Introduction à la langue de Maple

Maple est un logiciel de *calcul formel*. Cela signifie que Maple sait effectuer des calculs exacts sur des expressions contenant des variables, comme le font les calculatrices avancées, et non pas seulement du calcul numérique. Par exemple, l'instruction « `expand((x+1)^3);` » — développer $(x+1)^3$ — conduit Maple au résultat suivant : x^3+3x^2+3x+1. Ici, Maple sait manipuler la variable x sans jamais se poser la question de son éventuelle valeur.

1 Premier contact

Quand vous ouvrez Maple, une feuille de calcul vierge se présente à vous d'emblée, avec la possibilité d'écrire à droite du symbole « > », appelé *prompt* en anglais et *invite* en français. C'est ici que vous écrivez vos *instructions*. Au signal « Entrée », Maple obéit et vous livre son résultat.

```
> 2*3;              # 2 fois 3.
  2^4;              # 2 puissance 4.
  8/2:              # Maple n'affiche pas le résultat de cette instruction.
                                        6
                                       16
```

Que tirer de l'exemple de ces trois instructions très simples ?

- Trois symboles indispensables : l'étoile « `*` » pour la multiplication, l'accent circonflexe « `^` » pour les puissances et le slash « `/` » pour la division.
- Toute instruction doit s'achever par un point-virgule « `;` » ou par deux points « `:` ». La différence entre ces deux possibilités, c'est que Maple n'affiche pas le résultat de son calcul dans le cas de deux points.
- Tout ce qui suit un dièse « `#` » sur une même ligne est considéré comme un commentaire et n'est pas analysé par Maple. Aucun commentaire, bien entendu, n'est nécessaire.
- Pour passer à la ligne dans un même prompt « > », il ne faut pas appuyer sur « Entrée » mais sur « Shift + Entrée ». On n'appuie sur « Entrée » que quand on souhaite voir Maple effectuer les calculs en cours de rédaction.

Si vous oubliez le point-virgule ou les deux points à la fin d'une instruction, Maple vous le rappellera aussitôt ainsi :

```
> 6*7
Warning, incomplete statement or missing semicolon
```

1.1 Fonctions usuelles et nombres remarquables

Maple connaît toutes les fonctions usuelles : `exp`, `ln`, `sin`, `cos`, `tan`, `abs` (valeur absolue), `sqrt` (racine carrée, « square root » en anglais)… ainsi que les nombres remarquables : `Pi` pour π (attention à la majuscule), `exp(1)` pour e, `infinity` pour ∞, `-infinity` pour $-\infty$ et `I` pour le nombre complexe i (attention à la majuscule).

```
> exp(ln(x));
  exp(I*Pi);            # Attention : l'étoile * du produit est OBLIGATOIRE !
                                        x
                                       -1
```

1.2 Calculs numériques exact et approché

Par défaut, Maple effectue seulement des calculs exacts. L'exemple suivant montre qu'il simplifie de lui-même les fractions, mais aussi que les gros calculs ne lui font pas peur. Eventuellement il prend son temps — essayez de calculer 10000! par exemple.

```
> 4/3+3/4;
  100!;
                                       25
                                       --
                                       12
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

La fonction `evalc` — « `c` » comme « complexe » — renvoie tout nombre complexe sous forme algébrique.

```
> exp(I*x);
  evalc(exp(I*x));
```

$$e^{ix}$$
$$\cos x + i\sin x$$

Grâce à `evalf` — « `f` » comme « flottant », mais nous comprendrons cela plus tard — on peut quand même forcer Maple à faire du calcul approché. Par défaut, Maple garantit 10 chiffres significatifs. Notez bien que la virgule est un point !

```
> evalf(200/3);
  evalf(200/3,15);            # On précise : 15 chiffres significatifs.
```

$$66.66666667$$
$$66.6666666666667$$

1.3 Résolution d'équations

Dans l'exemple suivant, on résout l'équation $x^4+3x^2+2=0$ d'inconnue $x\in\mathbb{C}$, puis d'inconnue $x\in\mathbb{R}$, puis on résout le système $\left\{\begin{array}{ccccc} x & + & 2y & = & 1 \\ 2x & + & 5y & = & 3 \end{array}\right.$ d'inconnue $(x,y)\in\mathbb{R}^2$. La fonction `fsolve` se comporte comme la fonction `solve`, mais elle ne renvoie que les solutions réelles, sous forme approchée.

```
> solve(x^4-2*x^2-3=0,x);                    # Résolution exacte dans C.
  fsolve(x^4-2*x^2-3=0,x);                   # Résolution approchée dans R.
  solve({x+2*y=1,2*x+5*y=3},{x,y});          # Attention aux accolades !
```

$$\sqrt{3},-\sqrt{3},i,-i$$
$$-1.732050808, 1.732050808$$
$$\Big\{x=-1,y=1\Big\}$$

1.4 Manipulations de fonctions

On peut bien entendu créer soi-même ses propres fonctions.

```
> f:=x->exp(x)/(x^2+2);          # On crée une fonction f.
  f(1);                          # On l'évalue en 1 de manière exacte.
  evalf(");              # On calcule une valeur approchée du résultat précédent.
```

$$f:=x\longmapsto\frac{e^x}{x^2+2}$$
$$\frac{e}{3}$$
$$0.9060939426$$

- Maple distingue — comme il se doit ! — la *fonction* f de l'*expression* $f(x)$.
- Le symbole « `"` » contient le résultat du dernier calcul effectué par Maple : Maple l'utilise tel qu'il l'a stocké sans le re-calculer. Le résultat de l'avant-dernier calcul s'obtient de même avec la symbole « `""` ».

Maple sait composer les fonctions au moyen du symbole « `@` ».

```
> g:=x->x+1;
  (tan @ g)(x);
```

$$g:=x\longmapsto x+1$$
$$\tan(x+1)$$

Pour le calcul de limites, utiliser la fonction `limit`.

```
> limit((sin(x)-ln(1+x))/(exp(x)-cos(x))^2,x=0);
  limit(tan(x),x=Pi/2,right);
```

$$\frac{1}{2}$$
$$-\infty$$

Maple propose deux fonctions de dérivation : la fonction `D` pour les fonctions, la fonction `diff` pour les expressions.

```
> D(exp@sin);                    # On dérive la fonction exp o sin.
  (D@@3)(x->exp(2*x));           # On dérive trois fois...
  diff(x*ln(x),x);               # On dérive x*ln(x) par rapport à x.
  diff(x*ln(x),x$2);             # On dérive deux fois...
```

$$\exp \circ \sin \cos$$
$$x \longmapsto 8e^{2x}$$
$$\ln(x) + 1$$
$$\frac{1}{x}$$

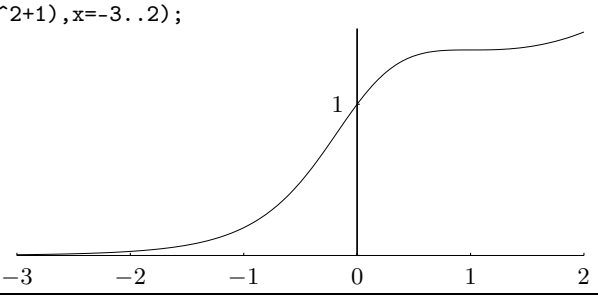
Pour la primitivation et le calcul intégral, utiliser la fonction `int`.

```
> int(x*ln(x),x);          # On calcule UNE primitiver de x*ln(x) par rapport à x.
  int(x*ln(x),x=1..2);     # Intégrale de 1 à 2 de x*ln(x) par rapport à x.
```

$$\frac{x^2}{2} \ln(x) - \frac{x^2}{4}$$
$$2\ln(2) - \frac{3}{4}$$

Enfin, pour obtenir un tracé du graphe de la fonction $x \longmapsto \dfrac{e^x}{x^2+1}$ sur $[-3, 2]$, on écrira :

```
> plot(exp(x)/(x^2+1),x=-3..2);
```



1.5 Sommes et produits

L'usage des symboles $\sum$ et $\prod$ est assez naturel :

```
> sum(1/k,k=1..100);
  evalf(");                # Evaluation approchée du résultat du calcul précédent.
  product(p,p=1..n);
  sum(1/k^2,k=1..infinity);              # Calcul exact d'une somme infinie !
```

$$\frac{14466636279520351160221518043104131447711}{2788815009188499086581352357412492142272}$$
$$5.187377518$$
$$\Gamma(n+1)$$
$$\frac{\pi^2}{6}$$

- Par défaut Maple fait du calcul exact — d'où le premier résultat affreux. La commande `evalf` fournit juste après un résultat approché. Quant au symbole « " », il contient le résultat du dernier calcul effectué par Maple : Maple l'utilise tel qu'il l'a calculé juste avant ; il ne le re-calcule pas.
- La troisième instruction demande le calcul du produit $\prod_{p=1}^{n} p$. On attendait comme résultat $n!$, mais Maple répond $\Gamma(n+1)$. Qu'est-ce que c'est que ce bazar ? De fait Maple connaît bien plus de mathématiques que vous, et souvent il vous répond comme si vous en saviez autant que lui. Heureusement pas de panique : l'aide Maple est très complète — en anglais. Pour tout savoir, cliquer sur « Topic search » dans l'onglet « Help » de la barre d'outils et chercher « GAMMA ».

1.6 Packages

On appelle *package* tout ensemble de fonctions groupées ensemble autour d'un thème mathématique commun. Maple connaît à l'avance tout un tas de fonctions mais elles ne sont pas toutes directement disponibles à l'ouverture d'une feuille de travail. Il convient dans ce cas de les charger à l'aide de la commande `with`. Voici quelques packages dont nous nous servirons : `plots` (pour faire des tracés de graphes notamment), `numtheory` (pour faire de l'arithmétique) et `linalg` (pour faire de l'algèbre linéaire, mais vous ne savez pas encore ce que c'est).

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

$$[animate, animate3d, changecoords, complexplot, complexplot3d, conformal, contourplot, contourplot3d, coordplot, coordplot3d, cylinderplot, densityplot, display, display3d, fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot, implicitplot3d, inequal, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, odeplot, pareto, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedraplot, replot, rootlocus, semilogplot, setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot, surfdata, textplot, textplot3d, tubeplot]$$

On a ici chargé le package `plots`. Maple nous liste alors l'ensemble des fonctions disponibles au sein de ce package.

2 Variables et affectations

Une *variable* n'est qu'un nom auquel on peut attacher — ou non — une valeur. Par exemple, pour donner à la variable x la valeur 5, on écrira :

```
> x:=5;
```

$$x := 5$$

L'essentiel ici, c'est qu'on a utilisé le symbole dit d'*affectation* « `:=` » et non l'égalité « `=` ». Il ne s'agit pas en effet d'affirmer l'égalité $x = 5$, ce qui supposerait que x a déjà la valeur 5, mais de donner un contenu (5) à une variable (x) — x est comme une boîte que l'on remplit du nombre 5. Pour ne pas confondre l'ordre des termes, on pourra lire l'instruction « `x:=5;` » de la façon suivante : « x reçoit 5 ».

Par défaut, quand on ouvre une feuille de calcul Maple, toutes les variables sont *indéfinies*, i.e. sans contenu aucun. Pour supprimer le contenu de la variable x, on utilisera la syntaxe suivante, qui signifie que l'on remplit la boîte x de son seul nom :

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

$$x := x$$

On peut aussi « dés-affecter » d'un coup d'un seul toutes les variables au moyen d'un `restart`.

```
> restart;
```

Tâchez de comprendre chacun des résultats de Maple reportés ci-dessous :

```
> y;
  y:=2;
  y;
  y:=y+1;
```

$$y$$
$$y := 2$$
$$2$$
$$y := 3$$

Remarquez notamment qu'on peut remplir une variable sans l'avoir vidée auparavant. L'ancien contenu est simplement oublié.

Comparez maintenant les résultats des deux tableaux suivants :

```
> a:=Pi;              # a reçoit pi.
  b:=exp(a);          # b reçoit exp(a), i.e. exp(pi).
  b/(b+1);            # On calcul b/(b+1), i.e. exp(pi)/(exp(pi)+1).
  a:='a':             # On vide a.
  b;                  # On re-calcule b, dont la valeur n'a pas bougé.
```

$$a := \pi$$
$$b := e^{\pi}$$
$$\frac{e^{\pi}}{e^{\pi}+1}$$
$$e^{\pi}$$

```
> b:=exp(a);              # b reçoit exp(a), mais a est encore vide.
  a:=Pi;                  # a reçoit pi.
  b/(b+1);                # On calcule b/(b+1), i.e. exp(a)/(exp(a)+1), i.e.
                          # exp(pi)/(exp(pi)+1).
  a:='a':                 # On vide a.
  b;                      # On re-calcule b, i.e. exp(a) où a est une boîte vide.
```

$$b := e^a$$
$$a := \pi$$
$$\frac{e^\pi}{e^\pi + 1}$$
$$e^a$$

Ces deux exemples montrent que Maple remplace toujours toutes les variables par leur valeur quand elles en ont une, jusqu'à ne plus obtenir que des variables indéfinies. Certaines affectations posent problème, du coup :

```
> z:=z+1;
Warning, recursive definition of name
```

$$z := z + 1$$

Maple nous prévient ici que nous venons de donner une définition *récursive* de z, i.e. une définition de z qui contient z, ce qui évidemment ne définit rien du tout. Après cela, Maple râlera vraiment si nous lui demandons la valeur de z, au motif que nous lui demandons un calcul infini : pour calculer z il doit connaître z, donc le calculer, donc le connaître, donc le calculer, etc.

```
> z;
Error, too many levels of recursion
```

Penchons-nous enfin sur la question de l'échange des valeurs de deux variables. Commençons par écrire des bêtises.

```
> u:=1;
  v:=2;
  u:=v;
  v:=u;
  u;
  v;
```

$$u := 1$$
$$v := 2$$
$$u := 2$$
$$v := 2$$
$$2$$
$$2$$

Problème : nous n'obtenons pas du tout ainsi l'échange souhaité. Mais c'est normal : l'instruction « `u:=v;` » donne à u la valeur 2, et de cette façon la précédente valeur de u, ici 1, est purement et simplement **OUBLIÉE**. On remédie à cela en introduisant une troisième variable de stockage provisoire. Il est impératif que vous compreniez bien et reteniez cette idée.

```
> u:=1;
  v:=2;
  temp:=u;            # Variable temporaire de stockage, indispensable !
  u:=v;
  v:=temp;
  u;
  v;
```

$$u := 1$$
$$v := 2$$
$$temp := 1$$
$$u := 2$$
$$v := 1$$
$$2$$
$$1$$

3 Types

Le *type* d'un objet Maple est l'ensemble des opérations auxquelles cet objet peut être soumis. On peut par exemple calculer la décomposition en produit de facteurs premiers d'un entier avec la fonction `ifactor`, mais pas d'un réel en général. Les entiers (relatifs) sont de type `integer`, les rationnels de type `fraction` ou `rational`, les réels de type `float`, `realcons` ou `numeric`, les nombres complexes de type `complex`, etc. Il ne nous sera pas utile de connaître l'architecture de ces types et le détail des restrictions qui leur sont liées. Seuls quelques types nous retiendront ici.

On peut tester le type d'un objet Maple à l'aide de la fonction `type`.

```
> type(4,integer);
  type(Pi,integer);
```

$$true$$
$$false$$

3.1 Type integer

Les entiers (relatifs) sont de type `integer`. En leur présence, Maple n'effectue que du calcul exact. Certaines fonctions leur sont dévolues, que nous aurons l'occasion d'utiliser quand nous ferons de l'arithmétique notamment : `ifactor` pour la décomposition en produit de facteurs premiers, `isprime` pour savoir si un entier est premier ou non, `mod` pour effectuer un calcul de reste modulo un entier, etc.

```
> ifactor(40);       # Factoriser de 40 en produit de facteurs premiers.
  isprime(36);       # 36 est-il premier ?
  17 mod 5;          # Calculer le reste de la division euclidienne de 17 par 5.
```

$$(2)^3(5)$$
$$false$$
$$2$$

3.2 Type float

Les *flottants*, de type `float`, sont les objets que Maple dédie au calcul approché. On a tendance à les identifier aux réels, mais les choses sont en fait plus compliquées. Un flottant se présente toujours sous forme décimale avec un nombre fixé de chiffres significatifs. Ce nombre de chiffres significatifs est un paramètre noté `Digits` et dont la valeur par défaut est 10. Attention, la virgule est toujours notée d'un point.

```
> Digits:=5;             # On fixe ici à 5 le nombres de chiffres significatifs.
  4.333333333*2;
```

$$Digits := 5$$
$$8.6666$$

Rappelons qu'on peut toujours forcer Maple à faire du calcul approché grâce à la fonction `evalf` — « `f` » comme « flottant ».

```
> evalf(Pi);             # Actuellement, Digits vaut 5.
  evalf(Pi,30);          # On demande 30 chiffres significatifs.
```

$$3.1416$$
$$3.14159265358979323846264338328$$

Attention enfin : Maple distingue scrupuleusement les entiers des flottants. Pour lui, 12 est un entier mais « 12. » (avec un point) est un flottant. Tâchez de comprendre ses réponses dans l'exemple ci-dessous.

```
> evalf(12);
  type(12,integer);
  type(12,float);
  ifactor(12);
  ifactor(12.);
```

$$12.$$
$$true$$
$$false$$
$$(2)^2(3)$$

Error, (in ifactor) invalid arguments

3.3 TYPE boolean

Les *booléens*, dont le type est `boolean`, sont au nombre de trois : `true`, `false` et `FAIL` — qui signifie « Je ne sais pas ». On peut les tester au moyen de la fonction `is`.

```
> is((x+1)^2=x^2+2*x+1);
  is(0=1);
  is(x<>1);
```

$$true$$
$$false$$
$$FAIL$$

- Notez bien qu'ici c'est le symbole d'égalité « = » qui est utilisé, et non le symbole d'affectation « := ».
- Le symbole « <> » signifie « $\neq$ ». Autres symboles apparentés : « <= » pour « $\leqslant$ », etc.

Maple sait calculer des négations, des conjonctions et des disjonctions au moyen des fonctions logiques `not`, `and` et `or`.

```
> true and false;
  true or FAIL;
  not FAIL;
```

$$false$$
$$true$$
$$FAIL$$

On peut par ailleurs faire des hypothèses sur une variable indéfinie grâce à la fonction `assume`.

```
> x;
  assume(x>0);
  x;
  about(x);
```

$$x$$
$$x\sim$$

```
Originally x, renamed x~ :
  is assumed to be : RealRange(Open(0),infinity)
```

- A la première instruction, Maple répond x, à savoir le nom de la variable indéfinie x. Après l'hypothèse `assume`, il répond au contraire $x\sim$ à la même question. Pourquoi ? L'ajout du suffixe « $\sim$ » n'est qu'une manière d'indiquer que la variable x est indéfinie au sens où elle n'a aucun contenu précis, mais tout de même astreinte à une hypothèse.
- La fonction `about` rappelle quelles hypothèses on a fait porter sur x. Ici, x est supposé élément de l'intervalle $]0,\infty[$ de $\mathbb{R}$.

Intérêt de la fonction `assume` : ce qui ne peut être décidé sans hypothèse peut parfois l'être avec !

```
> about(w);
  is(w^2>=0);
  assume(w,real);
  is(w^2>=0);
```

```
w :
  nothing known about this object
```

$$FAIL$$
$$true$$