Chapitre 2
Types fondamentaux de Maple

1. Introduction

Nous allons seulement présenter quelques types qui sont d'une utilisation fréquente et dont il convient de connaître les propriétés fondamentales. C'est souvent la méconnaissance de ces caractéristiques qui conduit certains utilisateurs à croire que Maple ne répond pas à leurs attentes.

2. Constantes

Les entiers, les fractions (représentations des nombres rationnels), les nombres à virgule flottante ainsi que les nombres complexes dont les parties réelle et imaginaire relèvent de l'un des cas précédents sont des constantes numériques pour Maple. Par ailleurs, la variable prédéfinie globale « constants » contient la séquence de tous les noms initialement connus par Maple comme des constantes symboliques.

> **constants**;

false, *γ*, ∞, *true, Catalan, FAIL, π*

Nous allons voir que dans cette réponse figurent des constantes numériques célèbres ainsi que des constantes booléennes.

2.1. Constantes numériques célèbres

Il convient de bien connaître la façon dont on désigne les constantes mathématiques usuelles dans la syntaxe du langage. La constante π est désignée par Pi (*avec un P majuscule*), à ne pas confondre avec la variable de nom π que l'on obtient par l'écriture pi (*avec un p minuscule*). Les calculs suivants montrent que Maple connaît maintes propriétés du nombre π.

> Pi; **evalf**(Pi); **cos**(Pi); **sin**(Pi); **arccos**(0);

$$\pi$$
$$3.141592654$$
$$-1$$
$$0$$
$$\pi/2$$

Maple connaît la constante d'Euler et celle de Catalan:

> gamma; **evalf**(gamma); Catalan; **evalf**(Catalan);

$$\gamma$$
$$0.5772156649$$
$$Catalan$$
$$0.9159655942$$

Les exemples précédents montrent aussi que Maple désigne l'infini par le mot-clé **infinity** :

> **infinity**; **-infinity** ;

$$\infty$$
$$-\infty$$

2.2. Autres constantes numériques usuelles

L'unité imaginaire qui permet l'écriture des nombres complexes est désignée par I. La majuscule a été préférée à la minuscule, pourtant usuelle en mathématiques, car i désigne souvent un compteur de boucle en programmation. En fait, « I » n'est pas une constante pour Maple mais un alias (*soit un synonyme*) pour la racine carrée principale de −1 :

```
> I^2; solve(x^2+4=0,x); sqrt(-1);
```

$$-1$$

$$2\,I,\ -2\,I$$

$$I$$

La base *e* de l'exponentielle népérienne ne figure pas au rang des constantes initialement connues par Maple mais s'obtient comme un effet du *pretty-printer* par :

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

$$e$$

$$2.718281828$$

3. Les Booléens

Les valeurs booléennes **true** et **false** sont répertoriées parmi les constantes initialement connues. À ces deux valeurs logiques présentes dans la plupart des langages informatiques vient s'ajouter un troisième état désigné par **FAIL** (en majuscules, contrairement à **true** et **false**). La réponse **FAIL** à une requête signifie que les éléments d'information mis à la disposition de l'interpréteur sont insuffisants pour prendre une décision. La logique mise en oeuvre par Maple comporte donc trois états. Les expressions logiques sont formées à partir de variables logiques et de connecteurs logiques comme **and**, **or**, **xor**, **implies** et **not**. Les tables de vérité binaires des connecteurs logiques usuels s'étendent naturellement à des tables à trois états, si l'on se souvient que **FAIL** signifie à peu près « je ne sais pas »

La commande **evalb** force l'évaluation d'expressions comprenant des opérateurs relationnels (test d'égalité "=", vérification d'inégalités comme "<", "<=", ">", ">=" et "<>") qui, sinon, seraient considérées comme des objets algébriques de type équation ou inéquation et ne seraient pas évaluées à leur valeur logique.

```
> 1=2;
```

$$1 = 2$$

Maple a accepté l'entrée proposée, mais ne fait rien car cet objet est de type équation et l'utilisateur n'indique pas d'action à exécuter sur cet objet. Le recours à **evalb** conduit Maple à évaluer la valeur logique de l'objet passé en argument, en le considérant comme un test.

Lorsque Maple ne dispose pas de suffisamment d'informations, **evalb** se comporte comme la plupart des commandes et se contente de répéter la question posée :

```
> evalb(1=2); evalb(x>0); evalb(2<sqrt(5));
```

false

$$0 < x$$

$$2 < \sqrt{5}$$

Afin de parvenir à ses fins dans ce dernier exemple, il convient de forcer la conversion numérique de l'objet symbolique **sqrt**(5) à l'aide de la commande **evalf** :

```
> evalb(2<evalf(sqrt(5)));
```

true

4. Nombres entiers

Au premier abord, le type entier relatif (type **integer**) ainsi que les notations des opérations arithmétiques sur les entiers ne distinguent pas Maple des autres langages de programmation :

```
> ((1+2)*(9-7))^2;
                              36
```

Les priorités entre opérations sont usuelles. En revanche, Maple en tant que système de calcul formel, est capable de représenter un nombre entier arbitrairement grand, dans la mesure des capacités matérielles du système sous-jacent[4]. C'est ce que montre l'exemple suivant qui fait intervenir un calcul de factorielle :

```
> a:=15!;
                    a := 1307674368000
```

Il est possible de connaître le nombre de chiffres d'un entier à l'aide de la commande **length** et d'obtenir une décomposition en facteurs premiers de
ce nombre avec la commande **ifactor.** Maple fournit un test de primalité à travers la commande **isprime**.

```
> length(a); ifactor(a);
                              13
                (2)^11 (3)^6 (5)^3 (7)^2 (11) (13)
> isprime(7); isprime(8);
                             true
                             false
```

Par ailleurs, voici une manière d'obtenir les dix premiers nombres premiers avec la commande **ithprime** (qui permet d'obtenir le *i*-ième nombre premier) et de déterminer le quotient et le reste d'une division euclidienne de deux nombres entiers, respectivement, à l'aide de commandes **iquo** et **irem** :

```
> seq(ithprime(k),k=1..10);
                2, 3, 5, 7, 11, 13, 17, 19, 23, 29
> iquo(31,10); irem(31,10);
                              3
                              1
```

Toujours dans le domaine de l'arithmétique, **igcd** et **ilcm** fournissent respectivement le plus grand commun diviseur et le plus petit commun multiple de deux entiers :

```
> igcd(12,30); ilcm(3,4);
                              6
                             12
```

Un entier peut être positive (>0) , negative (<0), nonneg (>=0) , posint (entier>0) , negint (entier<0) , nonnegint (entier>=0) , even (pair) , odd (impair) , primeint (premier).
Ce que l'on peut tester avec **type** ou **is**.

```
> type(-15,negint), is(13,even);
                          true, false
```

On peut sélectionner des éléments répondant à une condition booléenne avec ***select(condition,v);***

Exemple: sélectionner dans une liste de nombres ceux qui sont entiers >0

```
> select(x->type(x,posint),[-1/2,-7,Pi,3/4,0.25,18,2^5]);
                          [18, 32 ]
```

Le plus petit ou le plus grand:

```
> min(712,56,100,25,125),max(712,56,100,25,125);
                                25, 712
```

mod (a modulo n donne le reste de la division de a par n, ici 31=4*7+3) :

```
> 31 mod 4;
                                3
```

Fonction sign: cette fonction rend : -1 si n < 0 , 1 si 0 <= n

```
> n:=-7;sign(n);
                             n := -7
                              -1
```

Fonction valeur absolue:

```
> abs(n);
                                7
```

5. Les Rationnels

ce sont des objets de type *fraction* (ou *rational*) .
Un rationnel peut être *positive* (>0) , *negative* (<0) , *nonneg* (>=0) , ce qui peut être testé par ***type*** ou ***is*** (comme pour les entiers).
Quelques fonctions utilisant les rationnels:

```
> q:=21*(144/124);numer(q);denom(q);
```

$$q := \frac{756}{31}$$

$$756$$

$$31$$

6. Les Réels

un réel de type *float* , a une ***mantisse*** et un ***exposant***.
Exemple: 1234567=0.1234567*10^7 . La ***mantisse*** est 0.1234567 , et ***l'exposant*** 7 .
Le nombre s'écrit Float(0.1234567,7);

```
> evalf(Float(0.1234567,7));
                          0.1234567 10^7
```

L'utilisation du point. déclenche l'affichage en calcul flottant dans le second membre:

```
> 1/9=1.0/9;
```

$$\frac{1}{9} = 0.1111111111$$

Quelques fonctions utilisant les réels:
abs , ***sqrt*** (racine carrée) , ***exp*** , ***ln*** , ***log10*** , ***log[a]*** (logarithme de base a) , ***sin , cos , tan , cot*** (fonctions circulaires) , ***sinh , cosh , tanh , coth*** (fonctions hyperboliques) , ***arcsin ,arccos*** , ***arctan , arccot*** (fonctions circulaires réciproques) , ***arcsinh , arccosh , arctanh , arccoth***(fonctions hyperboliques réciproques).

```
> sqrt(1250);ln(exp(7));arctan(1);
```

$$25\sqrt{2}$$

$$7$$

$$\frac{1}{4}\pi$$

La fonction ***partie entière***: ***floor(x)*** est le plus grand entier inférieur ou égal à *x*.

```
> floor(-7.98);floor(7.98);
-8
7
```

7. Les Complexes

de type *complex* , sont constitués de deux réels , la partie réelle et la partie imaginaire .

```
> z:=1-5*I*sqrt(2):z,Re(z),Im(z);
```

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

evalc : fonction d'évaluation des nombres complexes

```
> z:=evalc((1+exp(1)^(I*Pi/6))/(1-I));
```

$$\frac{1}{4}+\frac{1}{4}\sqrt{3}+\left(\frac{3}{4}+\frac{1}{4}\sqrt{3}\right)I$$

8. Les Variables

Une ***variable*** informatique est l'adresse, dans la mémoire de l'ordinateur , d'un emplacement permettant de stocker un objet d'un type donné .
Exemples: AB, aB , X1, _var , x[a] sont des noms de variables MAPLE .
Si l'on affecte à une variable une valeur, la variable est dite ***assignée***.
Si elle n'est associée à aucune valeur, la variable est dite ***non assignée***.
La fonction ***restart*** permet de réinitialiser toutes les variables de la feuille de calcul.

```
> restart;
```

var est une variable assignée, dont la valeur est 3 :

```
> var:=3;
```

var := 3

La variable *a* et la variable indicée χ_a sont non assignées :

```
> a,x[a];
```

$a,\ \chi_a$

Une autre manière d'affecter une valeur à une variable est d'utiliser la fonction ***assign***:

```
> assign(nom,10-var);
> nom;
7
```

Pour désaffecter une variable assignée, lui affecter son nom écrit entre deux accents aigus

```
> nom:='nom':nom;
```

nom

On peut aussi utiliser la fonction ***evaln*** (évaluer en un nom) :

```
> var:=evaln(var):var;
```

var

9. Les Expressions

Elles sont évaluées de la gauche vers la droite , avec la priorité usuelle des opérateurs.
Pour afficher une expression, utiliser les fonctions ***lprint*** ou ***print*** :

```
> lprint(1-2/a+(a+1)/(a-5));
1-2/a+(a+1)/(a-5)
> print(1-2/a+(a+1)/(a-5));
```

$$1-\frac{2}{a}+\frac{a+1}{a-5}$$

Toute expression a un ***type*** que l'on peut obtenir grâce à la fonction ***whattype*** :
Parmi les types possibles figurent : + (somme) , * (produit) , ^ (puissance) , = (égalité) , <> (non égalité) , < , <= (inégalité au sens strict ou large) , and , or , not (et,ou,non) , integer (nb entier) , fraction (nb rationnel) , float (nb réel en virgule flottante) , complex (nb complexe) , numeric (numérique) , symbol (symbole), string (chaîne de caractères) , list (liste) , set (ensemble) , table (table) , array(tableau) , function (fonction) , name (nom) , .. (intervalle).

```
> whattype(a+b),whattype(a>b),whattype(-1.2245),whattype(a^7),whattype((a and b) or
(c or d));
```

+, <, *float*, ^, *or*

Autres expressions:

```
> expr:=x-3*y+5/(x+y);
```

$$expr := x - 3\,y + \frac{5}{x+\mathrm{y}}$$

nops(expression) donne le nombre d'opérandes de l'expression:

```
> nops(expr);
```

3

op(i,expression) donne le i-ème opérande de l'expression:

```
> op(2,expr);
```

$-3\,y$

op(i..j,expression) donne les opérandes de l'expression, du i-ème au j-ème:

```
> op(2..3,expr);
```

$$-3\,y\,,\ \frac{5}{x+\mathrm{y}}$$

10. Les Ensembles

Un ensemble Maple (variable de type **set**) s'obtient en délimitant une séquence par des accolades. Et contrairement à certains langages comme Caml, Maple n'exige pas que tous les éléments d'un ensemble relèvent du même type :

```
> E:={$1..6}; F:={1,2,3,a,2,a,b,4};
E := {1, 2, 3, 4, 5, 6}
F := {1, 2, 3, 4, a, b}
```

On constate sur l'exemple précédent que, conformément à la règle mathématique, Maple supprime les doublons. Par ailleurs, toujours selon cet usage, l'ordre des éléments d'un ensemble n'est pas contractuel :

```
> evalb({1,2,3}={2,1,3});
```

true

Cependant, même s'il faut pour des raisons évidentes éviter de faire reposer un calcul sur l'ordre des éléments d'un ensemble, on peut constater que cet ordre demeure figé tout au long d'une session et qu'il dépend seulement de la session. Les ensembles en Maple sont donc une implémentation des ensembles mathématiques finis.

On obtient le nombre d'éléments d'un ensemble avec la commande **nops** et la séquence sous-jacente à un ensemble grâce à la commande **op**. Pour accéder directement à un élément d'un ensemble, il suffit d'indiquer sa position dans l'ensemble, soit à l'aide de la commande **op** à laquelle on passe la position en premier argument soit directement à l'aide de l'opérateur de sélection « [] ». En revanche, cette syntaxe ne permet pas de modifier un élément de l'ensemble:

> **nops**(F); **op**(F); **op**(4,F); F[5]; F[5]:=12;

6

1, 2, 3, 4, *a*, *b*

4

a

Error, cannot reassign the entries in a set

Par ailleurs, un certain nombre d'opérations usuelles sur les ensembles sont implémentées dans Maple. Ainsi, l'union, l'intersection et la différence non symétrique s'obtiennent respectivement à l'aide des opérateurs **union**, **intersect** et **minus**. L'appartenance à un ensemble peut être testée avec la commande **in** et afin d'obtenir le résultat d'évaluation booléenne de l'expression, la demande doit être fait explicitement :

> A:={**seq**(2*i,i=1..10)}; B:={**seq**(3*i,i=1..10)};

A := {2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

B := {3, 6, 9, 12, 15, 18, 21, 24, 27, 30}

> A **union** B; A **intersect** B; A **minus** B;

{2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 27, 30}

{6, 12, 18}

{2, 4, 8, 10, 14, 16, 20}

> 2 **in** A; 4 **in** B ; **evalb**(2 **in** A); **evalb**(4 **in** B);

$2 \in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$

$4 \in \{3, 6, 9, 12, 15, 18, 21, 24, 27, 30\}$

true

false

11. Les Listes

On obtient une liste (variable de type **list**) en délimitant une séquence par des crochets. Maple, contrairement à certains langages comme Caml, n'exige pas que tous les éléments d'une liste relèvent du même type :

> L:=[$1..6]; F:=[1,2,3,a,2,a,b,4];

L := [1, 2, 3, 4, 5, 6]

F := [1, 2, 3, a, 2, a, b, 4]

Les listes se distinguent des ensembles : d'une part, les doublons éventuels sont conservés, comme le montre l'exemple précédent, et, d'autre part, l'ordre des éléments est contractuel, comme en témoigne l'exemple suivant :

> **evalb**([1,2,3]=[2,1,3]);

false

Comme dans le cas des ensembles le nombre d'éléments d'une liste s'obtient avec la commande **nops** et la séquence sous-jacente à une liste avec la commande **op.** L'accès à un élément se fait à laide de la commande **op** ou directement :

> T:=[1,3.2,.5,a,7]; **nops**(T); **op**(T);

$T := [1, 3.2, 0.5, a, 7]$

5

1, 3.2, 0.5, a, 7

> **op**(2,T); T[3];

3.2

0.5

De manière plus générale, on peut extraire d'une liste une sous-liste (d'éléments consécutifs) de la manière suivante :

> T[2..4];

$[3.2, 0.5, a]$

Enfin, contrairement au cas des ensembles, on peut modifier directement un élément de la liste avec cette même syntaxe :

> T[2]:=55; T;

$T_2 := 55$

$[1, 55, 0.5, a, 7]$

Maple propose diverses commandes particulièrement utiles pour manipuler les listes.
Parmi ces commande : **select**(f,L), qui permet de sélectionner dans une liste les éléments de la liste L qui satisfont à la fonction (*ou prédicat*) f à valeurs booléennes, La commande **remove** fonctionne comme **select**, mais elle retire les éléments qui satisfont au critère de sélection f, la commande **map**(f,L) permet d'appliquer la fonction f à tous les éléments de la liste L,

> **select**(**isprime**,[$1..20]); **remove**(**isprime**,[$1..20]);

[2, 3, 5, 7, 11, 13, 17, 19]

[1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20]

> **map**(f,[a,b,c,d]); **map**(x->x^2,[$1..10]);

$[f(a), f(b), f(c), f(d)]$

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

12. Les fonctions

12.1. Les fonctions prédéfinies

Maple nous propose une foule de fonctions prédéfinies. Parmi celles-ci, on compte les **sin**, **log**, **exp**, **sqrt** (racine carrée). En plus de ces grands classiques, d'autres relèvent de domaines spécialisés et beaucoup sont probablement inconnues. On peut les rencontrerez en déroulant le menu **Help**, en cliquant sur **Contents** et en ouvrant la section **Mathematics**. Le reste est une question d'exploration.

Notez que toutes les fonctions "fonctionnent" de la même manière. Pour en tirer un calcul et un résultat, il vous suffira de taper le nom de la fonction correctement et de lui fournir un argument entre parenthèses:

```
> abs(-5);
        5
```

Si vous connaissez le nom d'une fonction et que vous n'êtes pas certain(e) de ce qu'elle fait ou de la syntaxe de ses arguments, faites simplement : **?nom_de_fonction**

Par exemple, pour la fonction abs de "mise en valeur absolue":

```
>?abs
```

12.2. Définition d'autres fonctions

Il est très commode de savoir définir ses propres fonctions pour tailler les calculs sur mesure. Les deux principaux mécanismes de définition sont:

- la notation flechee ->
- l'operateur **unapply**

Notations fléchées

C'est certainement la façon la plus simple et elle rappelle la notation mathématique traditionnelle.

Voici, par exemple, une définition fléchée de la fonction "cosinus du carre".

```
> cos_carre :=x->cos(x^2);
```

$$\mathbf{cos\_carre := x \to \cos(x^2)}$$

Une fois définie, elle acceptera divers types d'arguments:

```
>cos_carre(5) ;
```

$$\mathbf{\cos(25)}$$

```
>cos_carre(x) ;
```

$$\mathbf{\cos(x^2)}$$

```
>cos_carre(1.5) ;
```

$$\mathbf{-.6281736227}$$

Une fonction à une variable :

```
>f :=x->exp(x)-x ;
```

$$\mathbf{f := x \to e^x - x}$$

On peut calculer facilement f(3*x+2)

Une fonction à plusieurs variables :

*>f :=(x,y,z,t) ->x^2-3*y+4*z-t ;*

Aussi on peut aisément calculer f(2*a,b,3*c,d)

Avec **unapply**, à partir d'une expression

On peut avoir d'excellentes raisons de désirer construire une fonction à partir d'une expression. Le cas le plus typique est celui du polynôme dont on veut tirer une fonction polynomiale. Prenons par exemple:

*>px :=55*x^5-37*x^4-35*x^3+97*x+50 ;*

$$px := -55\,x^5 - 37\,x^4 - 35\,x^3 + 97\,x + 50\ ;$$

On aimerait bien effectuer des évaluations en un point particulier de **px**, en faisant simplement:

>px(3) ;

$$-55\,x(3)^5 - 37\,x(3)^4 - 35\,x\,(3)^3 + 97\,x(3) + 50$$

On voit bien que ca ne marche pas du tout. La raison de ce comportement vient de ce que ***px*** n'est pas une fonction, mais une expression. Il faudra donc transformer ***px*** en une fonction. L'operateur **unapply** sert précisément à cette transformation:

>fonc_px := unapply(px,x) ;

$$fonc\_px := x \to -55\,x^5 - 37\,x^4 - 35\,x^3 + 97\,x + 50$$

dès lors on obtient facilement

>fonc_px(3) ;

$$9764$$

12.3. Composition des fonctions

Le symbole **@** permet de calculer la composition de fonctions

*>f :=u->u^3 : g :=t->2*t+1 : (f@g)(a);(g@f)(a);*

$$(2a+1)^3$$

$$2a^3+1$$

fofof correspond à :

>(f@@3)(u) ;

$$u^{27}$$

12.4. Extremums d'une fonction

Deux commandes de base sont disponibles pour trouver les extremums d'une fonction d'une ou de plusieurs variables :

Minimize (expression, option1, option2, …, option3)

Maximize (expression, option1, option2, …, option3)

Exemple 1 : Trouver le minimum de $(x-2)^2$ sur l'intervalle [1,5]

```
> restart ;
fonction := (x-2)^2 ;
minimize(fonction, x=1..5,location) ;
```

$$0,\{[\{x=2\},0]\}$$

location indiquera quelles sont les coordonnées du point ou se trouve ce minimum.

Exemple 2 : trouver le minimum et le maximum de $f(x,y)=x^2+y^2$ sur le domaine [-3,3]x[-4,4]

```
>restart ;
fonction := x^2 + y^2 ;
minimize(fonction, x = -3..3, y = -4..4, location) ;
maximize(fonction, x = -3..3, y = -4..4, location) ;
```

$$x^2 + y^2$$

$$0, \{[\{x = 0, y = 0\}, 0]\}$$

$$25, \{[\{x = -3, y = -4\}, 25], [\{x = -3, y = 4\}, 25], [\{x = 3, y = -4\}, 25], [\{x = 3, y = 4\}, 25]\}$$

12.5. Dérivation explicite

La syntaxe est :
Diff(expression, x1, x2, … xn);

Exemple 1 : soit à calculer $\partial f(x)/\partial x$ ou $f(x) = x^2+ax+1$

```
>f := x^2+ax+1 ;
```

$$x^2+ax+1$$

```
>df :=diff(f,x) ;
```

$$2x+a$$

Ou bien à calculer $\partial f(x)/\partial a$

```
>dfa :=diff(f,a) ;
```

$$x$$

Exemple 2 : soit à calculer $\partial f(x,y)/\partial x$ ou $f(x,y) = \sin(x+y)*\ln(xy)$

```
>expression := sin(x +y) * ln(x*y) : diff(expression,x) ;
```

$$\cos(x+y)\ln(xy) + \frac{\sin(x+y)}{x}$$

Exemple 3 : soit à calculer $\partial 2f(x,y)/\partial x\partial y$ ou $f(x,y) = \sin(x+y)*\ln(xy)$

```
>expression := sin(x +y) * ln(x*y) ;
>diff(expression, y, x) ;
```

$$-\sin(x+y)\ln(xy) + \frac{\cos(x+y)}{x} + \frac{\cos(x+y)}{y}$$

12.6. Intégration

12.6.1. Intégrale simple

La syntaxe est :
int(f(x), x = a..b) ; pour évaluer l'intégrale et afficher le résultat.
Int(f(x), x = a..b) ; pour retourner l'intégrale non évaluée.

Tel que :

- f(x) est la fonction à intégrer
- x est la variable d'intégration
- a,b sont les bornes d'intégration inferieure et supérieure respectivement

Exemple 1 : soit à intégrer la fonction f(x) = sin(x)

```
>int(sin(x),x) ;
                    -cos(x)
```

Soit à intégrer la fonction f(x) = sin(x) sur [-pi, pi]

```
>int(sin(x), x = -Pi .. Pi) ;
                      0
```

Exemple 2 : soit la fonction f(x) = x^2+ax+1 intégrer la fonction f^4

```
>f4 :=f**4 ;
              (x^2+ax+1)^4
>int(f4,x) ;
```

$$x+\frac{1}{9}x^9+\frac{1}{2}ax^8+\frac{1}{7}\left(4+6a^2\right)x^7+\frac{1}{6}\left(4a+4a\left(2+a^2\right)\right)x^6+\frac{1}{5}\left(2+8a^2+\left(2+a^2\right)^2\right)x^5+\frac{1}{4}\left(4a+4a\left(2+a^2\right)\right)x^4+\frac{1}{3}\left(4+6a^2\right)x^3+2ax^2$$

Exemple 3 : soit la fonction à intégrer f(x) = x/x^4+1

```
>Int(x/(x^4+1),x=0..1) : % = value(%);
```

$$\int_0^1 \frac{x}{x^4+1}\,dx=\frac{1}{8}\pi$$

Intégrale généralisée convergente:

```
> Int(x/(x^4+1),x=0..infinity) : % = value(%);
```

$$\int_0^\infty \frac{x}{x^4+1}\,dx=\frac{1}{4}\pi$$

Intégrale généralisée divergente:

```
> Int(exp(x)/x,x=1..infinity) : % = value(%);
```

$$\int_1^\infty \frac{e^x}{x}\,dx=\infty$$

12.6.2. Intégrale double

La syntaxe est :

Doubleint(f(x,y), x, y) l'intégrale se fera selon x et ensuite selon y
Doubleint(f(x,y), x = a..b, y = c..d)
Doubleint(f(x,y), x, y, domaine)
Tels que :

- f(x,y) est la fonction ou l'expression à intégrer.
- a, b, c, d sont les bornes d'intégration.
- Domaine est le nom du domaine qui apparaitra sous les intégrales

Pour utiliser cette commande, il est nécessaire d'ouvrir le module **student** a l'aide de **with(student)**. Voir exemple ci-dessous.

Exemple : soit à intégrer la fonction f(x,y) = x+y sur le domaine D=[1,2]x[3,4]

```
>with(student) ;
>doubleint(x+y, x = 1..2, y = 3..4) ;
```

$$\int_3^4 \int_1^2 (x+y)\, dx\, dy$$

Le package ***inttrans*** contient quelques outils pour la transformation d'intégrales.
Consulter les pages d'aide de **inttrans** pour plus de précision.

```
> with(inttrans);
```

[addtable, fourier, fouriercos, fouriersin, hankel, hilbert, invfourier, invhilbert, invlaplace, invmellin, laplace, mellin, savetable]

Exemple

```
>laplace(sin(omega*t), t, s);
```

$$\frac{\omega}{s^2+\omega^2}$$

12.6.3 Méthodes d'approximation d'intégrales:

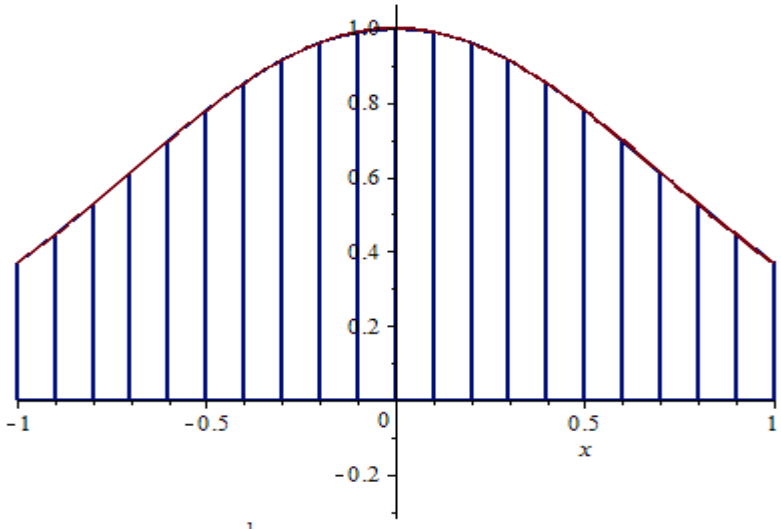
Méthode des trapèzes: calcul avec ***trapezoid(expr,x=a..b,n)***
Méthode de Simpson: calcul avec ***simpson(expr,x=a..b,n)*** , avec *n* pair.

Quelques outils du package ***student*** :

```
> with(student);
```

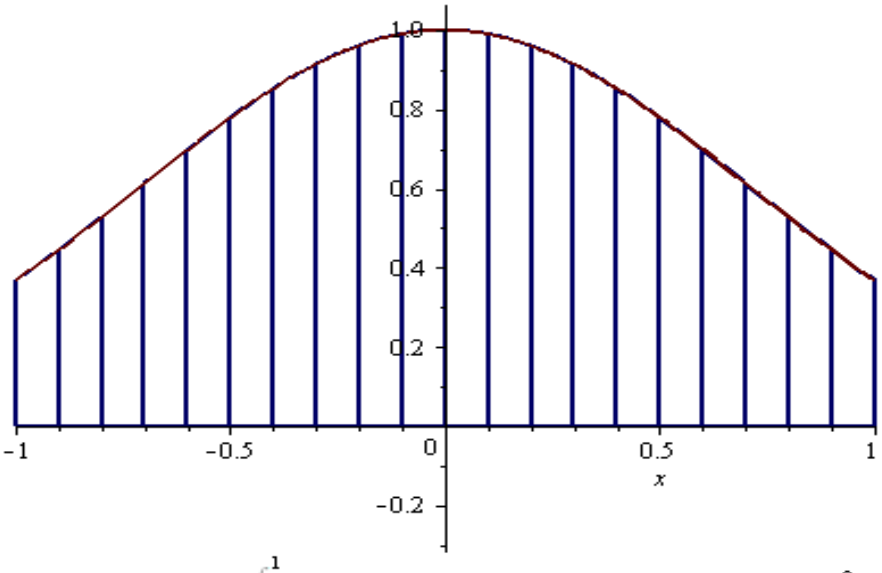
[D, Diff, Doubleint, Int, Limit, Lineint, Product, Sum, Tripleint, changevar, completesquare, distance, equate, integrand, intercept, intparts, leftbox, leftsum, makeproc, middlebox, middlesum, midpoint, powsubs, rightbox, rightsum, showtangent, simpson, slope, summand, trapezoid]

```
ApproximateInt(exp(-x^2), x = -1..1, method = trapezoid, output = plot,
    partition = 20);
```



An approximation of $\int_{-1}^{1} f(x)\,dx$ using trapezoid rule, where $f(x) = e^{-x^2}$
and the partition is uniform. The approximate value of the integral is
1.492421592. Number of subintervals used: 20.

```
ApproximateInt(exp(-x^2), x=-1..1, method=simpson, output=plot,
    partition=20);
```



An approximation of $\int_{-1}^{1} f(x)\,dx$ using Simpson's rule, where $f(x) = e^{-x^2}$
and the partition is uniform. The approximate value of the integral is
1.493648368 . Number of subintervals used:20.

12.7. Limites

On calcule la limite d'une fonction avec la commande **limit**

>Limit(f,x=a) ; $\lim_{x \to a} f$

Exemples

```
>limit(exp(x),x=infinity) ;
```

$$\infty$$

```
>limit(sin(x)/x,x=0) ;
```

$$1$$

```
> limit((2*t-3)/(3*t+4), t=infinity);
```

$$\frac{2}{3}$$

12.8. Graphisme

Grace aux commandes **plot** et **plot3d**, vous pourrez produire des graphes de plusieurs fonctions dans une même fenêtre, des graphes paramètres, etc. Nous en rencontrerons quelques exemples et le **?** fournira plusieurs autres.

Les commandes graphiques **plot** et **plot3d** sont les plus fréquemment utilisées. Le package **plots** en contient plusieurs autres. Pour obtenir des informations sur ce package : faites simplement **?plots**.

Tapez **with(plots)**: pour le charger et ainsi avoir accès à ces commandes supplémentaires.

12.8.1. Graphisme 2-D

Pour dessiner des courbes en deux dimensions, plusieurs commandes sont disponibles. Le choix de l'une de ces commandes se fait selon que la fonction est exprimée sous forme : explicite, implicite, paramétrique ou sous forme polaire.

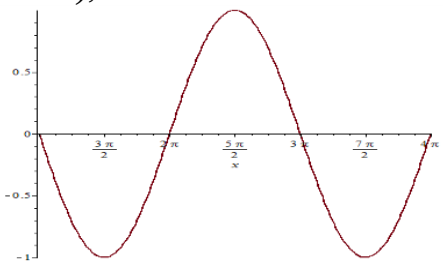
Forme explicite

Soit à dessiner la fonction f = f(x) sur l'intervalle [a,b].

Syntaxe : **plot**(f(x), x=a..b) ;

Exemple 1 : dessiner la fonction f(x) = sin(x) sur [pi, 4pi]

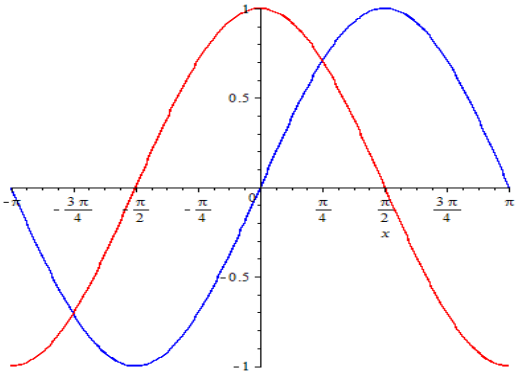
```
>plot(sin(x), x = Pi .. 4*Pi);
```



Exemple 2: Dessiner sur un même graphique les fonctions f(x) et g(x) ci-dessous, dont f(x) sera en bleu et g(x) en rouge, sur l'intervalle [-pi, pi]

f(x) = sin(x)
g(x) = cos(x)

```
>plot([sin(x),cos(x)], x = -Pi..Pi, color = [blue,red]);
```



Remarque: [sin(x),cos(x)] étant une liste de même que [blue, red] ainsi l'ordre des fonctions à dessiner suivra l'ordre de la liste des couleurs.

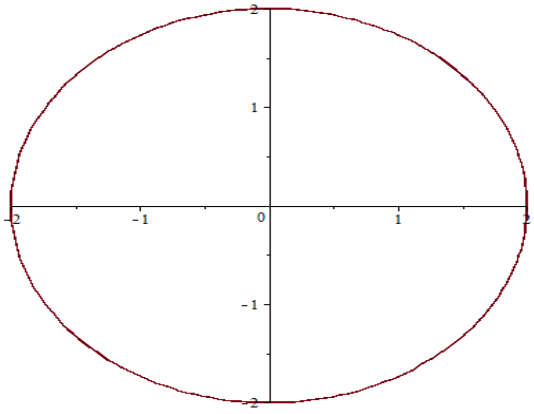
Forme paramétrique

Maple permet aussi de représenter des courbes définies paramétriquement, c'est-a-dire dont les composantes x et y dépendent toutes deux d'un même paramètre.
Soit à tracer une fonction sur l'intervalle [a,b] et dont les équations paramé>triques sont :
x = f(t)
y = g(t)
La syntaxe de la commande est :

>plot ([f(t), g(t), t = a .. b], scaling = constrained) ;

Exemple : soit à tracer un cercle de rayon R = 2 sur l'intervalle [0 .. 2*pi], dont les équations paramètriques sont :
x(t) = 2 * cos (t)
y(t) = 2 * sin(t)

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



Forme implicite

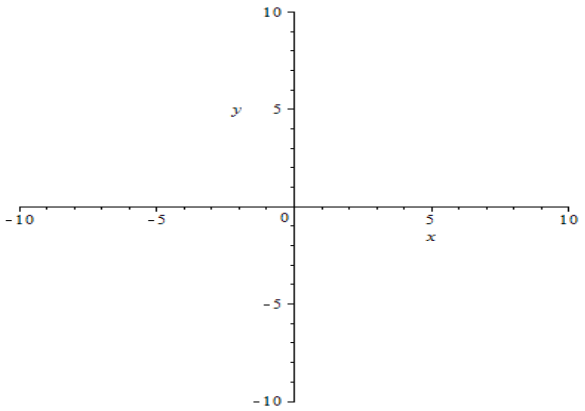
Soit à dessiner la fonction qui est donnée sous la forme implicite f(x, y) = 0. Pour cela, il est nécessaire d'introduire le module **plots**. Ainsi :

>with(plots) ;

implicitplot(f(x, y), x = a .. b, y = c .. d) ;

Exemple : Dessiner la courbe dont l'équation cartésienne est :
Exp(x*y) = cos(x-y) pour $0<x<\pi$ et $2<y<4$
Noter que y = y(x).

>with(plots) ;
*implicitplot(exp(x*y) – cos(x – y), x = 0 .. Pi, y = 2 .. 4) ;*



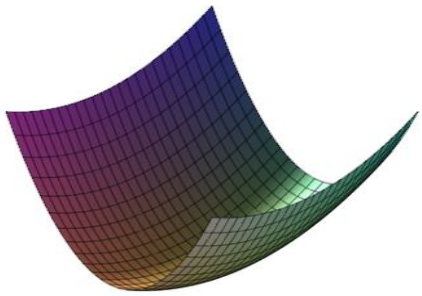
12.8.2. Graphisme 3D

Forme explicite
Soità dessiner une surface dont l'équation de cette surface est donnée sous forme z=f(x,y).
La syntaxe de la commande est :

>plot3d (f(x,y), x = a .. b, y = c .. d) ;

Exemple : soit à dessiner la surface $x^2 + y^2 = z$ pour $-1 < x < 1$ et $-2 < y < 2$

>plot3d(x^2 + y^2, x= -1..1, y=-2..2) ;



Forme implicite
Soit à dessiner la surface S lorsque l'équation de cette surface est donnée sous forme : f(x,y,z)=0.
Les commandes sont :

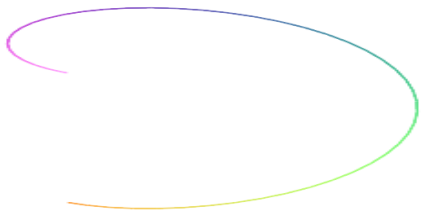
>with(plots) ;
implicitplot3d (f(x,y,z), x=a..b, y=c..d, z=e..f) ;

Forme paramétrique
Soit à dessiner une courbe dans l'espace dont on connait les équations paramétriques suivantes :
x = f(t)
y = g(t) b < t < a
z = h(t)
Les commandes sont :

```
>with(plots) ;
spacecurve([f(t),g(t),h(t)], t=a..b) ;
```

Exemple : soit à tracer dans l'espace l'hélice circulaire dont les équations paramétriques sont :
x = cos(t)
y = sin(t) $0 < t < 2\pi$
z = t
les commandes sont :

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



Soit à dessiner une surface dont on connait les équations paramétriques suivantes :
x = f(t,s)
y = g(t,s) a < t < b et c < s < d
z = h(t,s)
La commande est :

```
>plot3d([f(t,s),g(t,s),h(t,s)], t = a..b, s = c..d) ;
```

13. Développement en série de Taylor

13.1. Série de Taylor

La syntaxe est :

taylor (f(x), x = a, n) ;

tels que:

- f(x) est la fonction dont il faut écrire le développement en série de Taylor.
- a est le point autour duquel f(x) est développe en série de Taylor
- n est l'ordre du développement

Exemple : soit à développer en série de Taylor autour du point x=0 de f(x)=sin(x).

>serie_taylor := taylor(sin(x), x = 0,3) ;

serie_taylor := x + O(x^3)

13.2. Polynôme de Taylor

De la série de Taylor, il est possible d'extraire le polynôme de Taylor à l'aide de la commande **convert** et de l'option **polynom**.

Exemple :

>serie_taylor := taylor(sin(x), x = 0,3) ;

serie_taylor := x + O(x3)

>poly := convert(serie_taylor, polynom);

poly := x

Pour les séries entières, on peut utiliser le module **powseries** qui contient plusieurs commandes permettant de manipuler ces séries.

>restart ;

with(powseries) ;

14. Algèbre linéaire

Algèbre linéaire. Maple est particulièrement doué pour l'algèbre linéaire : addition, multiplication, inversion de matrices, produits scalaire et vectoriel, déterminant, transposée, résolution de systèmes, valeurs propres, base, espace des colonnes, orthogonalisation, ... il suffit de charger le package LinearAlgebra :

>with(LinearAlgebra)

```
[ &x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis,
    BezoutMatrix, BidiagonalForm, BilinearForm, CARE,
    CharacteristicMatrix, CharacteristicPolynomial, Column,
    ColumnDimension, ColumnOperation, ColumnSpace,
    CompanionMatrix, ConditionNumber, ConstantMatrix,
    ConstantVector, Copy, CreatePermutation, CrossProduct, DARE,
    DeleteColumn, DeleteRow, Determinant, Diagonal,
    DiagonalMatrix, Dimension, Dimensions, DotProduct,
    EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal,
    ForwardSubstitute, FrobeniusForm, GaussianElimination,
    GenerateEquations, GenerateMatrix, Generic,
    GetResultDataType, GetResultShape, GivensRotationMatrix,
    GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose,
    HessenbergForm, HilbertMatrix, HouseholderMatrix,
    IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal,
    IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm,
    KroneckerProduct, LA_Main, LUDecomposition, LeastSquares,
    LinearSolve, LyapunovSolve, Map, Map2, MatrixAdd,
    MatrixExponential, MatrixFunction, MatrixInverse,
    MatrixMatrixMultiply, MatrixNorm, MatrixPower,
    MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial,
    Minor, Modular, Multiply, NoUserValue, Norm, Normalize,
    NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm,
    QRDecomposition, RandomMatrix, RandomVector, Rank,
    RationalCanonicalForm, ReducedRowEchelonForm, Row,
    RowDimension, RowOperation, RowSpace, ScalarMatrix,
    ScalarMultiply, ScalarVector, SchurForm, SingularValues,
    SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector,
    SumBasis, SylvesterMatrix, SylvesterSolve, ToeplitzMatrix, Trace,
    Transpose, TridiagonalForm, UnitVector, VandermondeMatrix,
    VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm,
    VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip ]
```

14.1. Matrices

Initialisation de matrices avec Matrix :

```
>with(LinearAlgebra):
>A := Matrix([[2,3,1],[3,2,3],[0,3,2]]);
```

$$\begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 3 \\ 0 & 3 & 2 \end{bmatrix}$$

Où bien:

```
>C := Matrix ([[x-2,3,1],[3,x-2,3],[0,3,x-2]]);
```

$$C := \begin{bmatrix} x-2 & 3 & 1 \\ 3 & x-2 & 3 \\ 0 & 3 & x-2 \end{bmatrix}$$

Où bien:
>B :=matrix(3,3,[2,0,1,4,-1,1,2,0,-5]);

Une matrice ou le coef i,j est i^j :

>*M :=Matrix(3,3,(i , j)→ x* $_i{}^j$ *) ;*

$$\begin{bmatrix} x_1 & x_1^2 & x_1^3 \\ x_2 & x_2^2 & x_2^3 \\ x_3 & x_3^2 & x_3^3 \end{bmatrix}$$

On peut calculer le **déterminant** :

>*Determinant (M) ;*

$$x_1 x_2^2 x_3^3 - x_1 x_2^3 x_3^2 + x_1^3 x_2 x_3^2 - x_1^2 x_2 x_3^3 + x_1^2 x_2^3 x_3 - x_1^3 x_2^2 x_3$$

L'inverse d'une matrice avec **MatrixInverse**

>*MatrixInverse(A) ;*

$$\begin{bmatrix} \frac{5}{19} & \frac{3}{19} & -\frac{7}{19} \\ \frac{6}{19} & -\frac{4}{19} & \frac{3}{19} \\ -\frac{9}{19} & \frac{6}{19} & \frac{5}{19} \end{bmatrix}$$

Soit la matrice M : $M := \begin{bmatrix} 2 & 4 \\ 3 & 2 \end{bmatrix}$

On peut calculer valeurs propres avec : **Eigenvalues**

>*Eigenvalues(M) ;*

$$\begin{bmatrix} 2 + 2\sqrt{3} \\ 2 - 2\sqrt{3} \end{bmatrix}$$

Produit d'une matrice par un scalaire :

>*ScalarMultiply(A,2) ;*

$$\begin{bmatrix} 4 & 6 & 2 \\ 6 & 4 & 6 \\ 0 & 6 & 4 \end{bmatrix}$$

Produit de deux matrices :

>*Multiply(A,C) ;* **ou bien**

>*MatrixMatrixMultiply(A,C) ;*

$$\begin{bmatrix} 2x+5 & 3+3x & 9+x \\ 3x & 14+2x & 3+3x \\ 9 & 3x & 2x+5 \end{bmatrix}$$

Somme de deux matrices :

>*MatrixAdd(A,A) ;*

$$\begin{bmatrix} 4 & 6 & 2 \\ 6 & 4 & 6 \\ 0 & 6 & 4 \end{bmatrix}$$

14.2. Les vecteurs

Les vecteurs sont des matrices donc pour initialiser un vecteur on peut procéder comme suit:

>*b :=Matrix(3,1,[1,2,0]) ;* ou bien >*b :=Vector([1,2,0]) ;*

$$b := \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Transposée d'un vecteur ou matrice avec **Transpose**

>*Transpose(b) ;*

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

Produit vecteur /matrice

>*VectorMatrix(A,b) ;*

$$\begin{bmatrix} 8 \\ 7 \\ 6 \end{bmatrix}$$

Résolution d'un système d'équation AX=b ;

>LinearSolve(A,b) ;

$$\begin{bmatrix} \frac{11}{19} \\ -\frac{2}{19} \\ \frac{3}{19} \end{bmatrix}$$