

# Travaux pratiques 5 : Introduction à Maple

Cliquez sur les triangles "►" pour déployer le texte.

## Introduction

Maple est un logiciel de calcul formel, c'est-à-dire un logiciel qui permet de faire des mathématiques en manipulant des expressions symboliques. Contrairement à la plupart des langages classiques de programmation il peut traiter non seulement des quantités numériques (entières, réelles, complexes) mais aussi des polynômes, des fonctions, des séries,... et d'effectuer des opérations courantes, dérivation et intégration formelle, limites, simplifications,...

De plus Maple permet aussi de faire les calculs numériques classiques, solution d'équations ou de systèmes, résolution d'équations différentielles,... Des capacités graphiques évoluées autorisent aussi de faire des représentations graphiques ainsi que d'aborder une partie des questions de géométrie plane ou dans l'espace.

Une feuille de travail (worksheet) Maple se compose de texte, d'équations et de lignes de commande Maple comme la suivante:

> 4+5

9

(1.1)

Vous pouvez évaluer une telle commande en y plaçant le curseur et en tapant <Enter>. Les commandes ne sont pas évaluées automatiquement lors de l'ouverture de la feuille de travail. Vous pouvez évaluer toutes les commandes de la feuille en tapant <Ctrl+Shift+Enter>. (Ne l'essayez pas sur la feuille présente, ça risque de prendre du temps.)

## Guide (A LIRE ATTENTIVEMENT!)

Voici un petit guide pour vos premiers pas dans Maple:

1. Lisez attentivement l'onglet "Rédaction" ci-dessous.
2. Lisez les Sections 1, 2 et 3, en expérimentant sur une autre feuille de travail les commandes présentées.
3. Commencez à résoudre les exercices (Section 7), en vous référant aux autres sections ci-dessous lorsque nécessaire.
4. Comme pour Matlab, 1 exercice = 1 feuille de travail. Ne résolvez pas les exercices sur cette feuille de travail.
5. Utilisez la possibilité d'écrire du texte dans la feuille de travail pour expliquer EN DETAIL vos raisonnements et votre code. Pas de commentaires = 0 point.

## Rédaction

Pour rédiger un fichier Maple comme celui ci, utilisez la barre se trouvant juste au dessus de la fenêtre dans laquelle vous lisez ce texte. Vous y trouvez des options pour formater le texte.

Pour écrire des équations, cliquez sur le bouton "Math":  $x^2 + y^2 = z^2$ . Cliquez sur le bouton "Texte" pour revenir au mode texte.

Pour entrer une commande Maple, cliquez sur l'onglet "Insertion" dans le menu de la fenêtre principale et choisissez "Entrée Maple", ou plus simplement, tapez <Ctrl+M>:

> x := 3

x := 3

(1.2.1)

Les entrées Maple peuvent être évaluées en y plaçant le curseur et en tapant <Enter>. Tapez <Ctrl+Z> pour annuler l'évaluation. Pour supprimer une entrée Maple, placez-y le curseur et tapez <Ctrl+Delete>. Pour écrire une entrée Maple sur plusieurs lignes sans l'évaluer, utilisez <Shift+Enter>.

**IMPORTANT:** Si vous avez entré une commande Maple par erreur en mode texte ou math, elle ne peut pas être exécutée:

3 + 4

Dans ce cas, créez une ligne "Entrée Maple" (Maple input, <Ctrl+M>), et collez votre commande dedans pour pouvoir l'exécuter:

> 3+4

7

(1.2.2)

Pour créer des sections déroulantes comme dans ce fichier, utilisez l'icône d'indentation dans le menu de la fenêtre principale, ou <Ctrl+.> (point). Pour les supprimer, utilisez <Ctrl+Delete> comme pour les entrées Maple.

Toutes les actions peuvent être annulées avec <Ctrl+Z>.

Comme pour Matlab, vous devez rédiger un fichier Maple par exercice. Expliquez votre code et les commandes que vous utilisez au moyen de texte.

Pour la présentation, inspirez-vous de l'exemple d'exercice résolu sur Moodle. D'une manière générale, assurez-vous que vos solutions soient aisément lisibles et compréhensibles.

## 1. Bases

### Assignement de valeurs et égalité

L'assignation de valeurs aux variables se fait avec le symbole `:=`, comme en Pascal. En principe, toutes les commandes se terminent par un point virgule, mais dans les versions récentes, on peut omettre celui-ci. On peut toutefois l'utiliser pour écrire plusieurs commandes sur la même ligne (pas recommandé pour des raisons de lisibilité). Si on veut supprimer l'impression du résultat, on termine la commande par deux points à la place du point virgule :

> x:=2:y:=x^2;y+x;

y := 4

6

(2.1.1)

Le symbole "`=`" est utilisé pour exprimer une équation; la commande `solve("equation")` retourne la liste des solutions :

```
> solve(z^2=2);  
           $\sqrt{2}, -\sqrt{2}$ 
```

Si on a oublié que l'on a assigné la valeur 2 à `x` :

```
> solve(x^2=2);  
on n'obtient aucune solution! Evidemment, puisque x vaut 2 et qu'il n'y a pas de solution à l'équation  $2^2=2$ .
```

Si on veut de nouveau utiliser `x` comme variable, il faut lui assigner la valeur symbolique '`x`' :

```
> x:='x';  
           $x := x$ 
```

```
> solve(x^2=2);  
           $\sqrt{2}, -\sqrt{2}$ 
```

De même, si on écrit :

```
> solve(1=2);  
on n'obtient rien en retour. On peut même écrire :
```

```
> z:=x^2=2;  
           $z := x^2 = 2$ 
```

et `z` a pour valeur l'expression  $x^2=2$ . On peut donc écrire :

```
> solve(z);  
           $\sqrt{2}, -\sqrt{2}$ 
```

Si l'on pose

```
> z:=x=2;  
           $z := x = 2$ 
```

on aura assigné à `z` une expression booléenne, dont on peut connaître la valeur par la commande `evalb` (évaluation booléenne) :

```
> x:=1:evalb(z);  
          false
```

```
> x:=2:evalb(z);  
          true
```

## Aide

Maple propose un système d'aide très complet et qui fournit beaucoup d'exemples. Si on veut de l'aide sur une commande ou une procédure, taper `?"nom"` :

```
> ?+
```

ou encore

```
> ?plot
```

On peut aussi écrire :

```
> help(plot)
```

Enfin, on peut utiliser l'aide à partir de la barre de menu de Maple, par mot-clé ou par sujet. Essayez aussi la commande :

```
> help(help)
```

## Recommencer

Maple vous offre la possibilité de repartir à zéro, mais pas tout-à-fait; la commande

```
> restart;
```

efface les valeurs assignées aux variables et les éventuelles librairies chargées en mémoire (par exemple avec la commande `with(plots)`, utilisée au dernier paragraphe). Ainsi :

```
> a;  
          a
```

La variable "a" n'a plus qu'une valeur symbolique.

Cependant, cette commande ne remet pas à disposition la totalité de la mémoire utilisée jusque-là. Si le comportement de Maple est bizarre, il faut repartir à zéro en quittant l'application (après avoir sauvé votre feuille de travail), et la relancer ensuite.

## 2. Calculs avec les nombres

```
> 2+2;  
          4
```

```
> 2*4;  
          8
```

Le signe `%` rappelle le dernier résultat obtenu :

```
> %+2;  
          10
```

Le circonflexe précède un exposant :

```
> 3^2;
```

```
> 2^(3+2);
```

32

(3.5)

Les fractions sont représentées sous forme exacte :

```
> 9/7;
```

$\frac{9}{7}$

(3.6)

Si on veut le résultat sous forme de nombre décimal, il faut utiliser la fonction **evalf** ("f" pour "float", i.e. nombre réel à virgule flottante) :

```
> evalf(%);
```

1.285714286

(3.7)

et si on veut plus de décimales :

```
> evalf(9/7, 20);
```

1.285714285714285714

(3.8)

On peut forcer les nombres entiers à être traités comme des "float" en écrivant un 0 après la virgule (qui est un point dans Maple) :

```
> 2.0/3;
```

0.6666666667

(3.9)

Racine carrée se dit "square root" en anglais :

```
> sqrt(2);
```

$\sqrt{2}$

(3.10)

```
> evalf(%);
```

1.414213562

(3.11)

```
> sqrt(2.0);
```

1.414213562

(3.12)

On remarque ci-dessus que **sqrt(2)** est conservé sous forme symbolique par Maple; par contre, si on écrit **sqrt(2.0)** le **2.0** force Maple à travailler avec des représentations décimales approchées de nombres réels. L'intérêt de la forme symbolique est que le symbole  $\sqrt{2}$  est compris comme le nombre réel (positif) dont le carré vaut 2. Si on veut d'autres racines (cubique, *nième*), on utilise l'écriture exponentielle ou la fonction **root** :

```
> 2^(1/3);
```

$2^{1/3}$

(3.13)

```
> evalf(%);
```

1.259921050

(3.14)

Certains nombres sont prédefinis, comme le nombre **Pi**, qui est la moitié de la circonference du cercle de rayon 1 :

```
> Pi;
```

$\pi$

(3.15)

```
> evalf(Pi);
```

3.141592654

(3.16)

```
> evalf(Pi, 20);
```

3.1415926535897932385

(3.17)

ou le nombre d'Euler **e**, qui n'a pas de nom particulier dans Maple (contrairement à  $\pi$ ) :

```
> exp(1);
```

$e$

(3.18)

```
> evalf(% , 30);
```

2.71828182845904523536028747135

(3.19)

Maple est sensible aux majuscules-minuscules. Mais Maple n'est pas contrariant; si vous écrivez :

```
> pi;
```

$\pi$

(3.20)

```
> evalf(pi);
```

$\pi$

(3.21)

```
> Pi := 1;
```

*Error, attempting to assign to `Pi` which is protected. Try declaring `local Pi`; see ?protect for details.*

```
> pi := 1;
```

$\pi := 1$

(3.22)

Maple pense que **pi** est une nouvelle variable.

Pour assigner un nombre à une variable **x** :

```
> x := sqrt(2);
```

$x := \sqrt{2}$

(3.23)

```

> x^2;
2
(3.24)

> y := sqrt(2.0);
y := 1.414213562
(3.25)

> y^2;
1.999999999
(3.26)

```

Cet exemple montre l'intérêt à travailler avec les expressions exactes.

La décomposition en produit de facteurs premiers s'effectue au moyen de la commande **ifactor** (= "integer factorisation"). Attention, il existe des algorithmes plus efficaces que de trouver sa décomposition en facteur premier pour déterminer si un nombre est premier ou non. C'est pourquoi si vous devez savoir si un nombre est premier sans nécessairement connaître sa décomposition il faut utiliser la commande **isprime**.

```

> ifactor(2011);
(2011)
(3.27)

> isprime(2011);
true
(3.28)

> isprime(1899998676^5+125099^6);
false
(3.29)

> ifactor(1899998676^5+125099^6);
(470339374377617) (20442202663189) (2575297847702655029)
(3.30)

```

## 3. Expressions symboliques, fonctions, équations

### Expressions symboliques

On commence par nettoyer:

```

> restart;
> r := (a + b)^2;
r := (a + b)^2
(4.1.1)

> er := expand(%);
er := a^2 + 2 a b + b^2
(4.1.2)

> r - er;
(a + b)^2 - a^2 - 2 a b - b^2
(4.1.3)

> simplify(r - er);
0
(4.1.4)

```

L'exemple ci-dessus montre à quel point Maple respecte la volonté de l'utilisateur: les expressions ne sont simplifiées que sur demande.

**Maple considère **r** et **s** comme des expressions en les variables symboliques **a** et **b**. Pour substituer les symboles par des valeurs on utilise la fonction **subs**:**

```

> subs(a = 1, r);
(1 + b)^2
(4.1.7)

> subs(a = 2, b = 1, r);
9
(4.1.8)

```

Maple sait factoriser des polynômes, au moyen de la commande **factor**:

```

Error, missing operator or `;`

> factor(x^3 - y^3);
(x - y) (x^2 + x y + y^2)
(4.1.9)

```

et même des quotients de polynômes :

```

> factor((x^4 - y^4) / (x^3 + y^3));
(x - y) (x^2 + y^2)
x^2 - x y + y^2
(4.1.10)

```

Dans ce cas Maple a pris l'initiative de simplifier l'expression.

### Fonctions

On définit une fonction comme suit, au moyen du symbole :

```

> f := x -> x^2 - 1;
f := x -> x^2 - 1
(4.2.1)

```

```
> f(1);
```

0

```
> g := (x,y) -> x^2 + y^2 - 1;
```

$g := (x, y) \mapsto x^2 + y^2 - 1$

```
> g(2,1);
```

4

```
> g(1,0);
```

0

## Difference entre expressions et fonctions

Une expression est liée aux symboles utilisés dans sa définition; dans une fonction, au contraire, le nom du symbole représentant la variable peut être modifié :

```
> f(u);
```

$u^2 - 1$

```
> h := (x,y) -> x^2 - y^3 + 1;
```

$h := (x, y) \mapsto x^2 - y^3 + 1$

```
> h(y,x);
```

$-x^3 + y^2 + 1$

```
> h(u,v);
```

$-v^3 + u^2 + 1$

```
> E := x^2 - 1;
```

$E := x^2 - 1$

```
> subs(x = 1,E);
```

0

```
> subs(u = 1,E);
```

$x^2 - 1$

## Equations

Pour résoudre une équation :

```
> solve(x^2 - 1 = 0,x);
```

1, -1

```
> solve(x^2 + y^2 - 1 = 0,x);
```

$\sqrt{-y^2 + 1}, -\sqrt{-y^2 + 1}$

```
> solve({x^2 + y^2 - 1 = 0,x = y},{x,y});
```

$\{x = \text{RootOf}(2 \_Z^2 - 1), y = \text{RootOf}(2 \_Z^2 - 1)\}$

et si on veut tout savoir :

```
> allvalues(solve({x^2 + y^2 - 1 = 0,x = y},{x,y}));
```

$\left\{x = \frac{\sqrt{2}}{2}, y = \frac{\sqrt{2}}{2}\right\}, \left\{x = -\frac{\sqrt{2}}{2}, y = -\frac{\sqrt{2}}{2}\right\}$

On peut aussi utiliser d'autres formes :

```
> solve(x^2 + x - 1,x);
```

$\frac{\sqrt{5}}{2} - \frac{1}{2}, -\frac{\sqrt{5}}{2} - \frac{1}{2}$

ce qui montre que le signe **= 0** est facultatif; ou encore :

```
> solve(x^2 + x = 1,x);
```

$\frac{\sqrt{5}}{2} - \frac{1}{2}, -\frac{\sqrt{5}}{2} - \frac{1}{2}$

Les solutions peuvent être imaginaires :

```
> solve(x^2 + x + 1,x);
```

$-\frac{1}{2} + \frac{I\sqrt{3}}{2}, -\frac{1}{2} - \frac{I\sqrt{3}}{2}$

ou **I** est la racine de -1:

```
> I*I;
```

-1

Il est donc interdit d'utiliser **I** comme nom de variable:

```
> I := 1;
```

Error, illegal use of an object as a name

Les solutions sont données sous forme de liste. Voici comment accéder individuellement aux solutions :

```
> equat := x^2 - x + 1; sols := solve(equat,x);
```

$$equat := x^2 - x + 1$$

$$sols := \frac{1}{2} + \frac{I\sqrt{3}}{2}, \frac{1}{2} - \frac{I\sqrt{3}}{2}$$

(4.4.9)

```
> a := sols[1];
```

$$a := \frac{1}{2} + \frac{I\sqrt{3}}{2}$$

(4.4.10)

```
> subs(x = a, equat);
```

$$\left(\frac{1}{2} + \frac{I\sqrt{3}}{2}\right)^2 + \frac{1}{2} - \frac{I\sqrt{3}}{2}$$

(4.4.11)

Maple est minimaliste : il ne développe pas les expressions plus que nécessaire. Pour voir si notre **a** est vraiment solution, il faut faire ainsi:

```
> expand(subs(x = a, equat));
```

$$0$$

(4.4.12)

```
> expand(subs(x = sols[2], equat));
```

$$0$$

(4.4.13)

**fsolve** résoud les équations numériquement

```
> fsolve({x^2 + y^2 - 1 = 0, x = y}, {x, y});
```

$$\{x = -0.7071067812, y = -0.7071067812\}$$

(4.4.14)

```
> solve(exp(x) = 2, x);
```

$$\ln(2)$$

(4.4.15)

```
> fsolve(exp(x) = 2, x);
```

$$0.6931471806$$

(4.4.16)

Contrôlons:

```
> evalf(ln(2), 20);
```

$$0.69314718055994530942$$

(4.4.17)

Lorsqu'on a deux équations à deux inconnues, les solutions sont données sous la forme de liste d'expression de la forme **x = . . . , y = . . .**:

```
> sols := allvalues(solve({x^2 + y - 1 = 0, x = y}, {x, y}));
```

$$sols := \left\{x = \frac{\sqrt{5}}{2} - \frac{1}{2}, y = \frac{\sqrt{5}}{2} - \frac{1}{2}\right\}, \left\{x = -\frac{\sqrt{5}}{2} - \frac{1}{2}, y = -\frac{\sqrt{5}}{2} - \frac{1}{2}\right\}$$

(4.4.18)

Voici comment accéder aux solutions. La commande **op** donne accès aux "opérants", ou composantes d'une expression:

```
> sols[1];
```

$$\left\{x = \frac{\sqrt{5}}{2} - \frac{1}{2}, y = \frac{\sqrt{5}}{2} - \frac{1}{2}\right\}$$

(4.4.19)

```
> op(sols[1]);
```

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

(4.4.20)

```
> op(sols[1])[1];
```

$$x = \frac{\sqrt{5}}{2} - \frac{1}{2}$$

(4.4.21)

```
> op(sols[1])[2];
```

$$y = \frac{\sqrt{5}}{2} - \frac{1}{2}$$

(4.4.22)

```
> evalf(op(sols[1])[2]);
```

$$y = 0.6180339880$$

(4.4.23)

On peut aussi utiliser la commande **rhs** (= right hand side, ou membre de droite) :

```
> rhs(op(sols[1])[1]);
```

$$\frac{\sqrt{5}}{2} - \frac{1}{2}$$

(4.4.24)

Dans la commande **fsolve**, on peut préciser un intervalle dans lequel on veut chercher les solutions:

```
> fsolve(x^2 - 2, x, -2..0);
```

$$-1.414213562$$

(4.4.25)

## Dérivations

```
> g := (x, y) -> x^2 + y^2 - 1;
```

$$g := (x, y) \mapsto x^2 + y^2 - 1$$

(4.5.1)

Pour dériver  $g$  par rapport à  $x$ , on exécute

```
> diff(g(x,y),x);
```

$2x$

(4.5.2)

On itère la dérivation en répétant le nom des variables par rapport auxquelles on veut dériver :

```
> diff(g(x,y),x,x);
```

$2$

(4.5.3)

```
> diff(g(x,y),x,y);
```

$0$

(4.5.4)

Mais on peut changer le nom des variables, si on veut :

```
> diff(g(u,v),u);
```

$2u$

(4.5.5)

On peut aussi dériver une expression :

```
> f := x^2 + x*y^3 - x*y;
```

$f := x y^3 + x^2 - x y$

(4.5.6)

```
> diff(f,x,y);
```

$3y^2 - 1$

(4.5.7)

```
> diff(f,x,y,y);
```

$6y$

(4.5.8)

D'autres exemples :

```
> diff(exp(x^2+y^2),x,y);
```

$4xy e^{x^2+y^2}$

(4.5.9)

```
> diff(exp(x^2+y^2),y,x);
```

$4xy e^{x^2+y^2}$

(4.5.10)

```
> diff(sin(x^2+y^2),x,y,y);
```

$-4x \sin(x^2+y^2) - 8xy^2 \cos(x^2+y^2)$

(4.5.11)

```
> diff(sin(x^2+y^2),y,x,y);
```

$-4x \sin(x^2+y^2) - 8xy^2 \cos(x^2+y^2)$

(4.5.12)

## 4. Graphiques

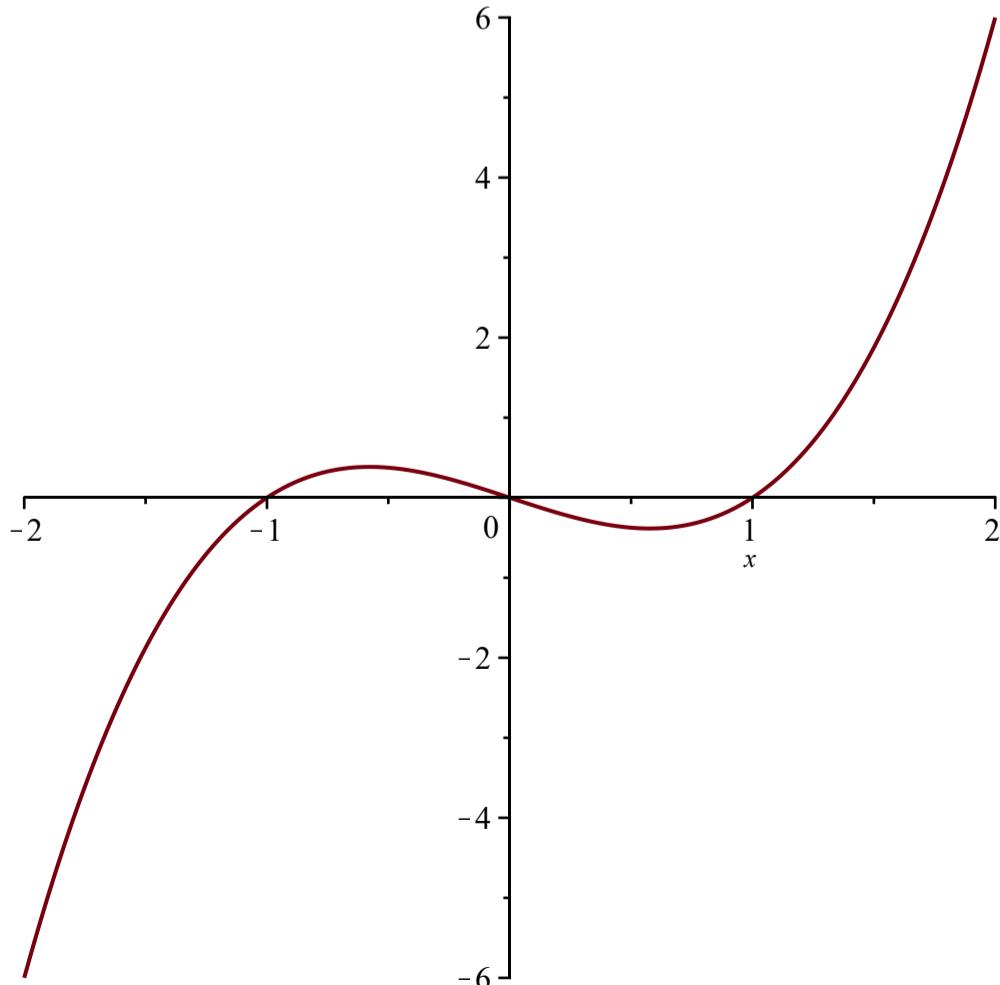
### Fonctions d'une variable

Le graphe d'une fonction à une variable peut être obtenu au moyen de la commande `plot`.

```
> restart;
> f := x -> x^3 - x;
> plot(f(x),x = -2 .. 2);
```

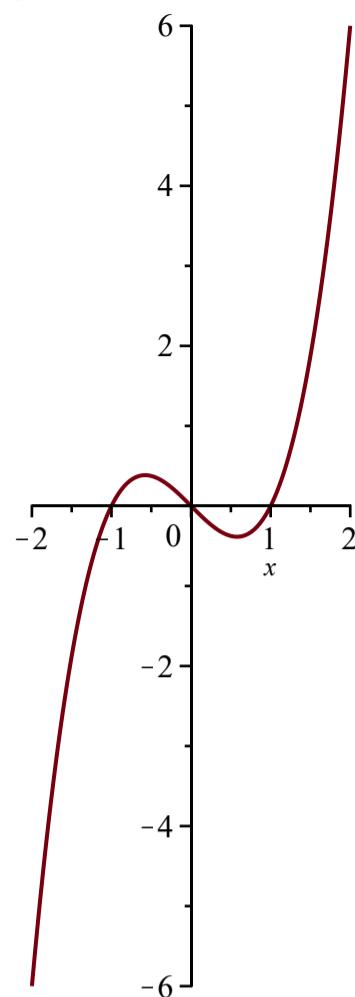
$f := x \mapsto x^3 - x$

(5.1.1)



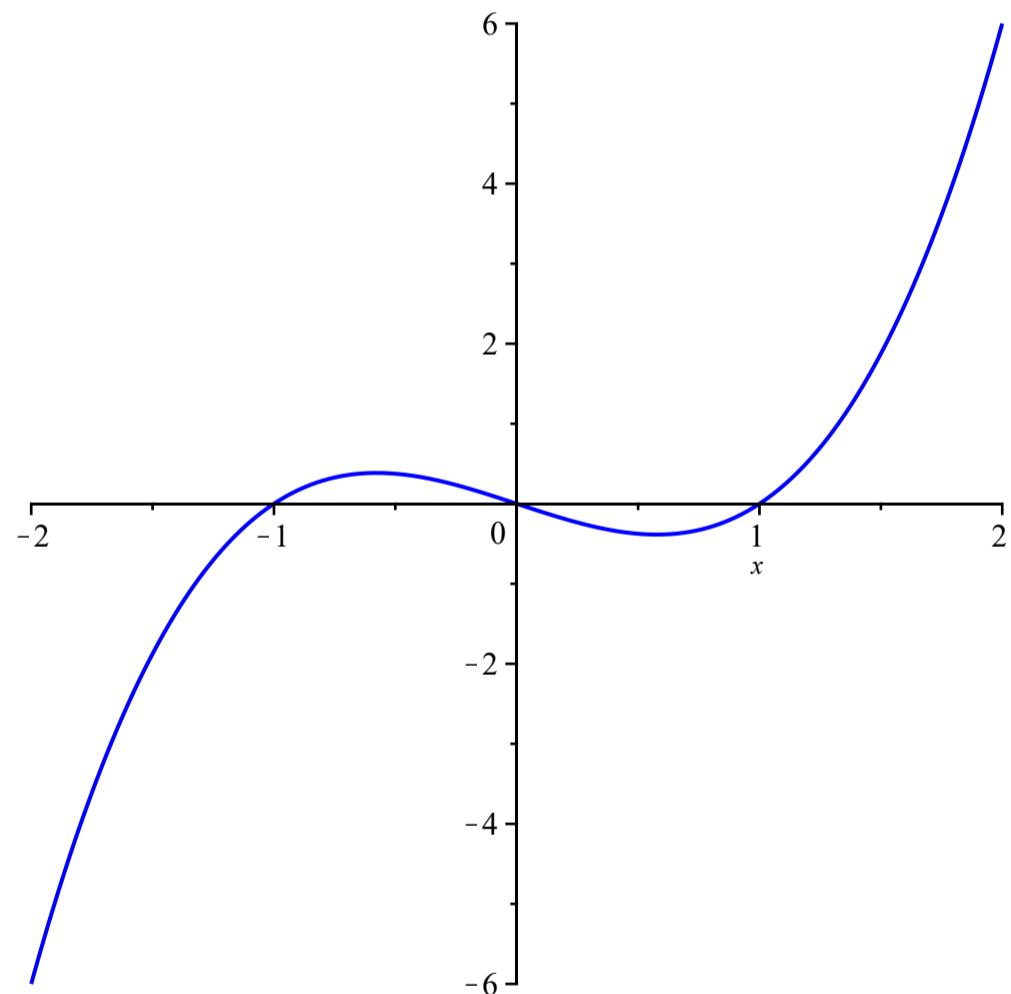
Si on veut la même échelle sur les 2 axes :

```
> plot(f(x),x = -2 .. 2, scaling = constrained);
```



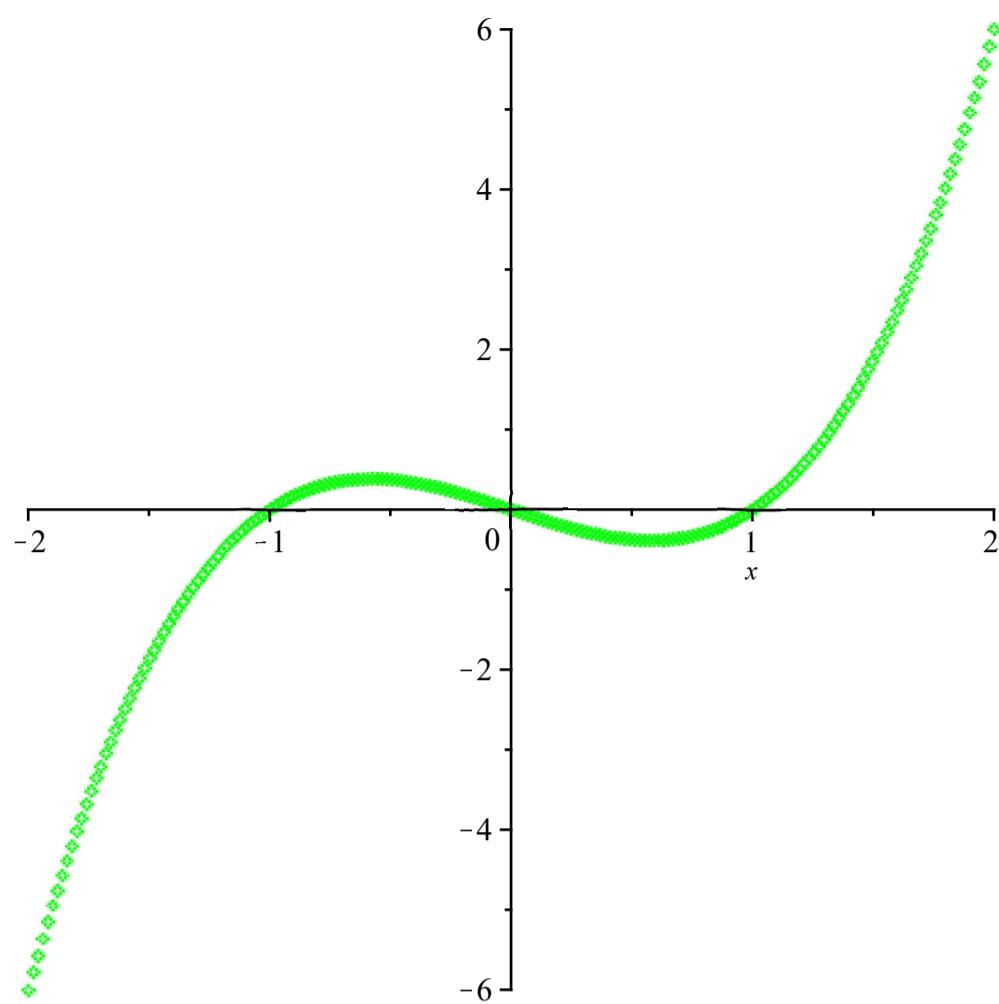
Par défaut, 50 points sont calculés. Si on en veut plus, et une autre couleur :

```
> plot(f(x),x = -2 .. 2, numpoints = 500, color = blue);
```



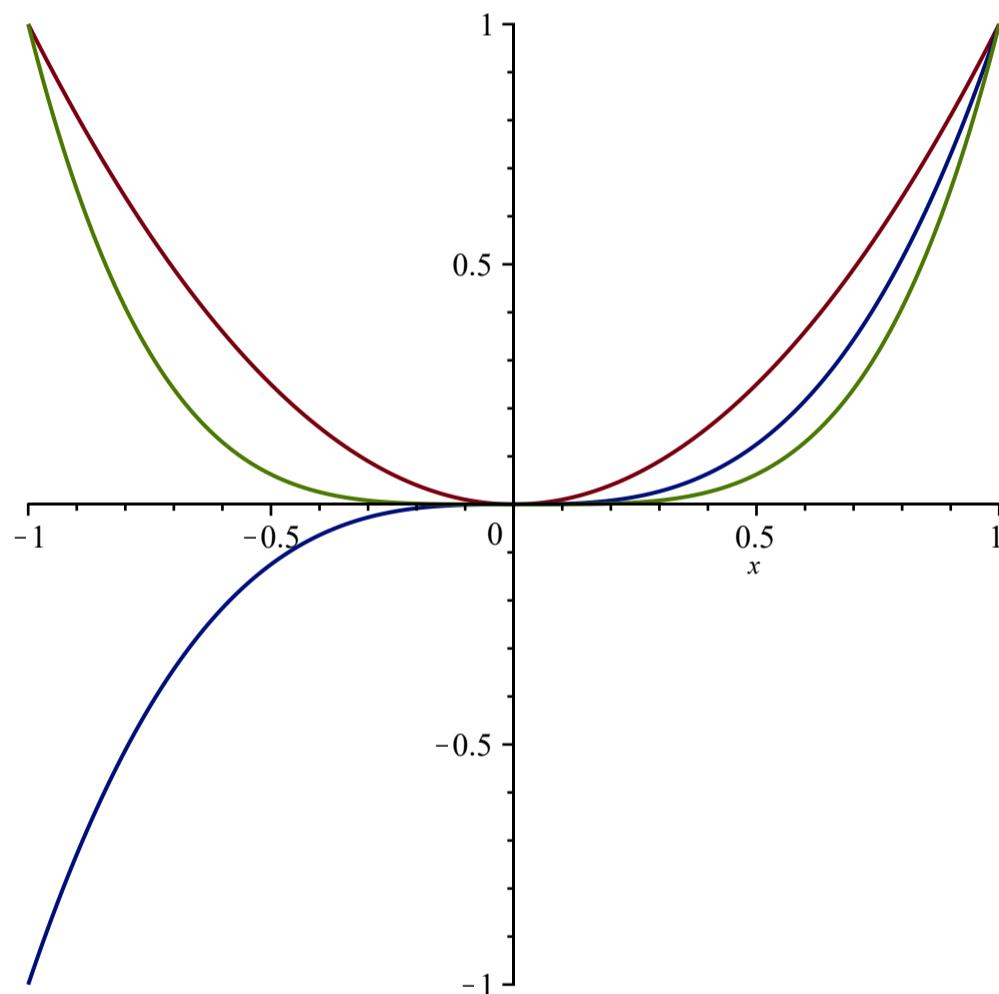
On peut aussi changer de style :

```
> plot(f(x),x = -2 .. 2, style = point, color = green);
```



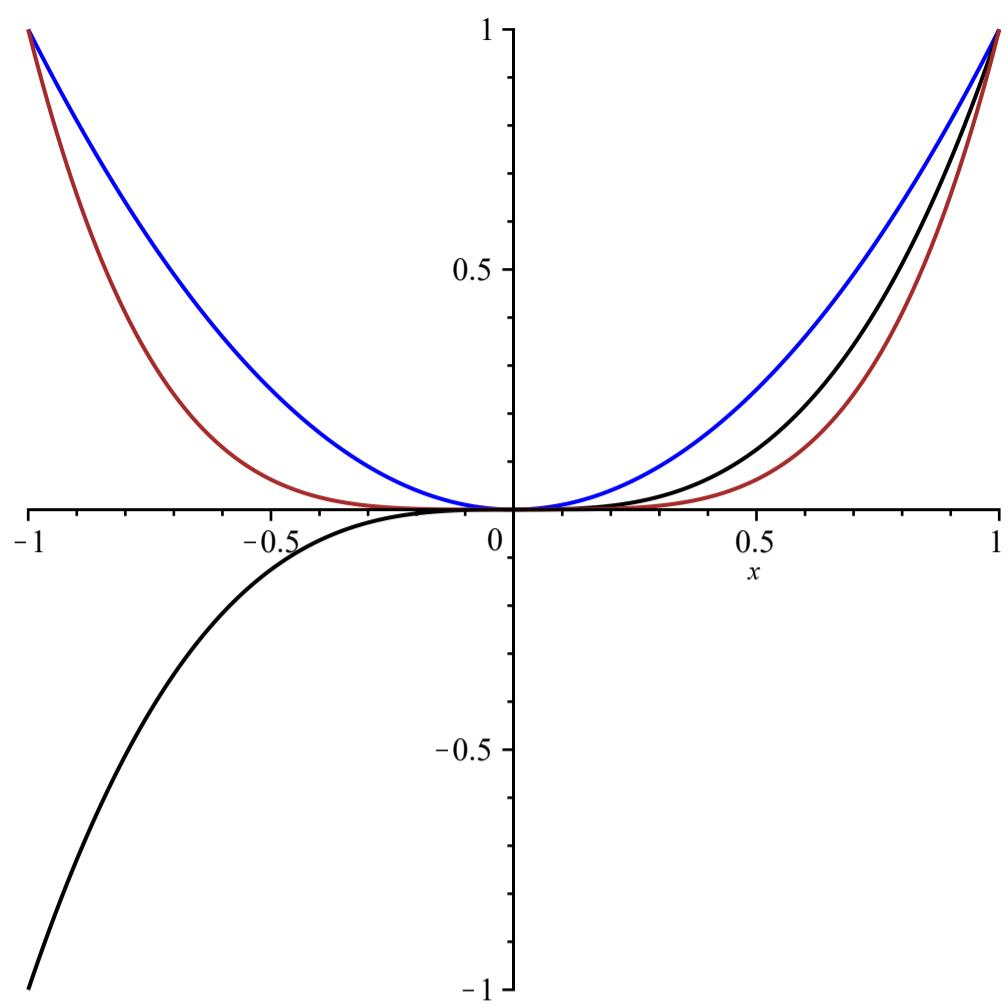
Si on veut superposer plusieurs graphes, il suffit d'écrire la liste des fonctions au lieu d'une seule fonction:

```
> plot([x^2,x^3,x^4],x = -1 .. 1);
```



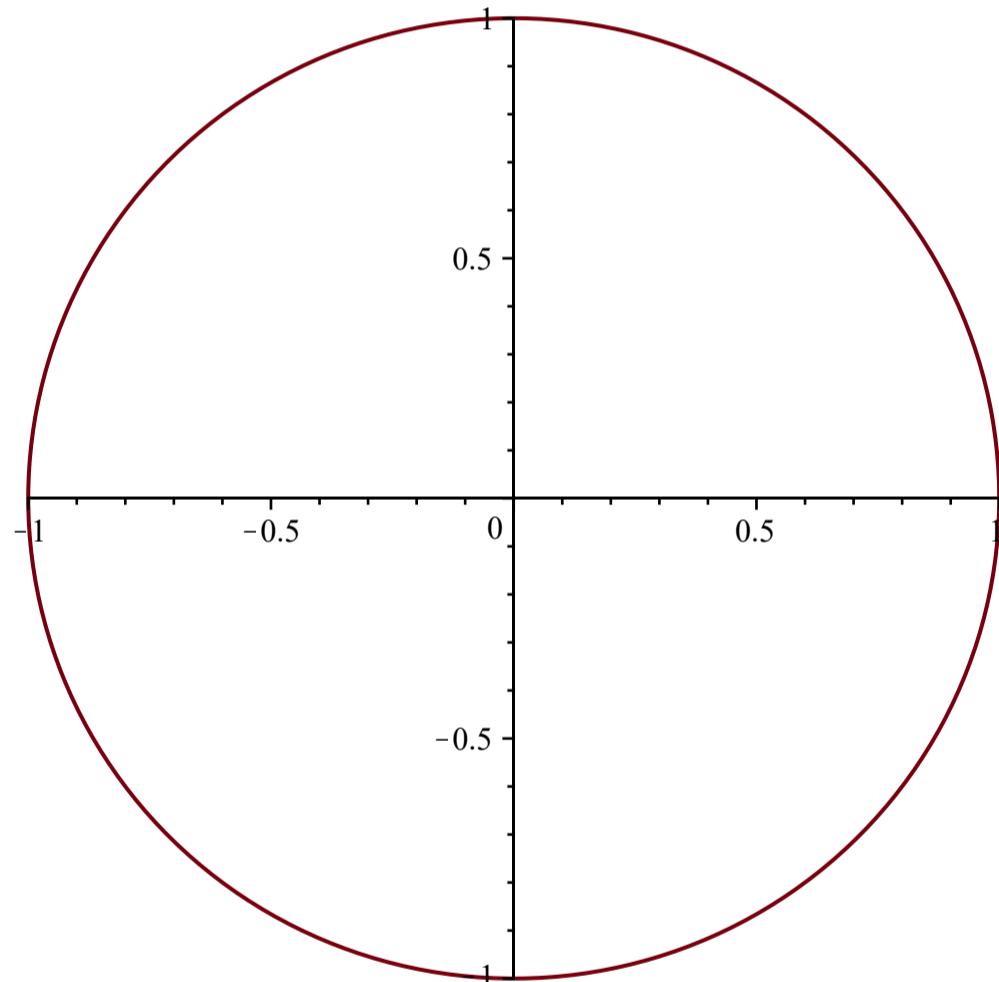
Si le choix des couleurs ne plaît pas, on peut donner sa propre liste :

```
> plot([x^2,x^3,x^4],x = -1 .. 1, color = [blue,black,brown]);
```



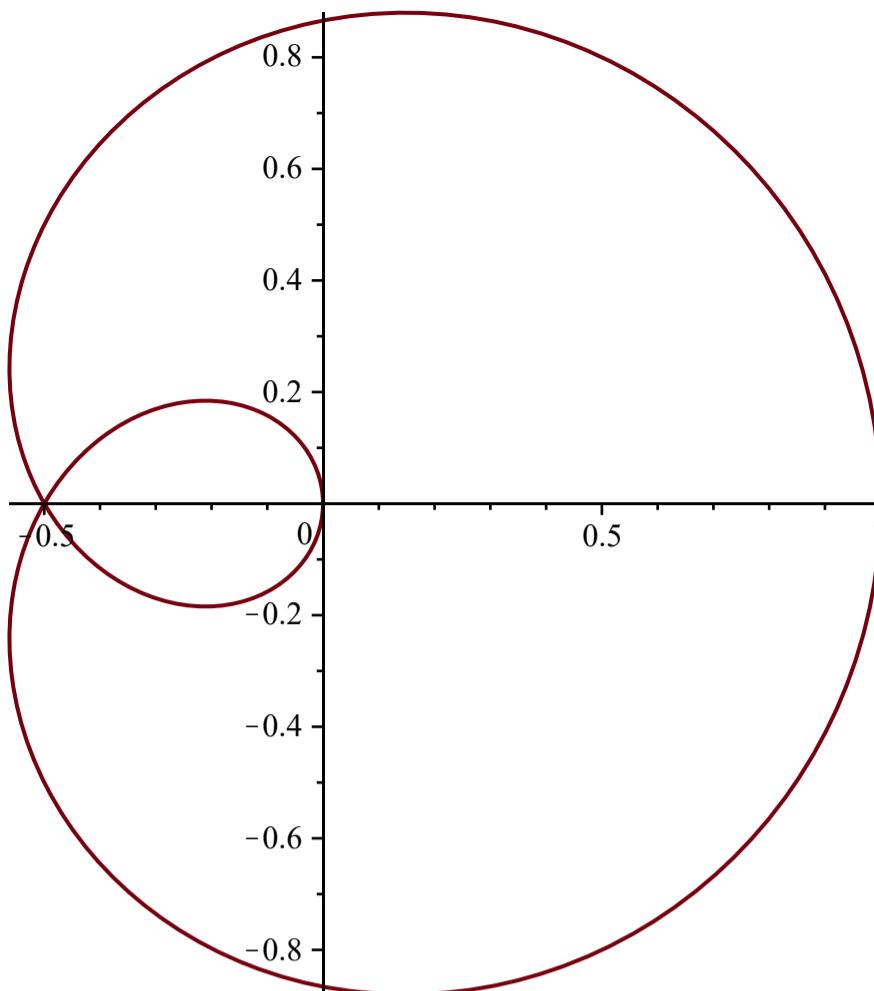
On peut utiliser les coordonnées polaires :

```
> plot(1, t = 0 .. 2*Pi, coords = polar);
```



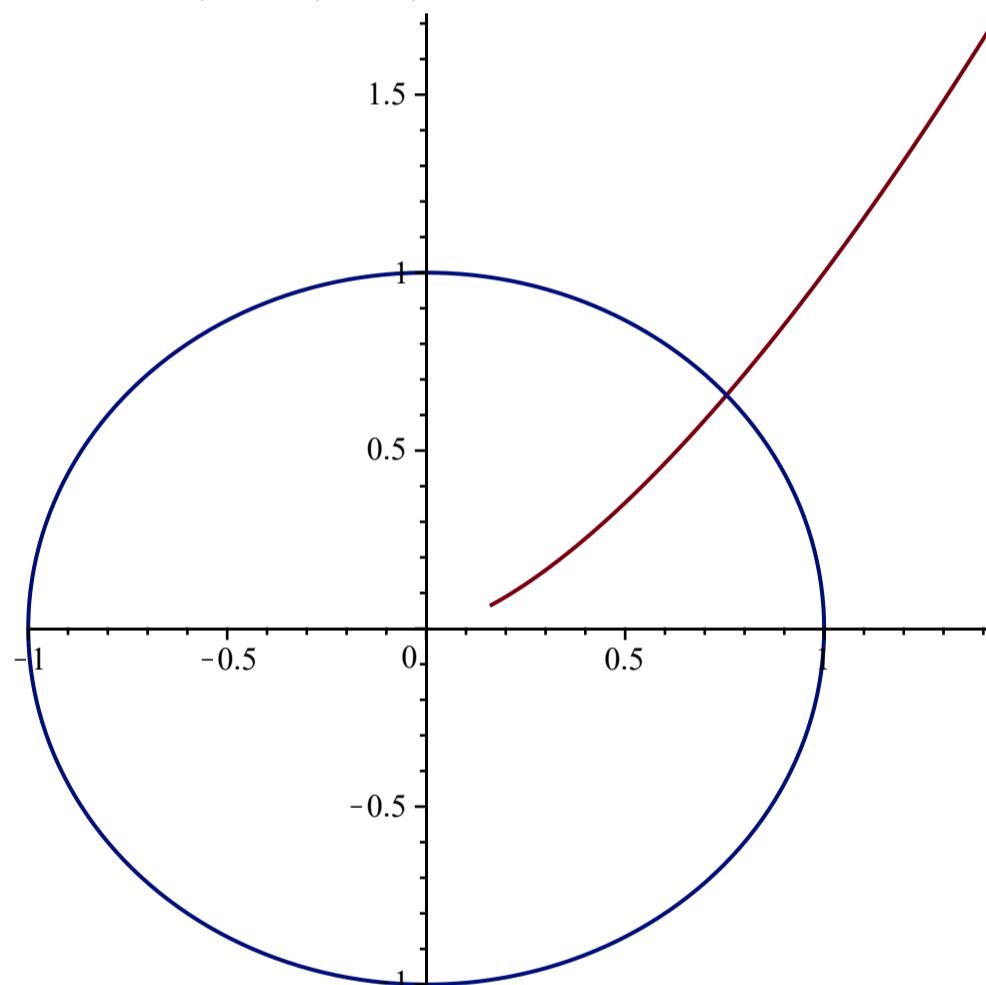
Un autre exemple :

```
> plot(cos(t/3), t = 0 .. 3*Pi, coords = polar, scaling = constrained);
```



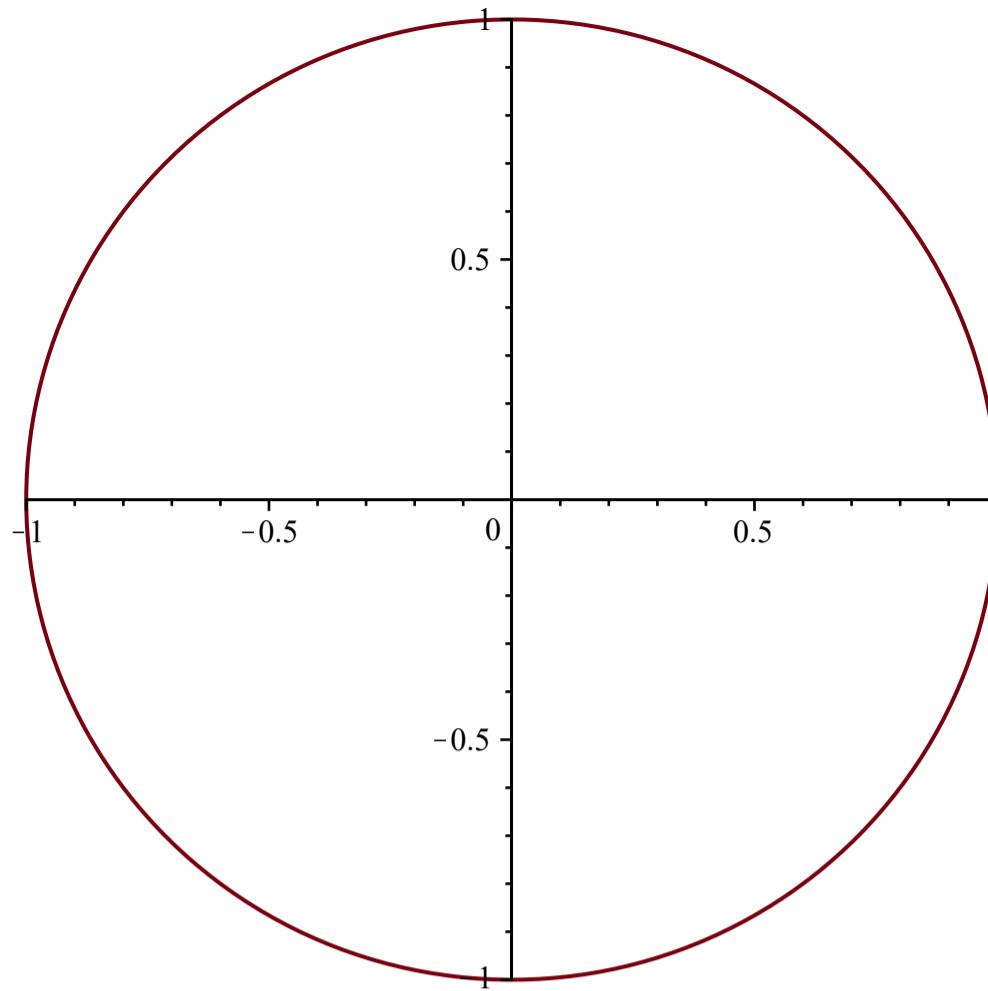
Si on veut dessiner des courbes paramétrées de manière différentes sur un même graphe, il est possible de faire ceci.

```
> plot({[cos(t), sin(t), t = 0 .. 2*Pi], [t^2, t^3, t = 0.4 .. 1.2]});
```



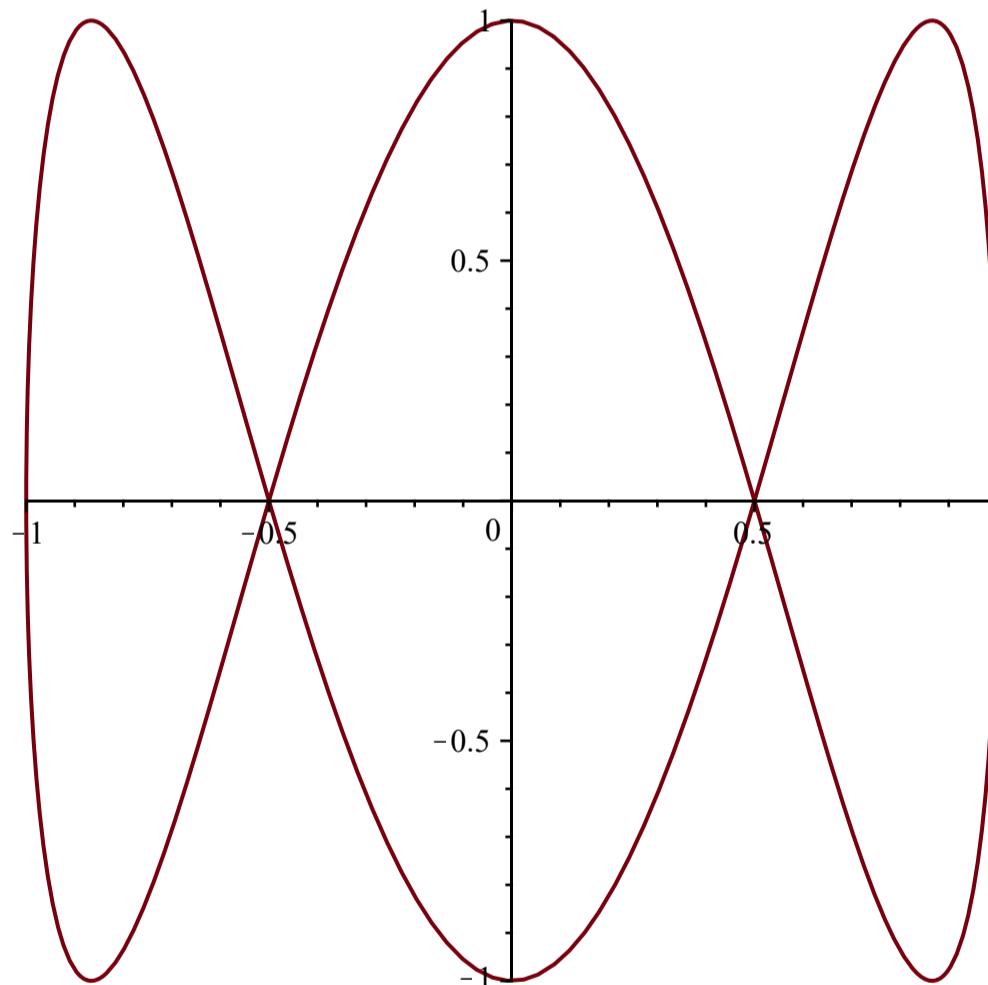
### ▼ Courbes paramétriques

```
> plot([cos(t), sin(t), t = 0 .. 2*Pi]);
```



Les figures de Lissajous sont la composition de mouvement sinusoïdaux perpendiculaires l'un à l'autre. Elles ont des paramétrisations de la forme  $(a \cos(kt), b \sin(ht))$

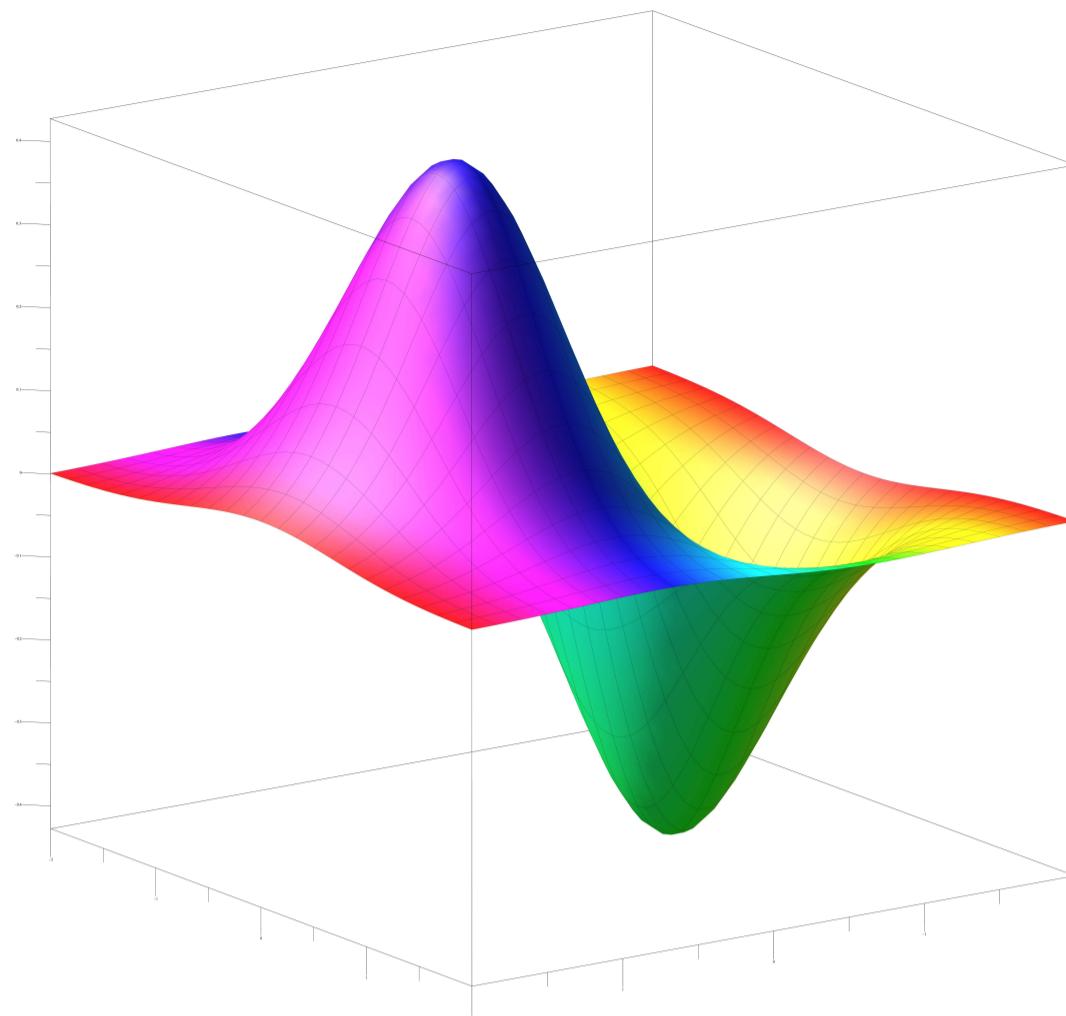
```
> plot([cos(t), sin(3*t), t = 0 .. 2*Pi]);
```



### Fonctions de deux variables.

Si vous voulez dessiner des surfaces, la fonction **plot3d** est la plus adaptée. Il est possible de faire tourner la représentation sur elle-même. Essayez sur l'exemple suivant.

```
> plot3d(x*exp(-x^2 - y^2), x = -2 .. 2, y = -2 .. 2, color = x);
```



**plot3d** a plusieurs options intéressantes (dans l'exemple précédent l'option color permet de colorier la surface en fonction d'un paramètre). Consultez l'aide de **plot3d** et essayez les exemples.

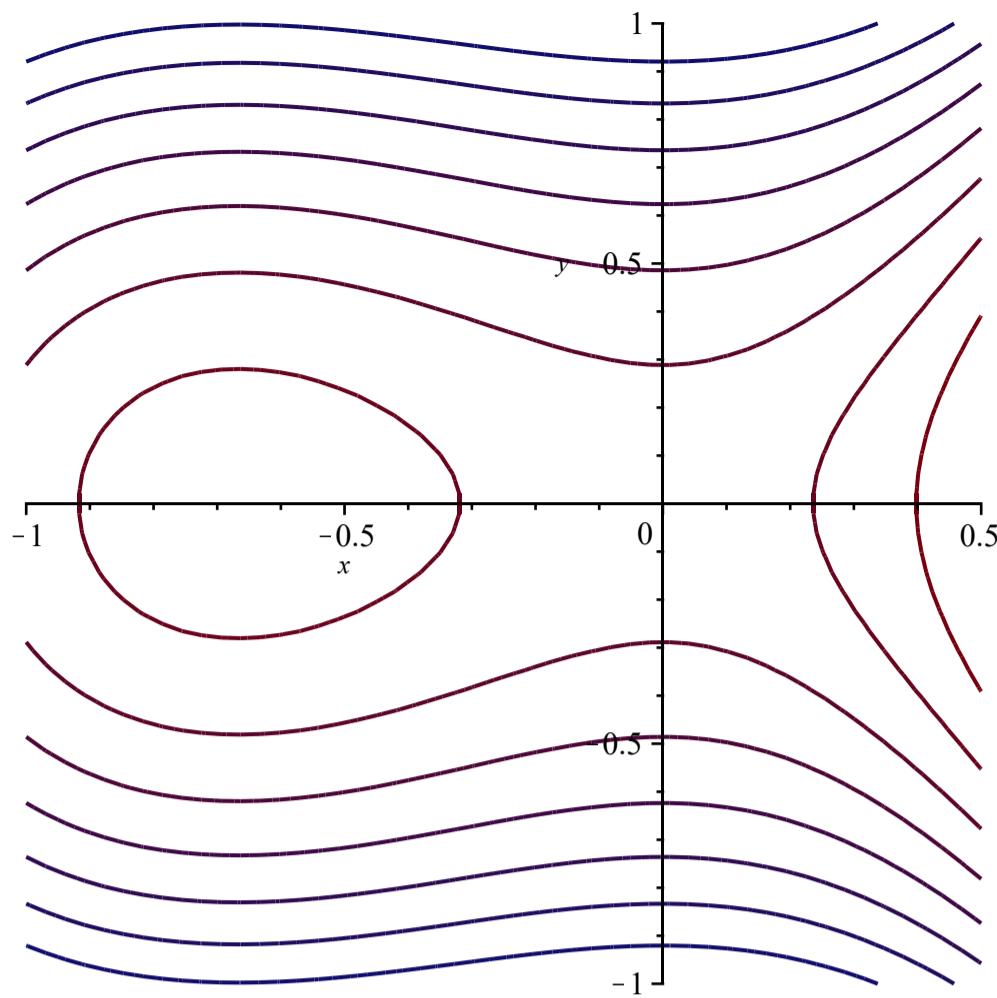
Il existe d'autres outils de dessins dans Maple, il sont regroupé dans des librairies de procédures , les plus utiles d'entre elles sont **plots** et **plottools**. On tape d'abord la commande suivante pour charger la librairie **plots**

```
> with(plots);
[animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors, setoptions, setoptions3d, shadebetween, spacecurve, sparsematrixplot, surldata, textplot, textplot3d, tubeplot] (5.3.1)
```

et on obtient comme réponse la liste des nouvelles procédures à notre disposition.

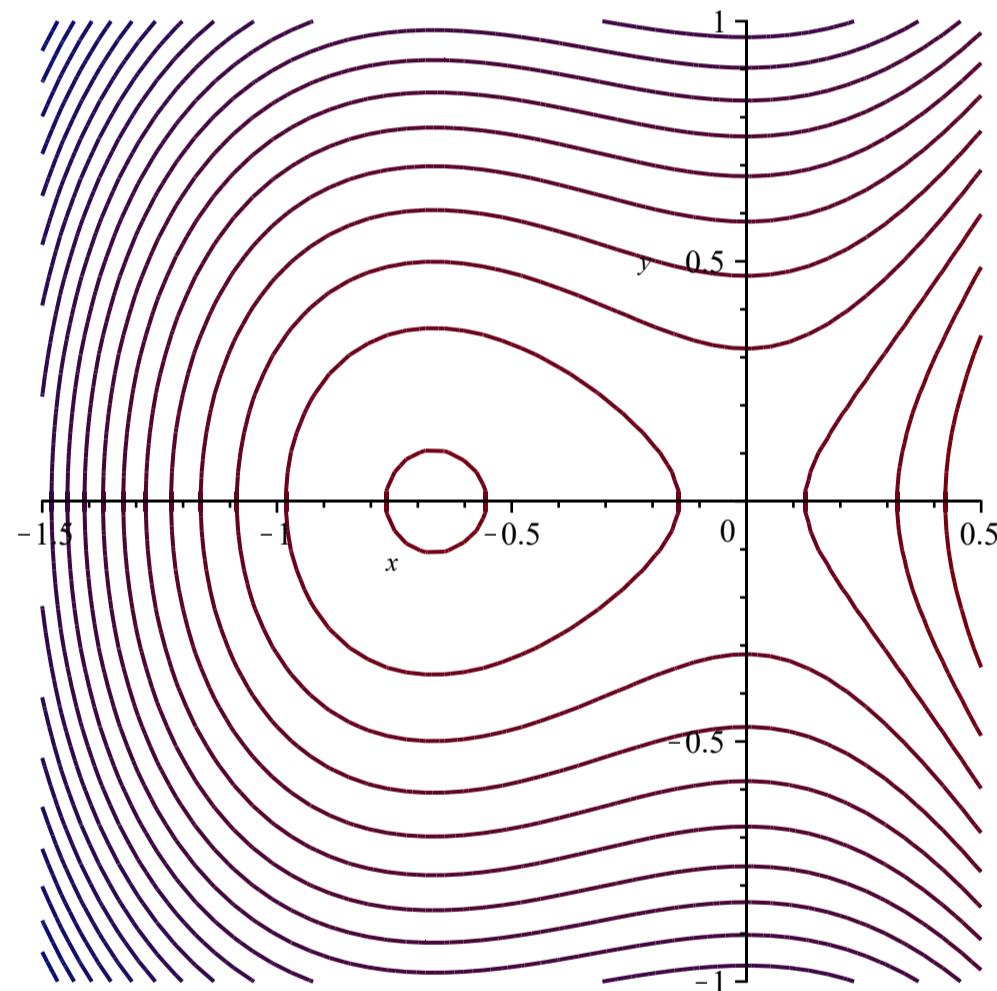
La commande suivante dessine les niveaux  $f(x,y) = c$  d'une fonction  $f$ , pour  $x$  et  $y$  variant entre les limites indiquées. Les valeurs de  $c$  sont réparties de manière uniforme entre les valeurs minimales et maximales de  $f$ , pour  $x$  et  $y$  dans les limites indiquées.

```
> contourplot(y^2 - x^3 - x^2, x = -1 .. 0.5, y = -1 .. 1);
```



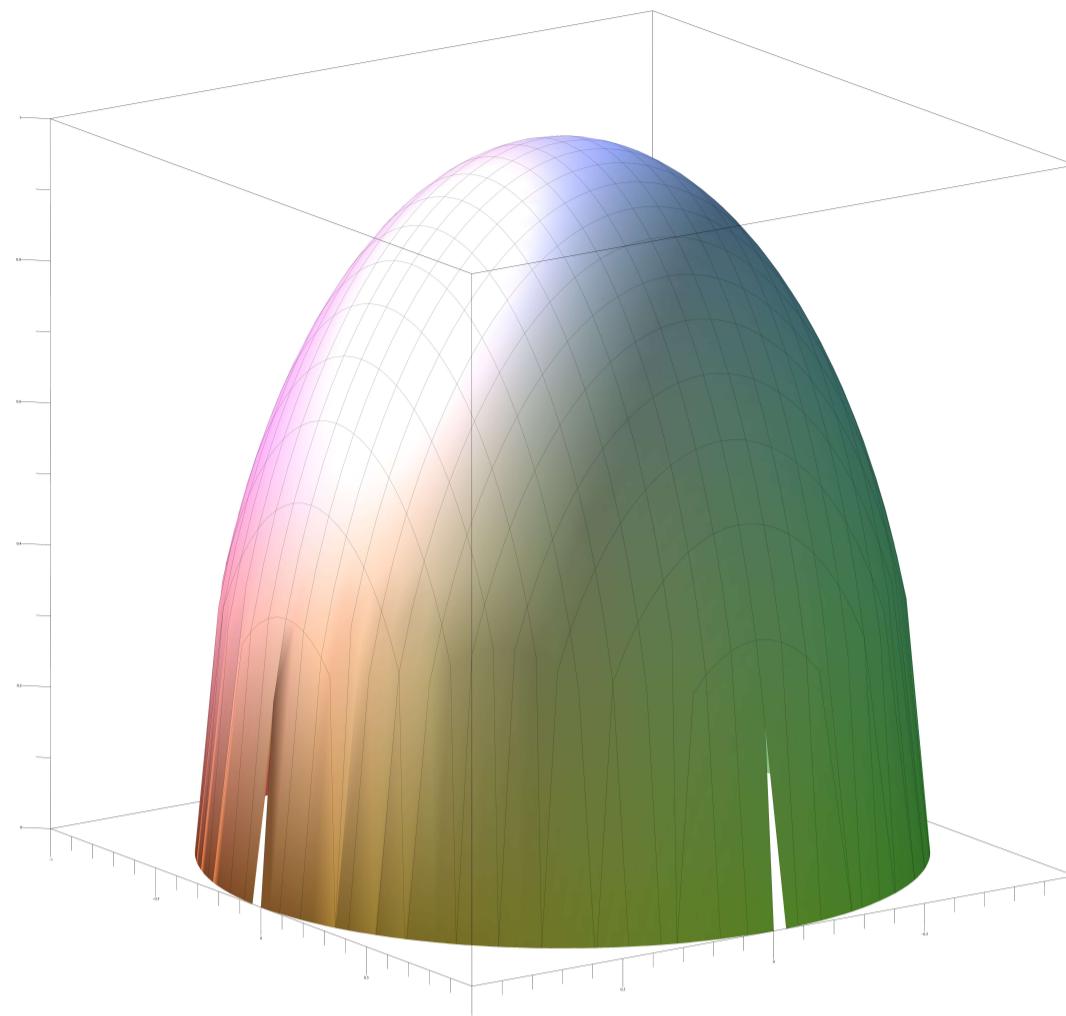
Par défaut, Maple dessine 8 contours. Si on en veut plus, il suffit de le dire :

```
> contourplot(y^2 - x^3 - x^2, x = -1.5 .. 0.5, y = -1 .. 1, contours = 20);
```



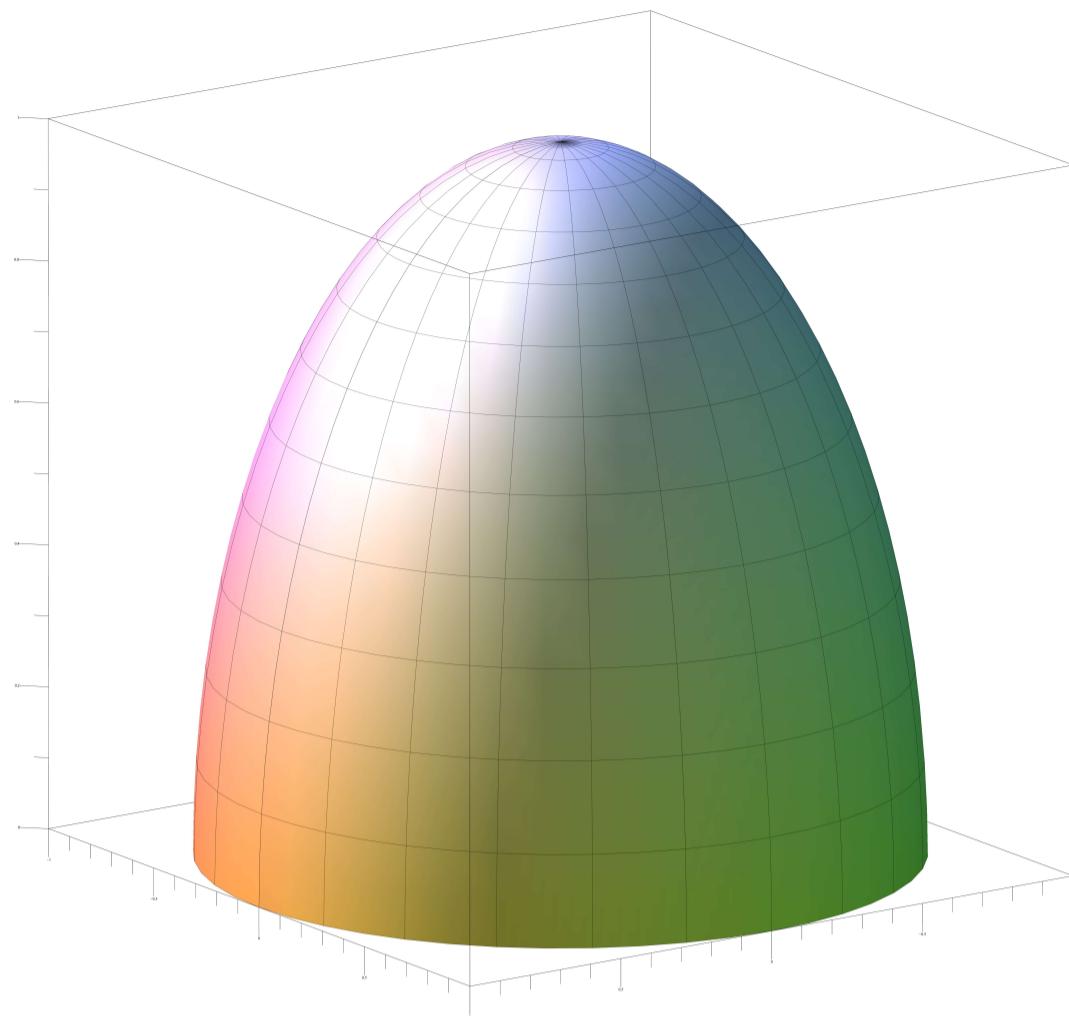
Pour dessiner des surfaces en 3 dimensions, rien de plus facile, utilisez une paramétrisation de votre surface. Pour la demi-sphère:

```
> plot3d(sqrt(1 - (y^2 + x^2)), x = -1 .. 1, y = -1 .. 1, contours = 20);
```



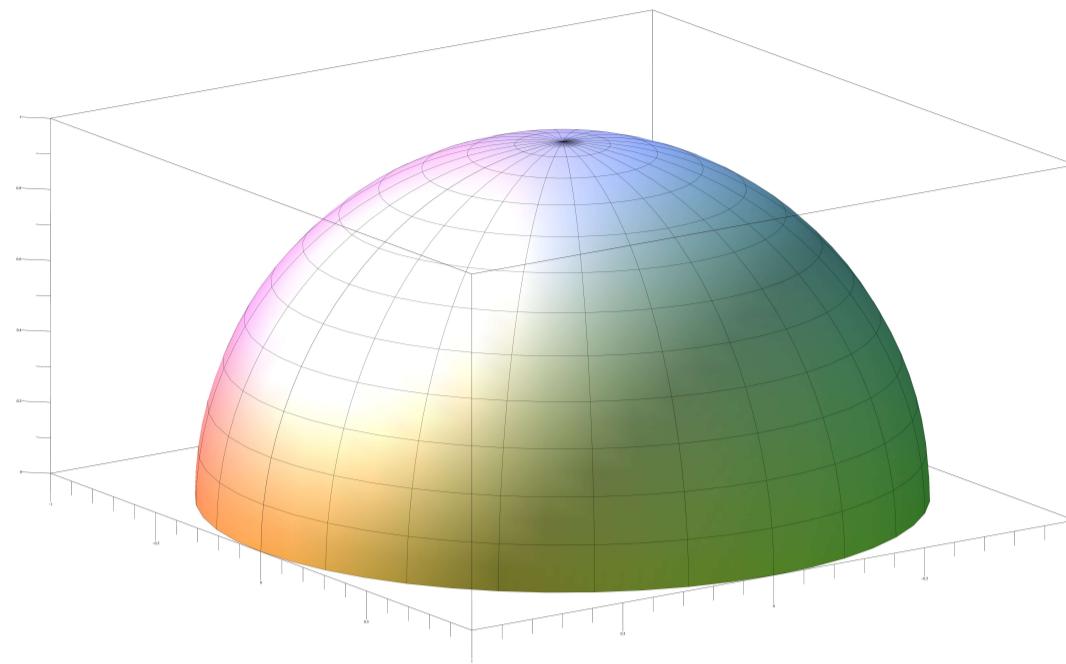
Le bord n'est pas génial, mais si on prend une autre paramétrisation cela va mieux

```
> plot3d([cos(x)*cos(y), sin(x)*cos(y), sin(y)], x = 0 .. 2*Pi, y = 0 .. Pi, contours = 20);
```



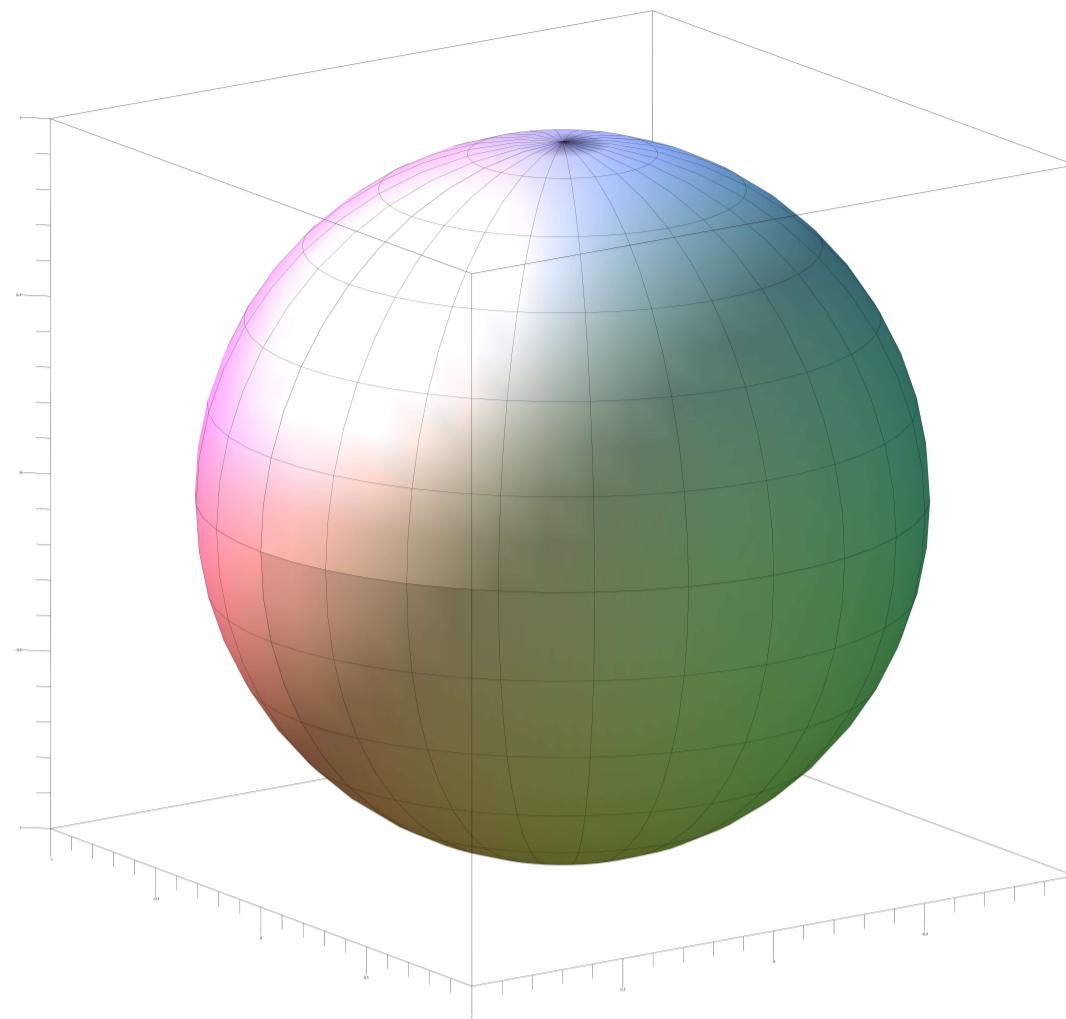
A l'oeil, ça n'a pas l'air d'une demi-sphère... Parce qu'il faut garder la même échelle sur les axes, comme dans le cas de la commande **plot**.

```
> plot3d([cos(x)*cos(y), sin(x)*cos(y), sin(y)], x = 0 .. 2*Pi, y = 0 .. Pi, scaling = constrained);
```



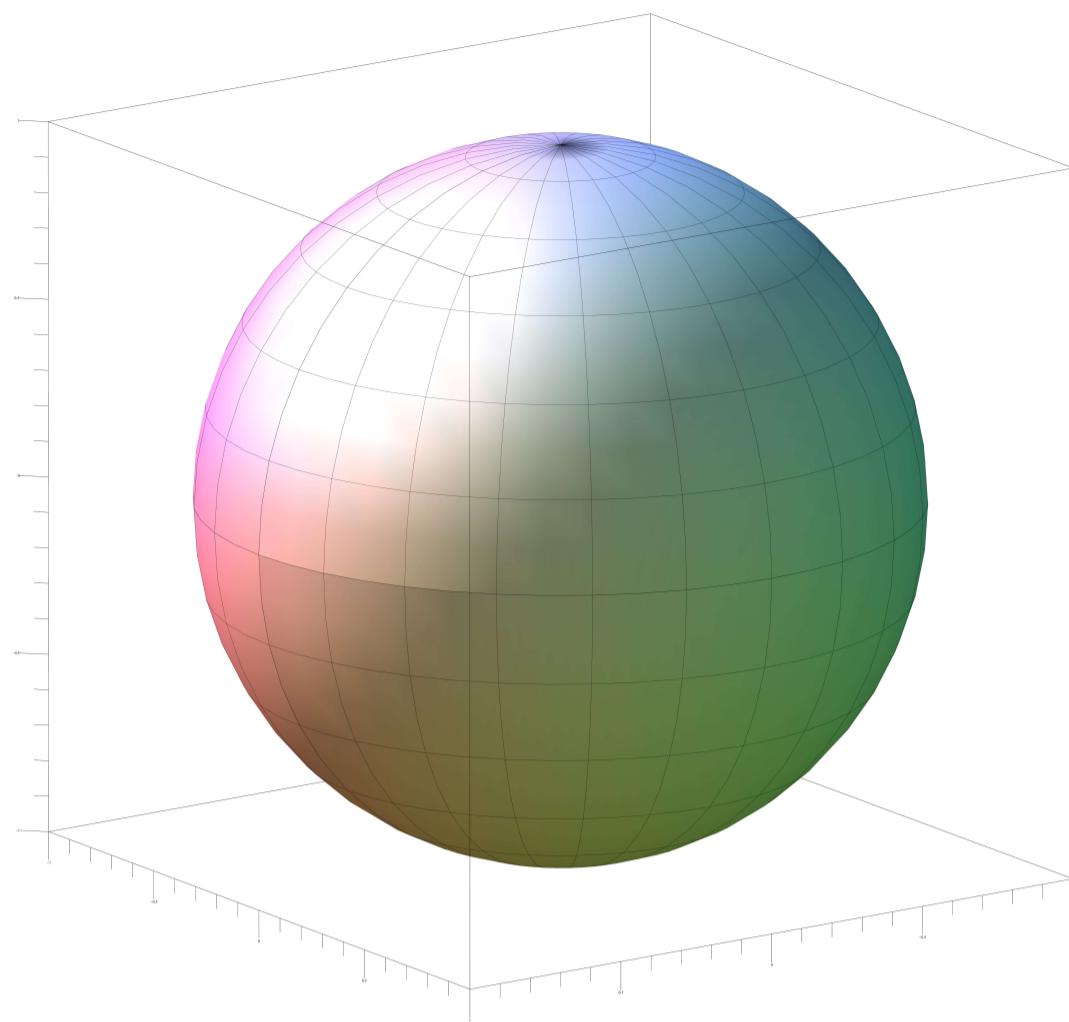
La sphère peut aussi se dessiner complètement.

```
> plot3d([cos(x)*cos(y),sin(x)*cos(y),sin(y)], x = 0 .. 2*Pi, y = 0 .. 2*Pi);
```

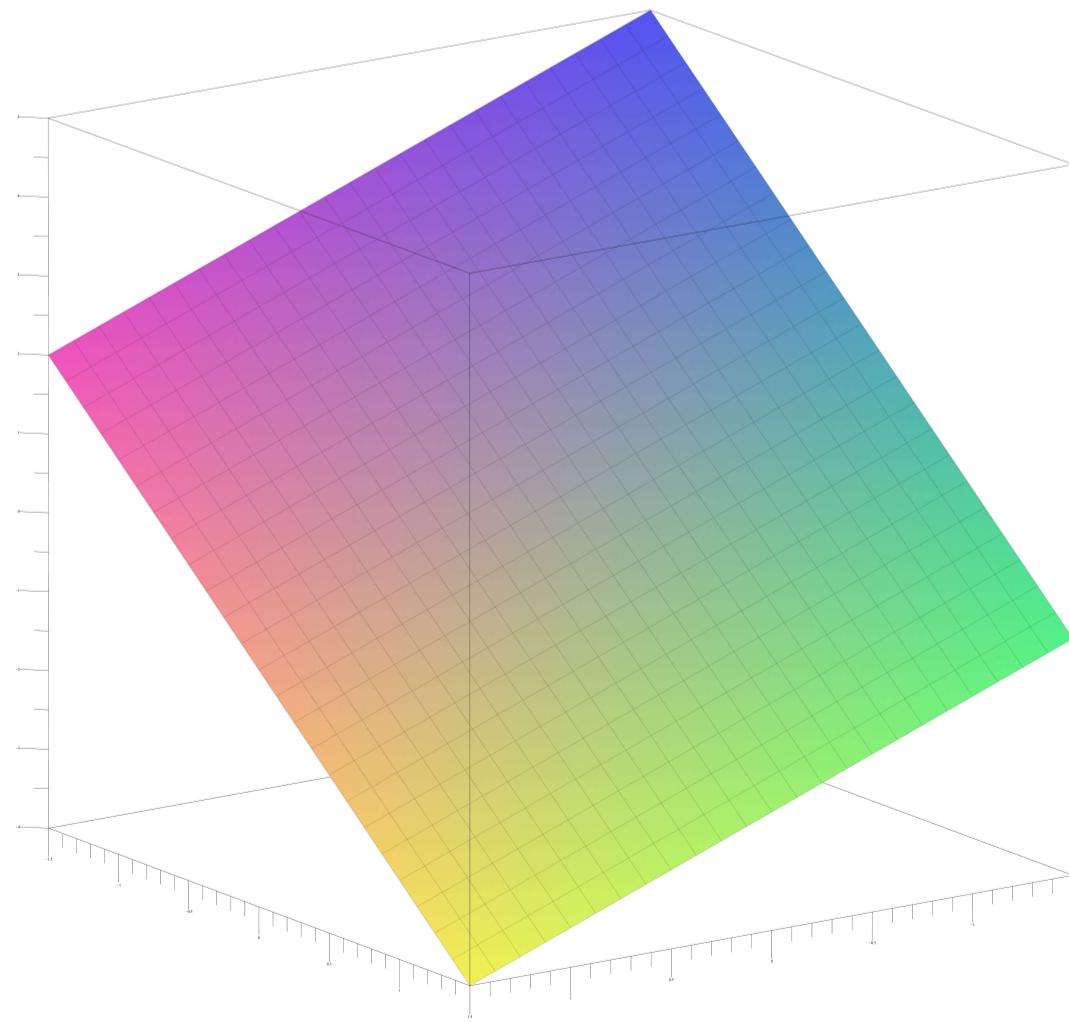


On peut aussi dessiner plusieurs surfaces ensemble

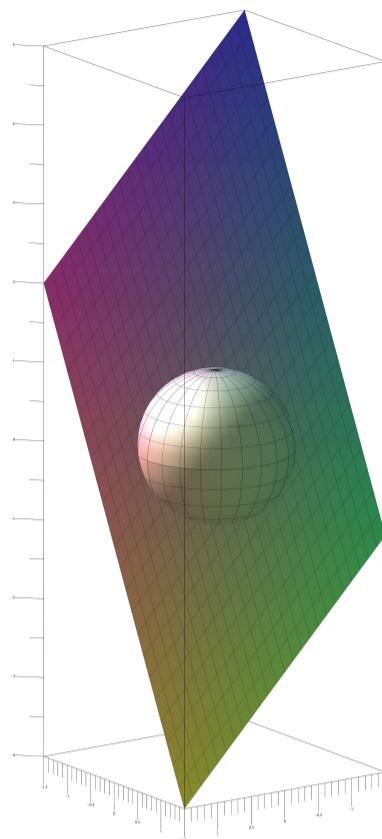
```
> sphere := plot3d([cos(x)*cos(y),sin(x)*cos(y),sin(y)], x = 0 .. 2*Pi, y = 0 .. 2*Pi);
```



```
> plan := plot3d(0.5 - x - 2*y, x = -1.5 .. 1.5, y = -1.5 .. 1.5);
```



```
> display({sphere,plan}, scaling = constrained);
```



## 5. Procédures dans Maple

### Un exemple et généralités

Voici une procédure qui calcule la somme des nombres de 1 jusqu'à n :

```
> f := proc(n)          # n est un paramètre que l'on peut passer à la procédure f
  local i, temp;        # On déclare les variables locales de la procédure.
  temp := 0;            # On assigne la valeur zero à la variable temp.
  # Une boucle, voir ci-dessous. La variable i va être incrémentée de 1 à n, et pour chaque valeur de i, on
  exécute la commande après "do".
  for i from 1 to n do
    temp := temp + i; # Commande répétée pour chaque valeur de i.
  end do;              # Indique que la boucle est terminée, indispensable!
end proc;            # Indique que le code de la procédure est terminé, indispensable!
f := proc(n) local i, temp; temp := 0; for i to n do temp := temp + i end do end proc
```

```
> f(1);               # On teste notre procédure
```

1

```
> f(10);
```

55

On introduit une procédure au moyen de la commande **proc**, suivie des arguments entre parenthèses, séparés par des virgules. Notez l'utilisation du point virgule. Pas de point virgule après la commande **proc**. Mais chaque ligne de la procédure se termine par un point virgule.

Il n'est pas nécessaire de mentionner quelle est la valeur renvoyée par la procédure. Celle-ci retourne par défaut le dernier calcul effectué. Dans le cas ci-dessus, c'est la dernière valeur prise par la variable **temp**. Il est aussi possible de mentionner explicitement quelles variables doivent être renvoyées, au moyen de la commande **return**.

On termine la procédure par la commande **end proc**, qui est indispensable.

Attention : on n'a pas le droit de modifier les paramètres passés à la procédure :

```
> double := proc(n) # calcule 2 fois n
  local temp;
  temp := n;
```

```

n := 2*temp;
end proc;
double := proc(n) local temp; temp := n; n := 2 * temp end proc

```

(6.1.4)

```

> double(3);
Error, (in double) illegal use of a formal parameter

```

Par contre:

```

> redouble:=proc(n)
  local temp;
  temp:=n;
  2*temp;
end proc;
redouble := proc(n) local temp; temp := n; 2 * temp end proc

```

```

> redouble(3);

```

6

(6.1.6)

## Boucles

Pour effectuer une commande plusieurs fois, on utilise une boucle, dont on a vu une utilisation dans l'exemple ci-dessus, avec la commande **for**. Celle-ci s'utilise comme suit

```

for [Variable] from [Valeur de départ de la variable] to [Valeur finale de la variable] do [Commandes à exécuter]
end do;

```

Notez la commande **end do**, qui est indispensable.

## If ... then ...

Quand on veut exécuter une commande seulement si une certaine condition est respectée, on utilise la commande **if ... then ...**, qui s'utilise comme suit:

```

if [Condition] then [Commande à exécuter si la condition est vraie]
elif [Condition 2] then [Commande à exécuter si la première condition est fausse et la deuxième est vraie]
else [Commande à exécuter si toutes les conditions sont fausses];
end if;

```

Notez la commande **end if**, qui est indispensable. Les parties introduites avec **elif** et avec **else** ne sont pas nécessaires pour que la commande fonctionne.

Les opérateurs logiques **and** et **or** sont souvent utiles pour formuler les conditions.

Voici un exemple à comprendre et à tester utilisant une commande **if ... then ...**:

```

> g:=proc(start,n_pas)
  local i,temp;
  temp:=start;
  for i from 1 to n_pas do
    if (temp mod 2)=1
      then temp:=3*temp+1;
      else temp:=temp/2;
    end if;
    print(temp);
  end do;
end proc;
g := proc(start, n_pas)
  local i, temp;
  temp := start; for i to n_pas do if temp mod 2 = 1 then temp := 3 * temp + 1 else temp := 1/2 * temp end if; print(temp) end do
end proc

```

(6.3.1)

```

> g(5,10);
16
8
4
2
1
4
2
1
4
2
5

```

(6.3.2)

```

> g(10,10);

```

5

16  
8  
4  
2  
1  
4  
2  
1  
4

(6.3.3)

(On constate expérimentalement que, quel que soit l'entier **start**, après un nombre de pas suffisamment grand on tombe sur la suite 4,2,1. On ne sait pas si c'est toujours vrai.)

## 6. Développements de Taylor

### A une variable

Pour avoir les 5 premiers termes du développement de Taylor d'une fonction d'une variable en  $x=0$ , on écrit :

```
> taylor(exp(x), x);
1 + x + 1/2 x^2 + 1/6 x^3 + 1/24 x^4 + 1/120 x^5 + O(x^6)
```

Mais on peut aussi en demander plus, et en un autre point :

```
> taylor(exp(x), x = 1, 8);
e + e (x - 1) + 1/2 e (x - 1)^2 + 1/6 e (x - 1)^3 + 1/24 e (x - 1)^4 + 1/120 e (x - 1)^5 + 1/720 e (x - 1)^6 + 1/5040 e (x - 1)^7 + O((x - 1)^8)
```

La fonction dont on cherche le développement doit être passée en paramètre **sous forme d'expression**. Le dernier paramètre (ci-dessus égal à 8) est le nombre de termes calculés, en commençant par le terme constant, c'est donc 1 de moins que l'ordre du développement. Pour en faire un vrai polynôme, il faut le convertir :

```
> t3 := convert(taylor(exp(x), x = 1, 3), polynom);
t3 := e + e (x - 1) + e (x - 1)^2/2
```

```
> subs(x = 0.5, t3);
0.625000000 e
```

> 8 i

(7.1.4)

## 7. Exercices

### Exercice 1. Résolution d'une équation

Résolvez l'équation  $4x^3 - 2x - 1 = 0$ .

Remplacez les solutions trouvées dans l'équation pour les vérifier.

### Exercice 2. Dérisation

Soit  $f(x, y, z) = \exp(2x^2 - 3y) \cos(3z)$

Calculez toutes les dérivées d'ordre 2 et vérifiez l'indépendance de l'ordre de dérisation pour les dérivées mixtes.

### Exercice 3. Factorisation

On définit les polynômes

$$f(x) = x^5 - 20x^3 + 64x$$

$$g(x) = x^4 + 10x^3 + 35x^2 + 50x + 24$$

Quel est leur plus grand diviseur commun? Simplifiez le quotient  $\frac{f}{g}$ .

### Exercice 4. Limites

Considérons la fonction  $f(x) = x^{\frac{x}{1-x}}$ .

1. Trouvez les valeurs singulières de  $f$ .
2. Vérifiez que  $f$  est continue sur le complément des valeurs singulières.
3. Calculez toutes les limites de  $f$  lorsque  $x$  s'approche d'une valeur singulière ou de  $\infty$ .
4. Plottez  $f$  pour vérifier vos réponses visuellement.

Indication : Utilisez les fonctions Maple **singular**, **iscont**, **limit**.

## Exercice 5. Graphes

Tracez sur le même graphique les graphes de  $\sin(x)$  et de  $1 - \frac{1}{2} \left( x - \frac{\pi}{2} \right)^2$  pour  $x$  variant de -3 à 3. Un phénomène intéressant devrait apparaître. Expliquez ce phénomène.

## Exercice 6. Graphe et solutions

On veut résoudre l'équation  $x^3 - \cos(x^2)\sin(x) = 0$ . Dessinez le graphe du membre de gauche pour identifier les régions où on risque de trouver des racines, puis utilisez la commande `fsolve(f(x), x = a .. b)`, qui cherche une solution de  $f(x) = 0$  dans l'intervalle  $[a, b]$ .

## Exercice 7. Cône

Dessinez un cône en utilisant la fonction `plot3d`.

## Exercice 8. Approximation de $\pi$

Définissez une fonction

$$f(n) = 4 \left( 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{2n+1} \right)$$

et vérifiez informellement que  $f(n)$  tend vers  $\pi$  si  $n$  tend vers l'infini.

*Indication:* Vous pouvez utiliser la procédure de Maple `sum`, tapez `help(sum)` pour savoir comment.

## Exercice 9. Nombres premiers

Ecrivez une procédure qui retourne le nombre de nombres premiers contenus entre  $n$  et  $n + 100$ . Utilisez votre procédure pour estimer qualitativement le comportement de la densité de nombres premiers en fonction de  $n$ , pour  $n$  grand par rapport à 100.

*Indication:* Par exemple, utilisez votre procédure avec quelques nombres  $n$  proches de 1000. Répétez l'opération avec quelques nombres proches de 1000000 et de 10000000000.

## Exercice 10. Nombre d'or

a) On considère l'expression suivante:

$$\Phi = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{\dots}}}}$$

formée d'une chaîne infinie de fractions imbriquées. Une telle expression est appelée une *fraction continue*.

Ecrivez une procédure qui calcule le *nième* convergent, c'est-à-dire la fraction continue, tronquée au *nième* niveau. Le 3*ième* convergent est par exemple

$$1 + \cfrac{1}{1 + \cfrac{1}{1 + 0}} = \frac{3}{2}$$

En utilisant cette procédure, calculer les 6 premiers convergents (sous forme de fraction).

b) On définit les *nombres de Fibonacci* par récurrence, comme suit:

$$f_1 = 1$$

$$f_2 = 1$$

$$f_{n+2} = f_{n+1} + f_n$$

Ecrivez une procédure calculant le *nième* nombre de Fibonacci. Utilisez cette procédure pour calculer les fractions  $\frac{f_{n+2}}{f_{n+1}}$ . Celles-ci devraient être identiques aux convergents calculés au point a). Expliquez pourquoi.

c) Remarquez que la fraction continue  $\Phi$  coincide avec le dénominateur du second terme dans l'équation définissant  $\Phi$  ci-dessus. Utilisez cette propriété pour trouver une équation satisfaite par  $\Phi$ . Résolvez cette équation pour trouver la valeur de  $\Phi$ .

$\Phi$  s'appelle le *nombre d'or*. Comme les convergents convergent nécessairement vers la fraction continue correspondante, on a prouvé dans cet exercice que la suite formée par les quotients de nombres de Fibonacci successifs converge vers le nombre d'or  $\Phi$ .