utbm

MAXIMA 5

Manuel d'utilisation

Utilisation de MAXIMA

1°) Aide de Maxima

Charger dans le navigateur intégé à Maxima l'url suivante :
http://www.univ-orleans.fr/EXT/ASTEX/astex/doc/fr/maxima/htmla/maxima.html
Sinon, une aide est intégrée directement à Maxima mais elle est en anglais.
La plus utile des commandes de l'aide en ligne est **describe** qui fournit de l'aide sur toutes les commandes contenant une chaîne particulière.
Exemple :

```
(C1) describe("sum");
```

Comme raccourci vous pouvez taper **? sum** au lieu de **describe("sum").**

2°) Expressions sous Maxima

```
(C1) Expr : x^3+2*x^2-3*x+1 ;
```

Il faut obligatoirement le "**;**" pour afficher et interpréter le résultat. Pour masquer et interpréter le résultat mettre un "**$**" à la place du "**;**"**.**

3°) Variables sous Maxima

```
(C1) a : expression ;
```

Affecte à la variable **a** la valeur défini par expression.
Pour éviter tous problèmes de variables réutilisées faîtes un restart accessible par le menu File (avec l'interface xmaxima).

4°) Diverses commandes utiles

% permet de rappeler le dernier résultat calculé.

Chaque commande s'écrit après une invitation noté (Cnombre).
Exemple :

```
(C1) A : 2+3 ;
```

Le résultat correspondant sera alors noté (Dnombre). Ici ce sera :

```
(D1) 5
```

On peut ainsi rappeler les résultats précédents de cette manière :

```
(C5) B : 3+(D18) ;
```

Opérateurs

Opérateur "!"

C’est l'opérateur factoriel qui est le produit de tous les entiers de 1 à son argument. Ainsi 5! = 1*2*3*4*5 = 120.

Opérateur "!!"

Il représente le double factoriel qui est défini comme produit de tous les entiers consécutifs impairs (ou pairs) de 1 (ou 2) à l'argument impair (ou pair). Ainsi 8!! est 2*4*6*8 = 384.

Opérateur "#"

Est l'opérateur logique "Non égal".

Opérateur "."

Est l'opérateur point, pour la multiplication (non-commutative) de matrices. Lorsque "." est utilisé de cette façon, des espaces devront être laissés des deux côtés, e.g. A . B.

Opérateur ":"

L'opérateur d'affectation. Par exemple A:3 donne à la variable A la valeur 3.

Opérateur ":="

L'opérateur de définition de fonction. Par exemple F(X):=SIN(X) définit une fonction F.

Opérateur "="

Indique une équation de MAXIMA.

Opérateur "^"

Est l’opérateur puissance.

Opérateurs usuels

+, -, *, /, <, <=, >, >= sont des opérateurs courants sous maxima.

Constantes

Constante "%PI"

Donne la valeur de Pi

Constante "%E"

Donne la valeur de e (=2.71)

Constante "%I"

Donne la racine complexe de -1

Constante "TRUE"

Donne la valeur booléenne “VRAI”

Constante "FALSE"

Donne la valeur booléenne “FAUX”

Constante "INF"

Donne la “valeur” infini

Constante "MINF"

Donne la “valeur” moins infini

Fonctions Mathématiques

Maxima reconnaît les fonctions mathématiques suivantes :

- `exp(x)`, `log(x)`,
- `?round(x)`, `?truncate(x)`
- `sqrt(x)`, `abs(x)`
- `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, `acos(x)`, `atan(x)`
- `sinh(x)`, `cosh(x)`, `tanh(x)`, `asinh(x)`, `acosh(x)`, `atanh(x)`

Nombres réels

1°) Calcul sur les réels

- **`sqrt(x)`** retourne la racine carrée positive de x
- **`abs(x)`** retourne la valeur absolue de x
- **`?round(x)`** retourne la partie entière de x
- **`?truncate(x)`** supprime la partie décimale de x

2°) Calcul sur les entiers

- **`gcd(a, b)`** retourne le plus petit diviseur commun entre a et b
- **`lcm(a, b)`** retourne le plus grand multiple commun entre a et b. Attention, charger la librarie *FUNCTS* : **`LOAD(FUNCTS) ;`**
- **`random(n)`** retourne un entier aléatoire entre 0 et n-1
- **`quotient(a, b)`** retourne le quotient de la division euclidienne de a par b
- **`mod(a, b)`** retourne le reste de la division euclidienne de a par b
- **`primep(n)`** retourne *TRUE* si le nombre est premier *FALSE* sinon
- **`factor(n)`** décompose le nombre n en nombres premier s

Les polynômes

Soit P un polynôme et soit Q = 1/ P

- **`factor(P)`** factorise le polynôme P
- **`expand(P)`** développe le polynôme P
- **`partfrac(Q, X)`** décompose la fraction rationnelle Q en éléments simples.
- **`divide(P1, P2, X)`** calcule le quotient et le reste du polynôme P1 divisé par le polynôme P2. Le résultat est une liste dont le premier élément est le quotient et le second élément le reste.
- **`gfactor(P)`** factorise le polynôme P dans l'ensemble complexe C

Sommation et produit

1°) Sommation

- **`sum(expr, var, start, end)`**

Retourne la somme de l'expression *expr* par rapport à la variable *var* entre les nombres *start* et *end*.

Exemple

```
(C1) sum(k^2, k, 1, 100) ;
```

Ici on retourne la somme des carrées entre 1 et 100

2°) Produit

- **`product(expr, var, start, end)`**

Retourne le produit de l'expression *expr* par rapport à la variable *var* entre les nombres *start* et *end*.

Exemple

```
(C1) product(k^2, k, 1, 100) ;
```

Ici on retourne le produit des carrées entre 1 et 100

Les matrices

1°) Définition

Exemple de définition de la matrice M :

```
(C20) M:matrix([4,-2,-2],[-2,4,-2],[-2,-2,4]);
```

2°) Opérations

Soit les matrices A et B

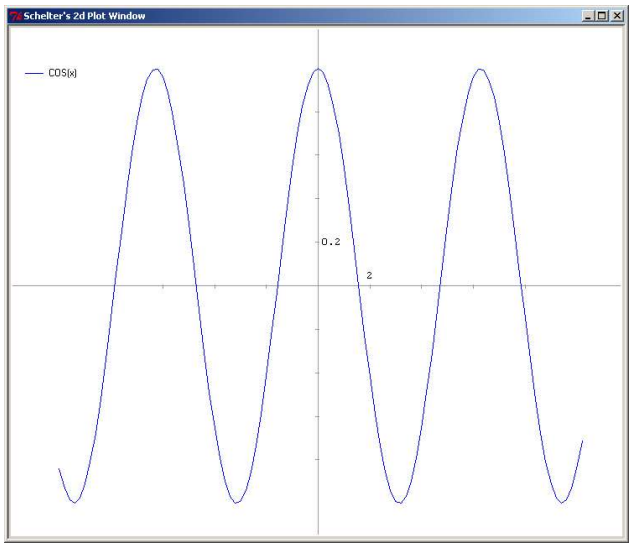
- Somme : A + B
- Produit Matriciel : A*B
- **`determinant(A)`** retourne le déterminant de la matrice A
- **`invert(A)`** retourne la matrice inverse de A
- **`transpose(A)`** retourne la matrice transposée de A
- **`mattrace(A)`** retourne la trace de la matrice A. Attention, il faut chargé la librairie *nchrpl* : **`LOAD(nchrpl) ;`**
- **`rank(A)`** retourne le rang de la matrice A
- **`expand(charpoly(A, x))`** calcule le polynôme caractéristique de la matrice A
- **`eigenvalues(A)`** retourne les valeurs propres de la matrice A
- **`eigenvectors(A)`** retourne les vecteurs propres de la matrice A

Tracé de courbes en 2D

1°) En coordonnées cartésiennes

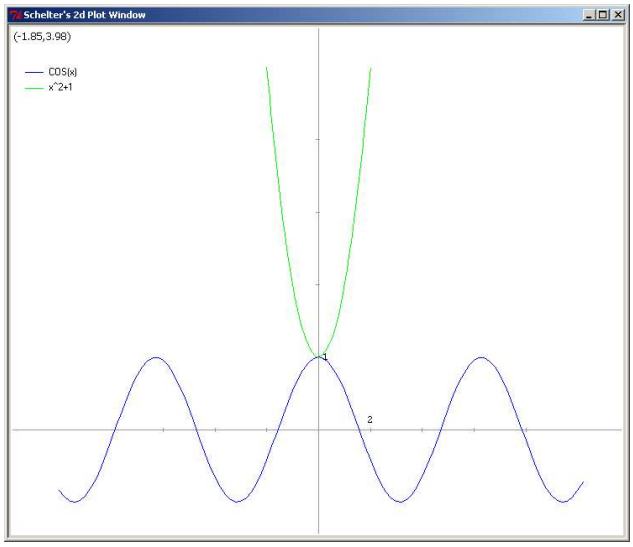
(C1) plot2d([cos(x)],[x,x1,x2],[y,y1,y2]);

Trace la courbe y=cos(x) pour x variant de x1 à x2 et y variant de y1 à y2. La variance de y est optionnelle.



(C2) plot2d([cos(x),(x^2+1)],[x,x1,x2]);

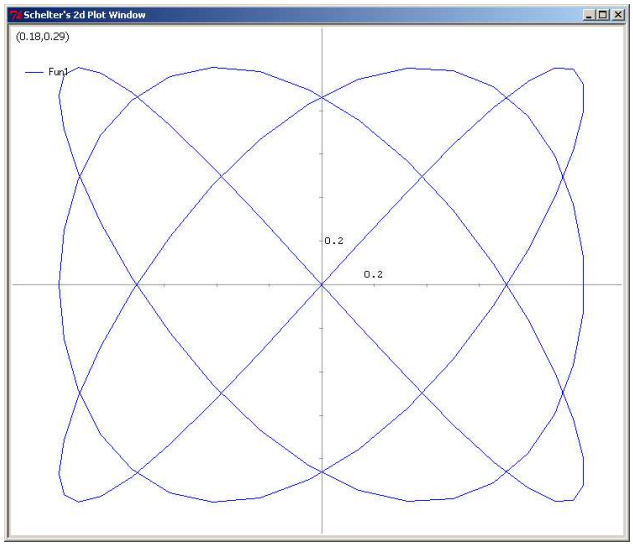
Trace les courbes y1=cos(x) et y2=x^2+1 sur le même graphe pour x variant de x1 à x2



2°) En coordonnées paramétriques

```
(C3) plot2d([parametric,cos(3*t), sin(4*t), [t,-5,5]]);
```

Trace la courbe paramétrée x=cos(3t) et y=sin(4t) pour t variant de -5 a 5



3°) En coordonnées polaires

Il faut tout d'abord charger la fonction polarplot qui permet de tracer une courbe en coordonnées polaires.
Pour cela placer le dossier mt26 dans le dossier

- **C:\Program Files\Maxima-5.9.0\share\maxima\5.9.0\share** pour Windows
- **/usr/share/maxima/5.9.0/share/** pour Linux

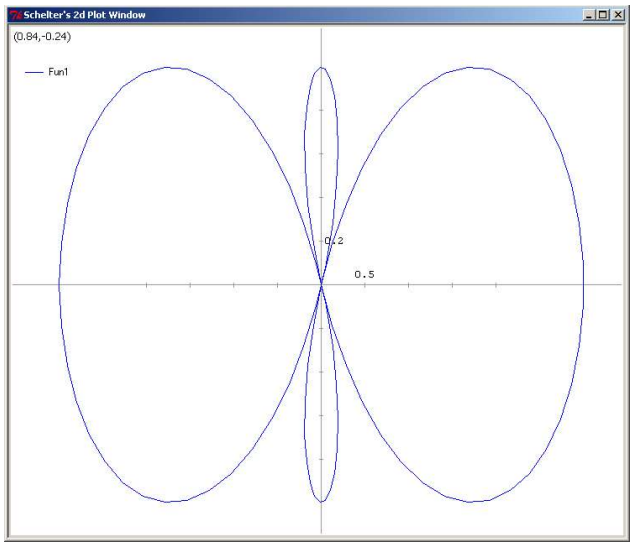
Ensuite taper :

```
(C1) LOAD("mt26/mt26.mc");
```

Maintenant vous êtes capables de tracer des courbes polaires
Exemple

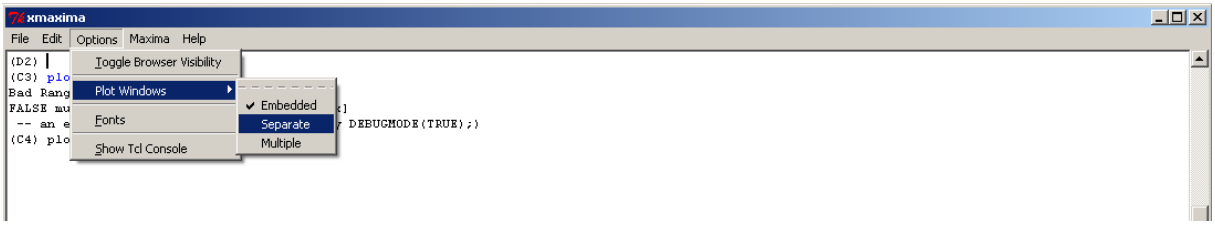
```
(C2) polarplot(sin(3*t)/sin(t),[t,-5,5]);
```

Trace la courbe polaire r=sin(3*t)/sin(t) pour t variant de -5 a 5

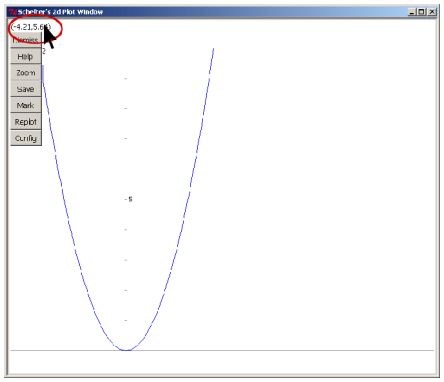


4°) Options

- Il est est possible de séparer la fenêtre graphique de la fenêtre de maxima. Pour cela allez dans le menu Options > Plots Windows > Separate (seulement avec l'interface xmaxima)



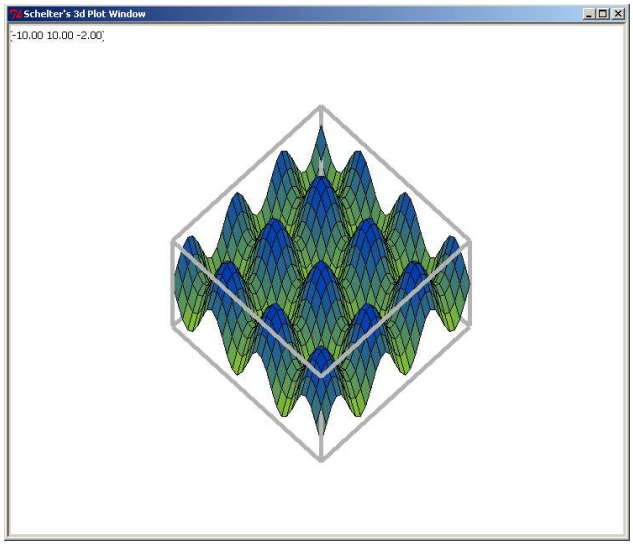
- Vous pouvez appliquer plusieurs options sur le graphique. Pour cela, amenez la souris en haut à gauche sur les coordonnées pour faire apparaître le menu des options.



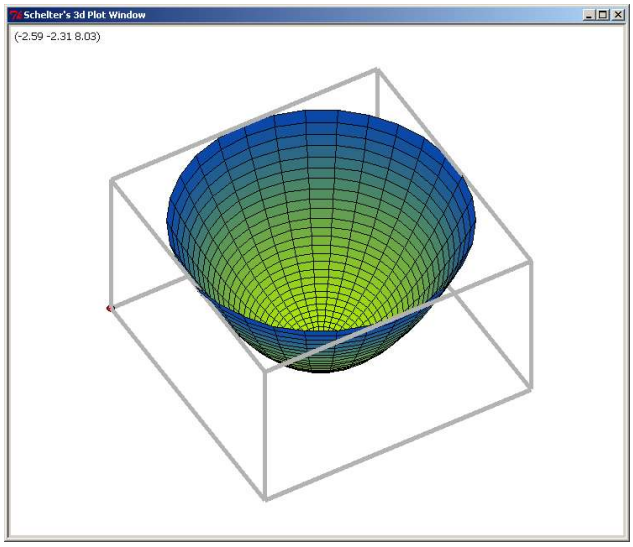
- Pour zoomer cliquez sur le bouton zoom puis cliquer à l'endroit souhaité. Pour dézoomer maintenez la touche SHIFT enfoncée et cliquez à l'endroit souhaité.

Tracé des courbes en 3D

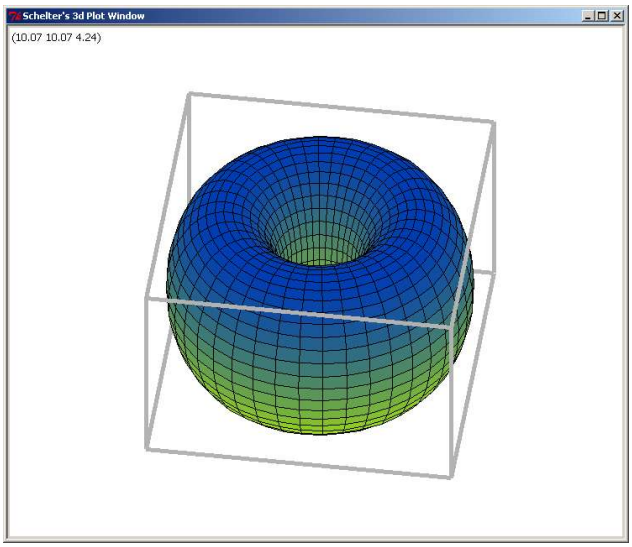
(C1) `plot3d(sin(x)+sin(y),[x,-10,10], [y,-10,10]);`



(C2) `plot3d([u*cos(v),2*u*sin(v),u^2],[u,0,5], [v,0,2*%PI]);`



```
(C3)  plot3d([cos(y)*(10.0+6*cos(x)),  sin(y)*(10.0+6*cos(x)),  -6*sin
(x)], [x,0,2*%pi],[y,0,2*%pi],['grid,40,40]);
```



Equations

1°) Résolution d'équations

Maxima résout directement dans C

- **`solve(x^4+1=0,x)`** résout l'équation x^4+1=0
- **`solve(a*x^2+b*x+c=0,x)`** résout l'équation a*x^2+b*x+c=0 suivant la variable x
- **`solve(sin(x)=0,x)`** résout l'équation sin(x)=0. Attention certaines valeurs peuvent êtres perdues.

2°) Systèmes d'équations

```
(C1) S1:[3*x+4*y-z=12, x-y+2*z=-3, x+y+z=2];
```

Définit le système d'équations linéaires S1

```
(C2) solve(S1, [x,y,z]);
```

Résout le système S1 suivant les variables x, y et z

Note : la fonction **`solve`** utilise alors dans ce cas la fonction **`linsolve`** permettant de résoudre les systèmes d'équations.

3°) Equations différentielles

Pour résoudre une équation différentielle il faut utiliser la fonction ode2
Exemple : résolution de y'+2*x*y=0

```
(C1) ode2('diff(y,x)+2*x*y=0,y,x);
```

Vous noterez le **'** devant l'expression **`diff(y,x)`**. En fait le **'** signifie que *y'* est définie implicitement. Ne l'oubliez pas sinon, vous risquez d'obtenir une erreur et de relancer le logiciel.

Fonctions Numériques

1°) Définition

(C1) f(x):=x^3+x^2-3*x+1;
Définit la fonction f de variable *x*. Attention pour définir une fonction il faut utiliser l'affectation **:=** et non pas l'affectation **:**

(C2) f(0);
Permet de calculer f(0)

(C3) g(x, y) := x SIN(y) + 2 x COS(y) + (- 3) x EXP(y) + 1;
Définit la fonction g de variables x et y

(C4) g(1,2);
Permet de calculer la valeur de g pour x=1 et y=2

2°) Limites

- **limit(1/x,x,2)** calcule la limite de 1/x en 2
- **limit(1/x,x,0, PLUS)** calcule la limite de 1/x en 0+
- **limit(1/x,x,0, MINUS)** calcule la limite de 1/x en 0-
- **limit(1/x,x,0, INF)** calcule la limite de 1/x en +infini
- **limit(1/x,x,0, MINF)** calcule la limite de 1/x en -infini

3°) Dérivation

- **diff(f(x),x)** calcule la dérivée de la fonction f selon la variable x
- **taylor(f(x), x, 0, n)** calcule le développement limité de la fonction f en 0 à l'ordre n

4°) Intégration

- **integrate(f(x), x)** calcule la primitive de la fonction f selon la variable x
- **integrate(f(x), x, x1, x2)** calcule l'intégrale de la fonction selon la variable x entre les bornes x1 et x2

Programmation

1°) Structure d'une fonction / procédure

Définition d'une fonction :

```
(C1) nom(paramètres d'entrée) := block([paramètres locaux],
     blocs d'instructions 1,

     /* Commentaire */

     blocs d'instructions 2,

     return(valeur)
)$
```

Définition d'une procédure :

```
(C1) nom(paramètres d'entrée) := block([paramètres locaux],
     blocs d'instructions 1,

     /* Commentaire */

     blocs d'instructions 2
)$
```

Voici un exemple simple de fonction : la fonction additionner 2 nombres a et b

```
(C1) somme(a, b) := block([c],
       c:a+b,
       return(c)
)$
(C2) c:somme(a,3);
(D2) 3+a
```

2°) La boucle for

Elle se définit comme suit :

FOR variable:première_valeur **STEP** valeur_incrémentation **THRU** valeur_limite **DO** (bloc d'instructions 1**,** blocs d'intructions 2 ...)**;**

Exemple :

```
(C3) for i:1 THRU 5 DO print(i);
```

3°) La boucle while

Elle se définit comme suit :

WHILE (condition) **DO** (blocs d'instructions 1**,** blocs d'instructions 2 ...)**;**

4°) L'instruction If ... Then

Elle se définit comme suit :

if (condition1) **then** (blocs d'instructions)
elseif (condition2) **then** (blocs d'instructions2)
else (blocs d'instructions3)**;**

Exemple

```
(C1) a:5;

(D1) 5

(C2) if (a < 5) then "a est inférieur à 5"
elseif (a = 5) then "c"
else "a est supérieur à 5";

(D2) a est égal à 5
```

Index

Utilisation de Maxima 2
1°) Aide de Maxima
2°) Expressions sous Maxima
3°) Variables sous Maxima
4°) Diverses commandes utiles

Opérateurs 3
Opérateur "!"
Opérateur "!!"
Opérateur "#"
Opérateur "."
Opérateur ":"
Opérateur ":="
Opérateur "="
Opérateur "^"
Opérateurs usuels

Constantes 4
Constante "%PI"
Constante "%E"
Constante "%I"
Constante "TRUE"
Constante "FALSE"
Constante "INF"
Constante "MINF"

Fonctions Mathématiques

Nombres réels 5
1°) Calcul sur les réels
2°) Calcul sur les entiers

Les polynômes

Sommation et produit 6
1°) Sommation
2°) Produit

Les matrices 7
1°) Définition
2°) Opérations

Tracé de courbes en 2D 8
1°) En coordonnées cartésiennes
2°) En coordonnées paramétriques 9
3°) En coordonnées polaires
4°) Options 10

Tracé de courbes en 3D 11

Equations 13
1°) Résolution d'équations
2°) Systèmes d'équations
3°) Equations différentielles

Fonctions numériques 14
1°) Définition
2°) Limites
3°) Dérivation
4°) Intégration

Programmation 15
1°) Structure d'une fonction / procédure
2°) La boucle for
3°) La boucle while 16
4°) L'instruction If ... Then