Université Ibn Zohr Agadir
Ecole Nationale des Sciences Appliquées
Département: Génie Informatique
Responsable: Prof. A. Elyousfi

Année universitaire: 2021-2022
Section: CP1
Module: MAPLE

MAPLE

Abderrahmane ELYOUSFI
elyousfiabdo@yahoo.fr

Département
Génie Informatique

Plan du cours

- **Généralité sur l'emploi du MAPLE**
- **Manipulation des nombres**
- **Calcul algébrique**
- **Complément sur les expressions et les fonctions.**
- **Tracés de graphiques**
- **Equation, dérivation, intégration et limites.**

Généralité sur l'emploi du MAPLE

Définition:

- Maple est un logiciel de calcul formel, c'est-à-dire qu'il est capable de manipuler des nombres mais aussi des symboles représentant des nombres ou des objets mathématiques plus élaborées: équations, fonctions, matrices, etc.
- Maple est un logiciel de calcul formel: il peut traiter des données numériques (entier, réels, complexes...) de précision arbitraire et aussi des données symboliques (polynômes, expressions...).
- Ce logiciel est également doté de capacités graphiques.

Exemple 1:

```
> Limit((sin(tan(x))-tan(sin(x)))/x^7,x=0)=limit
  ((sin(tan(x))-tan(sin(x)))/x^7,x=0);
```

$$\lim_{x \to 0} \frac{\sin(\tan(x)) - \tan(\sin(x))}{x^7} = \frac{-1}{30}$$

- Dans cet exemple, comme dans les suivants, la réponse de Maple présente une écriture proche de l'écriture mathématique usuelle pour le membre de gauche. Celui-ci résulte d'un effet du pretty-printer à travers l'utilisation des commandes Limit, Diff et Int, homologues inertes des commandes limit, diff et int.

Exemple 2:

```
> Diff(arcsin(ln(x^3)),x)=diff(arcsin(ln(x^3)),x);
```

$$\frac{d}{dx}\arcsin(\ln(x^3)) = \frac{3}{x\sqrt{1-\ln(x^3)^2}}$$

- Dans cet exemple, comme dans les suivants, la réponse de Maple présente une écriture proche de l'écriture mathématique usuelle pour le membre de gauche. Celui-ci résulte d'un effet du pretty-printer à travers l'utilisation des commandes Limit, Diff et Int, homologues inertes des commandes limit, diff et int.

Exemple 3:

Exemple 3:

```
> Int(arctan(x),x=0..1)=int(arctan(x),x=0..1);
```

$$\int_0^1 \arctan(x)\,dx = \frac{\pi}{4} - \frac{1}{2}\ln(2)$$

- Maple représente le résultat obtenu de manière exacte à l'aide de constantes mathématiques célèbres (au lieu de se limiter à donner une valeur décimale approchée du résultat).

Exemple 4:

Exemple 4:

```
> S:=solve(x^3-1=0);
```

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

- On remarque que Maple propose d'emblée une représentation exacte des trois racines cubiques de l'unité sous forme de nombres complexes (l'unité imaginaire étant notée I). Intéressons-nous à la deuxième racine trouvée :

Exemple 5:

Exemple 5:

```
> u:=S[2];
```

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

```
> v:=evalf(u);
```

$$v := -0.5000000000 + 0.8660254040\, I$$

Exemple 6:

Exemple 6:

```
> evalf[40](sqrt(2));
      1.4142135623730950488016887242096980785 70
```

- On obtient une valeur décimale approchée de la racine carrée de 2 avec 40 chiffres significatifs6 (alors qu'un logiciel effectuant des calculs numériques aurait, sur la même plateforme, une précision *a priori* limitée à 15 chiffres significatifs) :

Composants du logiciel

Le logiciel MAPLE offre un ensemble d'outils qui se déclinent en trois composants principaux :

- ***la feuille de travail ;***
- ***l'interpréteur de commandes ;***
- ***le langage de programmation.***

Nous allons détailler ces différents composants ainsi que certaines de leurs caractéristiques.

Composants du logiciel

1-La feuille de travail:

- La feuille de travail (*Maple worksheet*) est le composant auquel l'utilisateur est confronté dès qu'il a lancé le logiciel.
- Son environnement peut prendre différentes formes comme on le verra un peu plus loin.
- Dans les modes évolués, c'est-à-dire autres que le mode ligne de commande, l'interface présente des menus, des barres d'outils, des menus contextuels, ou encore des fenêtres graphiques permettant à l'utilisateur d'interagir avec le système.

Composants du logiciel

1-La feuille de travail:

- Au fur et à mesure des interventions de l'utilisateur, la feuille de travail, initialement vierge, va afficher les traces des interactions entre l'utilisateur et le logiciel.
- Dans la rubrique *Save* du menu *File*, figurent différentes manières d'enregistrer ou d'exporter le travail effectué.

Composants du logiciel

1-La feuille de travail:

- La feuille de travail peut comporter trois zones de travail:

- les **zones de texte** (*text regions*), modifiables à volonté,
- les **zones d'entrée** (*input regions*), modifiables également, et exécutables par une pression de la touche <Entrée>,
- les **zones de réponse** de Maple (*output regions*), qui peuvent comporter des résultats numériques ou littéraux, des graphiques, etc. Elles ne peuvent pas être modifiées en place mais peuvent être recopiées dans une zone de texte ou une zone d'entrée par les méthodes « copier/coller » classiques sous Windows.

Composants du logiciel

2- L'interpréteur:

- L'interpréteur va déclencher des calculs et afficher des résultats selon les instructions saisies par l'utilisateur.
- Les instructions font appel à des commandes, c'est-à-dire à des fonctions faisant partie du noyau ou des bibliothèques de Maple.
- Toutes les variables crées pendant une session sont conservées avec leurs contenus jusqu'à ce qu'une instruction modifie leur valeur ou que l'on décide de procéder à une réinitialisation complète de la mémoire de travail.

Composants du logiciel

2- L'interpréteur:

- En termes de langages de programmation, un *interpréteur* s'oppose a un *compilateur*.
- En pratique, un compilateur sert le plus souvent a traduire un code source écrit une fois pour toutes dans un langage de programmation en un autre langage, habituellement un langage d'assemblage ou un langage machine.
- Le programme en langage machine produit par un compilateur est appelé code objet et c'est lui qui est effectivement exécuté lorsqu'on veut utiliser le programme constitue par le code source.
- Contrairement a un compilateur, un interpreteur execute les instructions du programme (ou en evalue les expressions), au fur et a mesure de leur lecture pour interpretation.

Composants du logiciel

3- Le langage de programmation:

- Le logiciel met à disposition de l'utilisateur un langage de programmation impératif qui permet d'écrire des fonctions nouvelles élaborées à partir de commandes Maple existantes.
- Ce langage présente de grandes similitudes dans ses primitives et ses principes de fonctionnement avec des langages comme Pascal ou C.
- Les mots clés, les structures de contrôle ainsi que les fondements de la programmation avec Maple sont abordés dans la suite de cours.

Ordre des commandes:

Ordre des commandes:

Il est très important de comprendre que le comportement du logiciel dépend de l'ordre chronologique de validation des entrées et non de l'ordre apparent, typographique, des entrées sur la feuille de calcul.

Premiers pas avec Maple

L'invite.

- Après avoir lancé le logiciel Maple à travers l'interface classique, on obtient une feuille de travail vierge sur laquelle apparaît l'invite (*prompt*) qui, par défaut, est représentée par le symbole "> ".
- L'utilisateur attentif remarquera que l'invite est précédée par une sorte de symbole qui ressemble à un crochet ouvrant ("["), appelé *crochet d'exécution* dont le rôle sera expliqué plus loin.

Premiers pas avec Maple

Caractères de fin de saisie.

- Lorsque l'on utilise Maple en interagissant directement avec l'interpréteur, l'interaction avec le logiciel consiste en une succession d'instructions saisies par l'utilisateur et de réponses fournies par l'interpréteur.
- La feuille de travail présente les traces de cette interaction. L'utilisateur doit donc saisir une instruction à côté de l'invite et pour signifier la fin de sa saisie, il lui faut terminer la ligne de commande par le caractère ";" ou par le caractère ":".
- L'instruction saisie sera interprétée et exécutée (si la syntaxe est correcte) dès que l'utilisateur aura pressé la touche Entrée.

Premiers pas avec Maple

Caractères de fin de saisie.

- Lorsqu'une instruction se finit par ";", l'interpréteur de Maple exécute l'instruction et, si elle est syntaxiquement valide15, affiche le résultat obtenu:

```
> 2+3;
                                5

> exp(1.0);
                          2.718281828
```

Premiers pas avec Maple

Caractères de fin de saisie.

- si l'instruction est terminée par ":", Maple exécute l'instruction de façon muette, c'est-à-dire n'affiche pas le résultat :

```
> 2+3:
> exp(1.0):
```

- ▲▲ e de résultats volumineux.

Premiers pas avec Maple

Groupement d'instructions.

- Pour saisir plusieurs instructions sur une même ligne et ainsi les passer simultanément à l'interpréteur, il suffit de les séparer par des ";" ou des ":".
- Elles sont alors exécutées consécutivement, dans l'ordre où elles ont été présentées, et les résultats, s'il y a lieu, sont affichés l'un sous l'autre en une seule fois par l'interpréteur.:

```
> 1+2;2.0^(1/2);(2*3)^2;
                                3
                          1.414213562
                               36
```

Premiers pas avec Maple

***Groupement d'instructions*.**

- On peut aérer la présentation des instructions et passer à la ligne entre chacune d'elles plutôt que de les écrire sur la même ligne.
- Pour passer à la ligne au sein du même crochet d'exécution sans provoquer l'exécution individuelle d'une instruction au moment du changement de ligne, il faut presser simultanément les touches Majuscule et Entrée (Shift et Enter

```
> 1+2;
  2.0^(1/2);
  (2*3)^2;
```

```
                    3
              1.414213562
                   36
```

Premiers pas avec Maple

***Groupement d'instructions*.**

- Cette façon d'aller à la ligne sans déclencher l'interpréteur est particulièrement utile lorsqu'on programme avec Maple.
- Elle permet d'écrire des fonctions Maple dont le contenu s'étend sur plusieurs lignes.

```
> 1+2;
  2.0^(1/2);
  (2*3)^2;
```

$$3$$

$$1.414213562$$

$$36$$

Premiers pas avec Maple

***Groupement d'instructions*.**

- Pour écrire une instruction dont la longueur est supérieure à celle d'une ligne sur la feuille de travail, on peut formater la saisie en plaçant un caractère "\" chaque fois que l'on va à la ligne. Maple utilise aussi ce caractère pour signaler des réponses qui tiennent sur plusieurs lignes :

```
> 1234567890\
  1234567890\
  1234567890\
  1234567890\
  1234567890\
  1234567890\
  1234567890\
  1234567890\
  1234567890+1;

      12345678901234567890123456789012345678901\
      23456789012345678901234567890123456789012\
      34567891
```

Premiers pas avec Maple

Commentaire:

- Le caractère "#" permet d'introduire un commentaire dans une feuille de travail. Ce qui se trouve entre ce caractère et la fin de ligne n'est alors pas pris en compte par l'interpréteur, comme le montre l'exemple suivant :

```
> 1+2; # un premier calcul très simple
  # 2.0^(1/2); cette ligne n'est pas prise en compte
  (2*3)^2;
```

$$3$$

$$36$$

Premiers pas avec Maple

Réinitialisation:

- La commande **"restart"** permet de libérer la mémoire interne de la feuille de travail.

Manipulation des nombres

Taille des nombres:

Taille des nombres:

```
> 100!;
  933262154439441526816992388562667004907159682643816214685929638952175\
  999322991560894146397615651828625369792082722375825118521091686400000\
  00000000000000000
> %*101-101!;
  0
```

Il ne faut tout de même pas demander l'impossible :

```
> 10^(100!);
  Error, integer too large in context
```

Opérations de base sur les entiers:

Opérations de base sur les entiers:

Opération	Symbole	Exemple
addition	+	
soustraction	-	
multiplication	*	
division :		
quotient entier (*integer quotient*)	`iquo(...)`	`> iquo(355,113);` `3`

Opérations de base sur les entiers:

Opérations de base sur les entiers:

reste (*integer remainder*)	`irem(...)`	`> irem(355,113);` 16
exponentiation	`^`	`> 2^32;` 4294967296
regroupements de termes	`(...)`	`> 2*3+4; 2*(3+4);` 10 14
valeur absolue	`abs(...)`	`> abs(-10);` 10
factorielle	`!`	`> 8!;` 40320

Opérations plus élaborés sur les entiers:

Opérations plus élaborés sur les entiers:

Opération	Fonction	Exemple

test de primalité	`isprime()`	`> isprime(2495657);`
		true
		`> isprime(2495625);`
		false

factorisation	`ifactor()`	`> ifactor(2495625);` $(3)(5)^4(11)^3$

pgcd	`igcd()`	`> igcd(1691657,3314375);`
		5303
		`> igcd(13041,8505,22113);`
		567

ppcm	`ilcm()`	`> ilcm(555,1515);` 56055

Calculs sur les rationnels

Maple définit un fractionnel à l'aide de la barre de fraction /

```
> 3/5;
```

$$\frac{3}{5}$$

```
> 3/(-12);
```

$$\frac{-1}{4}$$

```
> 1/6+1/3;
```

$$\frac{1}{2}$$

Calculs sur les rationnels

Maple simplifié la fraction quand c'est possible réduit dénominateur quant c'est nécessaire.

```
> whattype(3/5);
```

fraction

```
> 12/3;
```

4

```
> whattype(%);
```

integer

Calculs sur les rationnels

Opérations disponibles sur les rationnels.

Opération	Symbole	Exemple
les 4 opérations élémentaires	+ - * /	
regroupements de termes	(. . .)	
numérateur (penser que la fraction est d'abord simplifiée !)	numer ( )	> numer(6/15); 2
dénominateur (la fraction est d'abord simplifiée)	denom ( )	> denom(6/15); 5

Calculs sur les rationnels

Opérations disponibles sur les rationnels.

exponentiation	^	`> (5/4)^2;` $\frac{25}{16}$
valeur absolue	abs()	`> abs(-2/5);` $\frac{2}{5}$

Calculs sur les réels et les complexes

Une façon d'approcher un réel en Maple est d'introduire le point décimal.

```
> 2/3;

> whattype(%);
> 2./3;
> whattype(%);
> 123.4*10^(-6);
```

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

fraction

.6666666667

float

.000123400000

Calculs sur les réels et les complexes

On peut changer le nombre de chiffres significatifs employés par Maple en modifiants la variable Digits prédéfinie.

```
> Digits:=20:
> 355./113;
                                3.1415929203539823009

> Digits:=10:
```

Calculs sur les réels et les complexes

Pour changer le nombre de chiffres dans un calcul isolé, il est plus commode d'employer la fonction `evalf` :

```
> evalf(Pi,20);
                    3.1415926535897932385
```

`Pi` est une constante réelle prédéfinie, connue par Maple avec un grand nombre de décimales. On peut par exemple demander `evalf(Pi,1000)`.

La fonction `evalf` peut être employée sans le deuxième argument, le nombre de chiffres utilisés a alors la valeur par défaut définie par la variable `Digits`. Nous verrons plus loin l'utilité de la fonction dans ce cas.

Attention, les symboles `pi` et `PI` donnent respectivement les *caractères* π et Π, et rien d'autre ! (C'est une cause fréquente d'erreur.)

Calculs sur les réels et les complexes

On peut définir un complexe en introduisant le nombre imaginaire i, qui doit être noté I pour Maple.

```
> (5+4*I)-(1+2*I);
```

$$4 + 2\,I$$

```
> (5+4*I)^3;
```

$$-115 + 236\,I$$

```
> sqrt(-4);
```

$$2\,I$$

```
> solve(x^2+5=0);
```

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

Calculs sur les réels et les complexes

Opérations disponibles sur les complexes.

Fonction ou opération	Syntaxe Maple
Partie réelle de z	`Re(z)`
Partie imaginaire de z	`Im(z)`
Conjugué de z	`conjugate(z)`
Module de z	`abs(z)`
Argument de z	`argument(z)`
Définition trigonométrique d'un complexe z	`z:=polar(r,theta)`

Calculs formels sur les réels et les complexes

Calcul décimal, approché :

```
> sqrt(3.); sqrt(9.);
```

1.732050808
3.000000000

Calcul formel, exact :

```
> sqrt(3); sqrt(9);
```

$\sqrt{3}$
3

```
> sqrt(I);
```

$\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2}$

```
> cos(Pi/6);
```

$\frac{1}{2}\sqrt{3}$

```
> arcsin(sqrt(3)/2);
```

$\frac{1}{3}\pi$

Quelques fonctions prédéfinies

Maple connaît de très nombreuses fonctions, les unes élémentaires, les autres dites « spéciales ». Ces dernières peuvent être définies par des intégrales, des séries, des relations de récurrences, etc. Voici des exemples de fonctions élémentaires connues par Maple et leurs notations :

Quelques fonctions prédéfinies

Fonctions	Noms des fonctions dans Maple
Exponentielle	`exp`
Logarithme népérien	`ln ou log`
Logarithme décimal	`log10`
Racine carrée	`sqrt`
Fonctions circulaires	`sin, cos, tan`
Fonctions circulaires inverses (avec les mêmes conventions de détermination qu'en math.)	`arcsin, arccos, arctan` (`arctan` peut s'employer avec 2 arguments : voir l'aide)
Fonctions hyperboliques	`sinh, cosh, tanh`
Fonctions hyperboliques inverses	`arcsinh, arccosh, arctanh`
Valeur absolue (pour un réel) ou module (pour un complexe)	`abs`

Quelques fonctions prédéfinies

Entier le plus proche	`round`
Entier le plus proche en allant vers 0	`trunc`
Plus grand entier inférieur ou égal à x (partie entière de x)	`floor(x)` (*floor* = sol, plancher)
Plus petit entier supérieur ou égal à x	`ceil(x)` (*ceiling* = plafond)
(Fausse) « partie fractionnaire » de x (attention si x<0 !)	`frac(x) = x-trunc(x)`
Le plus grand des réels x1,x2,...,xn	`max(x1,x2,...,xn)`
Le plus petit des réels x1,x2,...,xn	`min(x1,x2,...,xn)`

Quelques fonctions prédéfinies

```
> sqrt(-1);
```

$$I$$

```
> sqrt(I);
```

$$\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2}$$

```
> sqrt(-I);
```

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

Quelques fonctions prédéfinies

```
> sqrt(1-I);
```

$$\sqrt{1-I}$$

Malgré ce premier échec, tout n'est pas perdu. Il existe une fonction qui tente d'aller plus loin que les méthodes employées automatiquement par Maple, c'est la fonction `evalc` (avec un c, comme « complexe ») :

```
> evalc(%);
```

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

Quelques fonctions prédéfinies

Enfin, on a toujours la possibilité de demander une évaluation approchée, et cela de deux façons différentes :

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

$$1.098684113 - .4550898606\,I$$

```
> sqrt(1.-I);
```

$$1.098684113 - .4550898606\,I$$

Remarque : Maple considère la notation `z^(1/2)` comme exactement synonyme de `sqrt(z)`. Cela implique d'ailleurs qu'il en élargit la signification aux complexes, alors qu'en mathématiques on la réserve habituellement aux réels positifs ou nuls.

Calcul algébrique

Les variables

Noms des variables:

- Le nom d'un variable doit comporter au moins un caractère.
- Le premier caractère doit être une lettre non accentuée.
- Chacun des caractères suivants, s'il y en a, peut être une lettre non accentuée, un chiffre entre 0 et 9, ou un caractère de soulignement (under_score).

Par exemple:

```
n, x, y, X, Y, somme, produit, resultat1, resultat2 sont des
noms variables qui respectent ces règles. On peut par exemple écrire :
> n: = 5:
> produit := 7*n:
mais :
> autre produit := 12*n^2:    # provoque une erreur:
missing operator or `;`
> `autre produit` := 12*n^2: # ici tout rentre dans l'ordre
```

Les variables

Variables affectées et variables non affectées:

Considérons par exemple la ligne d'entrée suivante:

```
y := a*x^3+1:
```

et supposons que a et x n'aient subi aucune affectation auparavant. Alors:

- a et x sont des ***variables non affectées***, elles n'ont pas de valeurs particulières.
- y est une ***variable affectée***. Elle a une valeur, qui est ici non pas un nombre mais une expression.

Les variables

Commande d'affectation:

Une variable est affectée quand elle s'est déjà trouvé, au cours de la même session Maple, au premier membre d'une commande d'affectation, c'est-à-dire d'une entrée de la forme:

```
> variable := expression;
```

L'expression qui est au second membre peut contenir des nombres, des chaînes de caractère, des structures de données plus complexes, et des variables, affectées ou non.

Les variables

La règle d'évaluation complète:

Lorsque Maple doit interpréter une expression, il évalue généralement les variables qu'elle contient selon la règle d'évaluation complète c'est-à-dire selon le procédé suivant:

a. Maple cherche dans l'expression les variables déjà affectées et les remplace par leurs valeurs.
b. Si l'expression obtenue contient encore des variables affectées, Maple recommence à l'étape a.
c. L'expression est considérée comme évaluée lorsqu'elle ne contient plus que des variables non affectées et des constantes.

Les variables

La règle d'évaluation complète:

Voici des exemples:

```
> y := x^2;

> x := 1;

> y;
```

$$y := x^2$$

$$x := 1$$

$$1$$

Les variables

La règle d'évaluation complète:

Voici des exemples:

```
> x := 2;

> y;

> z := x^2;

> x := 3;

> y; z;
```

$$x := 2$$

$$4$$

$$z := 4$$

$$x := 3$$

$$9$$
$$4$$

Les variables

Réinitialisation d'une variable - Emploi des apostrophes:

Il faut d'abord savoir que l'on empêche l'évaluation d'un variable en la plaçant entre apostrophes:

```
> x := 7;

> 'x';

> x;
```

```
x := 7

x

7
```

Les variables

Réinitialisation d'une variable - Emploi de la fonction unassign ():

L'entrée:

```
> unassign('x1','x2',...);
```

Supprime toute affectation aux variables xl, x2,... passées en arguments.

Les variables

Variables indexées:

Comme en mathématique, un nom de variable peut être muni d'un indice (inférieur). Dans une zone d'entrée, l'indice se note entre crochets:

```
> x[0] := 0.; x[1] := 10.;
```

$$x_0 := 0.$$
$$x_1 := 10.$$

Les expressions

Formation des expressions - Exemples d'expressions algébriques:

```
> (sin(N*Phi/2)/(N*Phi/2))^2;
```

$$4\frac{\sin\left(\frac{1}{2}N\Phi\right)^2}{N^2\Phi^2}$$

```
> 6*x*y/2+lambda*sqrt(x^2-y^2);
```

$$3xy+\lambda\sqrt{x^2-y^2}$$

Les expressions

Formation des expressions - Les opérateurs utilisables:

Les opérateurs *déjà utilisés pour les* calculs numériques sont généralement applicables aux calculs formels. Il en est bien ainsi des opérateurs +, -, *, /, et ^.

Par contre, l'opérateur mod doit être employé avec la plus grande circonspection. Voici un exemple de difficulté:

```
> if (n mod 2 = 0) then "n est pair" else "n est impair"
  fi;
```

Les expressions

Formation des expressions - Fonctions sum et product:

Pour faciliter l'écriture des sommes de termes analogues, on utilise en mathématiques un grand sigma (∑). Maple dispose d'une ***fonction sum*** qui joue le même rôle. Sa syntaxe est la suivante:

```
sum(expression_en_i, i=a..b)
```

De même, pour faciliter l'écriture des produits de facteurs analogues, on utilise en mathématiques un grand pi (∏). Maple dispose d'une fonction **product** qui joue le même rôle. Sa syntaxe est la suivante:

```
product(expression_en_i, i=a..b)
```

Les expressions

Formation des expressions - Expressions de type chaîne ou de type nom:

À partir de chaînes de caractères, on peut en former de nouvelles à l'aide de l'opérateur de concaténation, ou opérateur point <<.>>

Par exemple:

```
> x:="début"; y:="fin";
                                        x := début
                                        y := fin

> x.y;
                                        xfin
```

Les expressions

Opérandes d'une expression:

Les opérandes d'une expression A sont les éléments ou les sous-expressions reliés par le ou les opérateurs de plus basse priorité dans l'expression A.

Soit par exemple:

```
> A:=x^3+5*x*y-cos(y);
```

$$A := x^3 + 5\,x\,y - \cos(y)$$

Alors l'opérateur de plus basse priorité est + et les opérandes de l"expression située au second membre sont:

$$x^3 \;,\; 5\,x\,y \;,\; -\cos(y)$$

Les expressions

Représentation interne d'une expression:

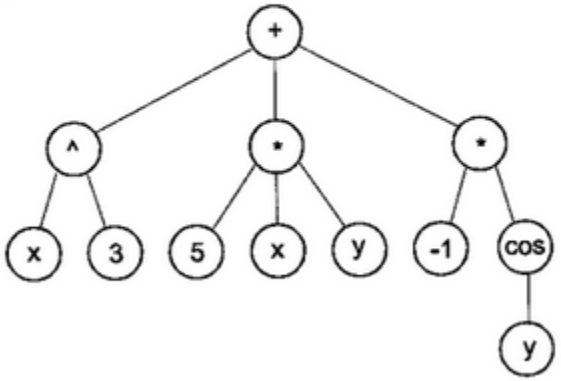
Quels que soient les détails techniques réellement mis en œuvre par Maple pour stocker une expression, nous pouvons nous représenter la structure de stockage à l'aide d'un schéma arborescent. Nous ne ferons pas de théorie générale sur cette question mais l'illustrerons par deux exemples:

Reprenons notre expression A de tout à l'heure :

```
> A:=x^3+5*x*y-cos(y);
```

$$A := x^3 + 5\,x\,y - \cos(y)$$

Sa construction correspond au schéma suivant :



Les expressions

Type d'une expression:

Objet ou expression	type renvoyé par `whattype`	Exemples
Nombres		
entier	integer	`3 , -7`
rationnel	fraction	`2/5`
complexe (en particulier, réel)	float	`3.1, .1*10^(-6), 2.*I`
Intervalle	..	`5..9 , imin..imax`
Chaîne de caractères	string	`"ma chaîne"`

Les expressions

Type d'une expression:

Noms		
symbole	symbol	y , `mon symbole`
symbole indexé	indexed	z[1] , t[max]
Expressions algébriques		
somme	+	x*y+z , x-z , 3.-2*I
produit	*	x*(y+z) , x/z
puissance	^	x^2 , 1/x , sqrt(x)
fonction	function	cos(1.2) , ln(1+x)
Expressions booléennes		
disjonction	or	a and b or c
conjonction	and	a and (b or c)
négation	not	not(b>=a)
égalité	=	x*y = z
non-égalité	<>	x+y<>z
inégalité stricte	<	x<0 , y>1
inégalité large	<=	x<=0 , y>=1

Les expressions

Simplifications et transformations d'expressions - La fonction ***simplify***

```
> 4^(1/2)+1;
```

$$\sqrt{4}+1$$

```
> simplify(%);
```

$$3$$

```
> (sin(x))^4-(cos(x))^4;
```

$$\sin(x)^4 - \cos(x)^4$$

```
> simplify(%);
```

$$1-2\cos(x)^2$$

```
> sin(x)^4 + 2*cos(x)^2 - 2*sin(x)^2 - cos(2*x);
```

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

```
> simplify(%);
```

$$\cos(x)^4$$

Les expressions

Simplifications et transformations d'expressions - Autres fonctions de transformation d'

Appel de fonction	Action
`expand(expr)`	Développe l'expression *expr* : distribue * sur +, développe `sin(a+b)`, transforme e^{a+b} en $e^a e^b$, etc.
`combine(expr,option)`	Fait en gros l'inverse de ce que fait `expand`. Il faut souvent lui préciser ce que l'on veut à l'aide du 2e argument. *option* peut prendre entre autres l'une des valeurs suivantes : `trig`, `exp`, `ln`, `power`
`collect(expr,variable)`	Organise une expression par rapport à une variable (on peut remplacer *variable* par un appel de fonction).
`normal(expr_rationnelle)`	Réduit au même dénominateur, puis simplifie de façon que numérateur et dénominateur soient premiers entre eux.
`factor(expr_polynomiale)` `factor(expr_rationnelle)`	Tente de factoriser en conservant des coefficients rationnels. Dans le cas d'une expression rationnelle, applique d'abord `normal` puis tente de factoriser numérateur et dénominateur.
`rationalize(expr)`	Tente de rendre rationnel (et donc réel) le dénominateur de *expr*
`convert(expr,option)`	Convertit une forme en une autre, par exemple une fonction circulaire en exponentielles. Consulter l'aide en tapant `?convert`
`sort`	Ordonne les opérandes selon certains critères. (Consulter l'aide en tapant `?sort`)

Les expressions

<u>Simplifications et transformations d'expressions</u> - Exemples d'emploi de ***expand***:

```
> (x+y+2*z)^2-(x+2*z)^2;
```

$$(x + y + 2\,z)^2 - (x + 2\,z)^2$$

```
> expand(%);
```

$$2\,x\,y + y^2 + 4\,y\,z$$

```
> cos(a+b)-sin(2*x)+exp(x+y);
```

$$\cos(a+b) - \sin(2\,x) + e^{(x+y)}$$

```
> expand(%);
```

$$\cos(a)\cos(b) - \sin(a)\sin(b) - 2\sin(x)\cos(x) + e^x\,e^y$$

Les expressions

Simplifications et transformations d'expressions - Exemples d'emploi de ***combine***

```
> combine(sin(a)*cos(b));
```

$$\frac{1}{2}\sin(a+b)+\frac{1}{2}\sin(a-b)$$

```
> combine(exp(x)*exp(y));
```

$$e^{(x+y)}$$

```
> combine(ln(x)+ln(y));
```

$$\ln(x)+\ln(y)$$

Les expressions

Simplifications et transformations d'expressions - Exemples d'emploi de ***collect***

```
> A:=5*x^2*y+3*x*y+(x+y)^3;
```

$$A := 5x^2y + 3xy + (x+y)^3$$

```
> collect(A,x);
```

$$x^3 + 8x^2y + (3y + 3y^2)x + y^3$$

```
> collect(A,y);
```

$$y^3 + 3xy^2 + (8x^2 + 3x)y + x^3$$

```
> B:=(1+cos(x))*(x+y^2)-z*cos(x);
```

$$(\cos(x) + 1)(x + y^2) - z\cos(x)$$

```
> collect(B,cos(x));
```

$$(x + y^2 - z)\cos(x) + x + y^2$$

```
> C:=int(x^3*exp(x),x); # calcul d'une intégrale
```

$$C := x^3e^x - 3x^2e^x + 6xe^x - 6e^x$$

```
> collect(C,exp(x));
```

$$(x^3 - 3x^2 + 6x - 6)e^x$$

Les expressions

<u>Simplifications et transformations d'expressions</u> - Exemples d'emploi de ***<u>normal</u>***

```
> (x^3+y^3)/(x^2-y^2);
```

$$\frac{x^3+y^3}{x^2-y^2}$$

```
> normal(%);
```

$$\frac{x^2-xy+y^2}{x-y}$$

```
> 1/(x+1)+x/(x^2-1);
```

$$\frac{1}{x+1}+\frac{x}{x^2-1}$$

```
> normal(%);
```

$$\frac{2x-1}{x^2-1}$$

Les expressions

<u>Simplifications et transformations d'expressions</u> - Exemples d'emploi de ***factor***

```
> 6*x^2-9*x-60;
```

$$6x^2-9x-60$$

```
> factor(%);
```

$$3(x-4)(2x+5)$$

```
> (x^3+y^3) /(2*x^3+3*y*x^2-2*y^2*x-3*y^3);
```

$$\frac{x^3+y^3}{2x^3+3yx^2-2y^2x-3y^3}$$

```
> factor(%);
```

$$\frac{x^2-xy+y^2}{(2x+3y)(x-y)}$$

Les expressions

Simplifications et transformations d'expressions - Exemples d'emploi de ***rationalize***

```
> phi:=(1+sqrt(5))/2;  # le "nombre d'or"
```

$$\phi := \frac{1}{2} + \frac{1}{2}\sqrt{5}$$

```
> rationalize(-1/phi); # l'autre racine de 1/x = x-1
```

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

```
> z:=(1+I*x)/(1-I*x);
```

$$z := \frac{1+I\,x}{1-I\,x}$$

```
> rationalize(z);
```

$$\frac{(1+I\,x)^2}{1+x^2}$$

Compléments sur les expressions et fonctions

Séquence

Définition:

Une séquence est une énumération finie d'objets Maple séparés par des virgules.

Une séquence n'est pas délimitée par des caractères spécifiques alors que des crochets ou des accolades délimitent respectivement les listes et les ensembles.

Maple appelle expression séquence, ou simplement séquence, une collection ordonnée d'expression séparées par des virgules.

Séquence

Définition:

Une séquence est une énumération finie d'objets Maple séparés par des virgules.

.

```
> s:=a,1,b,2;
```

$$s := a,\ 1,\ b,\ 2$$

```
> z:=10;
```

$$z := 10$$

```
> t:=2*Pi;
```

$$t := 2\,\pi$$

```
> r:=z,t,z;
```

$$r := 10,\ 2\,\pi,\ 10$$

Séquence

Accéder à un élément d'une séquence:

On peut accéder à un élément d'une séquence grâce à l'opérateur de sélection qui utilise une syntaxe faisant intervenir les crochets ("[" et "]"), comme il est usuel de le faire dans d'autres langages pour accéder aux éléments d'un tableau.

Ici, on choisit de récupérer le deuxième élément de la séquence s :

```
> s[2];
                                    1
```

Collections d'expressions

Les séquence - Remarque importante:

Les séquences sont des structures à lecture seule (read only), c'est-à-dire qu'on ne peut pas modifier une séquence existante.

```
> s3[4]:=x;
  Error, cannot assign to an expression sequence
```

Collections d'expressions

Les séquence: - La fonction seq:

On utilise toujours des séquences dont chaque membre est calculable à partir d'une formule générale. La fonction seq permet de construire une telle séquence. Sa syntaxe est la suivante:

```
seq(expression_en_k, k=kmin..kmax)
```

Voici des exemples:

```
> seq(k^2,k=1..10);
                    1, 4, 9, 16, 25, 36, 49, 64, 81, 100
> seq(x^k/k!,k=0..6);
```

$$1, x, \frac{1}{2}x^2, \frac{1}{6}x^3, \frac{1}{24}x^4, \frac{1}{120}x^5, \frac{1}{720}x^6$$

Séquence

Création d'une séquence:

Maple met à la disposition de l'utilisateur la commande **seq** pour créer des séquences présentant un motif régulier avec une syntaxe simple et évidente.

Noter, sur les exemples suivants, que l'intervalle définissant la variation de la variable muette qui intervient dans le motif est défini par deux nombres obligatoirement entiers séparés par deux points.

```
> k:=3;
```

$$k := 3$$

```
> seq(cos(k*Pi/23),k=1..6);
```

$$\cos(\frac{1}{23}\pi),\ \cos(\frac{2}{23}\pi),\ \cos(\frac{3}{23}\pi),\ \cos(\frac{4}{23}\pi),\ \cos(\frac{5}{23}\pi),\ \cos(\frac{6}{23}\pi)$$

Séquence

Création d'une séquence:

Au lieu de la fonction `seq`, on peut employer un **opérateur spécial noté** `$` :

```
> k^3 $ k=1..5;
```

1, 8, 27, 64, 125

Séquence

Création d'une séquence:

En fait, les deux méthodes ne sont pas toujours équivalentes : la fonction `seq` ne peut construire qu'une séquence parfaitement déterminée tandis que l'opérateur $ est capable de définir une séquence de façon formelle ; par exemple, la borne supérieure de l'indice peut être un paramètre (variable non affectée) :

```
> sf := k^3 $ k=1..kmax;
```

$$sf := k^3 \ \$ \ (k = 1 \,..\, kmax)$$

```
> subs(kmax=5, sf);
```

$$k^3 \ \$ \ (k = 1 \,..\, 5)$$

```
> eval(%);
```

$$1, 8, 27, 64, 125$$

La syntaxe de l'opérateur $ présente plusieurs variantes. Pour avoir des détails, consulter l'aide Maple en tapant `?$` .

La notion de séquence va nous permettre de construire très simplement d'autres collections d'expressions, comme les ensembles et les listes.

Séquence

Création d'une séquence:

L'opérateur "$" permet de créer des séquences comportant des répétitions d'objets identiques:

```
> x$5;
```

$$x, x, x, x, x$$

Ce procédé trouve une application fréquente dans le calcul de dérivées d'ordres élevés, comme dans l'exemple suivant où l'on calcule la dérivée cinquième de $x \mapsto \sin(3x+2)$

```
> diff(cos(3*x+2),x$5);
```

$$-243\sin(3x+2)$$

Séquence

Création d'une séquence:

L'opérateur "$" permet aussi de fabriquer des suites d'entiers consécutifs:

```
>   $2..6;
                                    2, 3, 4, 5, 6
```

Ensembles

Définition:

Un ensemble Maple (variable de type set) s'obtient en délimitant une séquence par des accolades :

```
> E:={$1..6};
```

$$E := \{1, 2, 3, 4, 5, 6\}$$

Contrairement à certains langages, Maple n'exige pas que tous les éléments d'un ensemble relèvent du même type :

```
> F:={1,2,3,a,2,a,b,4};
```

$$F := \{1, 2, 3, 4, a, b\}$$

Ensembles

Définition de la Ensembles:

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
```

On obtient le nombre d'éléments d'un ensemble avec la commande **nops** :

Ensembles

Définition de la Ensembles:

On obtient le nombre d'éléments d'un ensemble avec la commande **nops** :

```
> F:={1,2,3,a,2,a,b,4};
```

$$F := \{1, 2, 3, 4, a, b\}$$

```
> nops(F);
```

$$6$$

Ensembles

Affichage des éléments de l'ensemble:

On obtient la séquence sous-jacente à un ensemble grâce à la commande **op** :

```
> F:={1,2,3,a,2,a,b,4};
```

$$F := \{1, 2, 3, 4, a, b\}$$

```
> op(F);
```

$$1, 2, 3, 4, a, b$$

Ensembles

Accéder aux éléments de l'ensemble:

Pour accéder directement à un élément d'un ensemble, il suffit d'indiquer sa position dans l'ensemble, toujours à l'aide de la commande op à laquelle on passe la position en premier argument:

```
> op(4,F);
```

$$4$$

ou, plus directement, à l'aide de l'opérateur de sélection :

```
> F[5];
```

$$a$$

Ensembles

Accéder aux éléments de l'ensemble:

Cette syntaxe ne permet pas de modifier un élément de l'ensemble:

```
> F[5]:=12;

Error, cannot reassign the entries in a set
```

Collections d'expressions

Les ensembles - Définition:

Un ensemble est une collection non ordonnée d'expression toutes différentes. Il se note à l'aide d'une séquence entre accolades.

```
> el:={a,b,c,d};
```

$$el := \{c, b, d, a\}$$

Collections d'expressions

Les ensembles - Opération sur les ensemble:

Opération mathématique	Notation Maple	Exemples (avec e1 et e2 définis ci-dessus)
e1 ∪ e2	e1 union e2	> e1 union e2; {c, b, d, a, f, e, g}
e1 ∩ e2	e1 intersect e2	> e1 intersect e2; {c, d}
e1 \ e2	e1 minus e2	> e1 minus e2; {b, a} > e2 minus e1; {f, e, g}

Collections d'expressions

Les ensembles - Opérande d'un ensemble:

Dans Maple, un ensemble est aussi une expression. En tant les qu'expression, il possède des opérande, qui sont simplement les éléments de l'ensemble, et l'on peut manipuler ces opérandes à l'aide des fonctions nops et op.

```
> op(e1);
```

$$c, b, d, a$$

On voit que l'on obtient une séquence, et non un ensemble. À titre d'exemple, on pourra vérifier que :

`e1 union e2` équivaut à `{op(e1),op(e2)}`

Ensembles

Opéraions sur les ensembles:

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 :

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

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

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

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

```
> A union B;
```

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

```
> A intersect B;
```

$$\{6, 12, 18\}$$

```
> A minus B;
```

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

Ensembles

Opéraions sur les ensembles:

L'appartenance à un ensemble peut être testée avec la commande `in` :

```
> 1 in {1,2,3,4};
```

$$1 \in \{1, 2, 3, 4\}$$

Dans cet exemple, comme dans celui de la page 42, il faut expliciter sa demande pour obtenir l'évaluation booléenne de l'expression considérée :

```
> evalb(%);
```

$$true$$

Liste

Définition:

On obtient une liste (variable de type `list`) en délimitant une séquence par des crochets :

```
> L:=[$1..6];
```

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

Contrairement à certains langages comme Caml, Maple n'exige pas que tous les éléments d'une liste relèvent du même type :

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

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

Liste

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

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

false

Liste

Comme dans le cas des ensembles le nombre d'éléments d'une liste s'obtient avec la commande **nops** :

```
> T:=[1,3.2,5,a,1];
```

$$T := [1,\ 3.2,\ 5,\ a,\ 1]$$

```
> nops(T);
```

$$5$$

et la séquence sous-jacente à une liste avec la commande **op** :

```
> op(T);
```

$$1,\ 3.2,\ 5,\ a,\ 7$$

Liste

Comme dans le cas des listes, la commande op dotée d'un deuxième argument permet d'accéder à un élément de la liste considérée:

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

$$3.2$$

ce que l'opération de sélection fournit aussi directement :

```
> T[2];
```

$$3.2$$

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

```
> T[2..4];
```

$$[3.2,\ 5,\ a]$$

Liste

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

```
> T[2]:=55;
```

$$T_2 := 55$$

```
> T;
```

$$[1,\ 55,\ 5,\ a,\ 7]$$

Liste

Lors de l'appel **select(f, L)**, la commande ***select*** sélectionne dans une liste les éléments de la liste L qui satisfont à la fonction f à valeurs booléennes.

Voici comment ne conserver que les nombres premiers dans la liste des entiers compris entre 1 et 20 :

```
> select(isprime,[$1..20]);
                    [2, 3, 5, 7, 11, 13, 17, 19]
```

La commande **remove** fonctionne comme select, mais elle retire les éléments qui satisfont au critère de sélection f :

```
> remove(isprime,[$1..20]);
                [1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20]
```

Liste

L'appel **map(f, L)**, la commande ***map*** permet d'appliquer la fonction f à tous les éléments de la liste L, comme le montre l'exemple abstrait suivant :

```
> map(f,[a,b,c,d]);
```

$$[\mathrm{f}(a), \mathrm{f}(b), \mathrm{f}(c), \mathrm{f}(d)]$$

```
> map(x->x^2,[$1..10]);
```

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

Liste

La commande **member** permet de tester l'appartenance d'un objet à une liste :

```
> member(y, {x, y, z});
```

true

Les Fonctions

Avec Maple on peut définir des fonctions en utilisant l'opérateur ->.

Une variable peut être affectée d'une valeur de type *function* selon la syntaxe suivante, très proche de la notation mathématique :

```
nom_fonction := nom_variable -> expression
```

Par exemple :

```
> f:=x->x^2;
```

$$f := x \to x^2$$

Après cette définition, on peut appliquer la fonction f à une expression quelconque, constante ou symbolique, exactement comme on fait avec une fonction prédéfinie :

```
> f(5);
```

$$25$$

```
> f(a+b);
```

$$(a+b)^2$$

```
> expand(f(a+b));
```

$$a^2 + 2\,a\,b + b^2$$

Les Fonctions

On peut définir de la même façon des fonctions de plusieurs variables :

```
nom_fonction := (séquence_noms_variables) -> expression
```

Noter qu'ici les parenthèses sont obligatoires autour de la séquence des noms de variables (sinon on définirait une séquence, dont seul le dernier élément serait une fonction).

Voici un exemple :

```
> g:=(x,y)->arctan(y/x);
```

$$g := (x, y) \to \arctan\left(\frac{y}{x}\right)$$

```
> g(1,sqrt(3));
```

$$\frac{1}{3}\pi$$

Le résultat de la fonction peut être d'un type quelconque, par exemple une chaîne de caractères, une séquence, une liste, etc. :

```
> sincos:= theta->[sin(theta),cos(theta)];
```

$$sincos := \theta \to [\sin(\theta), \cos(\theta)]$$

```
> sincos(Pi/6);
```

$$\left[\frac{1}{2}, \frac{1}{2}\sqrt{3}\right]$$

Les Fonctions

Définition d'une fonction par morceaux

L'opérateur flèche permet de définir une fonction d'une variable réelle par morceaux (avec un nombre fini et connu de morceaux !) grâce à la fonction `piecewise`. La syntaxe générale, pour n morceaux, est la suivante :

```
nom_fonction := nom_variable ->
                piecewise( condition_1, expression_1,
                           condition_2, expression_2,...,
                           condition_(n-1), expression_(n-1),
                           expression_n );
```

L' *expression_n* est choisie lorsqu'aucune des conditions n'est satisfaite. Voici un exemple :

```
> f:= x -> piecewise(x<-1,-1,x<1,x,1);
```

$$f := x \to \text{piecewise}(x < -1, -1, x < 1, x, 1)$$

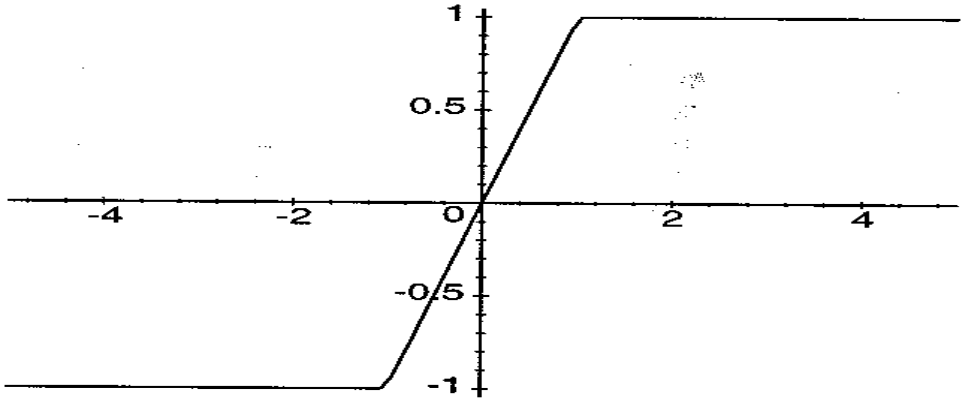
Les Fonctions

```
> f(x);
```

$$\begin{cases} -1 & x < -1 \\ x & x < 1 \\ 1 & otherwise \end{cases}$$

Une fonction ainsi définie peut subir **certaines** des manipulations applicables aux autres fonctions :

```
> plot(f,-5..5); # graphe de f sur [-5;+5]
```




```
> int(f(x),x); # calcul d'une primitive
```

$$\begin{cases} -x & x \le -1 \\ \frac{1}{2} + \frac{1}{2}x^2 & x \le 1 \\ x & 1 < x \end{cases}$$

Les Fonctions

Emploi de la fonction **unapplay**

La fonction unapply offre à peu près les mêmes possibilités que l'opérateur flèche, avec une syntaxe différente, qui est la suivante :

```
nom_fonction := unapply(expression, séquence_noms_variables)
```

La séquence des noms de variables peut bien sûr ne comporter qu'un seul élément. Reprenons des exemples déjà traités à l'aide de l'opérateur flèche :

```
> f:=unapply(x^2,x);
```

$$f := x \to x^2$$

```
> g:=unapply(arctan(y/x),x,y);
```

$$g := (x, y) \to \arctan\left(\frac{y}{x}\right)$$

Les Fonctions

Comparaision des deux méthodes unapplay et flèche

Les deux méthodes, emploi de l'opérateur flèche ou de la fonction `unapply`, donnent des résultats voisins, souvent même identiques. Cependant, la fonction `unapply` est plus pratique lorsqu'on veut transformer en fonction une expression qui vient d'être calculée par Maple ; il suffit en effet de remplacer l'argument *expression* par le signe `%`. La même technique ne donne rien de bon avec l'opérateur flèche :

```
> x^2;
```

$$x^2$$

```
> f:=unapply(%,x);
```

$$f := (x \to x)^2$$

La notation employée dans la réponse est fort contestable mais il s'agit bien de la fonction désirée :

```
> f(5);
```

25

Essayons maintenant avec l'opérateur flèche :

```
> x^2;
```

$$x^2$$

```
> f:=x->%;
```

$$f := x \to \%$$

```
> f(5);
```

On n'obtient RIEN !

Les Fonctions

La fonction **map**

La fonction **map** permet d'appliquer une fonction sur tous les opérandes d'une expression.

13.1. Application de map à une fonction d'une variable

Pour une fonction f d'une seule variable, la syntaxe est la suivante :

```
map(f, expression);
```

La fonction f peut être une fonction prédéfinie ou une fonction définie par l'utilisateur. L résultat est une expression de même type que l'expression initiale (sauf si des simplificatio automatiques modifient ce type) ; cette expression initiale n'est pas changée.

Commençons par des expressions algébriques :

```
> f:=x->x^2;
```

$$f := x \to x^2$$

```
> map(f,a+b*c);
```

$$a^2 + b^2\, c^2$$

Les Fonctions

Il s'agit ici d'une expression de type + et les opérandes sont a et b*c. Prenons maintenant une expression de type * :

```
> map(f,(a+b)*c);
```

$$(a+b)^2\,c^2$$

Il faut bien réfléchir à l'identification des opérandes si l'on ne veut pas avoir de mauvaises surprises :

```
> map(f,a-b*c);
```

$$a^2+b^2\,c^2$$

En effet, l'expression est de type + (il n'y a pas de type –) et le second opérande est –(b*c).

Si l'on a le moindre doute, exécuter d'abord `op(`*expression*`)` pour savoir sur quels opérandes la fonction `f` va agir.

Nous avons vu que les ensembles et les listes sont aussi des expressions. La fonction map peut donc leur être appliquée :

```
> s:=a,b,b,c; # une séquence
```

$$s := a, b, b, c$$

```
> e:={s}; # un ensemble
```

$$e := \{a, b, c\}$$

(on peut obtenir un ordre différent)

```
> li:=[s]; # une liste
```

$$li := [a, b, b, c]$$

En gardant la même fonction f que dans les exemples ci-dessus, on obtient avec l'ensemble e :

```
> map(f,e);
```

$$\{a^2, b^2, c^2\}$$

On a bien obtenu un ensemble. Opérons de même sur la liste :

```
> map(f,li);
```

$$[a^2, b^2, b^2, c^2]$$

On a bien obtenu une liste.

Noter que **la fonction** map **ne peut pas être appliquée à une séquence**. La raison en est que le deuxième élément de la séquence serait interprété comme un troisième argument de la fonction map, et que celui-ci a une signification différente, comme nous allons le voir.

Les Fonctions

La fonction **map**

La fonction **map** permet d'appliquer une fonction sur tous les opérandes d'une expression.

3.2. Application de map à une fonction de plusieurs variables

Si f est une fonction de plusieurs variables, la syntaxe de map est la suivante :

```
map(f, expression, arg2, arg3,...)
```

Dans ce cas, f prend comme premier argument tous les opérandes de expression successivement, tandis qu'à chaque fois le deuxième argument est arg2, le troisième est arg3, etc. Par exemple :

```
> f:= (x,n)->x^n;
```

$$f := (x, n) \rightarrow x^n$$

```
> map(f,e,4);
```

$$\{a^4, b^4, c^4\}$$

Moyen de Pregrammation

Programmation en Maple

Dans cette partie nous allons présenter les éléments de syntaxe et les règles essentielles qui concernent la programmation avec Maple.

Les opérateurs de comparaison

=	(égale)
<>	(est différent de)
<	(est strictement inférieur à)
>	(est strictement supérieur à)
<=	(est inférieur ou égal à)
>=	(est supérieur ou égal à)

Les opérateurs logiques

not	(non, le contraire de)
and	(et)
or	(ou)

Les opérateurs de comparaison, la fonction evalb

Exemple:

Les opérandes de la comparaison peuvent d'abord être des nombres :

```
> 355/113 = 3.1416;
```

$$\frac{355}{113} = 3.1416$$

On voit que l'évaluation de l'expression booléenne n'a pas eu lieu. C'est voulu, cela permet d'obtenir une présentation esthétique de l'expression et nous avons souvent eu recours à cette propriété. Pour forcer l'évaluation booléenne, on emploie la fonction `evalb` :

```
> evalb(%);
```

false

```
> evalb(355/113 < 3.1416);
```

true

Programmation en Maple

Les opérandes peuvent aussi être des expressions symboliques représentant des nombres :

```
> evalb(Pi=6*arcsin(1/2));
```

true

Dans le cas des opérateurs = et <> , les opérandes peuvent aussi être des chaînes de caractères ou des noms :

```
> a := "chaine"; b := "chaîne"; evalb(a=b);
```

a := "chaine"
b := "chaîne"
false

```
> a:=b; evalb(a=b);
```

a := "chaîne"
true

Comme l'ont montré les exemples précédents, la fonction `evalb` réalise quelques simplifications immédiates sur les opérandes d'une expression booléenne avant d'évaluer la vérité de celle-ci. Cependant, ces simplifications restent très limitées, ce qui peut mener à des résultats surprenants si l'on n'y réfléchit pas :

```
> evalb(x^2-y^2=(x-y)*(x+y));
```

false

La fonction `evalb` n'est donc pas chargée de faire du calcul algébrique. On peut aller beaucoup plus loin avec la fonction `is` (nom qu'on peut traduire, et paraphraser, par « est-il vrai que...? ») :

```
> is(x^2-y^2=(x-y)*(x+y));
```

true

Les structures de branchement

On peut soumettre l'exécution d'un groupe de commandes à une certaine condition grâce à la construction suivante:

```
if condition then commandes fi;
```

On peut choisir entre deux commandes ou groupes de commandes grâce à la construction suivante :

```
if condition
  then commandes_1
  else commandes_2
fi;
```

On peut choisir un groupe de commandes (éventuellement réduit à une seule commande) parmi plusieurs groupes grâce à la construction suivante :

```
if condition_1 then commandes_1
elif condition_2 then commandes_2
...
elif condition_n then commandes_n
« else commandes_(n+1) »
fi;
```

Les structures itératives ou boucles

while condition do commandes od;

Exemple :

Soit à trouver par la méthode d'Euclide le pgcd des deux nombres suivants :

```
> a:=896692: b:=1144905:
```

Voici une boucle qui résout le problème :

```
> while b<>0 do
      r:=irem(a,b): a:=b: b:=r
  od:
> a;
```

601

Programmation en Maple

Les structures itératives ou boucles

for *indice* **from** *valeur_début* **to** *valeur_fin* **by** *pas* **do** *commandes* **od;**

```
> for k from 5 to 11 by 2 do k^2 od;
                              25
                              49
                              81
                             121
```

Programmation en Maple

Ecriture des procédures

```
> f:=proc(x,y,z)
  [x,y,z]
  end proc;
```

$$f := \mathbf{proc}(x,\ y,\ z)\ [x,\ y,\ z]\ \mathbf{end\ proc}$$

```
> f(1,2,3);
```

$$[1,\ 2,\ 3]$$

Tracés de graphiques

Tracés de graphiques

1- Graphique dans le plan

La plupart des possibilités graphiques de Maple dans le plan sont fournies par la fonction Prédéfinie plot, dont la syntaxe générale prend l'une des formes suivantes, et les symboles et « » délimitent comme d'habitude des arguments facultatifs :

plot(*def «,options»*)

plot (*def, h_range «,* opti*ons* »)

plot (*def, h_range, v_range «,options»*)

La signification générale des symboles employés est la suivante :

- ***def*** indique la définition mathématique de ['ensemble à afficher,
- ***h_range*** indique l'intervalle affiche en abscisse (intervalle horizontal)
- ***v_range*** indique ['intervalle affiché en ordonnée (intervalle vertical »)
- *options* (facultatif) donne des indications supplémentaires.

➔ Si le paramètre *h_range* n'est pas fourni, l'intervalle -10 .. +10 est pris par défaut.

➔ Si le paramètre *r_range* n'est pas fourni, la fonction plot détermine un intervalle d'ordonnées contenant tous les points calculés sur l'intervalle choisi en abscisse.

Tracés de graphiques

Quelques options de fonction plot

4- L'option ***color***

L'option ***color = valeur*** permet de choisir Ia couleur de la courbe tracée. Le paramètre *valeur* peut être indiqué à l'aide d'identificateurs prédéfinis comme: red, green, blue, cyan, magenta, …. Pour obtenir des détails, consultez l'aide en tapant la commande : ***?plot, color;***

5- L'option ***title***

L'option ***title = chaîne de caractères*** permet de dessiner un titre.

Tracés de graphiques

Graphe multiple:

La fonction ***plot*** permet de tracer plusieurs courbes sur un même graphique. Il suffit pour cela de lui passer, comme premier argument, un ensemble (au sens de Maple) de définitions. On obtient ainsi la syntaxe suivante :

plot (*(defl, def2, ...}, h_range, «,v_range» «,options»)*

chaque élément def_i étant:

- ou bien une expression telle que ***f (x)*** **,** pour définir une courbe y=f(x),
- ou bien une liste de la forme **[f (t) , g (t) , t_range]**, pour une courbe paramétrique.

Une liste peut remplacer l'ensemble comme premier argument, d'où la syntaxe :

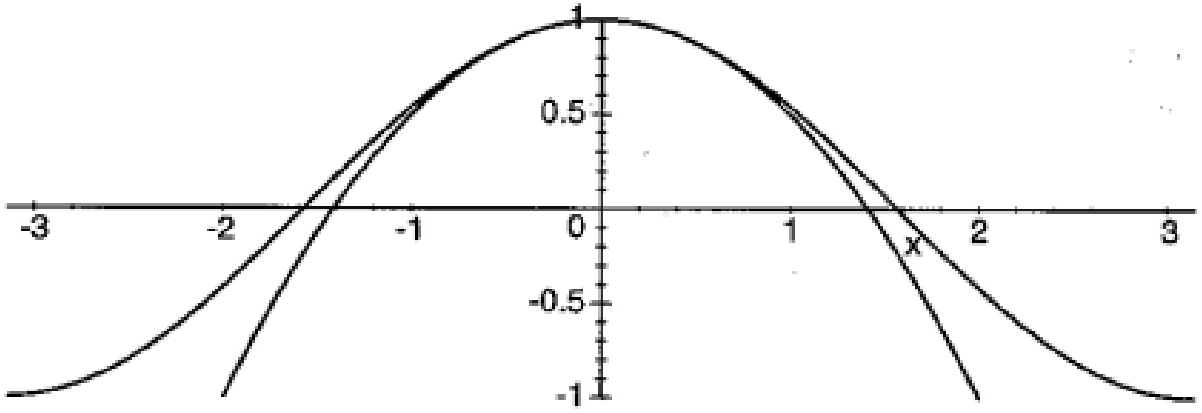
plot*((def1, def2, ...), h_range, «,v range» «,options»)*

Si l'on veut tracer plusieurs courbes paramétriques, on doit employer la même variable comme paramètre pour toutes ces courbes.

Tracés de graphiques

Exemple

```
> plot({cos(x),1-x^2/2}, x=-Pi..Pi,-1..1);
```



Tracés de graphiques

2- Graphique dans l'espace:

De même que la fonction ***plot*** permet de *tracer* des courbes dans le plan, la fonction ***plot3d*** permet de tracer des surfaces dans l'espace. Elle est tout aussi riche ct nous n'en citerons pas toutes les possibilités.

On pourra toujours se reporter à l'aide Maple en tapant la commande: ***?plot3d;***

La syntaxe générale de ***plot3d*** est de la forme :

plot3d(*def, range1, range2 «,options»)*

- ***def*** est la définition mathématique de la surface.
- ***range1*** *et* ***range2*** sont deux spécifications d'intervalles dont la nature exacte dépend de l'argument ***def***.

➢ Remarquons que, contrairement à ce qui se passait pour la **fonction plot, les deux intervalles sont toujours obligatoires.**

Equations, dérivation, intégration et limites

Résolution d'une seule équation à une seule inconnue

Syntaxe de la fonction *solve*

Un appel à la fonction solve, pour une équation à une inconnue, doit prendre la forme suivante :

solve ***(équation, var)***

où :

- ***équation*** est une égalité entre deux expressions, telle que f (x, ...) =g (x,...)
- ***var*** est une variable **non** affectée présente dans l'équation, considérée comme l'inconnue.

➔ Si l'équation comporte d'autres variables non affectées (en plus de ***var)***, ces variables jouent le rôle de paramètres.

Dérivation

I- Dérivées d'une expression

Les dérivées d'une expression s'obtiennent à l'aide de la fonction **diff** .

diff(*expression, var «,seq_var»)*

var est une variable non affectée et ***seq_var*** une séquence de telles variables. La fonction cherche la dérivée du premier argument par rapport aux variables indiquées et renvoie :

- l'expression de la dérivée cherchée si elle la trouve
- la commande non évaluée dans le cas contraire.

Pour les besoins de la présentation, il existe aussi une fonction « inerte » ***Diff*** de même syntaxe, qui se contente d'afficher la commande non évaluée. On peut ensuite forcer l'évaluation du résultat (si Maple sait le trouver) à l'aide de la fonction ***value***. Ainsi, **value (Diff*(args))*** a le même effet que **diff *(args).***

Dérivation

1- Calcul d'une dérivée première

Pour demander une dérivée première, on utilise seulement deux arguments, l'expression à dériver et la variable par rapport à laquelle on veut dériver :

```
> diff(tan(x),x);
```

$$1+\tan(x)^2$$

L'expression peut très bien dépendre de plusieurs variables non affectées :

```
> Diff(ln(x^3+a*x^2+1),x);
```

$$\frac{\partial}{\partial x}\ln\left(x^3+a\,x^2+1\right)$$

```
> value(%);
```

$$\frac{3x^2+2ax}{x^3+ax^2+1}$$

Dérivation

2- Dérivée d'ordre supérieur:

Pour déterminer une dérivée d'ordre supérieur, on doit fournir à la fonction ***diff*** l'expression à dériver puis, en séquence, les variables par rapport auxquelles elle doit dériver successivement :

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

$$\frac{\partial^2}{\partial y \partial x} f(x, y)$$

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

$$\frac{\partial^3}{\partial x^3} f(x, y)$$

Pour former la séquence des variables de dérivation, on peut bien sûr utiliser la fonction **seq** ou l'opérateur $. L'exemple précédent s'écrit ainsi :

```
> diff(f(x,y),x$3);
```

$$\frac{\partial^3}{\partial x^3} f(x, y)$$

Dérivation

II- Dérivée d'une liste d'expressions:

Supposons que nous voulions dériver tous les éléments d'une liste d'expressions tels que :

```
> li:=[f(x),g(x),h(x)];
```

$$li := [f(x), g(x), h(x)]$$

```
> diff(li,x);
```

$$\left[\frac{\partial}{\partial x} f(x), \frac{\partial}{\partial x} g(x), \frac{\partial}{\partial x} h(x)\right]$$

Intégration

I-Calcul d'une primitive:

Les calculs formels de primitives, ou « intégrales indéfinies », se font à l'aide de la fonction **int**, dont voici la syntaxe :

int *(expression,var)*

Le premier argument est l'expression à intégrer (et non la fonction elle-même), le second est le nom de la variable d'intégration.

II existe aussi une fonction inerte ***Int***, qui se contente d'afficher l'intégrale non évaluée. On peut ensuite forcer son évaluation à l'aide de la fonction ***value***. Tel que: ***value (Int (ags))*** equivaut a ***int (args).***

```
> int(tan(x),x);
```

$$-\ln(\cos(x))$$

```
> Int( x/(x^3+1), x );
```

$$\int \frac{x}{x^3+1}\,dx$$

```
> value(%);
```

$$-\frac{1}{3}\ln(x+1)+\frac{1}{6}\ln(x^2-x+1)+\frac{1}{3}\sqrt{3}\arctan\left(\frac{1}{3}(2x-1)\sqrt{3}\right)$$

Limites

1- Limite d'une fonction/expression algébrique

Maple sait déterminer certaines limites non évidentes. La fonction qui fait ce travail, ou qui tente de le faire, s'appelle naturellement ***limit***.

limit(fonction_x, x=a «,direction»)

- ***fonction_x:*** est une fonction/expression algébrique contenant une variable non affectée nommée ici x.
- **a:** est la valeur de x pour laquelle on cherche la limite de l'expression précédente. Ce peut être une constante ou une expression algébrique. Les constantes **infinity (pour +oo)** et **-infinity (pour -oo)** sont acceptées.
- ***direction:*** est une indication facultative qui peut valoir : left, right, real ou complex.

L'option real correspond au comportement par défaut. Dans ce cas, Maple cherche les limites réelles à gauche et à droite (sauf bien sin- pour a =+ ou -infinity) .

Noter là aussi la forme inerte **Limit** qui affiche la forme limite.

Limites

Exemples

Considérons la fonction f suivante :

```
> f:=x->2*(1-cos(x))/x^2;
```

$$f := x \to 2\frac{1-\cos(x)}{x^2}$$

```
> Limit(f(x),x=0): % = value(%);
```

$$\lim_{x\to 0} 2\frac{1-\cos(x)}{x^2} = 1$$

```
> limit(1/x,x=0);
```

undefined

```
> limit(1/x,x=0,right);
```

$$\infty$$

Limites

2- Limite d'une suite

```
> u:=n->(1+1/n)^n;
```

$$u := n \to \left(\frac{1}{n}+1\right)^n$$

```
> Limit(u(n),n=infinity): %=value(%);
```

$$\lim_{n\to\infty}\left(\frac{1}{n}+1\right)^n = e$$

Fin