int_0^x f(t)dt$ puis, $\lim_{x->+\infty}\int_0^x f(t)dt$

Réponses

1. On tape :

```
factor(x^4-2x^3+2x^2)
```

On obtient :

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

On tape :

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

On obtient :

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

Donc pour tout $x\in\mathbb{R}$, $x^4-2x^3+2x^2=x^2*(x-1)^2+x^2\geq 0$
donc pour tout $x\in\mathbb{R}$, $P(x)=x^4-2x^3+2x^2+1\geq 1$

2. On tape pour calculer la valeur de la dérivée de f en un point :

```
normal(derive((exp(x)^2-exp(x)+1)/(exp(x)^3+exp(x)),x))
```

On obtient :

```
(-(exp(x))^4+2*(exp(x))^3-2*(exp(x))^2-1)/
    ((exp(x))^5+2*(exp(x))^3+exp(x))
```

Le numérateur est négatif car il est égal à $-P(\exp(x))$ et le dénominateur est strictement positif car il est égal à une somme de termes strictement positifs. La fonction f est donc décroissante.
Pour chercher la limite de f en $+\infty$, on tape :

```
limit((exp(x)^2-exp(x)+1)/(exp(x)^3+exp(x)),x=+infinity)
```

On obtient :

```
0
```

Pour chercher la limite de f en $-\infty$, on tape :

```
limit((exp(x)^2-exp(x)+1)/(exp(x)^3+exp(x)),x=-infinity)
```

On obtient :

```
infinity
```

Pour tracer le graphe de f, on tape :

```
plotfunc(((exp(x))^2-exp(x)+1)/((exp(x))^3+exp(x)),x)
```

On obtient le graphe de f.

3. On définit la fonction f, on tape :

```
f(x):=(exp(x)^2-exp(x)+1)/(exp(x)^3+exp(x))
```

On calcule $f(0)$, on tape :

```
f(0)
```

On obtient

$$\frac{1}{2}$$

On définit la fonction df comme étant la dérivée de f, on tape :

```
df:=unapply(normal(diff(f(x),x)),x)
```

On calcule $df(0)$, on tape :

```
df(0)
```

On obtient :

$$-\frac{1}{2}$$

L'équation de la tangente au point d'abscisse 0 est donc :
$y = df(0) * x + f(0)$ c'est à dire $y = -\frac{1}{2}x + \frac{1}{2}$.
ou encore on tape :

```
equation(tangent(plotfunc(f(x)),0),[x,y])
```

On obtient :

```
y=(1/-2*x+1/2)y=(1/-2*x+1/2)
```

4. On calcule l'intégrale : $\int_0^x f(t)dt$

   On tape :

   ```
   int(f(t),t,0,x)
   ```

   On obtient :

   ```
   (ln((exp(x))^2+1)*exp(x)+(-(2*x))*exp(x)+2*exp(x)-2)*
   1/2/exp(x)-1/2*ln(2)
   ```

   Puis on calcule : $\lim_{x->+\infty} \int_0^x f(t)dt$, on tape :

   ```
   limit((ln((exp(x))^2+1)*exp(x)+(-(2*x))*exp(x)+2*exp(x)-2)
   *1/2/exp(x)-1/2*ln(2),x=+infinity)
   ```

   On obtient :

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

7.2 Calcul de primitives (niveau début université)

1. Calculer

   $$\int_1^2 \frac{1}{x^3+1}dx$$

   Réponse :
   On tape :

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

   On obtient apres simplification (en utilisant `normal}`

   ```
   (sqrt(3)*ln(2)+pi)*1/3/sqrt(3)
   ```

   Pour vérifier, on tape :

   ```
   partfrac(1/(1+t^3))
   ```

   On obtient :

   ```
   1/((t+1)*3)+(-1/3*t+2/3)/(t^2-t+1)
   ```

   Puis on intègre chaque terme séparément...

2. Décomposer, sur $\mathbb{R}$, en éléments simples : $\frac{t^2}{1-t^4}$.

   Calculer $\int \frac{t^2}{1-t^4}dt$ et $\int \frac{\sin(x)^2}{\cos(2x)}dx$

   Réponse :
   On tape :

   ```
   partfrac(t^2/(1-t^4))
   ```

On obtient :

```
-1/2/(t^2+1)+1/(4*(t+1))-1/4/(t-1)
```

On tape :

```
int(-1/2/(t^2+1)+1/(4*(t+1))-1/4/(t-1),t)
```

ou on tape :

```
int(t^2/(1-t^4),t)
```

On obtient :

```
1/(-2*atan(t))+1/(4*ln(abs(t+1)))+1/(-4*ln(abs(t-1)))
```

On tape :

```
normal(int(sin(x)^2/cos(2*x),x))
```

On obtient :

```
-1/2*x-1/-4*ln(abs((tan(1/2*x))^2-2*tan(1/2*x)-1))-
    1/4*ln(abs((tan(1/2*x))^2+2*tan(1/2*x)-1))
```

Ou on tape en linéarisant avant d'intégrer :

```
normal(int(tlin(sin(x)^2/cos(2*x))))
```

On obtient :

```
1/4*ln(abs(tan(x)+1))+1/-4*ln(abs(tan(x)-1))+1/-2*x
```

Ou encore on veut faire le changement de variable $\tan(x) = t$ et on tape pour avoir l'expression en fonction de la tangente, avant d'intégrer :

```
trigtan(texpand(sin(x)^2/cos(2x)))
```

On obtient :

```
(-((tan(x))^2))/((tan(x))^2-1)
```

On fait le changement de variable $x = \text{atan}(t)$ on tape :

```
subst('integrate(-tan(x)^2/(tan(x)^2-1),x)',x=atan(t))
```

ou on tape

```
subst(Int(-tan(x)^2/(tan(x)^2-1),x),x=atan(t))
```

On obtient

```
integrate((-(t^2))/((1+t^2)*(t^2-1)),t)
```

Soit, le remplacant t par $\tan(x)$:

```
1/-2*atan(tan(x))+1/4*ln(abs(tan(x)+1))+1/-4*ln(abs(tan(x)-1))
```

3. Calculer $\int \frac{1}{t^2}dt$, $\int \frac{1}{t(t^2+1)}dt$, et $\int \frac{t^2-t+1}{t^4+t^2}dt$

   Réponse :
   On tape :

   ```
   int(1/t^2,t)
   ```

   On obtient :

   ```
   1/(-t)
   ```

   On tape :

   ```
   int(1/(t*(t^2+1)),t)
   ```

   On obtient :

   ```
   1/-2*ln(t^2+1)+1/2*ln((abs(t))^2)
   ```

   On tape :

   ```
   int((t^2-t+1)/(t^2+t^4),t)
   ```

   On obtient :

   ```
   1/2*ln(t^2+1)-ln(abs(t))+(-t+1)/(-t)
   ```

7.3 Dévelopements limités

1. Donner un développement limité à l'ordre 7 au voisinage de $x = 0$ de :

   $$\sin(\sinh(x)) - \sinh(\sin(x))$$

   Réponse :
   On tape :

   ```
   series(sin(sinh(x))-sinh(sin(x)),x=0,7)
   ```

   On obtient :

   ```
   1/-45*x^7+x^8*order_size(x)
   ```

2. Donner un développement limité à l'ordre 4 au voisinage de $x = 0$ de :

   $$\frac{\ln(\cos(x))}{\exp(x+x^2)}$$

   Réponse :
   On tape :

   ```
   series(ln(cos(x))/exp(x+x^2),x=0,4)
   ```

   On obtient :

   ```
   1/-2*x^2+1/2*x^3+1/6*x^4+x^5*order\_size(x)
   ```

   La fonction *order_size* est telle que, pour tout $\alpha > 0$, x^{α}*order_size*(x) tend vers 0 quand x tend vers 0

7.4 Équations différentielles

1. Trouver les solutions de l'équation différentielle :

$$x(x^2-1)y'+2y=0$$

Réponse :
On tape :

```
desolve(x*(x^2-1)*y'+2*y=0,y)
```

On obtient :

```
(c\_0*x^2)/(x^2-1)
```

2. Trouver les solutions de l'équation différentielle :

$$x(x^2-1)y'+2y=x^2$$

Réponse :
On tape :

```
desolve(x*(x^2-1)*y'+2*y=x^2,y)
```

On obtient :

```
((ln(abs(x))+c\_0)*x^2)/(x^2-1)
```

7.5 Les matrices

1. Soit $M_a = \begin{bmatrix} 2a-1 & a & 2a-1 \\ a^2+a-2 & a^2-1 & a-1 \\ a^2+a-1 & a^2+a-1 & a \end{bmatrix}$
a) Pour quelles valeurs de a, M_a est-elle inversible ?
Préciser son rang lorsqu'elle n'est pas inversible.
b) Calculer l'inverse de M_2

Réponse :
On tape :

```
M:=[[2a-1,a,2a-1],[a^2+a-2,a^2-1,a-1],[a^2+a-1,a^2+a-1,a]]
```

On calcule le déterminant de M, on tape :

```
det(M)
```

On obtient :

```
2*a^4+-2*a^3+-2*a^2+2*a
```

Pour avoir l'inverse de M on tape :

```
inv(M)
```

On obtient :

$$\frac{1}{2a^4-2a^3-2a^2+2a}\begin{bmatrix} a-1 & 2a^3+3a+1 & -2a^3+a^2+a-1 \\ -a^2+1 & -2a^3+a^2+2a-1 & 2a^3-a^2-2a+1 \\ a^3-2a+1 & -a^3+2a-1 & a^3-2a^2+1 \end{bmatrix}$$

On tape :

```
solve(2a^4-2*a^3-2*a^2+2*a,a)
```

On obtient :

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

Donc la matrice est inversible si $a \notin [-1,0,1]$
Ou on tape :

```
factor(2a^4-2*a^3-2*a^2+2*a)
```

On obtient :

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

On tape :

```
rank(subst(M,a,-1))
```

On obtient :

```
2
```

On tape :

```
rank(subst(M,a,0))
```

On obtient :

```
2
```

On tape :

```
rank(subst(M,a,1))
```

On obtient :

```
1
```

On tape :

```
inv(subst(M,a,2))
```

On obtient : $A = \frac{1}{12}\begin{bmatrix} 1 & 11 & -7 \\ -3 & -9 & 9 \\ 5 & -5 & 1 \end{bmatrix}$
Remarque : pour éviter de faire des substitutions on peut définir la matrice M comme une fonction de a, il faut alors écrire :

```
M(a):={[[2a-1,a,2a-1],[a^2+a-2,a^2-1,a-1],[a^2+a-1,a^2+a-1,a]]}
```

surtout ne pas oublier { et }.
On peut alors taper : `inv(M(2))`.

2. Soit $A = \begin{bmatrix} 1 & 1 & a \\ 1 & a & 1 \\ a & 1 & 1 \end{bmatrix}$
Pour quelles valeurs de a, A est-elle diagonalisable ?
Réponse :

On tape :

```
A :=[[1,1,a],[1,a,1],[a,1,1]]
```

Pour avoir les valeurs propres de A on tape :

```
egvl(A)
```

On obtient :

$$\begin{bmatrix} -a+1 & 0 & 0 \\ 0 & a+2 & 0 \\ 0 & 0 & a-1 \end{bmatrix}$$

ce qui s'écrit :

```
[[-a+1,0,0],[0,a+2,0],[0,0,a-1]]
```

Si $a \neq 1$ il y a 3 valeurs propres distinctes $-a+1, a+2, a-1$ et
si $a = 1$ il y a une valeur propre double ($\lambda = 0$) et une valeur propre simple ($\lambda = 3$).
Puis on cherche la matrice de passage, on tape :

```
egv(A)
```

On obtient :

$$\begin{bmatrix} -1 & 1 & 1 \\ 0 & 1 & -2 \\ -1 & 1 & 1 \end{bmatrix}$$

ce qui s'écrit :

```
[[1,1,1],[0,1,-2],[-1,1,1]]
```

les vecteurs propres sont les colonnes de cette matrice.
Ou on tape pour avoir directement les deux informations, matrice de passage et réduite de Jordan :

```
jordan(A)
```

On obtient une liste de deux matrices $[P,B]$ (P est la matrice de passage et $B = P^{-1}AP$) :

$$\left[\begin{bmatrix} -1 & 1 & 1 \\ 0 & 1 & -2 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -a+1 & 0 & 0 \\ 0 & a+2 & 0 \\ 0 & 0 & a-1 \end{bmatrix}\right]$$

ce qui s'écrit :

```
[[[1,1,1],[0,1,-2],[-1,1,1]],[[-a+1,0,0],[0,a+2,0],[0,0,a-1]]]
```

On remarque qu'en faisant : `a :=1` puis `jordan(A)`
les valeurs propres doubles sont regroupées et on obtient :

$$\left[\begin{bmatrix} 1 & -3 & 0 \\ 1 & 0 & -3 \\ 1 & 3 & 3 \end{bmatrix}\begin{bmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right]$$

ce qui s'écrit :

```
[[[1,-3,0],[1,0,-3],[1,3,3]],[[3,0,0],[0,0,0],[0,0,0]]]
```

A est donc diagonalisable quelque soit a et $B = P^{-1}AP$.

8 Vrai ou Faux ? (d'un point de vue informatique)

Exercice 8.1 Les commandes suivantes affichent la valeur exacte 2 : répondre vrai ou faux et pourquoi ?

1. ☐ `1+1:;`
2. ☒ `3-1`
3. ☐ `1.5+1/2`
4. ☒ `4/2`
5. ☒ `sqrt(4)`
6. ☐ `evalf(sqrt(4))`
7. ☒ `1^(1+1)+1^(1+1)`
8. ☐ `(1+1)^(1+1)`
9. ☐ `1*1^(1+1)`
10. ☒ `1+1*1^1`
11. ☒ `(1+1)*1^(1+1)`

Exercice 8.2 Les commandes suivantes affectent la valeur exacte 2 à la variable `c` : répondre vrai ou faux et pourquoi ?

1. ☒ `c:=2:;`
2. ☒ `c:=2`
3. ☐ `c==2`
4. ☐ `c=2`
5. ☒ `c:=4/2`
6. ☐ `c:=3/1.5`
7. ☒ `c:=(2+2)/2`
8. ☐ `c:=(2.0+2)/2`
9. ☐ `c:=2a/a`
10. ☐ `c:=(2*a)/a`
11. ☒ `c:=2*a/a`
12. ☒ `c:=1:; c:=2*c`

Exercice 8.3 Les commandes suivantes affectent à la variable `c` une expression valide : répondre vrai ou faux et pourquoi ?

1. ☒ `c:=ab`
2. ☒ `c:=a*b`
3. ☐ `c==a`
4. ☒ `c:= c==a`
5. ☐ `c:=a+(a*b))/2`

6. ☐ `c=a+a*b`
7. ☒ `c:=a/b`
8. ☐ `c->a/b`
9. ☒ `a/b=>c`
10. ☒ `c:=a/0`
11. ☒ `c:=2*a/a`
12. ☐ `c:=1: c:=2*c`

Exercice 8.4 Les commandes suivantes affectent la valeur 1 à `b` : répondre vrai ou faux et pourquoi ?

1. ☐ `a:=1:; b=a`
2. ☒ `a:=1:; b:=a`
3. ☐ `a:=1:; b:='a':; a:=3:; b`
4. ☐ `a:=1:; b:="a"`
5. ☒ `b:=a/a`
6. ☒ `b:=a^0`

Exercice 8.5 Les commandes suivantes retournent la valeur exacte 2 : vrai ou faux et pourquoi ?

1. ☒ `1/2^-1`
2. ☒ `a:=2`
3. ☒ `2*a/a`
4. ☐ `sqrt(4*a^2)/a`
5. ☐ `simplify(sqrt(4*a^2)/a)`
6. ☐ `sqrt(4*a^4)/(a*a)`
7. ☒ `simplify(sqrt(4*a^4)/(a*a))`
8. ☐ `expand(sqrt(4*a^4)/(a*a))`
9. ☒ `normal(sqrt(4*a^4)/(a*a))`
10. ☐ `ln(a^2)/ln(a)`
11. ☒ `simplify(ln(a^2)/ln(a))`
12. ☐ `texpand(ln(a^2)/ln(a))`
13. ☒ `normal(texpand(ln(a^2)/ln(a)))`
14. ☒ `-ln(exp(-2))`
15. ☐ `1/exp(-ln(2))`
16. ☒ `exp2pow(1/exp(-ln(2)))`

Exercice 8.6 Les commandes suivantes définissent la fonction f qui à x associe x^2 : vrai ou faux et pourquoi ?

1. ☒ `f(x):=x^2`
2. ☒ `f(a):=a^2`
3. ☐ `f := x^2`
4. ☐ `f(x):=a^2`
5. ☒ `f := a->a^2`
6. ☐ `f(x):=evalf(x^2)`
7. ☐ `f(x):=simplify(x^3/x)`
8. ☐ `f(x):=simplify(x*x*a/a)`
9. ☒ `E:=x^2:;f:=unapply(E,x)`
10. ☒ `f:=unapply(simplify(x^3/x),x)`

Exercice 8.7 Les commandes suivantes définissent la fonction `f` qui au couple (x,y) associe le produit xy : vrai ou faux et pourquoi ?

1. ☐ `f:=x*y`
2. ☐ `f:=x->x*y`
3. ☒ `f:=(a,b)->a*b`
4. ☒ `f(x,y):=x*y`
5. ☐ `f(x,y):=xy`
6. ☒ `f:=((x,y)->x)*((x,y)->y)`
7. ☐ `f:=(x->x)*(y->y)`
8. ☒ `f:=unapply(x*y,x,y)`
9. ☒ `E:=x*y:;f:=unapply(E,x,y)`

Exercice 8.8 Les commandes suivantes définissent la fonction `f1` qui à x associe $2*x$: vrai ou faux et pourquoi ?

1. ☐ `f(x):=x^2:; f1(x):=diff(f(x))`
2. ☐ `f1:=diff(x^2)`
3. ☒ `f1:=unapply(diff(x^2),x)`
4. ☒ `f(x):=x^2:; f1:=function_diff(f)`
5. ☐ `f(x):=x^2:; f1:=diff(f)`
6. ☐ `f(x):=x^2:; f1:=diff(f(x))`
7. ☒ `f(x):=x^2:; f1:=unapply(diff(f(x),x),x)`
8. ☐ `f(x):=x^2:; f1:=x->diff(f(x))`

Exercice 8.9 Les commandes suivantes affectent à A l'expression $2*x*y$: vrai ou faux et pourquoi ?

1. ☒ `A:=diff(x^2*y)`
2. ☐ `A:=x->diff(x^2*y)`

3. ☒ `A:=diff(x^2*y,x)`
4. ☐ `A:=diff(x^2*y,y)`
5. ☐ `A:=diff(x*y^2,y)`
6. ☒ `A:=normal(diff(x*y^2,y))`
7. ☒ `A:=normal(diff(x^2*y^2/2,x,y))`
8. ☒ `A:=normal(diff(diff(x^2*y^2/2,x),y))`

Exercice 8.10 Les lignes de commande suivantes affichent un losange : vrai ou faux et pourquoi ?

1. ☒ `losange(1,i,pi/3)`
2. ☐ `losange((1,0),(0,1),pi/3)`
3. ☒ `losange(point(1,0),point(0,1),pi/3)`
4. ☐ `parallelogramme(0,1,1+i)`
5. ☒ `parallelogramme(0,1,1/2+i*sqrt(3)/2)`
6. ☒ `quadrilatere(0,1,3/2+i*sqrt(3)/2,1/2+i*sqrt(3)/2)`
7. ☒ `polygone(0,1,3/2+i*sqrt(3)/2,1/2+i*sqrt(3)/2)`
8. ☐ `polygonplot(0,1,3/2+i*sqrt(3)/2,1/2+i*sqrt(3)/2)`
9. ☐ `polygonplot([0,1,3/2,1/2],[0,0,sqrt(3)/2,sqrt(3)/2])`
10. ☐ `polygone_ouvert(0,1,3/2+i*sqrt(3)/2,1/2+i*sqrt(3)/2)`
11. ☒ `polygone_ouvert(0,1,3/2+i*sqrt(3)/2,1/2+i*sqrt(3)/2,0)`

Exercice 8.11 Les lignes de commande suivantes affichent le cercle unité : vrai ou faux et pourquoi ?

1. ☒ `cercle(0,1)`
2. ☐ `arc(-1,1,2*pi)`
3. ☒ `arc(-1,1,pi), arc(-1,1,-pi)`
4. ☐ `plot(sqrt(1-x^2))`
5. ☒ `plot(sqrt(1-x^2)), plot(-sqrt(1-x^2))`
6. ☒ `plotimplicit(x^2+y^2-1,x,y)`
7. ☐ `plotparam(cos(t),sin(t))`
8. ☒ `plotparam(cos(t)+i*sin(t))`
9. ☒ `plotparam(cos(t)+i*sin(t),t)`
10. ☒ `plotparam(exp(i*t))`
11. ☐ `plotparam(cos(t)+i*sin(t),t,0,pi)`
12. ☒ `plotparam(cos(t)+i*sin(t),t,0,2*pi)`
13. ☒ `plotpolar(1,t)`
14. ☒ `plotpolar(1,t,-pi,pi)`

15. ☒ `plotpolar(1,t,0,2*pi)`

Exercice 8.12 Les commandes suivantes retournent la liste $[1,2,3,4,5]$: vrai ou faux et pourquoi ?

1. ☒ `l:=[1,2,3,4,5]`
2. ☐ `l:=op([1,2,3,4,5])`
3. ☒ `l:=nop(1,2,3,4,5)`
4. ☐ `l:=seq(i,i=1..5)`
5. ☐ `l:=seq(j=1..5)`
6. ☐ `l:=seq(j,j=1..5)`
7. ☐ `l:=seq(j,j,1..5)`
8. ☒ `l:=seq(j,j,1,5)`
9. ☒ `l:=seq(j,j,1,5,1)`
10. ☒ `l:=[seq(j,j=1..5)]`
11. ☒ `l:=nop(seq(j,j=1..5))`
12. ☐ `l:=[k$k=1..5]`
13. ☒ `l:=[k$(k=1..5)]`
14. ☐ `l:=[k+1$(k=0..4)]`
15. ☒ `l:=[(k+1)$(k=0..4)]`
16. ☒ `l:=cumSum([1$5])`
17. ☐ `l:=sort(5,2,3,1,4)`
18. ☒ `l:=sort([5,2,3,1,4])`
19. ☐ `l:=makelist(k,1,5)`
20. ☒ `l:=makelist(x->x,1,5)`

Exercice 8.13 Les commandes suivantes retournent la liste $[1.0, 0.5, 0.25, 0.125, 0.0625]$: vrai ou faux et pourquoi ?

1. ☒ `0.5^[0,1,2,3,4]`
2. ☐ `2^(-[0,1,2,3,4])`
3. ☒ `2.0^(-[0,1,2,3,4])`
4. ☒ `2^-evalf([0,1,2,3,4])`
5. ☒ `evalf(2^(-[0,1,2,3,4]))`
6. ☐ `seq(2^(-n),n=0..4)`
7. ☒ `evalf([seq(2^(-n),n=0..4)])`
8. ☐ `1/evalf(2^n$(n=0..4))`
9. ☐ `evalf(2^n$(n=0..4))^(-1)`
10. ☒ `[evalf(2^n$(n=0..4))]^(-1)`

11. ☒ `evalf(nop(2^n$(n=0..4))^(-1))`
12. ☒ `a:=[]:; (a:=append(a,0.5^k))$(k=0..4):; a`
13. ☐ `makelist(k->2^(-k),0,4)`
14. ☒ `f:=x->2.0^(-x):; makelist(f,0,4)`

Exercice 8.14 Soit l la liste $[1,0,2,0,3]$. Les lignes de commande suivantes retournent l'entier 10203 : vrai ou faux et pourquoi ?

1. ☒ `l*10^[4,3,2,1,0]`
2. ☐ `l*10^[0,1,2,3,4]`
3. ☒ `revlist(l)*10^[0,1,2,3,4]`
4. ☒ `l*seq(10^n,n,4,0,-1)`
5. ☒ `expr(char(sum(l,48)))`
6. ☒ `l*nop(seq(10^n,n=(4..0)))`
7. ☒ `l*10^nop(j$(j=4..0))`
8. ☐ `l*10^(j$(j=4..0))`
9. ☐ `l*10^(j$(j=4..0))`
10. ☒ `l*nop(10^j)$(j=4..0))`

Exercice 8.15 Soit n l'entier 10203. Les lignes de commande suivantes retournent la liste d'entiers $[1,0,2,0,3]$: vrai ou faux et pourquoi ?

1. ☐ `(floor(n/10^k)-floor(n/10^(k+1))*10)$(k=4..0)`
2. ☒ `[(floor(n/10^k)-floor(n/10^(k+1))*10)$(k=4..0)]`
3. ☐ `seq(iquo(n,10^k)-10*iquo(n,10^(k+1)),k=4..0)`
4. ☒ `nop(seq(iquo(n,10^k)-10*iquo(n,10^(k+1)),k=4..0))`
5. ☒ `revlist(convert(n,base,10))`
6. ☒ `sum(asc(string(n)),-48)`
7. ☐ `string(n)`
8. ☐ `mid(string(n),k,1)$(k=0..4)`
9. ☐ `[mid(string(n),k,1)$(k=0..4)]`
10. ☒ `[expr(mid(string(n),k,1))$(k=0..4)]`

Exercice 8.16 Le polynôme $P = X^4+2X^2+3$ a été affecté par la commande `P:=X^4+2*X^2+3`. Les lignes de commande suivantes affichent le polynôme réciproque `3*X^4+2*X^2+1` : vrai ou faux et pourquoi ?

1. ☒ `poly2symb(revlist(symb2poly(P)))`
2. ☐ `X^4*subst(P,X,1/X)`
3. ☒ `normal(X^4*subst(P,X,1/X))`
4. ☐ `normal(subst(P,X,1/X))`
5. ☒ `normal(subst(P/X^4,X,1/X))`

6. ☒ `normal(X^degree(P)*subst(P,X,1/X))`
7. ☒ `getNum(subst(P,X,1/X))`
8. ☐ `f:=unapply(P,X):; part(f(1/X),1)`
9. ☒ `f:=unapply(P,X):; part(normal(f(1/X)),1)`

9 Exercices (niveau université)

Il y a souvent plusieurs manières d'obtenir le même résultat en Xcas. *On s'efforcera de choisir les solutions les plus compactes.*

Exercice 9.1 Vérifier les identités suivantes.

1. $(2^{1/3}+4^{1/3})^3-6(2^{1/3}+4^{1/3})=6$
2. $\pi/4=4\arctan(1/5)-\arctan(1/239)$
3. $\sin(5x)=5\sin(x)-20\sin^3(x)+15\sin^5(x)$
4. $(\tan(x)+\tan(y))\cos(x)\cos(y)=\sin(x+y)$
5. $\cos^6(x)+\sin^6(x)=1-3\sin^2(x)\cos^2(x)$
6. $\ln(\tan(x/2+\pi/4))=\arg\sinh(\tan(x))$

Exercice 9.2 Transformer la fraction rationnelle

$$\frac{x^4+x^3-4x^2-4x}{x^4+x^3-x^2-x}$$

en les fractions suivantes

$$\frac{(x+2)(x+1)(x-2)}{x^3+x^2-x-1}\,,\quad \frac{x^4+x^3-4x^2-4x}{x(x-1)(x+1)^2}\,,\quad \frac{(x+2)(x-2)}{(x-1)(x+1)}\,,$$

$$\frac{x^2}{(x-1)(x+1)}-4\frac{1}{(x-1)(x+1)}\,.$$

Exercice 9.3 Transformer la fraction rationnelle

$$2\frac{x^3-yx^2-yx+y^2}{x^3-yx^2-x+y}$$

en les fractions suivantes

$$2\frac{x^2-y}{x^2-1}\,,\quad 2\frac{x^2-y}{(x-1)(x+1)}\,,$$

$$2-\frac{y-1}{x-1}+\frac{y-1}{x+1}\,,\quad 2-2\frac{y-1}{x^2-1}\,.$$

Exercice 9.4 On considère les fonctions f définies par

$$f(x) = \sqrt{e^x - 1}\,, \quad f(x) = \frac{1}{x\sqrt{1+x^2}}\,,$$

$$f(x) = \frac{1}{1+\sin(x)+\cos(x)}\,, \quad f(x) = \frac{\ln(x)}{x(x^2+1)^2}\,.$$

Pour chacune de ces fonctions :

1. Calculer une primitive F.
2. Calculer $F'(x)$ et montrer que $F'(x) = f(x)$ après simplifications.

Exercice 9.5 On considère les intégrales définies $I = \int_a^b f(x)\,dx$ suivantes.

$$\int_{-2}^{-1} \frac{1}{x}\,dx\,, \int_0^1 x\arctan(x)\,dx\,,$$

$$\int_0^{\pi/2} \sqrt{\cos(x)}\,dx\,, \int_0^{\pi/2} x^4\sin(x)\cos(x)\,dx\,.$$

Pour chacune de ces intégrales :

1. Calculer la valeur exacte, puis approchée de l'intégrale I.
2. Pour $n = 100$, puis $n = 1000$, et pour tout $j = 0,\ldots,n$, on pose $x_j = a + j(b-a)/n$, et $y_j = f(x_j)$. Calculer la valeur approchée de l'intégrale I par la méthode des rectangles à gauche :

$$I_r = \sum_{j=0}^{n-1} f(x_j)(x_{j+1} - x_j)\,.$$

3. Même question avec la méthode des trapèzes :

$$I_t = \sum_{j=0}^{n-1} \frac{1}{2}(f(x_j) + f(x_{j+1}))(x_{j+1} - x_j)\,.$$

Exercice 9.6 On considère la fonction f qui au couple (x,y) associe $f(x,y) = \cos(xy)$.

1. On pose $x_0 = y_0 = \pi/4$. Définir la fonction qui à (u,v,t) associe

$$f(x_0 + ut, y_0 + vt)\,.$$

2. Définir la fonction g qui à t associe la dérivée partielle par rapport à t de la fonction précédente (dérivée directionnelle).
3. Calculer le gradient de la fonction f au point (x_0,y_0), puis le produit scalaire de ce gradient avec le vecteur (u,v). Donner ce résultat en fonction de g

Exercice 9.7 On considère l'équation $x^3 - (a-1)x^2 + a^2x - a^3 = 0$ comme une équation en x.

1. Représenter graphiquement la solution x en fonction de a à l'aide de la fonction `plotimplicit`.
2. Calculer les trois solutions de l'équation, en utilisant `rootof` pour la première, en éliminant la première avec `quo` et en trouvant les deux dernières solutions en résolvant l'équation du second degré (utiliser `coeff` pour calculer le discriminant de l'équation).
3. Représenter graphiquement chacune des trois racines sur le même graphique avec une couleur différente, et pour les valeurs de a telles que ces solutions soient réelles (on pourra utiliser `resultant` pour trouver les valeurs de a pour lesquelles l'équation possède une racine multiple en x, ces valeurs sont les bornes possibles des intervalles en a où chacune des racines sont réelles).
4. Donner la valeur des solutions pour $a = 0, 1, 2$.

Exercice 9.8 On considère les limites suivantes.

$$\lim_{x\to 0} \frac{\sin(x)}{x},\ \lim_{x\to 0^+} (\sin(x))^{1/x},\ \lim_{x\to+\infty} (1+1/x)^x,\ \lim_{x\to+\infty} (2^x+3^x)^{1/x}$$

Pour chacune d'entre elles :

1. Donner sa valeur exacte.
2. Trouver une valeur de x telle que la distance de $f(x)$ à la limite soit inférieure à 10^{-3}.

Exercice 9.9 Représenter les fonctions f suivantes, en choisissant l'intervalle des abscisses et des ordonnées, de façon à obtenir la représentation la plus informative possible.

1. $f(x) = 1/x$.
2. $f(x) = e^x$.
3. $f(x) = 1/\sin(x)$.
4. $f(x) = x/\sin(x)$.
5. $f(x) = \sin(x)/x$.

Exercice 9.10 On considère la fonction $f(x) = 3x^2 + 1 + \frac{1}{\pi^4}\ln((\pi - x)^2)$.

1. Vérifier que cette fonction prend des valeurs négatives sur $\mathbb{R}^+$. Représenter la fonction sur l'intervalle $[0, 5]$.
2. Déterminer $\varepsilon > 0$ tel que `Xcas` donne une représentation correcte de la fonction sur l'intervalle $[\pi - \varepsilon, \pi + \varepsilon]$.

Exercice 9.11

1. Représenter la fonction $\exp(x)$ sur l'intervalle $[-1, 1]$. Sur ce graphique, tracer aussi les représentations des polynômes de Taylor de cette fonction en $x = 0$, aux ordres $1, 2, 3, 4$.
2. Même question pour l'intervalle $[1, 2]$.

3. Représenter la fonction $\sin(x)$ sur l'intervalle $[-\pi,\pi]$. Sur le même graphique, superposer les représentations des polynômes de Taylor de cette fonction en $x = 0$, aux ordres $1,3,5$.

Exercice 9.12 Superposer les représentations suivantes sur le même graphique, allant de 0 à 1 en abscisse et en ordonnée.

1. La première bissectrice ($y = x$).
2. Le graphe de la fonction $f : x \mapsto 1/6 + x/3 + x^2/2$.
3. La tangente au graphe de la fonction f au point $x = 1$.
4. Un segment vertical allant de l'axe des x au point d'intersection de la fonction f et de la première bissectrice, et un segment horizontal allant de ce point d'intersection à l'axe des y.
5. Les indications "point fixe" et "tangente", positionnées sur le graphique comme chaînes de caractères.

Exercice 9.13 Le but de l'exercice est de représenter sur un même graphique des familles de fonctions. On choisira le nombre de courbes, l'intervalle de représentation, les échelles en x et y ainsi que le pas de discrétisation des abscisses, de façon à obtenir la représentation la plus informative possible.

1. Fonctions $f_a(x) = x^a e^{-x}$, pour a allant de -1 à 1.
2. Fonctions $f_a(x) = 1/(x-a)^2$, pour a allant de -1 à 1.
3. Fonctions $f_a(x) = \sin(ax)$, pour a allant de 0 à 2.

Exercice 9.14 Pour chacune des courbes paramétrées suivantes, on choisira un intervalle de valeurs du paramètre assurant une représentation complète et suffisamment lisse.

1. $$\begin{cases} x(t) &= \sin(t) \\ y(t) &= \cos^3(t) \end{cases}$$
2. $$\begin{cases} x(t) &= \sin(4t) \\ y(t) &= \cos^3(6t) \end{cases}$$
3. $$\begin{cases} x(t) &= \sin(132t) \\ y(t) &= \cos^3(126t) \end{cases}$$

Exercice 9.15 Le but de l'exercice est de visualiser de différentes manières la surface définie par $z = f(x,y) = xy^2$. Ouvrir une fenêtre de géométrie 3-d.

1. Choisir un domaine de représentation et les pas de discrétisation, de manière à obtenir une représentation informative avec `plotfunc`.
2. Créer un paramètre a modifiable à la souris avec la fonction `assume`. Représenter la courbe définie par $z = f(a,y)$, puis faites varier le paramètre à la souris.

3. Créer un paramètre b modifiable à la souris. Représenter la courbe définie par $z = f(x,b)$, puis faites varier le paramètre à la souris.

Exercice 9.16 Le but de l'exercice est de visualiser un cône de différentes manières.

1. Représenter la surface d'équation $z = 1 - \sqrt{x^2+y^2}$.
2. Représenter la surface paramétrée définie par :

$$\begin{cases} x(u,v) &= u\cos(v) \\ y(u,v) &= u\sin(v) \\ z(u,v) &= 1-u\,. \end{cases}$$

3. En choisisant une valeur de a suffisamment grande, représenter la courbe paramétrée définie par :

$$\begin{cases} x(t) &= t\cos(at) \\ y(t) &= t\sin(at) \\ z(t) &= 1-t\,. \end{cases}$$

4. Représenter la famille de courbes paramétrées définies par :

$$\begin{cases} x(t) &= a\cos(t) \\ y(t) &= a\sin(t) \\ z(t) &= 1-a\,. \end{cases}$$

5. Représenter le même cône en utilisant la fonction `cone`.

Exercice 9.17

1. Engendrer une liste l de 100 entiers au hasard entre 1 et 9.
2. Vérifier que l'ensemble des valeurs de l est contenu dans $\{1,\ldots,9\}$.
3. Extraire de la liste l toutes les valeurs ≥ 5.
4. Pour tout $k = 1,\ldots,9$, compter combien de valeurs de la liste l sont égales à k.

Exercice 9.18 Si x est un réel, la fraction continue à l'ordre n de x est une liste $[a_0,\ldots,a_n]$ d'entiers, dont le premier terme a_0 est la partie entière de x. Pour tout $n \geq 0$, a_n est la partie entière de l'inverse de la partie décimale de a_{n-1}. La liste $[a_0,\ldots,a_n]$ est associée au rationnel

$$u_n = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}$$

Pour $x \in \{\pi, \sqrt{2}, e\}$ et $n \in \{5, 10\}$:

1. Calculer $[a_0,\ldots,a_n]$.
2. Comparer votre résultat avec celui que donne la fonction `dfc` de `Xcas`.
3. Calculer u_n, et donner la valeur numérique de $x - u_n$.

Exercice 9.19 Ecrire (sans utiliser de boucle) les séquences suivantes :

1. Nombres de 1 à 3 par pas de 0.1.
2. Nombres de 3 à 1 par pas de -0.1.
3. Carrés des 10 premiers entiers.
4. Nombres de la forme $(-1)^n n^2$ pour $n = 1, \ldots, 10$.
5. 10 "0" suivis de 10 "1".
6. 3 "0" suivis de 3 "1", suivis de 3 "2",..., suivis de 3 "9".
7. "1", suivi de 1 "0", suivi de "2", suivi de 2 "0",..., suivi de "8", suivi de 8 zéros, suivi de "9".
8. 1 "1" suivi de 2 "2", suivis de 3 "3",..., suivis de 9 "9".

Exercice 9.20

1. Définir les polynômes de degré 6 suivants.
   (a) polynôme dont les racines sont les entiers de 1 à 6.
   (b) polynôme dont les racines sont 0 (racine triple), 1 (racine double) et 2 (racine simple).
   (c) polynôme $(x^2-1)^3$.
   (d) polynôme x^6-1.
2. Ecrire (sans utiliser la fonction `companion`) la matrice compagnon A associée à chacun de ces polynômes. On rappelle que la matrice compagnon associée au polynôme :
$$P = x^d + a_{d-1}x^{d-1} + \cdots + a_1 x + a_0 \ ,$$
est :
$$A = \begin{pmatrix} 0 & 1 & 0 & \ldots & & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & & & & \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \ldots & & \ldots & 0 & 1 \\ -a_0 & -a_1 & & \ldots & & -a_{d-1} \end{pmatrix} .$$
3. Calculer les valeurs propres de la matrice A.
4. Calculer le polynôme caractéristique de A.

Exercice 9.21

1. Ecrire la matrice carrée A d'ordre 4, telle que $a_{j,k} = a$ si $j = k$ et $a_{j,k} = b$ si $j \neq k$, où a et b sont des variables.
2. Calculer et factoriser le polynôme caractéristique de A.
3. Déterminer une matrice orthogonale P telle que ${}^tPAP$ soit une matrice diagonale.
4. Utiliser la question précédente pour définir la fonction qui à un entier n associe la matrice A^n.

5. Calculer A^k, pour $k = 1, \ldots, 6$ en effectuant les produits matriciels, et vérifier que la fonction définie à la question précédente donne bien le même résultat.

Exercice 9.22

1. Ecrire la matrice carrée N d'ordre 6, telle que $n_{j,k} = 1$ si $k = j+1$ et $n_{j,k} = 0$ si $k \neq j+1$.
2. Calculer N^p, pour $p = 1, \ldots, 6$.
3. Ecrire la matrice $A = xI + N$, où x est une variable.
4. Calculer A^p, pour $p = 1, \ldots, 6$.
5. Calculer $\exp(At)$ en fonction de x et t :

$$\exp(At) = I + \sum_{p=1}^{\infty} \frac{t^p}{p!} A^p \ .$$

Exercice 9.23 Ecrire les fonctions suivantes, sans utiliser de boucle.

1. La fonction f prend en entrée un entier n et deux réels a, b et retourne la matrice A dont les termes diagonaux valent a, tous les autres termes étant égaux à b.
2. La fonction g prend en entrée un entier n et trois réels a, b, c et retourne la matrice $A = (a_{j,k})_{j,k=1,\ldots,n}$ dont les termes diagonaux sont égaux à a, les termes $a_{j,j+1}$ égaux à b et termes $a_{j+1,j}$ égaux à c, pour $j = 1, \ldots, n-1$ (les autres termes sont nuls).
3. La fonction H prend en entrée un entier n et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\ldots,n}$ définie par $a_{j,k} = 1/(j+k+1)$ (matrice de Hilbert). Comparer le temps d'exécution de votre fonction avec celui de la fonction `hilbert`
4. La fonction V prend en entrée un vecteur $x = (x_j)_{j=1,\ldots,n}$ et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\ldots,n}$ définie par $a_{j,k} = x_k^{j-1}$ (matrice de Vandermonde). Comparer le temps d'exécution de votre fonction avec celui de la fonction `vandermonde`
5. La fonction T prend en entrée un vecteur $x = (x_j)_{j=1,\ldots,n}$ et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\ldots,n}$ définie par $a_{j,k} = x_{|j-k|+1}$ (matrice de Toeplitz).

Exercice 9.24 Ecrire les fonctions suivantes. Toutes prennent en entrée une fonction f (de $\mathbb{R}$ dans $\mathbb{R}$), et trois valeurs x_{min}, x_0 et x_{max} (supposées telles que $x_{min} \leq x_0 \leq x_{max}$).

1. `derive` : Elle calcule et représente graphiquement la dérivée de f sur l'intervalle $[x_{min}, x_{max}]$. Elle retourne la une valeur de $f'(x_0)$.
2. `tangente` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$, elle superpose sur le même graphique la tangente au graphe de f au point x_0, et retourne l'équation de cette tangente comme un polynôme du premier degré.
3. `araignee` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$, ainsi que la droite d'équation $y = x$ (première bissectrice). Elle calcule et retourne les 10 premiers itérés de f en x_0 ($x_1 = f(x_0), x_2 = f \circ f(x_0), \ldots$). Elle représente la suite de segments, alternativement verticaux et horizontaux, permettant de visualiser les itérations : segments joignant $(x_0, 0)$, (x_0, x_1), (x_1, x_1), (x_1, x_2), (x_2, x_2), ...(comparer avec la fonction `plotseq`)

4. `newton_graph` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$. Elle calcule et retourne les dix premiers itérés de la suite définie à partir de x_0 par la méthode de Newton : $x_1 = x_0 - f(x_0)/f'(x_0)$, $x_2 = x_1 - f(x_1)/f'(x_1)\ldots$ Les valeurs de la dérivée sont approchées. La fonction représente sur le même graphique les segments permettant de visualiser les itérations : segments joignant $(x_0,0)$, $(x_0, f(x_0))$, $(x_1,0)$, $(x_1, f(x_1))$, $(x_2,0)$, $(x_2, f(x_2))$,... (comparer avec la fonction `newton`)

Exercice 9.25 On note D le carré unité : $D =]0,1[^2$. Soit Φ l'application définie sur D par

$$\Phi(x,y) = (z(x,y),t(x,y)) = \left(\frac{x}{1+y}, \frac{y}{1+x}\right) .$$

1. Calculer l'inverse de l'application Φ.
2. Déterminer et représenter graphiquement l'image par Φ du domaine D : $\Delta = \Phi(D)$.
3. Soit $A(x,y)$ la matrice jacobienne de Φ en un point (x,y) de D, et $B(z,t)$ la matrice jacobienne de Φ^{-1} en un point (x,y) de Δ. Calculer ces deux matrices, vérifier que $B(\Phi(x,y))$ et $A(x,y)$ sont inverses l'une de l'autre.
4. Soit $J(z,t)$ le déterminant de la matrice B. Calculer et simplifier $J(z,t)$.
5. Calculer
$$I_1 = \iint_D \left(\frac{1+x+y}{(1+x)(1+y)}\right)^3 dxdy .$$
6. Calculer
$$I_2 = \iint_\Delta (1+z)(1+t)\, dzdt ,$$
et vérifier que $I_1 = I_2$.

A Table des matières et index

Table des matières

Index