

FAQ MAXIMA

Version 2.0 du 5 février 2025

Auteur Michel Gosse

Mel : michel.gosse@free.fr

Cette Faq a pour objectif de répondre de manière concrète et pratique à toutes les questions que l'on peut se poser sur Maxima et son utilisation. La dernière version de cette faq est téléchargeable à l'adresse, rubrique Documentation :

<https://maxima-french-doc.fr>

Certaines réponses sont extraites de la liste internationale de Maxima dont nous remercions tous les participants.

Table des matières

1	Maxima	3
1.1	Le statut de Maxima	3
1.1.1	Maxima est-il gratuit ?	3
1.1.2	Maxima est-il performant ?	4
1.1.3	Quelles différences avec des logiciels comme Maple ou Mathematica ?	4
1.1.4	Qui est l'auteur de Maxima ?	4
1.1.5	Quelle est la différence entre Macsygma et Maxima	4
1.1.6	Quel est le site officiel de Maxima	4
1.2	Les versions de Maxima	4
1.2.1	Quelle est la dernière version de Maxima ?	4
1.2.2	Les différentes versions sont-elles compatibles ?	4
1.2.3	Quelles sont les plate-formes supportées par Maxima	4
1.2.4	Y a-t-il de nouvelles versions de prévu ?	4
1.2.5	Qui développe Maxima	4
1.3	Les interfaces graphiques à Maxima	5
1.3.1	L'interface xmaxima (tcl)	5
1.3.2	WxMaxima	5
1.3.3	Texmacs	5
2	Calcul numérique	5
2.1	Comment trouver une valeur approchée d'un nombre ?	5
2.2	Comment résoudre numériquement une équation ?	6
2.3	Comment trouver les solutions positives d'une équation	7
2.4	La fonction partie entière	7
2.5	La fonction valeur absolue	7
2.6	Majorer une somme par la somme des valeurs absolues	8
2.7	Simplifier des expressions comportant des racines carrées	8
2.7.1	En utilisant la commande sqdnst	8
2.7.2	avec la commande ratsimp	8
2.8	Comment trouver le maximum d'une liste	8
2.9	Comment calculer des lignes trigonométriques	9
3	Arithmétique	9
3.1	Quotient et diviseur	9
3.2	Comment obtenir le reste d'une division	9

3.3	Comment changer de base	9
4	Calcul algébrique	10
4.1	Comment résoudre une inéquation ?	10
4.2	Comment décomposer une fraction en éléments simples ?	10
4.3	Comment simplifier une somme de logarithmes	10
4.4	Comment développer un produit dans un logarithme	10
4.5	Comment ordonner les termes d'un polynôme	11
4.6	Comment obtenir le coefficient du terme d'un polynome	11
4.7	Comment générer un polynome à partir des coefficients	11
4.8	Linéariser un polynome trigonométrique	11
4.9	Transformer une expression sous forme exponentielle	12
4.10	Comment obtenir un développement en série	12
4.11	Comment éliminer une racine carrée d'un dénominateur	12
4.12	Substituer à une variable une valeur	12
4.13	Substituer complètement à une variable une expression	13
4.14	Substituer complètement et simplifier une expression	13
4.15	Manipuler une équation	13
5	Nombres complexes	14
5.1	Définition d'un nombre complexe	14
5.2	Module d'un nombre complexe	14
5.3	Argument d'un nombre complexe	14
5.4	Forme algébrique d'un nombre complexe	14
5.5	Forme exponentielle	15
5.6	Formules d'Euler	15
5.7	Transformer une exponentielle complexe	15
6	Analyse	16
6.1	Comment calculer une limite ?	16
6.2	Comment calculer une dérivée d'ordre n ?	16
6.3	Calcul d'intégrales	17
6.3.1	Comment intégrer par parties ?	17
6.3.2	Un package pour l'intégration par parties	17
6.4	Intégrer numériquement avec la méthode de Romberg	17
6.5	La fonction dilogarithme	18
7	Statistiques	18
7.1	Comment étudier une série statistique à une variable	18
8	Algèbre linéaire	19
8.1	Vecteurs	19
8.1.1	Comment noter un vecteur	19
8.1.2	Comment calculer un produit scalaire	19
8.1.3	Comment calculer un produit vectoriel	19
8.2	Matrices	20
8.2.1	Comment écrire une matrice	20
8.2.2	Comment extraire une ligne d'une matrice	20
8.2.3	Comment extraire une colonne d'une matrice	20
8.2.4	Comment extraire un élément d'une matrice	20
8.2.5	Comment ajouter un élément ou une liste à une liste	20
8.2.6	Comment ajouter une colonne ou une ligne à une matrice	20
8.2.7	Comment transposer une matrice	21
8.2.8	Trouver une matrice échelonnée réduite avec Maxima	21

8.2.9	Transformer une matrice en un vecteur ligne	21
8.2.10	Comment générer automatiquement une matrice	22
8.2.11	Comment multiplier deux matrices	22
8.2.12	Comment inverser une matrice	22
8.2.13	Comment calculer une puissance de matrices	23
8.3	Comment résoudre une récurrence linéaire	23
9	Graphisme	24
9.1	Courbes planes cartésiennes	24
9.1.1	Représenter une fonction	24
9.1.2	Représenter une fonction en escalier	25
9.1.3	Représenter la fonction partie entière	25
9.1.4	Représenter une famille de fonctions	26
9.2	Comment représenter un cercle	26
9.3	Courbes planes polaires	27
9.3.1	Comment tracer une courbe en polaire	27
9.4	Courbes planes paramétriques	28
9.4.1	Comment tracer une courbe en paramétrique ?	28
10	Entrées et sorties	28
10.1	Variables réservées	28
10.1.1	Comment noter plus l'infini ?	28
10.1.2	Comment noter moins l'infini ?	28
10.2	Effacer la valeur d'une variable	28
10.3	Fonctions usuelles	29
10.3.1	Comment écrire la fonction logarithme népérien	29
10.3.2	Comment définir la fonction logarithme décimal	29
10.3.3	Comment définir une fonction à partir d'une dérivée	30
10.4	Comment transformer une expression en fonction	30
10.5	Définir une fonction de manière formelle	31
10.6	Comment récupérer un élément d'une liste	31
10.7	Comment additionner les éléments d'une liste	31
10.8	Comment simplifier une expression trigonométrique	31
10.9	Comment tester une égalité	32
10.10	Comment déclarer qu'un nombre est entier	32
10.11	Comment déclarer qu'un nombre est positif	32
10.12	Comment savoir si une expression contient une variable	33
10.13	Etiqueter une sous-expression	33
10.14	Comment exporter en tex	33
10.15	Comment choisir un fichier pour l'exportation	34
10.16	Comment charger des commandes au lancement de maxima	34
10.17	Comment utiliser une fonction comme argument	34
10.18	Comment composer n fois une fonction	35
10.19	Comment collecter des termes d'une expression	35
10.20	Exporter en Latex	35

1 Maxima

1.1 Le statut de Maxima

1.1.1 Maxima est-il gratuit ?

Totalement. Le code de Maxima est en open source. Vous avez droit de le copier, de le modifier, de le distribuer librement. Vous pouvez installer le logiciel sur autant de postes que vous le désirez.

1.1.2 Maxima est-il performant ?

Tout à fait. Le code de Maxima existe depuis plus de 15 ans. Maxima est utilisé avec succès pour résoudre de nombreux problèmes mathématiques, scientifiques ou numériques.

1.1.3 Quelles différences avec des logiciels comme Maple ou Mathematica ?

Maxima est gratuit et distribué en Open Source. Il n'a rien à envier aux logiciels commerciaux. Ces derniers proposent cependant quelques packages très spécialisés dont Maxima ne dispose pas forcément. Le support technique de Maxima est basé sur la communauté des développeurs et utilisateurs, tandis que les logiciels commerciaux peuvent proposer un support payant.

1.1.4 Qui est l'auteur de Maxima ?

Les informaticiens du département de l'Energie américain ont élaboré le premier code source du logiciel. Ce code fut d'abord vendu à une société privée qui l'a développé afin de commercialiser le logiciel Macsygma. Le docteur William Schelter a ensuite obtenu l'autorisation d'exploiter le code du logiciel original, qui fut mis en open source sous le nom de Maxima. Le docteur Schelter l'a maintenu et amélioré durant une quinzaine d'année. A sa mort prématurée en 2001, le développement du logiciel a été repris par une équipe internationale de développeurs bénévoles.

1.1.5 Quelle est la différence entre Macsygma et Maxima

Macsygma était un logiciel commercial dont le code source a été récupéré pour construire Maxima. Ce logiciel Macsygma n'est plus commercialisé. La version en open source s'appelle Maxima et est activement développée.

1.1.6 Quel est le site officiel de Maxima

Le site officiel de Maxima est hébergé par Sourceforge à l'adresse :

<https://maxima.sourceforge.io/>

1.2 Les versions de Maxima

1.2.1 Quelle est la dernière version de Maxima ?

La dernière version stable est la 5.47-0 en date du 31 mai 2023, utilisée pour cette FAQ.

1.2.2 Les différentes versions sont-elles compatibles ?

ATTENTION : depuis la version 5.9.2, Maxima différencie les majuscules et les minuscules, à la fois dans le nom des fonctions et dans les variables. Cela implique que certains programmes ou scripts anciens sont à adapter. A part cela, les fichiers élaborés avec des versions anciennes de Maxima sont interprétés correctement par les dernières versions de Maxima.

1.2.3 Quelles sont les plate-formes supportées par Maxima

Maxima existe en version Linux, Windows et Mac Os.

1.2.4 Y a-t-il de nouvelles versions de prévu ?

Oui, la version 6.0 est en construction. Elle doit permettre de nettoyer le code du logiciel et de supprimer des bugs rapportés par les utilisateurs. Cette version comportera des améliorations majeures, notamment au niveau de l'interface. De nouvelles versions intermédiaires paraissent régulièrement et font de Maxima un logiciel évolutif.

1.2.5 Qui développe Maxima

?

Une équipe de développeurs bénévoles travaille sur le code de Maxima. Ils communiquent entre eux grâce à Internet. Un site CVS dédié permet de faire les mises à jour du code. Les utilisateurs communiquent les bugs trouvés qui sont éradiqués.

1.3 Les interfaces graphiques à Maxima

Maxima est le moteur de calcul, programmé en Lisp. On peut se connecter au logiciel par le biais de différents logiciels, qui fournissent ou non une interface graphique. Dans ce cas, Maxima est le serveur, et l'interface est le client. Parmi les interfaces existantes, les suivantes sont les plus répandues :

1.3.1 L'interface xmaxima (tcl)

C'était l'interface graphique qui était livrée par défaut avec le programme il y a quelques années. Elle est ancienne, sommaire et peu conviviale. Cette interface n'est quasiment plus utilisée, même si ce programme existe encore et fonctionne sans problème.

1.3.2 WxMaxima

Interface moderne et fonctionnelle, qui permet d'entrer une grande partie des commandes de Maxima à l'aide d'icônes et de menus. Ses fonctionnalités d'édition et sa simplicité en font un logiciel idéal pour utiliser Maxima. Elle fonctionne sous toutes les plate-formes. Elle est installée par défaut avec Maxima, et s'est imposée comme l'interface standard de Maxima. Le site de référence est :

<https://wxmaxima-developers.github.io/wxmaxima/index.html>

1.3.3 Texmacs

Texmacs est un traitement de textes scientifiques, qui permet d'entrer directement des commandes Maxima. Son intérêt est de produire des documents d'une qualité remarquable. Par contre, toutes les commandes Maxima doivent être entrées au clavier. Le site de référence :

<https://www.texmacs.org/>

2 Calcul numérique

2.1 Comment trouver une valeur approchée d'un nombre ?

La commande `ev(x, numer)` renvoie une valeur approchée de x .

La commande `bfloat(x)` renvoie le nombre x en virgule flottante, à la précision donnée par `fpprec` que l'on peut modifier :

(C7) `x:%pi;`

(D7) π

(C8) `ev(x, numer);`

(D8) 3.141592653589793

(C9) `fpprec:10;`

(D19) 10

(C20) `bfloat(x);`

3.141592654B0

(C21) `fpprec:100;`

(D21) 100

```
(C22) bfloat(x);
```

```
3.1415926535897932384626433832795028841971693993751058209749445923078
16406286208998628034825342117068B0
```

2.2 Comment résoudre numériquement une équation ?

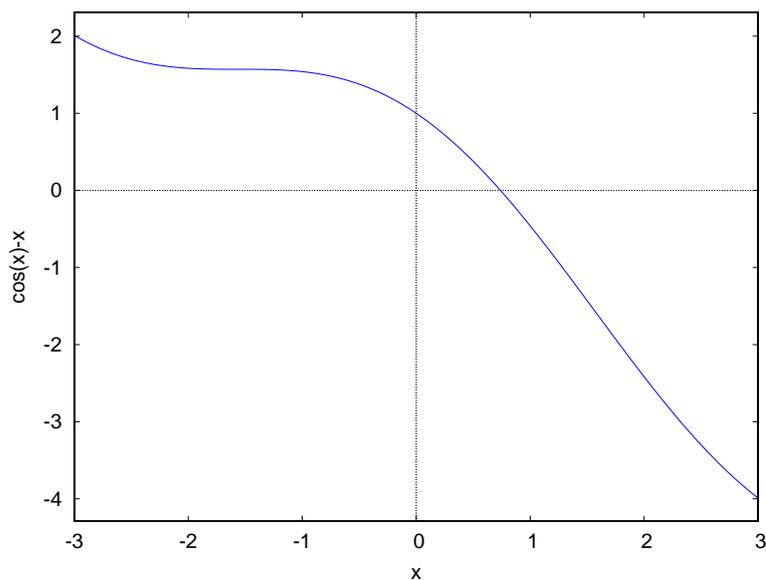
Pour déterminer des valeurs approchées d'une ou des racines d'une équation, on commence par localiser grossièrement cette racine, par exemple avec un graphique, puis on utilise la commande `find_root()`. Pour illustrer cette technique, on s'aperçoit que l'équation $\cos(x) - x = 0$ a une solution appartenant à l'intervalle $[0, 1]$, puis on utilise la commande `find_root()`.

- On peut aussi utiliser le package `newton1` pour résoudre ce problème. La commande `newton(expression, variable, point de départ, précision)` donne une valeur approchée de la solution recherchée.
- Le package `mnewton` effectue un travail similaire, mais peut résoudre en valeur approchée un système d'équations.

On peut aussi utiliser le package `newton` pour résoudre ce problème :

Remarque : la commande graphique utilisée est spécifique à Texmacs (`tm_plot2d`). Pour maxima, il faut juste utiliser directement la commande `plot2d` avec les paramètres indiqués ci-dessous.

```
(%i2) tm_plot2d(cos(x)-x, [x, -3, 3])$
```



```
(%i4) find_root(cos(x)-x=0,x,0,1);
```

```
(%o4) 0.7390851332151607
```

```
(%i8) load(newton1);
```

```
(%o8) /usr/share/maxima/5.45.1/share/numeric/newton1.mac
```

```
(%i9) newton(cos(x)-x,x,1,1/1000);
```

```
(%o9) 0.7391128909113617
```

```
(%i10) load(mnewton);
```

```
(%o10) /usr/share/maxima/5.45.1/share/mnewton/mnewton.mac
```

```
(%i11) mnewton(cos(x)-x=0, [x], [0]);
```

```
(%o11) [[x = 0.7390851332151607]]
```

2.3 Comment trouver les solutions positives d'une équation

Solution proposée par Monsieur Rodriguez sur la liste Maxima : Il définit une fonction positive, qui renvoie les éléments d'une liste dont le membre de droite est positif. Cette fonction agit uniquement sur la liste renvoyée par la commande de résolution d'une équation.

```
(%i12) positive(res):=sublist(res, lambda([z],member(sign(rhs(z[1])),['pos,
'pz])))

(%o12) positive(res) := sublist(res, lambda([z], member(sign(rhs(z1)), ['pos, 'pz])))

(%i13) algsys([x^2+2*x-3], [x])

(%o13) [[x = -3], [x = 1]]

(%i14) positive(%)

(%o14) [[x = 1]]

(%i29) positive([[x=1], [x=-2], [x=5]])

(%o29) [[x = 1], [x = 5]]
```

2.4 La fonction partie entière

La partie entière d'un réel x est donnée par entier(x) :

```
(%i30) entier(2.526);

(%o30) 2

(%i31) entier(0.489);

(%o31) 0

(%i32) entier(-1.45);

(%o32) -2
```

2.5 La fonction valeur absolue

La valeur absolue d'un réel x est donnée par abs(x) :

```
(%i33) abs(2);

(%o33) 2

(%i34) abs(-1);

(%o34) 1

(%i35) abs(x^2);

(%o35) x2

(%i36) abs(x^2-x);

(%o36) |x2 - x|

(%i37) assume(x>1);

(%o37) [x > 1]

(%i38) abs(x^2-x);

(%o38) x2 - x
```

Comme on le voit, on peut imposer une condition sur x pour permettre à Maxima de déterminer la valeur absolue d'une expression. Cette condition est donnée par l'argument de la commande assume().

2.6 Majorer une somme par la somme des valeurs absolues

Soit $X = a + b - c$. Pour majorer X par $|a| + |b| + |c|$, on écrit :

```
(%i39) X:a+b+-c;
(%o39) -c + b + a
(%i40) apply(op(%),map(abs,args(%)));
(%o40) |c| + |b| + |a|
```

2.7 Simplifier des expressions comportant des racines carrées

2.7.1 En utilisant la commande sqdnst

La commande sqdnst permet certaines simplifications d'expressions avec des radicaux :

```
(%i42) a:sqrt(2)+sqrt(6-4*sqrt(2));
```

```
(%o42)  $\sqrt{6 - 2^{\frac{5}{2}} + \sqrt{2}}$ 
```

```
(%i45) sqrtidenest(a);
```

```
(%o45) 2
```

2.7.2 avec la commande ratsimp

Par défaut, la commande ratsimp de Maxima ne simplifie pas les expressions contenant des racines carrées. Mettre l'option algebraic à true permet d'imposer les simplifications. La commande radcan(expression) a aussi pour but de simplifier les expressions comportant des radicaux :

```
(%i48) q1:1/(sqrt(5)-1);
```

```
(%o48)  $\frac{1}{\sqrt{5} - 1}$ 
```

```
(%i49) ratsimp(q1);
```

```
(%o49)  $\frac{1}{\sqrt{5} - 1}$ 
```

```
(%i50) algebraic:true;
```

```
(%o50) true
```

```
(%i51) ratsimp(q1);
```

```
(%o51)  $\frac{\sqrt{5} + 1}{4}$ 
```

```
(%i57) (4-sqrt(48))/2=ratsimp((4-sqrt(48))/2);
```

```
(%o57)  $\frac{4 - 4\sqrt{3}}{2} = 2 - 2\sqrt{3}$ 
```

```
(%i58) radcan((4-sqrt(48))/2);
```

```
(%o58)  $2 - 2\sqrt{3}$ 
```

2.8 Comment trouver le maximum d'une liste

Il suffit d'appliquer la commande max à une liste grâce à la fonction apply :

```
(%i1) apply(max,[5,7,2,3]);
```

```
(%o1) 7
```

2.9 Comment calculer des lignes trigonométriques

Par défaut, Maxima connaît la valeur de quelques lignes trigonométriques. Le package ntrig permet d'obtenir des résultats supplémentaires :

```
(%i59) cos(%pi/4);
```

```
(%o59)  $\frac{1}{\sqrt{2}}$ 
```

```
(%i60) cos(%pi/5);
```

```
(%o60)  $\cos\left(\frac{\pi}{5}\right)$ 
```

```
(%i61) load(ntrig);
```

```
(%o61) /usr/share/maxima/5.45.1/share/trigonometry/ntrig.mac
```

```
(%i62) cos(%pi/5);
```

```
(%o62)  $\frac{\sqrt{5}+1}{4}$ 
```

3 Arithmétique

3.1 Quotient et diviseur

La commande `divide(m,n)` renvoie le diviseur et le reste de la division de l'entier `m` par `n`. Cette commande fonctionne également avec les polynômes.

```
(%i63) divide(100,7);
```

```
(%o63) [14, 2]
```

```
(%i64) divide(3693,3);
```

```
(%o64) [1231, 0]
```

```
(%i68) divide(X^3-1,X-1);
```

```
(%o68) [X^2 + X + 1, 0]
```

```
(%i69) divide(X^4+X^2-3,X^2+1);
```

```
(%o69) [X^2, -3]
```

3.2 Comment obtenir le reste d'une division

La commande `mod(a,b)` renvoie le reste de la division de `a` par `b` :

```
(%i70) mod(25,4);
```

```
(%o70) 1
```

```
(%i71) mod(2345,23);
```

```
(%o71) 22
```

3.3 Comment changer de base

La base de départ est définie par le paramètre `ibase`, celle d'arrivée par le paramètre `obase`. Il suffit d'affecter les valeurs désirées à ces deux paramètres pour que Maxima effectue les changements de base attendus :

```
(%i77) ibase:10$
(%i76) obase:6$
(%i78) [5,6,7,8];
(%o210) [5,10,11,12]
(%i79) obase:12$
(%i80) [10,11,12,13,2576];
(%o68) [0 A,0 B,10,11,15 A 8]
```

Ces commandes ne permettent pas de d'écrire un nombre de la base $b > 10$ en un nombre écrit en base 10.

4 Calcul algébrique

4.1 Comment résoudre une inéquation ?

La résolution d'inéquations est très sommaire dans Maxima. Quelques inéquations simples sont résolues grâce à la commande `solve_rat_ineq()` du package `solve_rat_ineq` :

```
(%i91) load(solve_rat_ineq);
(%o77) /usr/share/maxima/5.45.1/share/solve_rat_ineq/solve_rat_ineq.mac
(%i92) solve_rat_ineq(2*x+5>7+x);
(%o78) [[x > 2]]
(%i93) solve_rat_ineq(x^2-1>=0);
(%o79) [[x ≤ -1], [x ≥ 1]]
```

4.2 Comment décomposer une fraction en éléments simples ?

On utilise la commande `partfrac(expression, variable)` :

```
(%i2) partfrac((x^2+8*x+4)/(x^2-4),x);
```

$$(\%o2) \frac{2}{x+2} + \frac{6}{x-2} + 1$$

4.3 Comment simplifier une somme de logarithmes

On emploie la fonction `logcontract(expression)` :

```
(%i3) logcontract(log(x)+log(x+1));
(%o3) log(x(x+1))
(%i4) logcontract(log(x)-log(x+1));
(%o4) log\left(\frac{x}{x+1}\right)
(%i5) logcontract(10*log(x));
(%o5) log(x^10)
```

4.4 Comment développer un produit dans un logarithme

Il s'agit de l'opération inverse. Pour cela, on écrit l'expression en utilisant la directive `logexpand=super` :

```
(%i6) expression:(log(x^2*(x+1)^5));
(%o6) log(x^2(x+1)^5)
(%i7) expression, logexpand=super;
(%o7) 5 log(x+1) + 2 log(x)
```

4.5 Comment ordonner les termes d'un polynôme

On utilise la commande `declare(x,mainvar)`, qui déclare `x` comme variable principale. Dans ce cas, le polynôme s'ordonnera selon les puissances de `x`. Par exemple :

```
(%i8) expand((x+y)^3);
(%o8) y^3 + 3xy^2 + 3x^2y + x^3
(%i9) declare(x,mainvar);
(%o9) done
(%i10) expand((x+y)^3);
(%o10) x^3 + 3yx^2 + 3y^2x + y^3
```

Autre solution proposée par Richard Fateman, l'utilisation de la commande `ratexpand(polynôme,variable)` qui développe en respectant la variable `x` :

```
(%i11) rat(expand(x+y)^3,x);
(%o11) x^3 + 3yx^2 + 3y^2x + y^3
(%i12) rat(expand(x+y+z)^3);
(%o12) x^3 + (3z + 3y)x^2 + (3z^2 + 6yz + 3y^2)x + z^3 + 3yz^2 + 3y^2z + y^3
(%i13) rat(expand(x+y+z)^3,x);
(%o13) x^3 + (3z + 3y)x^2 + (3z^2 + 6yz + 3y^2)x + z^3 + 3yz^2 + 3y^2z + y^3
```

4.6 Comment obtenir le coefficient du terme d'un polynome

La commande `ratcoeff(polynome,x^n)` donne le coefficient du terme en x^n :

```
(%i14) ratcoeff((z+1)^2*(t+y)^2,t^2);
(%o14) z^2 + 2z + 1
(%i15) ratcoeff((z+1)^2*(t+y)^2,t);
(%o15) 2yz^2 + 4yz + 2y
(%i16) ratcoeff(a*x^2+b*x+c,x^2);
(%o16) a
```

4.7 Comment générer un polynome à partir des coefficients

```
(%i17) coefficients:[5,4,3,2];
(%o17) [5, 4, 3, 2]
(%i18) sum(coefficients[i+1]*x^i,i,0,length(coefficients)-1);
(%o18) 2x^3 + 3x^2 + 4x + 5
```

4.8 Linéariser un polynome trigonométrique

La fonction `trigrat(expression)` permet de linéariser une expression trigonométrique :

```
(%i19) trigrat(sin(x)^2);
```

```
(%o19) 
$$\frac{\cos(2x) - 1}{2}$$

```

```
(%i20) trigrat(cos(x)^4+sin(x)^3);
```

```
(%o20) 
$$\frac{\cos(4x) - 2\sin(3x) + 4\cos(2x) + 6\sin(x) + 3}{8}$$

```

4.9 Transformer une expression sous forme exponentielle

Pour exprimer une fonction trigonométrique ou trigométrique réciproque à l'aide de la fonction exponentielle réelle ou complexe, il faut utiliser la commande `exponentialize` :

```
(%i21) exponentialize(cos(x));
```

```
(%o21) 
$$\frac{e^{ix} + e^{-ix}}{2}$$

```

```
(%i22) exponentialize(tanh(x));
```

```
(%o22) 
$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

```

4.10 Comment obtenir un développement en série

La commande `taylor(fonction,variable,point,ordre)` donne la partie principale du développement en série de Taylor d'une fonction :

```
(%i1) taylor(cos(x),x,0,7);
```

```
(%o1) 
$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \dots$$

```

```
(%i2) taylor(log(x),x,1,4);
```

```
(%o2) 
$$x - 1 - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \dots$$

```

4.11 Comment éliminer une racine carrée d'un dénominateur

On peut utiliser la commande `ratsimp(expression)`, à condition de positionner l'option `algebraic` à `true`. A la fin de cet exemple, on utilise la commande `remvalue(x)` qui permet de réinitialiser la variable `x` pour la suite du document.

```
(%i3) x:(1+sqrt(5))/(2-sqrt(5));
```

```
(%o3) 
$$\frac{\sqrt{5} + 1}{2 - \sqrt{5}}$$

```

```
(%i4) algebraic:true;
```

```
(%o4) true
```

```
(%i7) ratsimp(x);
```

```
(%o7) 
$$-3\sqrt{5} - 7$$

```

```
(%i8) remvalue(x);
```

```
(%o8) [x]
```

4.12 Substituer à une variable une valeur

La fonction `subst(valeur,variable,expression)` remplace dans `expression` la variable par `valeur`.

(%i9) eq:2*x^2+1/x-(x+1)^3/(1-x)

(%o9) $-\frac{(x+1)^3}{1-x} + 2x^2 + \frac{1}{x}$

(%i10) subst(3,x,eq);

(%o10) $\frac{151}{3}$

4.13 Substituer complètement à une variable une expression

Lorsque l'on désire remplacer une variable par une autre (en éliminant la première variable), il faut utiliser à la place de subst la commande ratsubst :

(%i11) eq;

(%o11) $-\frac{(x+1)^3}{1-x} + 2x^2 + \frac{1}{x}$

(%i12) subst(y,1-x,eq);

(%o12) $-\frac{(x+1)^3}{y} + 2x^2 + \frac{1}{x}$

(%i13) ratsubst(y,1-x,eq)

(%o13) $\frac{3y^4 - 13y^3 + 24y^2 - 23y + 8}{y^2 - y}$

4.14 Substituer complètement et simplifier une expression

Si a est tel que $a^2 = 1 + a$, on veut simplifier l'expression $X = (a + 1)^5$. Pour cela, la commande fullratsubst(expression), du package lrats, est à utiliser :

(%i14) Y:(a+1)^5;

(%o14) $(a + 1)^5$

(%i15) load(lrats);

(%o15) /usr/share/maxima/5.45.1/share/simplification/lrats.mac

(%i16) fullratsubst(1+a,a^2,Y);

(%o16) $55a + 34$

4.15 Manipuler une équation

Les opérations élémentaires (somme, multiplication) s'appliquent à chacun des membres d'une équation donnée. Certaines commandes peuvent aussi s'appliquer (expand par exemple). Ce comportement ne fonctionne pas si l'on essaye d'appliquer une fonction à une équation :

(%i17) eq:2*x-5=5*x+3;

(%o17) $2x - 5 = 5x + 3$

(%i18) eq+5;

(%o18) $2x = 5x + 8$

(%i19) eq/2;

(%o19) $\frac{2x - 5}{2} = \frac{5x + 3}{2}$

```
(%i20) eq^2;
(%o20) (2x - 5)^2 = (5x + 3)^2
(%i21) sin(eq);
(%o21) sin(2x - 5) = sin(5x + 3)
(%i22) expand(eq^2);
(%o22) 4x^2 - 20x + 25 = 25x^2 + 30x + 9
```

5 Nombres complexes

5.1 Définition d'un nombre complexe

Le complexe $z = a + ib$ se définit avec Maxima par `z:a+%i*b`;

```
(%i23) z1:sqrt(3)+%i;
(%o23) i + sqrt(3)
(%i27) z2:sqrt(2)-%i*sqrt(2);
(%o29) sqrt(2) - sqrt(2)i
(%i30) (%i)^2
(%o30) -1
```

5.2 Module d'un nombre complexe

La fonction `cabs(nombre complexe)` renvoie le module :

```
(%i31) cabs(z1);
(%o31) 2
(%i33) cabs(z2);
(%o33) 2
(%i34) cabs(%i)
(%o34) 1
```

5.3 Argument d'un nombre complexe

La fonction `carg(nombre complexe)` renvoie un argument en radians du complexe donné :

```
(%i35) carg(z1);
(%o35) pi/6
(%i36) carg(z2);
(%o36) -pi/4
```

5.4 Forme algébrique d'un nombre complexe

La commande `rectform(z)` renvoie la forme algébrique de z :

```
(%i38) z1:(1+%i)/(3-2*i);
```

```
(%o38)  $\frac{i+1}{3-2i}$ 
```

```
(%i39) rectform(z1);
```

```
(%o39)  $\frac{5i}{13} + \frac{1}{13}$ 
```

```
(%i40) z2:2*exp(1+%i);
```

```
(%o40)  $2e^{i+1}$ 
```

```
(%i41) rectform(z2);
```

```
(%o41)  $2e^{i\sin(1)} + 2e^{\cos(1)}$ 
```

5.5 Forme exponentielle

La commande `polarform(z)` renvoie la forme exponentielle de z :

```
(%i42) z3:1-%i;
```

```
(%o42)  $1 - i$ 
```

```
(%i43) polarform(z3);
```

```
(%o43)  $\sqrt{2}e^{-\frac{i\pi}{4}}$ 
```

5.6 Formules d'Euler

La fonction `exponentialize[nombre complexe]` permet l'application des formules d'Euler sur un nombre complexe :

```
(%i44) exponentialize(cos(x));
```

```
(%o44)  $\frac{e^{ix} + e^{-ix}}{2}$ 
```

```
(%i45) exponentialize(sin(2*x)+sin(x));
```

```
(%o45)  $\frac{i(e^{2ix} - e^{-2ix})}{2} - \frac{i(e^{ix} - e^{-ix})}{2}$ 
```

```
(%i46) exponentialize(cos(x)^2);
```

```
(%o46)  $\frac{(e^{ix} + e^{-ix})^2}{4}$ 
```

```
(%i47) expand(%);
```

```
(%o47)  $\frac{e^{2ix}}{4} + \frac{e^{-2ix}}{4} + \frac{1}{2}$ 
```

5.7 Transformer une exponentielle complexe

Pour transformer une exponentielle complexe en nombre complexe, on utilise la fonction `demoivre` :

```
(%i50) %e^(%i*x)+%e^(-%i*x)=demoivre(%e^(%i*x)+%e^(-%i*x));
```

```
(%o50)  $e^{ix} + e^{-ix} = 2 \cos(x)$ 
```

```
(%i54) %e^(%i*x)-%e^(-%i*x) = demoivre(%e^(%i*x)-%e^(-%i*x))
```

```
(%o54)  $e^{ix} - e^{-ix} = 2i \sin(x)$ 
```

6 Analyse

6.1 Comment calculer une limite ?

La fonction `limit(expression,variable,point,direction)` répond à cette question. La direction (facultative) est donnée par `plus` pour la limite par valeur supérieure, et `moins` pour la limite par valeur inférieure :

```
(%i55) 'limit(sin(x)/x,x,0)=limit(sin(x)/x,x,0)
```

```
(%o55)  $\lim_{x \rightarrow 0} \frac{\sin(x)}{x} = 1$ 
```

```
(%i56) 'limit(exp(x)/x^2,x,inf)=limit(exp(x)/x^2,x,inf)
```

```
(%o56)  $\lim_{x \rightarrow \infty} \frac{e^x}{x^2} = \infty$ 
```

```
(%i57) 'limit(1/(x^2+1),x,minf)=limit(1/(x^2+1),x,minf)
```

```
(%o57)  $\lim_{x \rightarrow -\infty} \frac{1}{x^2+1} = 0$ 
```

```
(%i58) 'limit(1/t,t,0,plus)=limit(1/t,t,0,plus)
```

```
(%o58)  $\lim_{t \downarrow 0} \frac{1}{t} = \infty$ 
```

```
(%i60) 'limit(1/t,t,0,moins)=limit(1/t,t,0,moins)
```

```
(%o60)  $\lim_{t \uparrow 0} \frac{1}{t} = -\infty$ 
```

```
(%i61) limit(1/t,t,0)
```

```
(%o61) infinity
```

Dans le cas précédent, Maxima indique que le résultat est soit $+\infty$ soit $-\infty$ sans pouvoir aller plus loin.

6.2 Comment calculer une dérivée d'ordre n ?

La commande `diff(f(x),x,n)` renvoie l'expression de la dérivée n-ième de la fonction `f` par rapport à la variable `x`.

```
(%i1) f(x):=x^5+1/x;
```

```
(%o1)  $f(x) := x^5 + \frac{1}{x}$ 
```

```
(%i2) diff(f(x),x,2);
```

```
(%o2)  $20x^3 + \frac{2}{x^3}$ 
```

```
(%i3) diff(f(x),x,6);
```

```
(%o3)  $\frac{720}{x^7}$ 
```

6.3 Calcul d'intégrales

6.3.1 Comment intégrer par parties ?

Le code suivant permet d'effectuer une intégration par parties. u est la fonction à intégrer, et v est la fonction à différentier :

```
(%i4) intpart(u,v,a,b):=subst(x=b,integrate(u,x))*subst(x=b,v)-subst(x=a,
integrate(u,x))*subst(x=a,v)-integrate(integrate(u,x)*diff(v,x),x,a,b);
```

```
(%o4) intpart(u,v,a,b):=subst(x=b,integrate(u,x))subst(x=b,v)-subst(x=a,integrate(u,
x))subst(x=a,v)-integrate(integrate(u,x)diff(v,x),x,a,b)
```

```
(%i5) 'integrate(x*log(x),x,1,%e)=intpart(x*log(x),1,%e);
```

```
(%o5)  $\int_1^e x \log(x) dx = \frac{e^2}{2} - \frac{e^2}{2} - \frac{1}{2}$ 
```

```
(%i6) 'integrate(x*exp(2*x),x,0,1)=intpart(x,exp(2*x),0,1);
```

```
(%o6)  $\int_0^1 x e^{2x} dx = \frac{e^2}{4} + \frac{1}{4}$ 
```

6.3.2 Un package pour l'intégration par parties

Le package `bypart` permet d'effectuer une intégration par parties sans bornes définies. On dispose alors de la fonction `byparts(intégrande,variable,u,dv)` :

```
(%i10) load(bypart);
```

On charge le package avec la commande précédente

```
(%o10) /usr/share/maxima/5.45.1/share/integration/bypart.mac
```

```
(%i11) byparts(log(x),x,log(x),1);
```

On intègre par parties la fonction logarithme népérien en décomposant en $\log(x)*1$

```
(%o11)  $x \log(x) - x$ 
```

```
(%i12) byparts(x*exp(x),x,x,exp(x));
```

```
(%o12)  $x e^x - e^x$ 
```

6.4 Intégrer numériquement avec la méthode de Romberg

La commande `romberg(fonction, variable, borne inférieure, borne supérieure)` est à utiliser dans ce cas (l'entrée `%i17` montre que Maxima ne sait pas intégrer numériquement l'intégrale choisie, ce qui justifie bien a posteriori l'utilisation de Romberg)

```
(%i14) f(x):=sqrt(1+sin(1+x^2));
```

```
(%o14)  $f(x) := \sqrt{1 + \sin(1 + x^2)}$ 
```

```
(%i15) integrate(f(x),x,0,1);
```

```
(%o15)  $\int_0^1 \sqrt{\sin(x^2 + 1) + 1} dx$ 
```

```
(%i16) romberg(f(x),x,0,1);
```

```
(%o16) 1.388657330137037
```

```
(%i17) float(integrate(f(x),x,0,1));
```

```
(%o17)  $\int_{0.0}^{1.0} \sqrt{\sin(x^2 + 1.0) + 1.0} dx$ 
```

6.5 La fonction dilogarithme

Il s'agit de la fonction définie par $x \mapsto \int_1^x \frac{\ln(t)}{1-t} dt$ pour $0 < x < 1$. Elle est définie en standard dans Maxima et se note `li[2]` :

Remarque : cette fonction est de manière équivalente définie aussi par (voir `%i28`)

$$x \mapsto - \int_0^x \frac{\ln(1-t)}{t} dt$$

`(%i25) li[2](1/2);`

`(%o26)` $\frac{\pi^2}{12} - \frac{\log(2)^2}{2}$

`(%i29) 'integrate(log(t)/(1-t),t,1,1/2)=expand(integrate(log(t)/(1-t),t,1,1/2));`

`(%o29)` $-\int_{\frac{1}{2}}^1 \frac{\log(t)}{1-t} dt = -\frac{\log(2)^2}{2} + \log(-1) \log(2) + \text{li}_2(2) - \frac{\pi^2}{6}$

`(%i28) diff(li[2](x),x);`

`(%o28)` $-\frac{\log(1-x)}{x}$

7 Statistiques

7.1 Comment étudier une série statistique à une variable

Il faut charger le package `descriptive.mac`. A partir d'une série de données appelée par exemple `data`, le package fournit les commandes suivantes permettant d'analyser cette série :

Indicateur de la série	Commande
Calcul de la moyenne	<code>mean(data)</code>
Médiane	<code>median(data)</code>
Etendue	<code>range(data)</code>
Variance (de la population)	<code>var(data)</code>
Ecart-type	<code>std(data)</code>

Tableau 1.

Dans l'exemple ci-dessous, on génère une série aléatoire de 15 données avec la fonction `random` puis on l'étudie :

`(%i30) data:makelist(random(50),i,1,15);`

`(%o30)` [12, 2, 34, 35, 4, 41, 29, 35, 48, 3, 35, 15, 40, 26, 39]

`(%i32) load(descriptive);`

`(%o32) /usr/share/maxima/5.45.1/share/descriptive/descriptive.mac`

`(%i34) mean(data);float(mean(data));`

`(%o34)` $\frac{398}{15}$

`(%o35)` 26.533333333333333

`(%i36) median(data);`

`(%o36)` 34

`(%i37) range(data);`

`(%o37)` 46

```
(%i40) var(data);float(var(data));
```

```
(%o40)  $\frac{49376}{225}$ 
```

```
(%o41) 219.4488888888889
```

```
(%i43) std(data);float(std(data));
```

```
(%o43)  $\frac{4\sqrt{3086}}{15}$ 
```

```
(%o44) 14.81380737315322
```

8 Algèbre linéaire

8.1 Vecteurs

8.1.1 Comment noter un vecteur

On définit un vecteur par ses coordonnées que l'on note entre deux crochets. Par exemple, la commande `u:[a,b,c]`; définit le vecteur \vec{u} de coordonnées (a, b, c) .

```
(%i45) u: [1,2,3];
```

```
(%o45) [1,2,3]
```

```
(%i46) v: [-1,2,3];
```

```
(%o46) [-1,2,3]
```

```
(%i47) u+v;
```

```
(%o47) [0,4,6]
```

8.1.2 Comment calculer un produit scalaire

Le produit scalaire se note avec un point, précédé et suivi d'un espace :

```
(%i48) a: [1,2,3];
```

```
(%o48) [1,2,3]
```

```
(%i49) b: [0,-1,5];
```

```
(%o49) [0,-1,5]
```

```
(%i50) a . b;
```

```
(%o50) 13
```

8.1.3 Comment calculer un produit vectoriel

Il faut charger le package `vect` par la commande `load("vect")`; Ensuite, le produit vectoriel se note \sim , et le calcul effectif du produit vectoriel s'effectue par la commande `express(vecteur)`;

```
(%i51) load("vect");
```

```
(%o51) /usr/share/maxima/5.45.1/share/vector/vect.mac
```

```
(%i52) [x,y,z] ~ [x1,y1,z1];
```

```
(%o52) ([x,y,z],[x1,y1,z1])
```

```
(%i53) express(%);
```

```
(%o53) [y z1 - y1 z, x1 z - x z1, x y1 - x1 y]
```

8.2 Matrices

8.2.1 Comment écrire une matrice

On définit une matrice A par la commande `A:matrix([1,2,3],[-1,5,2],[4,3,0]);`

```
(%i54) A:matrix([1,2,3],[-1,5,2],[4,3,0]);
```

```
(%o54)  $\begin{pmatrix} 1 & 2 & 3 \\ -1 & 5 & 2 \\ 4 & 3 & 0 \end{pmatrix}$ 
```

8.2.2 Comment extraire une ligne d'une matrice

On entre le nom de la matrice et entre crochets le numéro de la ligne désirée.

```
(%i55) A[1];
```

```
(%o55) [1, 2, 3]
```

8.2.3 Comment extraire une colonne d'une matrice

La fonction `col(matrice,numéro de colonne)` renvoie la colonne désirée :

```
(%i56) col(A,1);
```

```
(%o56)  $\begin{pmatrix} 1 \\ -1 \\ 4 \end{pmatrix}$ 
```

8.2.4 Comment extraire un élément d'une matrice

L'élément de la ligne i et de la colonne j s'obtient avec `matrice[i,j]` :

```
(%i57) A[1,2];
```

```
(%o57) 2
```

```
(%i58) A[3,1];
```

```
(%o58) 4
```

8.2.5 Comment ajouter un élément ou une liste à une liste

Une matrice est constituée de listes. La commande `append(liste1,liste2)` génère une nouvelle liste constituée des éléments des 2 listes (dans l'ordre défini de chaque liste). Les éléments de la liste2 sont ajoutés après ceux de la liste 1. La commande `cons(variable,liste)` construit une nouvelle liste avec `variable` en premier élément :

```
(%i59) append([1,2,3],[x]);
```

```
(%o59) [1, 2, 3, x]
```

```
(%i60) append([1,2,3],[4,5,6]);
```

```
(%o60) [1, 2, 3, 4, 5, 6]
```

```
(%i63) cons(p,[1,2]);
```

```
(%o63) [p, 1, 2]
```

8.2.6 Comment ajouter une colonne ou une ligne à une matrice

Pour ajouter une colonne ou une ligne à une matrice donnée :

```
(%i1) matrix(append([1,2],[p]),append([4,5],[q]));
```

```
(%o1)  $\begin{pmatrix} 1 & 2 & p \\ 4 & 5 & q \end{pmatrix}$ 
```

```
(%i2) matrix(cons(valeurs, [2,0,9]), cons(effectif, [1,1,5]));
```

```
(%o2) ( valeurs 2 0 9
      effectif 1 1 5 )
```

```
(%i3) addrow(matrix([1,2,3], [4,5,6]), [10,20,30]);
```

```
(%o3) ( 1 2 3
      4 5 6
      10 20 30 )
```

8.2.7 Comment transposer une matrice

La fonction `transpose(matrice)` effectue cette opération.

```
(%i5) A:matrix([1,2,3], [-1,5,2], [4,3,0]);
```

```
(%o5) ( 1 2 3
      -1 5 2
      4 3 0 )
```

```
(%i6) transpose(A);
```

```
(%o6) ( 1 -1 4
      2 5 3
      3 2 0 )
```

8.2.8 Trouver une matrice échelonnée réduite avec Maxima

La commande `echelon(matrice)` renvoie une matrice triangulaire supérieure obtenue avec des combinaisons élémentaires de lignes et de colonnes, dont les éléments diagonaux sont égaux à 1.

```
(%i7) A:matrix([1,2,3], [-1,5,2], [4,3,0]);
```

```
(%o7) ( 1 2 3
      -1 5 2
      4 3 0 )
```

```
(%i8) echelon(A);
```

```
(%o8) ( 1 3/4 0
      0 1 8/23
      0 0 1 )
```

8.2.9 Transformer une matrice en un vecteur ligne

Soit la matrice $A = \begin{pmatrix} 3 & 2 \\ 4 & 5 \\ 6 & 1 \end{pmatrix}$. On souhaite la transformer en un vecteur ligne de 6 éléments. Pour

cela, on définit une nouvelle fonction (appelée `flatten_matrix`) qui, appliquée à une matrice, renvoie la liste des éléments de cette dernière :

```
(%i9) A:matrix([3,2], [4,5], [6,1]);
```

```
(%o9) ( 3 2
      4 5
      6 1 )
```

```
(%i10) flatten_matrix(m) := apply(append, args(transpose(m)))$
```

```
(%i11) flatten_matrix(A);
```

```
(%o11) [3, 4, 6, 2, 5, 1]
```

```
(%i12) B:matrix([1,2,3,4],[p,q,r,s]);
```

```
(%o12)  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ p & q & r & s \end{pmatrix}$ 
```

```
(%i13) flatten_matrix(B);
```

```
(%o13) [1, p, 2, q, 3, r, 4, s]
```

8.2.10 Comment générer automatiquement une matrice

On définit une suite double, par exemple $f[i,j]$, qui va générer les coefficients de la matrice. On termine avec la fonction `genmatrix`(suite utilisée, nombre de lignes, nombre de colonnes) :

```
(%i14) f[i,j]:=i^2+j^2;
```

```
(%o14)  $f_{i,j} := i^2 + j^2$ 
```

```
(%i15) genmatrix(f,4,5);
```

```
(%o15)  $\begin{pmatrix} 2 & 5 & 10 & 17 & 26 \\ 5 & 8 & 13 & 20 & 29 \\ 10 & 13 & 18 & 25 & 34 \\ 17 & 20 & 25 & 32 & 41 \end{pmatrix}$ 
```

On peut utiliser cette possibilité pour écrire une matrice sous forme générale :

```
(%i16) genmatrix(a,3,3);
```

```
(%o16)  $\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$ 
```

8.2.11 Comment multiplier deux matrices

La multiplication des matrices se note avec le point (\cdot), et non pas avec l'étoile ($*$) :

```
(%i17) M:matrix([a,b],[c,d]);
```

```
(%o17)  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 
```

```
(%i18) M.M;
```

```
(%o18)  $\begin{pmatrix} bc+a^2 & bd+ab \\ cd+ac & d^2+bc \end{pmatrix}$ 
```

8.2.12 Comment inverser une matrice

La matrice inverse de A se note $A^{(-1)}$:

```
(%i19) M:matrix([a,b],[c,d]);
```

```
(%o19)  $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ 
```

```
(%i20) M^{(-1)};
```

```
(%o20)  $\begin{pmatrix} \frac{d}{ad-bc} & -\frac{b}{ad-bc} \\ -\frac{c}{ad-bc} & \frac{a}{ad-bc} \end{pmatrix}$ 
```

8.2.13 Comment calculer une puissance de matrices

La matrice A^n se note $A^{\wedge}n$:

```
(%i21) M:matrix([u,v],[w,t]);
```

```
(%o21) ( u v )
      ( w t )
```

```
(%i22) M^2;
```

```
(%o22) ( vw+u^2 uv+tv )
      ( uw+tw vw+t^2 )
```

```
(%i23) M^3;
```

```
(%o23) ( u(vw+u^2)+(uv+tv)w v(vw+u^2)+t(uv+tv) )
      ( v(vw+t^2)+u(uw+tw) t(vw+t^2)+v(uw+tw) )
```

```
(%i24) expand(%);
```

```
(%o24) ( 2uvw+tvw+u^3 v^2w+u^2v+tuv+t^2v )
      ( vw^2+u^2w+tuw+t^2w uvw+2tvw+t^3 )
```

8.3 Comment résoudre une récurrence linéaire

Le package `solve_rec` permet dans certains cas d'exprimer, pour une suite récurrente linéaire, le terme d'ordre n en fonction de n . Pour cela, on dispose de la fonction `solve_rec`(définition suite récurrente,suite,valeur initiale) :

```
(%i2) load(solve_rec);
```

```
(%o2) /usr/share/maxima/5.45.1/share/solve_rec/solve_rec.mac
```

Etude de la suite récurrente définie par :

$$u_n = \frac{n u_{n-1}}{1+n}$$

```
(%i3) solve_rec(u(n)=n*u(n-1)/(1+n),u(n));
```

```
(%o3) u(n) = %k1 / (n+1) (%k1 désigne une constante quelconque)
```

On rajoute la condition initiale $u_1=3$:

```
(%i4) solve_rec(u(n)=n*u(n-1)/(1+n),u(n),u(1)=3);
```

```
(%o4) u(n) = 6 / (n+1)
```

Etude de la suite récurrente d'ordre 2 définie par :

$$u_n = u_{n-1} + u_{n-2}$$

```
(%i28) solve_rec(u(n)=u(n-1)+u(n-2),u(n));
```

```
(%o28) u(n) = ((sqrt(5)-1)^n %k1 (-1)^n) / 2^n + ((sqrt(5)+1)^n %k2) / 2^n
```

puis avec les conditions initiales $u_0 = 1$ et $u_2 = 5$:

```
(%i20) solve_rec(u(n)=u(n-1)+u(n-2),u(n),u(0)=1,u(2)=5);
```

```
(%o20) u(n) = ((sqrt(5)+1)^n (7*sqrt(5)+5)) / 10*2^n - ((sqrt(5)-1)^n (7*sqrt(5)-5) (-1)^n) / 10*2^n
```

9 Graphisme

9.1 Courbes planes cartésiennes

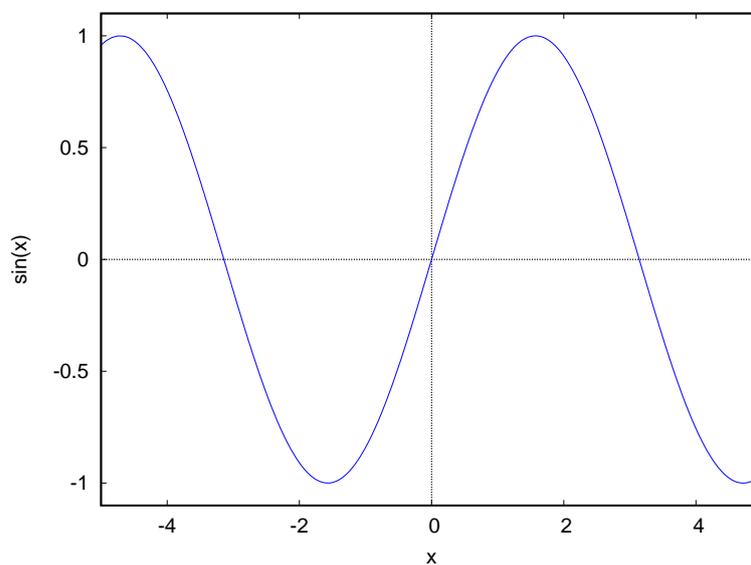
9.1.1 Représenter une fonction

On fait appel à la fonction `plot2d(fonction,[variable, min, max])`. Il est conseillé de terminer la commande par `$` plutôt que par `;` pour éviter les confirmations de Maxima.

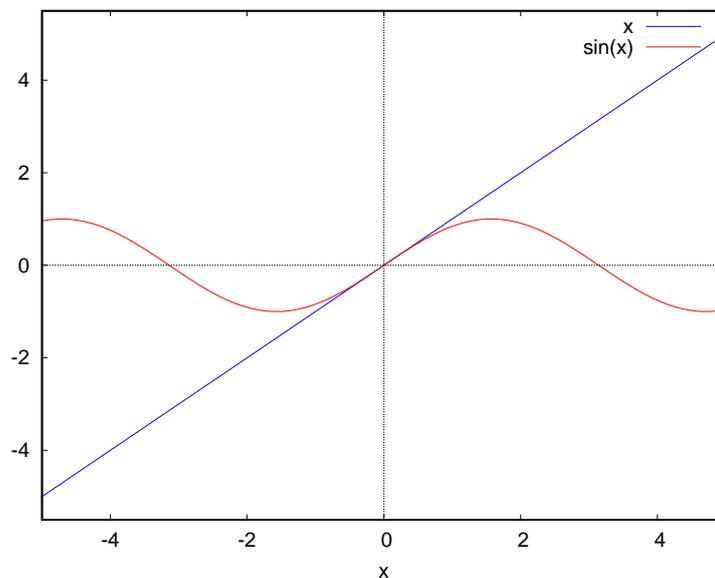
Remarque 1 : uniquement pour TeXmacs, il faut faire précéder la commande graphique par `tm_` pour qu'elle s'affiche à l'intérieur du document.

Remarque 2 : si on utilise l'interface wxMaxima, on peut utiliser la commande `wxplot2d` qui permet d'insérer le graphique dans le notebook.

```
(%i10) tm_plot2d(sin(x), [x, -5, 5])$
```



```
(%i8) tm_plot2d([x, sin(x)], [x, -5, 5])$
```



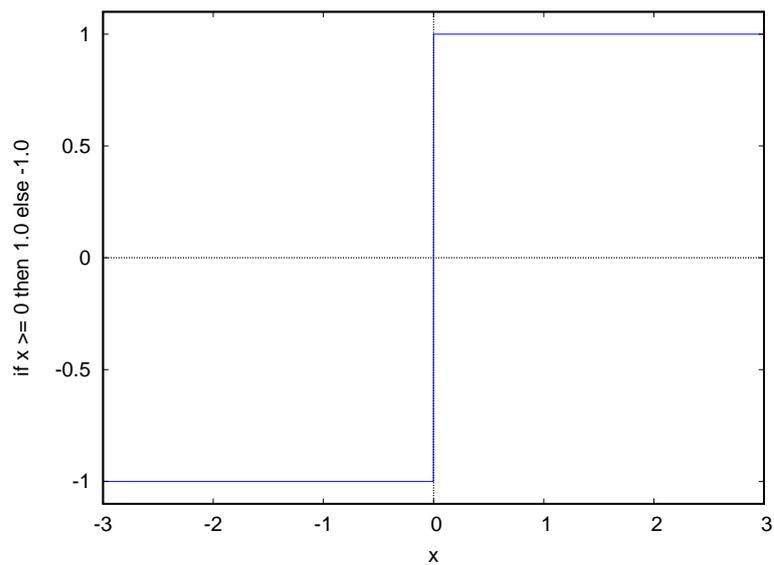
9.1.2 Représenter une fonction en escalier

Maxima renvoie un message d'erreur si l'on essaye de représenter une fonction en escalier. Pour y arriver, il faut utiliser une évaluation différée (ce qui se fait en précédant la fonction par `'`). Par exemple, représentons la fonction f définie sur \mathbb{R} par $f(x) = -1$ si $x < 0$ et $f(x) = 1$ si $x \geq 0$:

```
(%i11) f(t):=if t>=0 then 1.0 else -1.0;
```

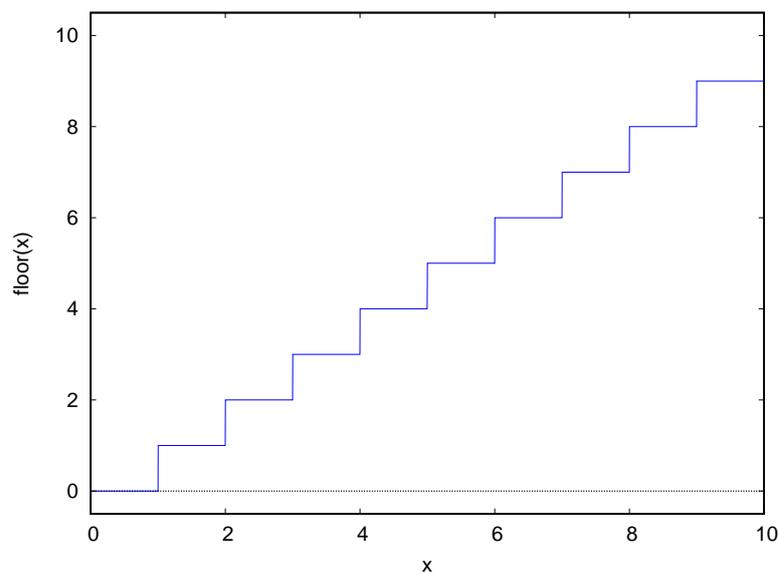
```
(%o11) f(t) := if t ≥ 0 then 1.0 else - 1.0
```

```
(%i13) tm_plot2d('(f(x)), [x, -3, 3])$
```



9.1.3 Représenter la fonction partie entière

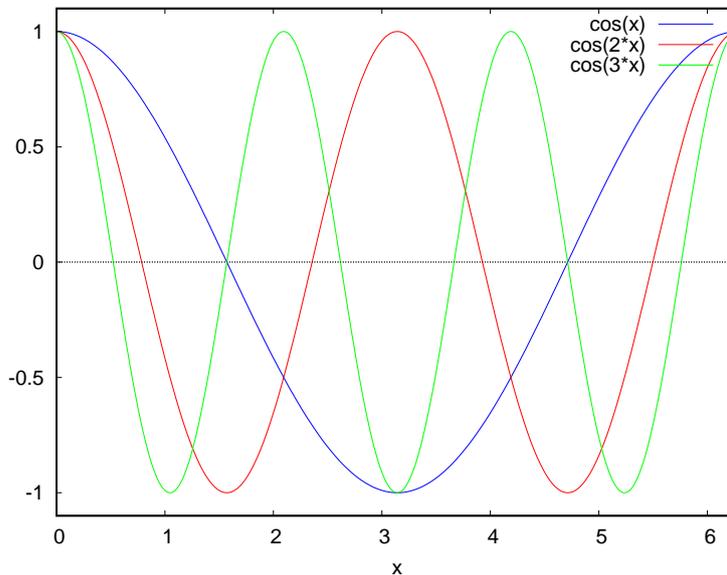
```
(%i14) tm_plot2d('(float(entier(x))), [x, 0, 10])$
```



9.1.4 Représenter une famille de fonctions

Pour représenter la famille de fonctions $x \rightarrow \cos(n x)$, pour n variant de 1 à 3, on entre la commande :

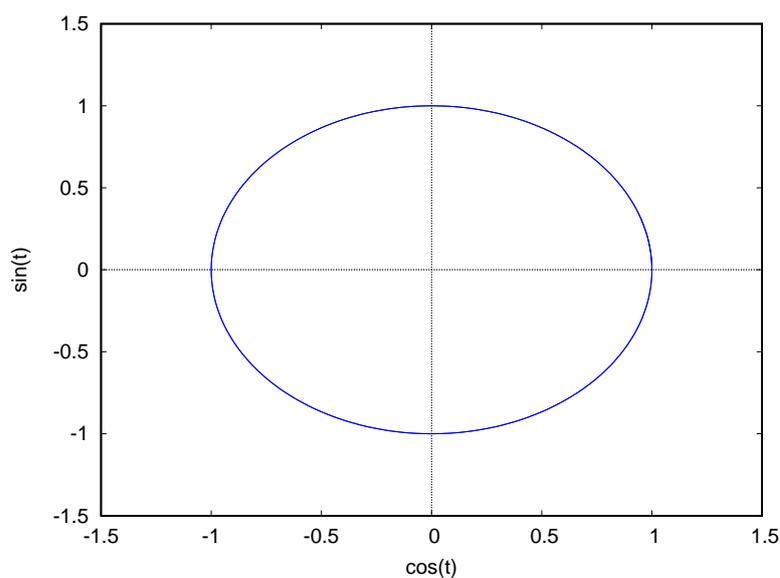
```
(%i15) tm_plot2d(makelist(cos(n*x),n,1,3),[x,0,2*%pi])$
```



9.2 Comment représenter un cercle

Pour tracer le cercle d'équation $x^2 + y^2 = 1$, on utilise la représentation paramétrique $x = \cos(t)$ et $y = \sin(t)$. Pour ce faire, la commande plot2d prend le paramètre parametric en entrée :

```
(%i23) tm_plot2d([parametric,cos(t),sin(t)],[t,-%pi*2,%pi*2]],[x,-1.5,1.5],[y,-1.5,1.5])$
```



```
(%i24)
```

9.3 Courbes planes polaires

9.3.1 Comment tracer une courbe en polaire

Cette solution est proposée par Stavros Macrakis : on entre le programme suivant :

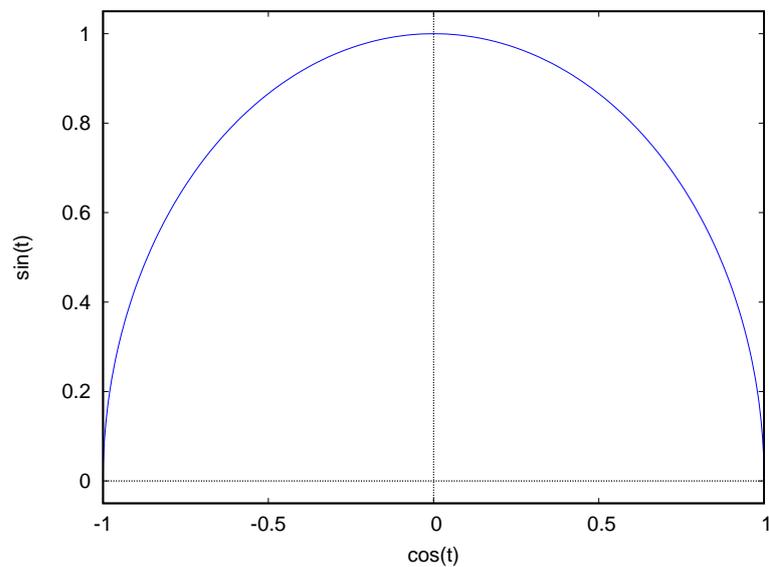
Remarque : il faut adapter ce programme car il est écrit pour fonctionner avec Texmacs tout simplement en enlevant `tm_` avant le `plot2d`

```
(%i31) plot_polar(expr,range) := block([theta_var: range[1]], tm_plot2d(
    ['parametric, cos(theta_var)*expr, sin(theta_var)*expr, range]));
```

```
(%o31) plot_polar(expr, range) := block ([theta_var: range[1], tm_plot2d(['parametric,
cos(theta_var) expr, sin(theta_var) expr, range]))
```

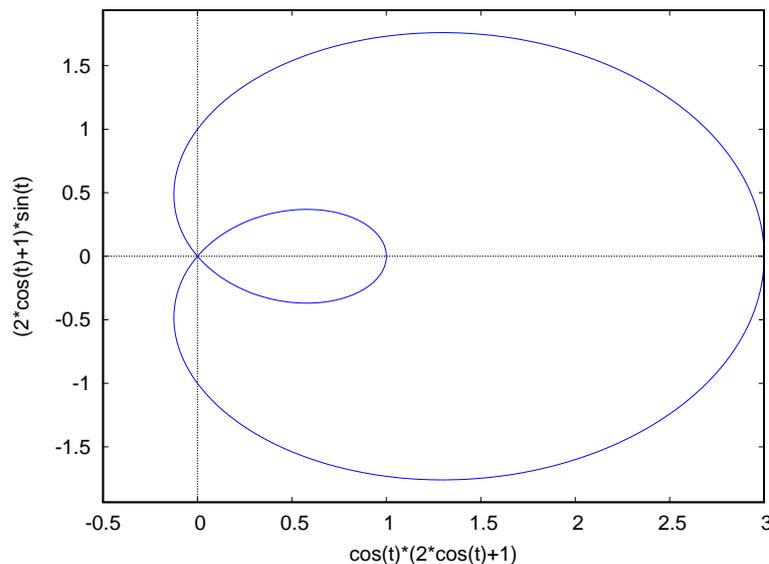
Le tracé de la courbe d'équation $\rho=1$ sur $[0, \pi]$ s'obtient par :

```
(%i35) plot_polar(1, [t,0,%pi])$
```



Le tracé d'une cardioïde s'obtient par :

```
(%i37) plot_polar( 1 + 2 * cos(t), [t,0,2*pi])$
```

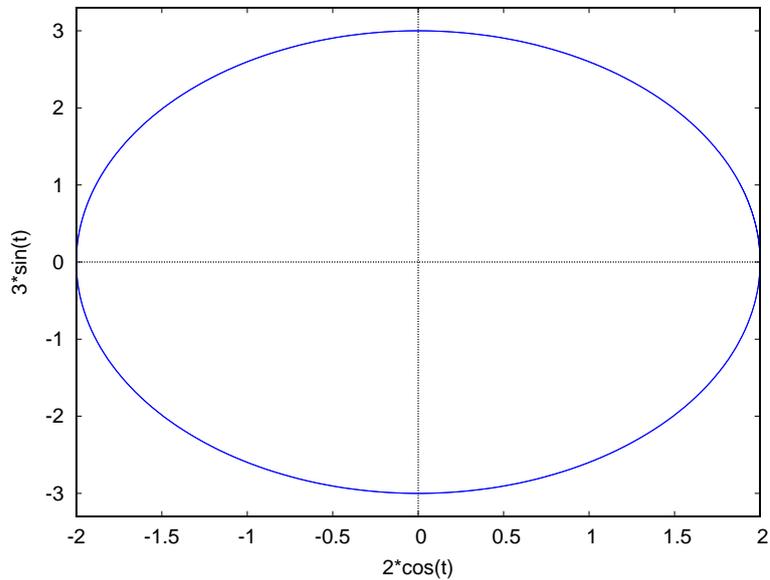


9.4 Courbes planes paramétriques

9.4.1 Comment tracer une courbe en paramétrique ?

On utilise la fonction `plot2d` (voir paragraphe précédent) avec le paramètre `parametric` :

```
(%i41) tm_plot2d([parametric,2*cos(t),3*sin(t),[t,-%pi*2,%pi*2]])$
```



10 Entrées et sorties

10.1 Variables réservées

10.1.1 Comment noter plus l'infini ?

Maxima utilise la variable réservée `inf` :

```
(%i1) limit((x+1)/(x+5),x,inf);
```

```
(%o1) 1
```

```
(%i2) integrate(exp(-x^2),x,0,inf)
```

```
(%o2)  $\frac{\sqrt{\pi}}{2}$ 
```

10.1.2 Comment noter moins l'infini ?

Maxima utilise la variable réservée `minf` :

```
(%i3) limit(exp(x),x,minf)
```

```
(%o3) 0
```

10.2 Effacer la valeur d'une variable

La commande `kill(variable)` efface la définition précédemment entrée :

```
(%i4) x:3+sqrt(5);
```

```
(%o4)  $\sqrt{5} + 3$ 
```

```
(%i5) x;
```

```
(%o5)  $\sqrt{5} + 3$ 
```

```
(%i6) kill(x);
```

```
(%o6) done
```

```
(%i7) x;
```

```
(%o7)  $x$ 
```

10.3 Fonctions usuelles

10.3.1 Comment écrire la fonction logarithme népérien

Elle se note log sous Maxima :

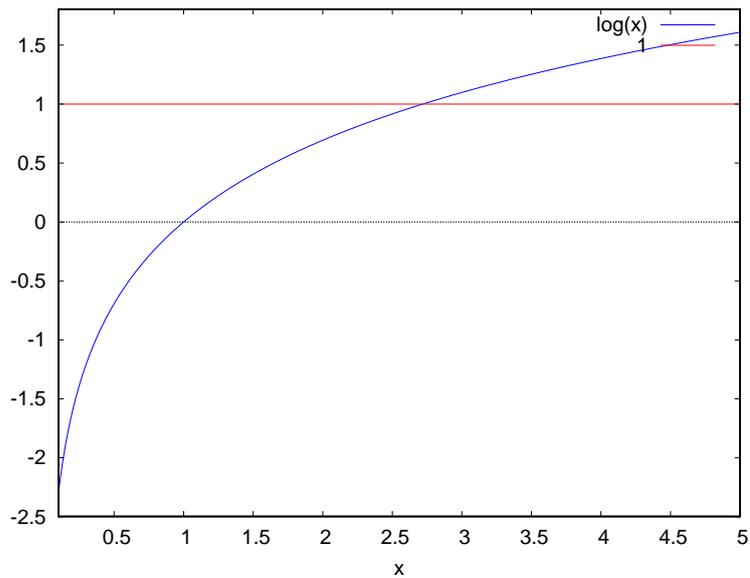
```
(%i8) log(1);
```

```
(%o8) 0
```

```
(%i9) log(exp(x));
```

```
(%o9)  $x$ 
```

```
(%i14) tm_plot2d([log(x),1],[x,0.1,5])$
```



```
(%i18) diff(log(x),x);
```

```
(%o18)  $\frac{1}{x}$ 
```

```
(%i19)
```

10.3.2 Comment définir la fonction logarithme décimal

```
(%i15) log10(x):=log(x)/log(10);
```

```
(%o15)  $\log_{10}(x) := \frac{\log(x)}{\log(10)}$ 
```

```
(%i16) log10(10);
```

```
(%i17) diff(log10(x),x);
```

```
(%o17)  $\frac{1}{\log(10)x}$ 
```

10.3.3 Comment définir une fonction à partir d'une dérivée

On veut définir la fonction dérivée d'une fonction f , pour pouvoir la réutiliser :

```
(%i23) expression:x^2+1/x;
```

```
(%o23)  $x^2 + \frac{1}{x}$ 
```

```
(%i24) define(f(x),expression);
```

```
(%o24)  $f(x) := x^2 + \frac{1}{x}$ 
```

```
(%i25) define(df(x),diff(f(x),'x));
```

```
(%o25)  $df(x) := 2x - \frac{1}{x^2}$ 
```

```
(%i26) df(1);
```

```
(%o26) 1
```

10.4 Comment transformer une expression en fonction

Si on a défini une expression appelée expr, on peut définir une fonction f à l'aide de la commande $f(x):=(expr)$; On utilise deux fois la simple quote. Une autre syntaxe possible est $define(f(x),expr)$; Par exemple :

```
(%i27) expr:logcontract(integrate(2/(x^2-1),x));
```

```
(%o27)  $\log\left(\frac{x-1}{x+1}\right)$ 
```

```
(%i28) f(x):=''(expr);
```

```
(%o28)  $f(x) := \log\left(\frac{x-1}{x+1}\right)$ 
```

```
(%i29) f(5);
```

```
(%o29)  $\log\left(\frac{2}{3}\right)$ 
```

```
(%i30) expand(factor(diff(f(x),x)));
```

```
(%o30)  $\frac{2}{x^2-1}$ 
```

```
(%i34) define(g(x),expand(factor(diff(f(x),x))));
```

```
(%o34)  $g(x) := \frac{2}{x^2-1}$ 
```

```
(%i35) g(5);
```

```
(%o35)  $\frac{1}{12}$ 
```

```
(%i37) logcontract(integrate(g(x),x));
```

```
(%o37) log( $\frac{x-1}{x+1}$ )
```

10.5 Définir une fonction de manière formelle

Il est possible de définir une fonction f en précisant uniquement la variable dont elle dépend. Cela permet d'effectuer notamment du calcul différentiel avec des fonctions indéfinies. La commande à utiliser est dans ce cas `depends(fonction, variable)` :

```
(%i38) depends(f,x);
```

```
(%o38) [f(x)]
```

```
(%i39) depends(g,x);
```

```
(%o39) [g(x)]
```

```
(%i40) diff(f*g,x);
```

```
(%o40)  $f\left(\frac{d}{dx}g\right) + \frac{d}{dx}fg$ 
```

```
(%i41) diff(f/g,x);
```

```
(%o41)  $\frac{\frac{d}{dx}f}{g} - \frac{f\left(\frac{d}{dx}g\right)}{g^2}$ 
```

```
(%i42) ratsimp(%);
```

```
(%o42)  $-\frac{f\left(\frac{d}{dx}g\right) - \frac{d}{dx}fg}{g^2}$ 
```

10.6 Comment récupérer un élément d'une liste

Les commandes `first(liste)`, `second(liste)`, `third(liste)` renvoient respectivement les premier, deuxième et troisième élément de la liste nommée liste indiquée dans l'argument. Par exemple :

```
(%i43) first([a,b,c,d]);
```

```
(%o43) a
```

```
(%i44) second([a,b,c,d]);
```

```
(%o44) b
```

```
(%i45) third([a,b,c,d]);
```

```
(%o45) c
```

10.7 Comment additionner les éléments d'une liste

```
(%i46) apply("+", [1,5,7,-1]);
```

```
(%o46) 12
```

10.8 Comment simplifier une expression trigonométrique

On peut utiliser la fonction `trigsimp(expression)` :

```
(%i47) trigsimp(tan(x));
```

```
(%o47)  $\frac{\sin(x)}{\cos(x)}$ 
```

```
(%i48) tan(x);
```

```
(%o48)  $\tan(x)$ 
```

10.9 Comment tester une égalité

La fonction `is(equal(expression1,expression2))` renvoie vrai ou faux selon que l'égalité est vérifiée. Lorsque Maxima ne sait pas statuer, il renvoie la réponse `unknown` :

```
(%i49) is(equal(4,2+2));
```

```
(%o49) true
```

```
(%i50) is(equal((x+1)^2,x^2+2*x+1));
```

```
(%o50) true
```

```
(%i51) is(equal((x+1)^2,x^2+1));
```

```
(%o51) unknown
```

```
(%i53) is(equal(5,1+3));
```

```
(%o53) false
```

10.10 Comment déclarer qu'un nombre est entier

La commande `declare(n,integer)` le réalise facilement.

```
(%i56) sin(n*%pi);
```

```
(%o56)  $\sin(\pi n)$ 
```

```
(%i57) declare(n,integer);
```

```
(%o57) done
```

```
(%i58) sin(n*%pi);
```

```
(%o58) 0
```

10.11 Comment déclarer qu'un nombre est positif

La commande `assume(x>0)` permet à Maxima de savoir que le réel x est positif. Le renseignement sera utilisé dans les calculs :

```
(%i66) abs(x);
```

```
(%o66)  $|x|$ 
```

```
(%i67) assume(x>0);
```

```
(%o67)  $[x > 0]$ 
```

```
(%i68) abs(x);
```

```
(%o68)  $x$ 
```

```
(%i69) kill(x);
```

```
(%o69) done
```

10.12 Comment savoir si une expression contient une variable

La commande `freeof(x,expression)` renvoie `true` si `expression` contient `x`, et `false` dans le cas contraire.

```
(%i70) freeof(x,2*a+5*y^2);
```

```
(%o70) true
```

```
(%i71) freeof(x,x^2-5*y+6);
```

```
(%o71) false
```

Attention, la commande n'est pas récursive, et donc le résultat renvoyé est faux si l'expression a été définie précédemment :

```
(%i72) x:5*a+2;
```

```
(%o72) 5 a + 2
```

```
(%i73) freeof(x,a);
```

```
(%o73) true
```

10.13 Etiqueter une sous-expression

La commande `pickapart(expression, profondeur)` attribue des étiquettes d'expression intermédiaires aux sous-expressions de `<expression>` à une profondeur `<n>`, `n` étant un entier.

Les sous-expressions à des profondeurs supérieures ou inférieures ne reçoivent pas d'étiquette.

`pickapart` renvoie une expression en termes d'expressions intermédiaires équivalente à l'expression originale `<expression>`.

```
(%i82) expression:(sqrt(7)-1)/(1+sqrt(3));
```

```
(%o82)  $\frac{\sqrt{7}-1}{\sqrt{3}+1}$ 
```

```
(%i87) pickapart(expression,1);
```

```
(%o87)  $\frac{\%t75}{\%t76}$ 
```

```
(%i88) pickapart(expression,2);
```

```
(%o88)  $\frac{\%t80-1}{\%t81+1}$ 
```

```
(%i89) pickapart(expression,3);
```

```
(%o89)  $\frac{\sqrt{7}-1}{\sqrt{3}+1}$ 
```

10.14 Comment exporter en tex

La commande `tex(expression)` renvoie l'expression codée en tex (on rajoute à la fin de la commande `$` plutôt que `;` afin d'éviter les messages de Maxima) :

```
(%i91) tex(sqrt(2)+1/2)$
```

```
$$\sqrt{2}+{\1}\over{2}$$
```

```
(%i95) 'integrate(1/(x+1),x)=integrate(1/(x+1),x);
```

```
(%o95)  $\int \frac{1}{x+1} dx = \log(x+1)$ 
```

```
(%i98) tex(integrate(1/(x+1),x))$
$$\log \left(x+1\right)$$
```

10.15 Comment choisir un fichier pour l'exportation

Avec wxMaxima, il suffit d'exporter la feuille de calcul du notebook en tex avec le menu Fichier → exporter. On peut aussi exporter le fichier au format html.

Il est possible de réaliser une exportation directement à partir de commandes Maxima. Pour cela, on utilise un fichier dont le nom est contenu dans une variable. Pour l'utiliser avec la commande tex, on utilise les syntaxes suivantes :

```
(%i99) nomfichier:"test.tex";
(%o99) test.tex
(%i100) tex(sqrt(3)+sqrt(2),'nomfichier);
(%o100) false
```

La commande précédente sauve l'expression $\sqrt{3} + \sqrt{2}$ au format tex dans le fichier test.tex défini précédemment, et ceci dans le répertoire courant.

On peut aussi indiquer le chemin sur le disque quand on définit le nom du fichier afin d'effectuer la sauvegarde dans un répertoire donné.

10.16 Comment charger des commandes au lancement de maxima

Maxima charge automatiquement un fichier de configuration nommé **maxima-init.mac** (ou **maxima.rc** sous Linux) situé dans le dossier utilisateur. C'est dans ce fichier (qu'il faut éventuellement créer s'il n'existe pas) que les commandes à charger sont à écrire sous forme de liste de commandes. Par exemple, on peut charger automatiquement un package donné.

10.17 Comment utiliser une fonction comme argument

Voici les codes proposés par Barton Willis pour le calcul de :

$$\text{xsum}(f, n) = \sum_{k=1}^n f(k)$$

où f est une fonction donnée et n un entier. Les tests se font avec la fonction identité puis la fonction carrée.

```
(%i101) xsum(f,n):=block([s:0],for k:1 thru n do (s:s + apply(f,[k])),s)$
(%i102) f(x):=x$
(%i103) g(x):=x^2$
(%i104) xsum(f,2);

(%o104) 3
(%i105) xsum(g,3);

(%o105) 14
```

Autre solution, utilisant la commande translate :

```
(%i106) ysum(f,n):=block([s:0],for k:1 thru n do (s:s+f(k)),s)$
(%i107) ysum(f,2);

(%o107) 3
```

```
(C12) ysum(g,3);
```

```
(D12) 6 (On constate ici une erreur)
```

```
(%i108) translate(ysum)$
```

```
warning: f is a bound variable in f(k), but it is used as a function.
note: instead I'll translate it as: apply(f,[k])
```

```
(%i109) ysum(g,3);
```

```
(%o109) 14
```

10.18 Comment composer n fois une fonction

On définit une procédure qui permet de composer n fois une fonction (astuce de Stavros Macrakis) :

```
(%i1) compose_n(f, x, n) := if n = 0 then x else f(compose_n(f, x, n-1));
```

```
(%o1) compose_n(f, x, n) := if n = 0 then x else f(compose_n(f, x, n - 1))
```

```
(%i2) f(x) := x^2 + 1;
```

```
(%o2) f(x) := x^2 + 1
```

```
(%i3) compose_n(f, x, 3);
```

```
(%o3) ((x^2 + 1)^2 + 1)^2 + 1
```

```
(%i4) expand(compose_n(f, x, 3));
```

```
(%o4) x^8 + 4x^6 + 8x^4 + 8x^2 + 5
```

```
(%i5) kill(f);
```

```
(%o5) done
```

```
(%i6) compose_n(f, x, 3);
```

```
(%o6) f(f(f(x)))
```

10.19 Comment collecter des termes d'une expression

On veut exprimer un polynôme de plusieurs variables en fonction des puissances de l'une d'entre elles (astuce de Barton Willis) :

```
(%i117) p : expand((a+b+c)^3);
```

```
(%o117) c^3 + 3bc^2 + 3ac^2 + 3b^2c + 6abc + 3a^2c + b^3 + 3ab^2 + 3a^2b + a^3
```

```
(%i118) collectterms(p, c);
```

```
(%o118) c^3 + (3b + 3a)c^2 + (3b^2 + 6ab + 3a^2)c + b^3 + 3ab^2 + 3a^2b + a^3
```

```
(%i119) facsum(p, c);
```

```
(%o119) c^3 + 3(b + a)c^2 + 3(b + a)^2c + (b + a)^3
```

10.20 Exporter en Latex

Par défaut, Maxima dispose d'une fonction pour générer du TeX. Pour obtenir du Latex, il suffit d'utiliser le package mactex-utilities :

```
(%i114) tex(x/y)$  
$$\frac{x}{y}$$  
(%i115) load("mactex-utilities");  
(%o115) /usr/share/maxima/5.45.1/share/utls/mactex-utilities.lisp  
(%i116) tex(x/y)$  
$$\frac{x}{y}$$
```