

Démarrer en Xcas

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 de la calculatrice TI89. 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 première année d'université), 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, en exécutant les commandes proposées une par une pour en comprendre l'effet.

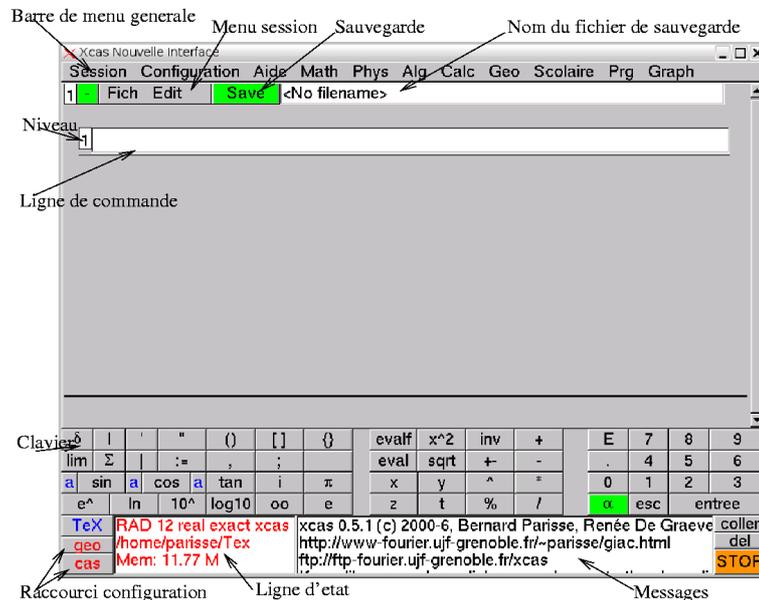
Table des matières

1	Pour commencer	3
1.1	Interface	3
1.2	Aide en ligne	4
1.3	Entrer des commandes	4
2	Les objets du calcul formel	5
2.1	Les nombres	5
2.2	Les variables	6
2.3	Les expressions	7
2.4	Développer et simplifier	7
2.5	Les fonctions	8
2.6	Listes, séquences, ensembles	10
2.7	Les chaînes de caractères	11
2.8	Temps de calcul, place mémoire	12
3	Outils pour l'Analyse	12
3.1	Dérivées	12
3.2	Limites et développements limités	13
3.3	Primitives et intégrales	14
3.4	Résolution d'équations	15
3.5	Equations différentielles	16
4	Outils pour l'Algèbre	16
4.1	Arithmétique des entiers	16
4.2	Polynômes et fractions rationnelles	17
4.3	Trigonométrie	18
4.4	Vecteurs et matrices	19
4.5	Systèmes linéaires	20
4.6	Réduction des matrices	22
5	Représentations graphiques	22
5.1	Tracés de courbes	22
5.2	Objets graphiques 2-d	23
5.3	Objets graphiques 3-d	24
6	Programmation	25
6.1	Le langage	25
6.2	Quelques exemples	27
6.3	Style de programmation	28
7	Vrai ou Faux ?	29
8	Exercices	34

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 menus gris cliquables : Session, Configuration, Help, Math, Phys,...
- une zone de gestion de la session avec les menus File, Edit, un bouton Save, et une zone affichant un nom de fichier de sauvegarde.
- une zone rectangulaire blanche numérotée 1 dans laquelle vous pouvez taper votre première commande : cliquez d'abord dans cette zone puis tapez $3+5$, suivi de la touche "Entrée" ("Enter" ou "Return" selon les claviers). Le résultat apparaît au-dessous, et une nouvelle fenêtre s'ouvre, numérotée 2.

Vous pouvez modifier l'aspect de l'interface et sauvegarder vos modifications pour les utilisations futures. En particulier, dans le menu Configuration, vous pouvez choisir de faire apparaître un clavier (Keyboard) ressemblant à celui d'une calculatrice, qui peut faciliter vos saisies.

Vous n'avez pour l'instant qu'à entrer des commandes dans les fenêtres successives. Si vous utilisez la version html de ce cours, vous pouvez copier-coller les commandes proposées. 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 modifier des commandes antérieures. Essayez par exemple de changer les commandes précédentes en :

```
1/3+3/4
sqrt(5)^2
solve(a*x+b*x+c,x)
500!
```

Le menu `Edit` vous permet de préparer des sessions plus élaborées qu'une simple succession de commandes. Vous pouvez créer des sections, grouper les commandes en niveaux et sous-niveaux, ajouter des commentaires ou fusionner des niveaux en un seul niveau.

1.2 Aide en ligne

Les commandes sont regroupées par thèmes dans les menus du bandeau gris supérieur : `Math`, `Phys`, `Alg`, `Calc`, `Geo`,... Lorsqu'on sélectionne une commande dans un menu, une aide succincte s'affiche dans la fenêtre blanche en bas à droite (double-cliquer pour afficher le message en entier), et le manuel s'ouvre dans votre navigateur à la bonne page.

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

Si vous connaissez le nom d'une commande et que vous désirez vérifier sa syntaxe (par exemple `solve`), vous pouvez saisir `?solve` pour avoir une aide en réponse. Si le nom que vous avez saisi n'est pas reconnu, des commandes proches vous sont suggérées.

Vous pouvez aussi taper le début du nom d'une commande puis la touche de tabulation (à gauche de la touche `A` sur un clavier français). Une fenêtre apparaît alors avec les complétions possibles et l'aide succincte. 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` puis `OK`.

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 :

- $a * 2 + b$ retourne $2a + b$
- $a / 2 * b$ retourne $\frac{1}{2} ab$
- $a / 2 / b$ retourne $\frac{a}{2b}$
- $a ^ 2 * b$ retourne $a^2 b$

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` est 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 d'environ 15 chiffres significatifs (précision relative de 53 bits pour les réels normalisés). Elle peut être changée, 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 $\text{infinity}(\infty)$, de $+\text{infinity}(+\infty)$ et de $-\text{infinity}(-\infty)$.

```
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	$+\infty$
-infinity	$-\infty$

2.2 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, redevienne 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 pour Vrai ou 0 pour 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

```

2.3 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 : $\cos(-x)=\cos(x)$, $\tan(\pi/4)=1$...

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

2.4 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 transcendentes (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 `Math` du bandeau supérieur, les 4 sous-menus de réécriture 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	
<code>simplify</code>	simplifier
<code>tsimplify</code>	simplifier (moins puissant)
<code>normal</code>	forme normale
<code>ratnormal</code>	forme normale (moins puissant)
<code>expand</code>	développer
<code>factor</code>	factoriser
<code>assume</code>	rajout d'hypothèses
<code>convert</code>	transformer en un format spécifié

2.5 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 `Math`.

Fonctions classiques	
abs	valeur absolue
round	arrondi
floor	partie entière (plus grand entier \leq)
ceil	plus petit entier \geq
abs	module
arg	argument
conj	conjugué
sqrt	racine carrée
exp	exponentielle
log	logarithme naturel
ln	logarithme naturel
log10	logarithme en base 10
sin	sinus
cos	cosinus
tan	tangente
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 $\text{subst}(E, x=2)$ 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 $f : x \mapsto E$. Par contre $E:=x^2-1; f:=\text{unapply}(E, x)$ définit bien la fonction $f : x \mapsto x^2 - 1$.

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
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}^1 par exemple $f(x, y) := (x + 2 * y, x - y)$

2.6 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 par `nop`. Le nombre d'éléments d'une liste est donné par `size`.

```
se:=(1,2,4,2)
li:=[se]
op(li)
nop(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 []. Il suffit d'écrire à la suite une séquence, puis une virgule, puis une autre séquence pour les concaténer. 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 %[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
size(li)	nombre d'éléments
sum	somme des éléments
product	produit des éléments
cumSum	sommes cumulées des éléments
apply(f,li)	appliquer une fonction à une liste
map(li,f)	appliquer une fonction à une liste
poly2symb	polynôme associé à une liste
symb2poly	coefficients d'un polynôme

2.7 Les chaînes de caractères

Une chaîne de caractères est encadrée par des guillemets ("). Un caractère est une chaîne ayant un seul élément. Il est possible de fabriquer des chaînes par extraction de parties et concaténation. Il est aussi possible de transformer une chaîne en expression et inversement.

```
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 de caractères	
size	nombre de caractères
concat ou +	concaténation
mid	morceau de chaîne
head	premier caractère
tail	chaîne sans le premier caractère
string	nombre ou expression transformé en chaîne
expr	chaîne transformée en nombre ou expression
asc	chaîne transformée en liste de codes ASCII
char	liste de codes ASCII transformée en chaîne

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 (en rouge à bas à gauche). 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 orange `stop` en bas 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)
```

Il ne faut pas définir la fonction dérivée par `f1(x):=diff(f(x))`, car `x` aurait dans cette définition 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, ce qui serait inefficace (dans la définition d'une fonction le membre de droite n'est pas évalué). Il faut donc soit utiliser `f1:=function_diff(f)`, soit `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	
<code>diff(ex)</code>	dérivée d'une expression
<code>function_diff(f)</code>	dérivée d'une fonction
<code>diff(ex,x\$n,y\$m)</code>	dérivées partielles
<code>grad</code>	gradient
<code>divergence</code>	divergence
<code>curl</code>	rotationnel
<code>laplacian</code>	laplacien
<code>hessian</code>	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 demandé est celui 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 sous la forme $x^a \text{order_size}(x)$, où $x^a \text{order_size}(x)$ est une fonction bornée. 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),x=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	
<code>limit(ex,x,a)</code>	limite en a
<code>limit(ex,x,a,1)</code>	limite à droite en a
<code>limit(ex,x,a,-1)</code>	limite à gauche en a
<code>taylor(ex,x=a)</code>	développement limité en a ordre 6
<code>series(ex,x=a,n)</code>	développement limité en a ordre n

3.3 Primitives et intégrales

La fonction `int` calcule une primitive d'une expression par rapport à x ou par rapport à la variable donnée en argument. Si l'expression comporte d'autres variables que x , il faut 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	
<code>int(E)</code>	primitive d'une expression
<code>int(E,x,a,b)</code>	intégrale exacte
<code>romberg(E,x,a,b)</code>	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 activer l'option `Complex` à partir du bouton rouge sur fond gris `cas` (en bas à gauche). 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	
<code>solve(eq,x)</code>	résolution exacte d'une équation
<code>solve([eq1,eq2],[x,y])</code>	résolution exacte d'un système
<code>fsolve(eq,x)</code>	résolution approchée d'une équation
<code>fsolve([eq1,eq2],[x,y])</code>	résolution approchée d'un système
<code>newton</code>	méthode de Newton
<code>linsolve</code>	système linéaire
<code>proot</code>	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_plotode` 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	
<code>desolve</code>	résolution exacte
<code>odesolve</code>	résolution approchée
<code>plotode</code>	tracé de trajectoire
<code>plotfield</code>	tracé d'un champ de vecteurs
<code>interactive_plotode</code>	interface cliquable

4 Outils pour l'Algèbre

4.1 Arithmétique des entiers

Les opérations sur les entiers figurent dans le menu `Math->Integer`. Les calculs modulo p se font en utilisant `%p`. Une fois défini un entier modulo p , disons $a := 3\%5$, tous les calculs seront effectués 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`.

```

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

```

Nombres entiers	
a%p	a modulo p
powermod(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 figurent dans le menu Alg->Polynomes.

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 réels. Pour les complexes, il faut activer l'option `Complex` à partir du bouton rouge `cas`. On peut aussi déclarer les coefficients comme des entiers modulo p pour travailler dans $\mathbb{Z}/p\mathbb{Z}$. Exécutez les commandes suivantes avant et après avoir activé l'option `Complex`.

```

P:=x^4-1
factor(P)
divis(P)
propfrac(x^4/P)
partfrac(4/P)
Q:=(x^4+1)%3
factor(Q)
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 coefficients
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 Math->Transcendental contient les fonctions circulaires et hyperboliques ainsi que leurs inverses. Pour linéariser et développer on utilise tlin et texpand. Beaucoup d'autres récritures sont accessibles à partir des menus

- Math->Recriture_trig : transformation des tangentes en sin et cos (tan2sincos), transformations en $\tan(x/2)$ (halftan),...

- `Math->Recriture_trig_exp`: transformation des fonctions trigonométriques en exponentielles par les formules d'Euler (`trig2exp`), des exponentielles en fonctions trigonométriques (`exp2trig`), des exponentielles en puissances (`exp2pow`)...
- `Math->Recriture_trig_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(E1)
tsimplify(E)
tsimplify(E1)
tsimplify(E-E1)
halftan(E)
trig2exp(E1)
Et:=trigtan(E)
tan2sincos(Et)
tan2sincos2(Et)
tan2cossin2(Et)
```

Trigonométrie	
<code>tlin</code>	linéariser
<code>tcollect</code>	linéariser et regrouper
<code>texpand</code>	forme polynomiale
<code>trig2exp</code>	trigonométrique vers exponentielle
<code>exp2trig</code>	exponentielle vers trigonométrique
<code>hyp2exp</code>	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 par une étoile. 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 $A = \begin{bmatrix} 0 & 2 \\ 1 & 3 \\ 2 & 4 \end{bmatrix}$ 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
<code>cross(v,w)</code>	produit vectoriel
$A*B$	produit matriciel
$A.*B$	produit terme à terme
$1/A$	inverse
<code>tran</code>	transposée
<code>rank</code>	rang
<code>det</code>	déterminant
<code>ker</code>	base du noyau
<code>image</code>	base de l'image
<code>idn</code>	matrice identité
<code>ranm</code>	matrice à coefficients aléatoires

4.5 Systèmes linéaires

Il existe trois moyens différents de résoudre un système linéaire.

- La fonction `linsolve` résout une liste d'équations linéaires, avec la même syntaxe que `solve`.
- La fonction `simult` peut résoudre plusieurs systèmes d'équations linéaires qui ne diffèrent que par leur second membre. Elle prend 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.
- La fonction `rref` calcule la réduction d'une matrice quelconque sous forme de Gauss-Jordan. Si on borde la matrice du système avec le second membre, `rref` retourne une matrice contenant le vecteur solution.

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	
<code>linsolve</code>	résolution d'un système
<code>simult</code>	résolution simultanée de plusieurs systèmes
<code>rref</code>	réduction de Gauss-Jordan
<code>rank</code>	rang
<code>det</code>	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)
log(A)
sin(A)
```

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

5 Représentations graphiques

5.1 Tracés de courbes

Chaque commande graphique crée un objet graphique, 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. En même temps que la fenêtre de réponse, l'objet graphique est affiché dans la fenêtre `DispG` (Display Graphics) que vous faites apparaître par le menu du bandeau supérieur `Session->Show->Show DispG` ou avec la commande `DispG()`. La différence est que les graphiques successifs sont tracés individuellement dans chaque fenêtre de réponse, et ils sont superposés dans la fenêtre `DispG`. Vous pouvez effacer la

fenêtre `DispG` par la commande `erase()`. Les paramètres par défaut (en particulier les intervalles de représentation en abscisse et ordonnée) peuvent être changés dans le menu accessible depuis le bouton `geo` rouge sur fond gris (en bas à gauche).

Pour afficher une courbe, on utilise l'instruction `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 `color=` 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,color=rouge);tangent(plot(E),0)
DispG()
plot([sin(x),x,x-x^3/6],x=-2..2,color=[rouge,bleu,vert])
erase
li:=[(x+k*0.5)^2$(k=-5..5)];
plot(li,x=-8..8,color=[k$(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)
erase
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	
<code>plot</code>	graphe d'une expression
<code>color</code>	paramètre de <code>plot</code> pour la couleur
<code>couleur</code>	choisir la couleur d'un tracé
<code>tangent</code>	tangente à une courbe
<code>plotparam</code>	courbe paramétrique
<code>plotpolar</code>	courbe en polaires
<code>plotimplicit</code>	courbe implicite

5.2 Objets graphiques 2-d

Xcas étant aussi un logiciel de géométrie, de nombreuses figures peuvent être tracées par des commandes du menu `Geo`, par exemple des polygones, des coniques... Les arguments de ces commandes sont des points (commande `point`) qui peuvent en général être saisis directement par leur

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

5.3 Objets graphiques 3-d

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 les plages de valeurs des variables, et leurs pas de discrétisation.

```
plotfunc(x^2-y^2, [x, y])
plotfunc(x+y^2, [x=-5..5, y=-2..2], xstep=0.5, ystep=0.1)
```

On obtient une surface en dimension 3. Pour modifier le point de vue, cliquer 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.

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 sont 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 tels que

```

plan(z=x+y)
droite(x=y,z=y)
A:=point(1,2,3); B:=point(2,-1,1); C:=point(1,0,0);
couleur(plan(A,B,C),cyan)
affichage(line_width_3)
droite(A,B)

```

Graphiques 3-d	
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	cône donné par centre, axe, angle d'ouverture

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, 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[1]:=2` crée une table `a`. Pour que `a` soit une liste, il faut d'abord affecter `a` avec une liste, et la modifier ensuite. Par exemple `a:=[0$10]` (si on connaît a priori la taille de la liste) ou `a:=[]` (affecter la liste vide). 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 des tests et des boucles est celle du langage C++. Pour les tests :

```
if (condition) {clause_vraie} else {clause_fausse}
```

(le `else` est facultatif). La condition est un booléen, résultat d'une expression logique, utilisant les opérateurs habituels.

Opérateurs logiques			
==	=	!=	non
<	<	>	>
<=	≤	>=	≥
&&	et		ou

Par exemple :

```
testif(a,b):={  
  if ( (a==10) || (a<b) )  
    b:=b-a;  
  else  
    a:=a-b;  
  return [a,b];  
};
```

La syntaxe de `for` est la suivante.

```
for(initialisation;test;incrementation){ corps_de_boucle }
```

Par exemple :

```
testfor(a,b):={  
  local j,(s:=0);  
  for (j:=a;j<=b;j++)  
    s:=s+1/j^2;  
  return s;  
};
```

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 `Add->Programme`. Le menu `Prog` 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(a,b):={
  local q,r;
  if (b!=0) {
    q:=iquo(a,b);
    r:=irem(a,b);
  }
  else {
    q:=0;
    r:=a;
  }
  return [q,r];
}
```

Saisissez cette fonction 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(a,b):={
  local r;
  while (b!=0) {
    r:=irem(a,b);
    a:=b;
    b:=r;
  }
  return a;
};
```

```
pgcd_recuratif(a,b):={
  if (b==0) return a;
  return pgcd_recuratif(b,irem(a,b));
};
```

Il arrive parfois qu'un programme ne fonctionne pas du premier coup comme prévu (!) Il est 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))`. On peut visualiser l'état des variables `a`, `b`, `r` avec le bouton `watch` et exécuter 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}
```

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++.

7 Vrai ou Faux ?

Exercice 7.1 Les commandes suivantes affichent la valeur exacte 2 : 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)`
12. `1^2/-1`
13. `1/2^-1`

Exercice 7.2 Les commandes suivantes affectent la valeur exacte 2 à la variable `c` : 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 7.3 Les commandes suivantes affectent à la variable `c` une expression valide : 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 7.4 Les commandes suivantes affectent la valeur 1 à b : vrai ou faux et pourquoi ?

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

Exercice 7.5 Les commandes suivantes retournent la valeur exacte 2 : vrai ou faux et pourquoi ?

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

Exercice 7.6 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. $f(x) := \text{evalf}(x^2)$
7. $f(x) := \text{simplify}(x^3/x)$
8. $f(x) := \text{simplify}(x*x*a/a)$
9. $E := x^2 ; f := \text{unapply}(E, x)$
10. $f := \text{unapply}(\text{simplify}(x^3/x), x)$

Exercice 7.7 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. $f := x \rightarrow x*y$
3. $f := (a, b) \rightarrow a*b$
4. $f(x, y) := x*y$
5. $f(x, y) := xy$
6. $f := ((x, y) \rightarrow x) * ((x, y) \rightarrow y)$
7. $f := (x \rightarrow x) * (y \rightarrow y)$
8. $f := \text{unapply}(x*y, x, y)$
9. $E := x*y ; f := \text{unapply}(E, x, y)$

Exercice 7.8 Les commandes suivantes définissent la fonction $f1$ qui à x associe $2*x$: vrai ou faux et pourquoi ?

1. $f(x) := x^2 ; f1(x) := \text{diff}(f(x))$
2. $f1 := \text{diff}(x^2)$
3. $f1 := \text{unapply}(\text{diff}(x^2), x)$
4. $f(x) := x^2 ; f1 := \text{function_diff}(f)$
5. $f(x) := x^2 ; f1 := \text{diff}(f)$
6. $f(x) := x^2 ; f1 := \text{diff}(f(x))$
7. $f(x) := x^2 ; f1 := \text{unapply}(\text{diff}(f(x), x), x)$
8. $f(x) := x^2 ; f1 := x \rightarrow \text{diff}(f(x))$

Exercice 7.9 Les commandes suivantes affectent à A l'expression $2*x*y$: vrai ou faux et pourquoi ?

1. $A := \text{diff}(x^2*y)$
2. $A := x \rightarrow \text{diff}(x^2*y)$
3. $A := \text{diff}(x^2*y, x)$
4. $A := \text{diff}(x^2*y, y)$
5. $A := \text{diff}(x*y^2, y)$
6. $A := \text{normal}(\text{diff}(x*y^2, y))$
7. $A := \text{normal}(\text{diff}(x^2*y^2/2, x, y))$
8. $A := \text{normal}(\text{diff}(\text{diff}(x^2*y^2/2, x), y))$

Exercice 7.10 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 7.11 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 7.12 Les commandes suivantes retournent la liste [1,2,3,4,5] : vrai ou faux et pourquoi ?

1. [1,2,3,4,5]
2. op([1,2,3,4,5])
3. nop(1,2,3,4,5)
4. seq(i,i=1..5)
5. seq(j=1..5)
6. seq(j,j=1..5)
7. seq(j,j,1..5)
8. seq(j,j,1,5)

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

Exercice 7.13 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 7.14 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. `1*10^(j$(j=4..0))`
9. `1*10^(j$(j=4..0))`
10. `1*nop(10^j$(j=4..0))`

Exercice 7.15 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 7.16 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)`

8 Exercices

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

Exercice 8.1 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 8.2 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 8.3 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 8.4 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 8.5 On considère les intégrales définies $I = \int_a^b f(x) dx$ suivantes.

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

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

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 $j = 0, \dots, n$, 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 8.6 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 8.7 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 8.8 On considère les limites suivantes.

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

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 8.9 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 8.10 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 8.11

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 8.12 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 à 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 chaînes de caractères "point fixe" et "tangente", positionnées sur le graphique.

Exercice 8.13 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 8.14 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 8.15 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 8.16 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 choisissant 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 8.17

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 $\{1, \dots, 9\}$.
3. Extraire de la liste l toutes les valeurs ≥ 5 .
4. Pour tout $k = 1, \dots, 9$, compter combien de valeurs de la liste l sont égales à k .

Exercice 8.18 Si x est un réel, la fraction continue à l'ordre n de x est une liste $[a_0, \dots, a_n]$ 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 $[a_0, \dots, a_n]$ est associée au rationnel

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

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

1. Calculer $[a_0, \dots, a_n]$.
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 8.19 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 $n = 1, \dots, 10$.
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 8.20

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 :

$$P = x^d + a_{d-1}x^{d-1} + \dots + a_1x + a_0,$$

est :

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \\ -a_0 & -a_1 & \dots & & -a_{d-1} \end{pmatrix}.$$

3. Calculer les valeurs propres de la matrice A .
4. Calculer le polynôme caractéristique de A .

Exercice 8.21

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 .

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

Exercice 8.22

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

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

Exercice 8.23 Ecrire les fonctions suivantes, sans utiliser de boucle.

- 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 .
- La fonction g prend en entrée un entier n et trois réels a, b, c et retourne la matrice $A = (a_{j,k})_{j,k=1,\dots,n}$ 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 $j = 1, \dots, n - 1$ (les autres termes sont nuls).
- La fonction H prend en entrée un entier n et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\dots,n}$ 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`
- La fonction V prend en entrée un vecteur $x = (x_j)_{j=1,\dots,n}$ et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\dots,n}$ 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`
- La fonction T prend en entrée un vecteur $x = (x_j)_{j=1,\dots,n}$ et retourne en sortie la matrice $A = (a_{j,k})_{j,k=1,\dots,n}$ définie par $a_{j,k} = x_{|j-k|+1}$ (matrice de Toeplitz).

Exercice 8.24 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 $x_{min} \leq x_0 \leq x_{max}$).

- `derive` : Elle calcule et représente graphiquement la dérivée de f sur l'intervalle $[x_{min}, x_{max}]$. Elle retourne la valeur de $f'(x_0)$.
- `tangente` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$, elle superpose sur le même graphique la tangente à f au point x_0 , et retourne l'équation de cette tangente comme un polynôme du premier degré.
- `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 ($x_1 = f(x_0), x_2 = f \circ f(x_0), \dots$). 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) , \dots (comparer avec la fonction `plotseq`)
- `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 : $x_1 = x_0 - f(x_0)/f'(x_0), x_2 = x_1 - f(x_1)/f'(x_1) \dots$. Les valeurs de la dérivée sont approchées. La fonction représente sur le même graphique les segments permettant de visualiser les itérations : segments joignant $(x_0, 0)$, $(x_0, f(x_0))$, $(x_1, 0)$, $(x_1, f(x_1))$, $(x_2, 0)$, $(x_2, f(x_2))$, \dots (comparer avec la fonction `newton`)

Exercice 8.25 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 (z, t) 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

$$I_1 = \iint_D \left(\frac{1+x+y}{(1+x)(1+y)} \right)^3 dx dy.$$

6. Calculer

$$I_2 = \iint_{\Delta} (1+z)(1+t) dz dt,$$

et vérifier que $I_1 = I_2$.

Index

abs, 9
acos, 9
acosh, 9
addition, 5
affectation, 6
affichage graphique, 23
aide en ligne, 4
ans, 5
append, 11
apply, 11
arc
 cosinus, 9
 sinus, 9
 tangente, 9
argument
 cosinus hyperbolique, 9
 sinus hyperbolique, 9
 tangente hyperbolique, 9
arithmétique, 16
arrondi, 9
asc, 12
asin, 9
asinh, 9
assume, 6
atan, 9
atanh, 9

Bezout, 17, 18
blockmatrix, 20
boucle, 26
break, 26

canonical_form, 17
caractère, 11
ceil, 9
cercle, 24
chaîne, 11
char, 12
coeff, 17
color, 23
composée, 9
concat, 12
cone, 25
conj, 9
conjugué, 9

continue, 26
convert, 8
cos, 9
cosh, 9
cosinus, 9
 hyperbolique, 9
couleur, 23
courbe
 en polaires, 23
 implicite, 23
 paramétrique, 23
cumSum, 11
curl, 13
cyclotomic, 18

debug, 27
décomposition en éléments simples, 17
degree, 17
dérivée
 d'une expression, 12
 d'une fonction, 12
 partielle, 12
desolve, 16
det, 20
déterminant, 20
développement limité, 13
développer, 7
diagonalisation, 22
diff, 12
Digits, 5
DispG, 23
divergence, 13
divergence, 13
divis, 18
division, 5
droite, 24, 25

e, 6
effacer, 23
égalité, 6
egcd, 18
eigenvals, 22
eigenvects, 22
else, 26
ensemble, 10

- équation, 6
 - différentielle, 16
 - résolution, 15
- equation, 24
- erase, 23
- erreur, 26
- espace, 25
- et, 26
- evalf, 5
- exact, 5
- exp, 9
- exp2trig, 19
- expand, 7, 17
- expr, 12
- expression, 7, 9

- facteurs premiers, 17, 18
- factor, 7, 18
- factors, 18
- floor, 9
- fonction, 9
 - appliquée à une liste, 11
 - composition des, 9
 - définition d'une, 9
 - graphe d'une, 23
 - primitive d'une, 14
- for, 26
- fraction rationnelle, 17
- fsolve, 15
- function_diff, 12

- Gauss-Jordan, 21
- gcd, 17, 18
- genpoly, 17
- grad, 13
- gradient, 13

- halftan, 19
- head, 12
- help, 4
- hermite, 18
- hessian, 13
- horner, 17
- hyp2exp, 19
- hypothèse, 6

- i, 6
- iabcuv, 17

- idivis, 17
- idn, 20
- iegcd, 17
- if, 26
- ifactor, 17
- ifactors, 17
- im, 9
- image, 20
- infinity, 6
- int, 14
- intégrale, 14
- inter, 24
- interactive_plotode, 16
- interface, 3
- iquo, 17
- iquorem, 17
- irem, 17
- is_prime, 17
- itération, 10
- iteratif, 27

- jordan, 22

- ker, 20

- lagrange, 18
- laguerre, 18
- langage
 - fonctionnel, 25
 - interprété, 28
 - non typé, 25
- laplacian, 13
- laplacien, 13
- lcm, 17, 18
- lcoeff, 17
- legend, 24
- ligne polygonale, 24, 25
- limit, 13
- limite, 13
 - à droite, 13
 - à gauche, 13
- linsolve, 15, 20
- liste, 10
 - des coefficients, 11
 - vide, 11
- ln, 9
- log, 9
- log10, 9

- logarithme
 - en base 10, 9
 - naturel, 9
- makelist, 10
- makemat, 20
- map, 11
- matrice, 19
 - aléatoire, 20
 - déterminant d'une, 20
 - hessienne, 13
 - identité, 20
 - image d'une, 20
 - noyau d'une, 20
 - rang d'une, 20
- matrix, 20
- menus, 3
- mid, 12
- modulo, 16
- multiplication, 5
- newton, 15
- nextprime, 17
- nombre
 - approché, 5
 - complexe, 6
 - d'éléments, 10
 - exact, 5
 - premier, 17
- non, 26
- nop, 10
- normal, 7, 17
- nuage de points, 24
- NULL, 11
- odesolve, 16
- op, 10
- ou, 26
- parameq, 24
- parenthèses, 5
- partfrac, 17
- partie entière, 9
- partie imaginaire, 9
- partie réelle, 9
- pcar, 22
- pcoeff, 17
- peval, 17
- PGCD, 17, 18
- pi, 6
- plan, 25
- plotpolar, 23
- plot, 23
- plotfield, 16
- plotfunc, 25
- plotimplicit, 23
- plotode, 16
- plotparam, 23, 25
- pmin, 22
- point, 24, 25
- poly2symb, 11, 17
- polygone, 24
- polygone_ouvert, 24
- polygonplot, 24
- polynôme, 17
 - aléatoire, 18
 - caractéristique, 22
 - cyclotomique, 18
 - de Hermite, 18
 - de Lagrange, 18
 - de Laguerre, 18
 - de Tchebyshev, 18
 - minimal, 22
- powermod, 17
- PPCM, 17, 18
- précision, 5
- previousprime, 17
- primitive, 14
- priorités, 5
- product, 11
- produit, 11
 - matriciel, 19, 20
 - scalaire, 20
 - terme à terme, 20
 - vectorel, 20
- programmation, 25
- proot, 15
- propre
 - valeur, 22
 - vecteur, 22
- ptayl, 17
- puissance, 5
- purge, 6
- quo, 18

quotient, 17, 18
 racine carrée, 9
 randpoly, 18
 rank, 20
 ranm, 20
 ratnormal, 7
 re, 9
 récursif, 27
 rem, 18
 résolution
 approchée, 15
 exacte, 15
 reste, 17, 18
 romberg, 14
 rotationnel, 13
 round, 9
 rref, 20

 seq, 10
 séquence, 10
 vide, 11
 series, 13
 simplifications automatiques, 7
 simplifier, 7
 simplify, 7
 simult, 20
 sin, 9
 sinh, 9
 sinus, 9
 hyperbolique, 9
 size, 10, 12
 solve, 15
 somme, 11
 sommes cumulées, 11
 soustraction, 5
 sphere, 25
 sqrt, 5, 9
 string, 12
 subst, 7
 somme, 11
 symb2poly, 11, 17
 système
 d'équations, 15
 linéaire, 20

 table, 25
 tail, 12

 tan, 9
 tan2sincos, 19
 tangente, 9
 hyperbolique, 9
 tangente, 23
 tanh, 9
 taylor, 13
 tchebyshev1, 18
 tchebyshev2, 18
 tcoeff, 17
 tcollect, 19
 temps de calcul, 12
 test, 26
 texpand, 19
 texte, 24
 time, 12
 tlin, 19
 trig2exp, 19
 trigonométrie, 18
 trigtan, 19
 tsimplify, 8

 unapply, 9

 valeur absolue, 9
 valeurs propres, 22
 valuation, 17
 variable
 affectée, 6
 formelle, 6
 vecteur, 19
 vecteurs propres, 22

 zoom, 22