V. Ripoll
vripoll@dma.ens.fr

MK1 "Calcul formel" Maple

TP2 : Séquences, ensembles, listes, fonctions, représentations graphiques

N.B.: ces TPs sont très largement inspirés des TPs écrits par Cécile Armana.

Accéder à Maple en dehors des heures de TPs :
Maple est disponible sur les ordinateurs des salles libre-service du SCRIPT ; consultez les plannings pour connaître les horaires et les salles. En cas de besoin, renseignez-vous auprès des techniciens du SCRIPT.

Version de démonstration de Maple :
Vous pouvez installer une version de démonstration de Maple (gratuite, mais limitée) sur votre ordinateur personnel, quelque soit son système d'exploitation (Windows, Linux, MacOS,...) : il suffit de la télécharger sur le ftp de Maplesoft, la société qui édite Maple :
ftp://ftp.maplesoft.com/pub/maple/demo
La version de Maple proposée est V Release 4. En TP, vous utilisez la version 9. Il existe de (petites) différences entre les deux.

Restart au début de chaque exercice :
Il est très fortement recommandé de mettre une commande :

```
> restart;
```

au début de chaque nouvel exercice.

Interface de Maple :
Pour créer une nouvelle ligne d'invite dans Maple, cliquer sur l'icone [> .
Pour taper du texte non interprété par Maple (par exemple, "Exercice n°1"), cliquer sur l'icone T.
Pour mettre des commentaires au milieu de commandes Maple, on utilise le symbole # (tout ce qui suit le symbole jusqu'à la fin de la ligne est alors ignoré) .

Et surtout, n'oubliez pas de vous (et de me) poser des questions !

1. Les collections d'expressions

La semaine dernière, nous avons utilisé des variables, en leur affectant une donnée simple (un nombre entier, un nombre complexe,...). Il peut etre intéressant de *regrouper plusieurs "objets" dans une seule variable.* Par exemple, si on affecte dans une variable l'ensemble des solutions d'une équation, cela permettra de manipuler toutes les solutions à la fois. Il existe différentes façons de collecter des expressions sous Maple.

1.1 Les séquences

Pour Maple, une *séquence* est une collection **ordonnée** d'expressions séparées par des **virgules**. Ces expressions peuvent etre du meme type (par exemple des nombres entiers) ou des objets de natures différentes (on peut former une séquence composée d'une fonction, d'un nombre entier, et d'un nombre réel approché).

```
> a,b,c ; # exemple de séquence
```

$$a, b, c \tag{1}$$

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

$$exprseq \tag{2}$$

Le type des séquences est *exprseq*.
Pour former une séquence, **une première méthode** est donc d'en lister les éléments dans l'ordre en les séparant par des virgules. On peut affecter cette séquence à une variable (c'est-à-dire lui "donner un nom") :

```
> s:=a,b,c;
```

$$s := a, b, c \tag{3}$$

Voici sur des exemples comment rappeler les éléments d'une séquence (attention, on utilise des *crochets*) :

```
> s[1]; # premier élément
  s[3]; # troisième élément
  s[4];
```

$$a$$
$$c \tag{4}$$

```
Error, invalid subscript selector
```

Bien sur, il n'existe pas de quatrième élément, donc Maple nous retourne un message d'erreur.

```
> s[2..3];
```

$$b, c \tag{5}$$

La séquence vide (aucun élément) se note NULL :

```
> vide:=NULL;
```

$$vide := \tag{6}$$

On peut former une séquence en mettant bout à bout (concaténation) plusieurs séquences existantes : il suffit de séparer les séquences par des *virgules* :

```
> t:=1,2,3;
```

$$t := 1, 2, 3 \tag{7}$$

```
> u:=t,NULL,s;
```

$$u := 1, 2, 3, a, b, c \tag{8}$$

Remarque : les séquences sont des **structures à lecture seule.** Cela signifie qu'on ne peut pas modifier une séquence existante, par exemple en affectant

l'un de ses éléments :

```
> u[5];
```

$$b \tag{9}$$

```
> u[5]:=0;
```

```
Error, invalid assignment (1, 2, 3, a, b, c)[5] := 0; cannot assign to an
expression sequence
```

Par contre, rien ne nous empeche de créer une nouvelle séquence à partir de "morceaux" de la séquence existante.
Une deuxième méthode pour créer une séquence est la fonction *seq*. Elle permet de former seulement les séquences définies à partir de "formules" (voir les exercices).

1.2 Les ensembles

Pour Maple, un *ensemble* est une collection **non ordonnée** d'expressions **toutes différentes**. Il se note à l'aide d'une séquence entre **accolades** { , }. La collection est non ordonnée : l'ordre des éléments que nous donnons à Maple en entrée n'est pas toujours respecté. D'ailleurs, il peut changer d'une fois sur l'autre.

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

$$e := \{a, b, d, c\} \tag{10}$$

Les expressions sont toutes différentes : si Maple voit plusieurs fois apparaitre la meme expression, il élimine les doublons :

```
> f:={a,a,a,a,b};
```

$$f := \{a, b\} \tag{11}$$

```
> whattype(e);
```

$$set \tag{12}$$

Le type des ensembles est *set* (ensemble, en anglais). Les méthodes de construction de séquences permettent aussi d'obtenir des ensembles : il suffit d'ajouter des accolades. Par exemple :

```
> {seq(i,i=1..15)};
```

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\} \tag{13}$$

Remarque : comme les séquences, les ensembles sont des **structures à lecture seule.**
On peut effectuer les manipulations usuelles sur les ensembles grace aux fonctions suivantes :
– la réunion de deux ensembles : *union*
– l'intersection de deux ensembles : *intersect*
– la différence de deux ensembles : *minus*

```
> e minus {a,d};
```

$$\{b, c\} \tag{14}$$

On obtient le nombre d'éléments de l'ensemble par la commande *nops* (pour "*number of operands*") :

```
> nops(e);
```

$$4 \tag{15}$$

et la *séquence* des éléments de l'ensemble par la commande *op* (pour "*operands*") :

```
> op(e);
```

$$a, b, c, d \tag{16}$$

1.3 Les listes

Pour Maple, une *liste* est une collection **ordonnée** d'expressions, notée entre **crochets**. C'est une "séquence entre crochets" et il peut y avoir des doublons. On peut la construire à partir d'une séquence.

```
> L:=[seq(1/(2*i),i=1..30)]; whattype(L);
```

$$L := \left[\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{10}, \frac{1}{12}, \frac{1}{14}, \frac{1}{16}, \frac{1}{18}, \frac{1}{20}, \frac{1}{22}, \frac{1}{24}, \frac{1}{26}, \frac{1}{28}, \frac{1}{30}, \frac{1}{32}, \frac{1}{34}, \frac{1}{36}, \frac{1}{38}, \frac{1}{40}, \frac{1}{42}, \frac{1}{44}, \frac{1}{46}, \frac{1}{48}, \frac{1}{50}, \frac{1}{52}, \frac{1}{54}, \frac{1}{56}, \frac{1}{58}, \frac{1}{60}\right] \tag{17}$$

$$list$$

Pour accéder au k–ième élément, on procède comme pour les séquences :

```
> L[4];L[10..15];
```

$$\frac{1}{8} \tag{18}$$

$$\left[\frac{1}{20}, \frac{1}{22}, \frac{1}{24}, \frac{1}{26}, \frac{1}{28}, \frac{1}{30}\right]$$

Remarque : comme les séquences et les ensembles, les listes sont des **structures à lecture seule.**
Pour obtenir le nombre d'éléments de la liste :

```
> nops(L);
```

$$30 \tag{19}$$

Pour obtenir la *séquence* formée des éléments de la liste :

```
> op(L);
```

$$\frac{1}{2}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{10}, \frac{1}{12}, \frac{1}{14}, \frac{1}{16}, \frac{1}{18}, \frac{1}{20}, \frac{1}{22}, \frac{1}{24}, \frac{1}{26}, \frac{1}{28}, \frac{1}{30}, \frac{1}{32}, \frac{1}{34}, \frac{1}{36}, \frac{1}{38}, \frac{1}{40}, \frac{1}{42}, \frac{1}{44}, \frac{1}{46}, \frac{1}{48}, \frac{1}{50}, \frac{1}{52}, \frac{1}{54}, \frac{1}{56}, \frac{1}{58}, \frac{1}{60} \tag{20}$$

On souhaite fondre deux listes en une.

```
> restart;L1:=[a,b,c];L2:=[d,e,f];
```

$$L1 := [a, b, c] \tag{21}$$
$$L2 := [d, e, f]$$

```
> L1,L2;
  whattype(L1,L2);
```

$$[a, b, c], [d, e, f] \tag{22}$$
$$exprseq$$

L'objet que l'on obtient en les séparant par une virgule est une séquence de deux listes. Ce n'est pas le résultat voulu. La façon de fondre ces deux listes en une est de concaténer les séquences des éléments de L1 et L2 et de créer une nouvelle liste :

```
> L:=[op(L1),op(L2)];
```

(23)

$$L := [a, b, c, d, e, f] \quad (23)$$

2. Les fonctions

Nous avons vu que Maple connait des fonctions par défaut : *exp, ln, sin, sqrt, ...* Vous pouvez aussi définir vos propres fonctions. Pour cela, il y a plusieurs méthodes.

2.1 L'opérateur flèche ->

Prenons un exemple :

```
> restart; f(x):=x^2; f(3);
```

$$f(x) := x^2$$
$$f(3) \quad (24)$$

Maple ne sait pas calculer f(3), parce que nous ne lui avons pas défini correctement la fonction f !

Règle importante : sous Maple, on **ne définit pas** une fonction par f(x):= ... On a juste défini une expression en x.

L'opérateur flèche -> permet de définir une fonction d'une façon très proche de la notation mathématique. La syntaxe est: nom_fonction : = nom_variable -> expression_en_variable

```
> f:=x->x^2; f(3); f(10);
```

$$f := x \to x^2$$
$$9$$
$$100 \quad (25)$$

On peut aussi définir des fonctions de plusieurs variables : nom_fonction : = (séquence_noms_variables) -> expression_en_variables

```
> g:=(a,b)->a^b; g(2,3);
```

$$g := (a, b) \to a^b$$
$$8 \quad (26)$$

(attention à bien mettre la séquence des variables entre parenthèses)

Le résultat d'une fonction peut etre d'un type quelconque, par exemple une liste :

```
> h:=x->[cos(x),sin(x)]; h(Pi/6);
```

$$h := x \to [\cos(x), \sin(x)]$$
$$\left[\frac{1}{2}\sqrt{3}, \frac{1}{2}\right] \quad (27)$$

2.2 La fonction *unapply*

Une autre méthode pour définir une fonction est d'utiliser *unapply* dont la syntaxe est : unapply(expression, séquence_noms_variables). Elle peut etre pratique quand on veut transformer une expression déjà calculée par Maple en une fonction.

```
> restart; f:=unapply(x^2+3*x+7,x);f(3);
```

$$f := x \to x^2 + 3x + 7$$
$$25 \quad (28)$$

Pour une fonction de plusieurs variables :

```
> g:=unapply([x^2+y^2,x*y],x,y);
```

$$g := (x, y) \to \left[x^2 + y^2, x\,y\right] \quad (29)$$

2.3 La composition des fonctions

La composition des applications se fait à l'aide du symbole @.

```
> restart;f:=exp@ln;
```

$$f := \exp@\ln \quad (30)$$

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

$$x \quad (31)$$

On peut composer plusieurs fois une application avec elle-meme (par exemple on note g^(5)=gogogogog). Chercher dans l'aide la syntaxe Maple pour définir g^(5).

3. Les fonctions add et mul

Elles permettent de calculer des *sommes* et des *produits*. Leur syntaxe est la suivante :

add(f(i),i=a..b) calcule la somme des f(i), pour i entier variant de a à b.

mul(f(i),i=a..b) calcule le produit des f(i), pour i entier variant de a à b.

Bien sur, i, a et b peuvent avoir d'autres noms. Mais i doit etre une variable non affectée.

```
> add(cos(x)^k,k=0..7);
```

$$1 + \cos(x) + \cos(x)^2 + \cos(x)^3 + \cos(x)^4 + \cos(x)^5 + \cos(x)^6 + \cos(x)^7 \quad (32)$$

4. La fonction map

La fonction *map* permet, entre autres, d'appliquer une fonction (d'une variable) à tous les éléments d'un ensemble ou d'une liste.

```
> restart; f:=x->x^3;
```

$$f := x \to x^3 \quad (33)$$

```
> E:={a,b,c,d,e}; L:=[a,b,d,c,e];
```

$$E := \{a, b, c, e, d\}$$
$$L := [a, b, d, c, e] \quad (34)$$

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

$$\{a^3, b^3, c^3, e^3, d^3\} \quad (35)$$

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

$$[a^3, b^3, d^3, c^3, e^3] \quad (36)$$

Attention, on ne peut pas appliquer la fonction *map* à une séquence ! Voici un exemple :

```
> S:=a,b,c,d,e; map(f,S);
```

$$S := a, b, c, d, e$$
$$a^3 \quad (37)$$

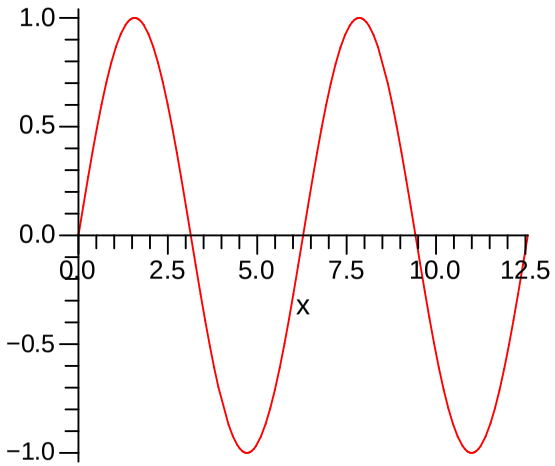
On voit que la fonction s'est appliquée uniquement au premier élément de la séquence.

5. Les représentations graphiques

Nous allons voir comment tracer le graphique d'une fonction f définie sur **R** (ou un intervalle de **R**) à valeurs dans **R**. On utilise pour cela la fonction *plot* (qui permet de tracer également d'autres types de graphiques). La syntaxe est

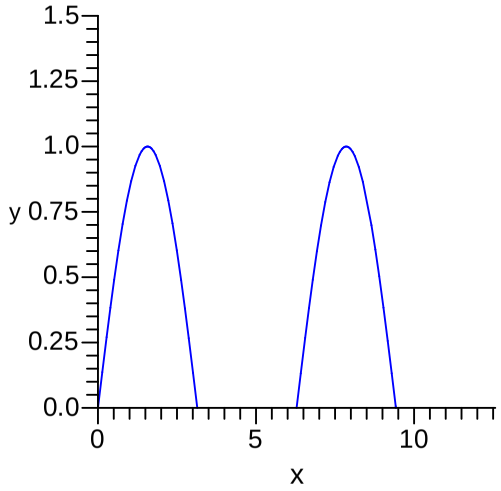
la suivante : plot(f(x) , x=a..b) où a et b désignent les bornes de l'intervalle (de l'axe des abscisses) sur lequel on trace *f*.

```
> restart;
  plot(sin(x),x=0..4*Pi);
```



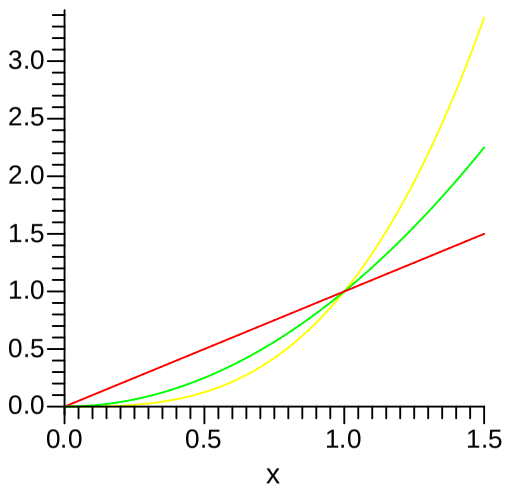
Référez-vous à l'aide de *plot* pour en savoir plus. On peut par exemple spécifier un intervalle en ordonnée pour le tracé et choisir la couleur de la courbe.

```
> plot(sin(x),x=0..4*Pi,y=0..1.5,color=blue);
```



Pour tracer plusieurs fonctions *f1,...,fn* à la fois, on utilise la syntaxe *[f1(x),...,fn (x)]* :

```
> plot([x,x^2,x^3],x=0..1.5);
```



Il existe des fonctions avancées permettant de tracer d'autres types de graphiques. La plupart sont contenues dans la **bibliothèque** (*package* en anglais) nommée **plots**. Une bibliothèque est un ensemble de commandes et fonctions relatives à un thème (ici, la représentation graphique) qui ne sont pas disponibles par défaut au lancement de Maple. Si vous souhaitez utiliser ces fonctions, vous devez charger la bibliothèque avec la commande *with*.

```
> with(plots);
```

(38)

```
Warning, the name changecoords has been redefined
[Interactive, animate, animate3d, animatecurve, arrow, changecoords,
    complexplot, complexplot3d, conformal, conformal3d, contourplot,
    contourplot3d, coordplot, coordplot3d, cylinderplot, densityplot, display,
    display3d, fieldplot, fieldplot3d, gradplot, gradplot3d, graphplot3d,
    implicitplot, implicitplot3d, inequal, interactive, interactiveparams,
    listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot,
    logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot,
    pointplot3d, polarplot, polygonplot, polygonplot3d,
    polyhedra_supported, polyhedraplot, replot, rootlocus, semilogplot,
    setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot,
    surfdata, textplot, textplot3d, tubeplot]
```

Maple affiche la liste des nouvelles fonctions disponibles (si vous ne voulez pas que toutes ces lignes s'affichent, pensez à utiliser " : " en fin de ligne au lieu de " ; " à chaque fois que vous chargez une bibliothèque avec *with*). Par exemple, pour placer des points dans un plan en donnant leurs coordonnées, on peut utiliser la fonction *pointplot*. Cherchez la commande *pointplot* dans l'aide :

```
> ?pointplot
> pointplot({[2,5],[-3,4],[7,2]},symbol=BOX);
```