Introduction à MAPLE

Version 3.3, Septembre 2002
S. Gonnord
Lycée du parc, Lyon
`http://perso.wanadoo.fr/stephane.gonnord/`

Table des matières

1 Présentation générale

1.1 The name of the game

Maple est la traduction de *érable* (d'où le logo). Il n'est en aucun cas question d'une pomme qui aurait gagné un P on ne sait pas trop comment. On retiendra donc :

Maple ne prend qu'un P

1.2 Pourquoi et comment utiliser MAPLE

MAPLE est un langage **interprété**, permettant d'effectuer des calculs **formels** et **numériques** ; (très) accessoirement, on peut utiliser MAPLE comme traitement de texte scientifique (tsss...).

De façon schématique, on peut dire que MAPLE réalise *formellement* tous les calculs pour lesquels nous avons un algorithme : dérivation, intégration de certaines fonctions, résolution de systèmes d'équations linéaires ou récurrentes, factorisation de polynômes, réduction en algèbre linéaire, résolution de certaines équations différentielles, etc... *Numériquement*, on peut évaluer des intégrales, sommes de séries, racines de polynômes... MAPLE possède également un certain nombre de fonctions graphiques, permettant d'obtenir des représentations de courbes en 2D et 3D, ainsi que des surfaces.

On lance MAPLE à la souris en double-cliquant sur l'icône de l'application (présente normalement sur le bureau de Windows). On dispose alors d'une fenêtre avec un prompt "`>`". On exécute une instruction `toto` en tapant : "`>toto;`" (suivi de ENTER...). L'oubli du point-virgule est la première source de perte de temps du débutant... et donc de son prof !

En permanence, on a accès à l'aide de MAPLE : soit en utilisant le menu HELP, soit en tapant "`>?toto;`" pour avoir la description de la commande "`toto`". Il faut s'habituer à utiliser l'aide pour voir la syntaxe et l'usage précis d'une fonction. En pratique, je commence toujours par aller voir les exemples fournis à la fin des pages d'aides : ils répondent en général à ma question.

MAPLE dispose d'une grande librairie de fonctions et procédures, que l'on peut utiliser directement ou à l'intérieur de programmes (procédures). On décrira essentiellement le second aspect dans ce poly, la familiarisation avec les fonctions usuelles se faisant en TD et TP ("khôlles").

1.3 Les objets manipulés par MAPLE

1.3.1 Expressions algébriques

C'est la structure de base de MAPLE. Il faut s'habituer à voir une expression comme un arbre[1], pour comprendre dans quel ordre Maple exécute un calcul.

Les expressions sont construites avec des opérateurs $(+, -, *, /, !, \ldots)$, des fonctions, des variables et des constantes. En exercice, on représentera les arbres correspondant aux expressions suivantes : `evalf(1/3*3)`, `3*x^2-5*sin(x+y)`, `evalf(1/3)*3`.

1.3.2 Types

MAPLE travaille sur de nombreux *types* courants, ce qui évite à l'utilisateur d'avoir à redéfinir par exemple un type `Entierlong`, ou `complexe` comme en pascal par exemple. On dispose donc essentiellement des types suivants :

1. Les entiers (`integer`). Ils sont arbitrairement longs, comme on peut le vérifier en faisant : `>1234^1234;`
2. Les fractions (`fraction`). C'est la donnée de deux entiers : le numérateur et le dénominateur. A noter que MAPLE simplifie les fractions systématiquement.
3. Les décimaux (`float`). Un décimal est la donnée de deux entiers relatifs : la mantisse et l'exposant. Par exemple, MAPLE représente 3.14 comme le couple $(314, -2)$. Se souvenir que la notation est anglo-saxonne : les "." remplacent les ",". Par ailleurs, 3 est un entier alors que 3. est un décimal. A noter que

[1] même si c'est en réalité un peu plus compliqué...

les décimaux n'ont pour MAPLE qu'un nombre "localement fixe" de chiffres significatifs : il s'agit *par défaut* de 10. On peut modifier directement ce chiffre qui est stocké dans la variable globale `Digits`.

4. Les booléens (`boolean`). C'est un type qui contient essentiellement les constantes `true,false`. Il s'agit en particulier du type du résultat de l'évaluation d'égalités, ou d'inégalités... mais une fonction peut également retourner un booléen (par exemple, `isprime`).
5. Les suites d'expressions (`exprseq`). Ce sont des expressions séparées par des virgules. La suite vide est notée `NULL`. On les produit en général avec `seq` ou $.
6. Les listes (`list`) et ensembles (`set`). Ce sont des familles d'expressions séparées par des virgules. Une liste se note `[a,12,x+2]`, alors qu'un ensemble se note `{13/4,5.1,12,true}`. Les deux différences essentielles entre ces deux structures sont d'une part le fait que les listes sont ordonnées alors que les ensembles ne le sont pas, et d'autre part un élément ne peut pas être répété deux fois dans un ensemble, alors que c'est possible dans une liste :

   ```
   >map(evalb,[{1,2}={2,1},[1,2]=[2,1],{1,1}={1},[1,1]=[1]]);
                    [true,false,true,false]
   ```

   Dans l'exemple précédent, on a appliqué (`map`) la fonction `evalb` (qui évalue une expression booléenne) à une liste de quatre expressions. Le résultat est une liste contenant le résultat des évaluations *(relisez calmement cette phrase, je vous assure qu'elle a un sens...)*.

   Les fonctions `op,nops,member,has,union,intersect` permettent de manipuler ces structures. A titre d'exercice, on peut chercher à concaténer deux listes, ou changer des listes en ensemble et vice-versa : cf TD.
7. Les procédures (`proc`). Voir la section 3.
8. Les intervalles (`range`) entiers ou réels : `-infinity..x` ou `3..17`. Ils interviennent dans les intégrales (`int`) et sommes (`sum`).

On peut vérifier le type d'une expression par la fonction `type()`, prenant en entrée une expression et un type ; le résultat est un booléen.

```
> type(17,integer),type(17.0,integer),type(17.0,float);
                          true, false, true
```

1.3.3 Constantes

MAPLE connaît les constantes usuelles... et quelques autres. Celles dont vous vous servirez sont : `Pi`, `exp(1)`, `I` (le i complexe, qui permet d'utiliser les complexes de façon transparente ; tapez par exemple `(1+I)^2;`), `gamma` (constante d'Euler), `infinity`, `true`, et `false`. A noter que `e` n'est pas une constante, pas plus que `E`...

1.4 Les " ;" et les " :"

Ce sont des séparateurs. Ils sont mis à la fin d'une instruction pour en provoquer l'exécution. La différence entre " ;" et " :" est *grosso modo* que dans le premier cas, le résultat de l'instruction (s'il existe) est affiché, alors que dans le second cas, le résultat n'est pas affiché. L'usage du " :" est conseillé quand on ne veut pas que MAPLE affiche des résultats "intermédiaires".

A l'intérieur des programmes, les règles pour mettre ou ne pas mettre un séparateur sont **cruciales**, et décrites dans le paragraphe 3.

1.5 WARNING

1.5.1 Maple est *case sensitive*

Cela signifie qu'il distingue les majuscules et les minuscules. Si on a défini une fonction `toto`, Maple ignore donc la fonction `Toto`...

1.5.2 Affectation vs égalité

Les phrases "x a la valeur 10" et "je donne à x la valeur 10" ont deux sens très différents : dans le premier cas, il s'agit d'une *proposition*, qui est vraie ou fausse. Dans le deuxième cas, il s'agit d'une *action*, qui change la valeur de x.

De même, `x=5` désigne en Maple une *égalité*, qui peut s'évaluer à `true` ou `false`, alors que `x :=5` est une *affectation* : c'est une instruction qui donne la valeur 5 à la variable x.

Mélanger les affectations et les égalités aura un effet dramatique en Maple... comme en maths, d'ailleurs.

2 Quelques possibilités offertes par MAPLE

Nous donnons ici un petit catalogue **très limité** de ce que l'on peut faire en MAPLE :

2.1 Calculs formels

MAPLE sait dériver formellement, résoudre des équations différentielles, calculer des limites, des séries de Taylor, des développements asymptotiques, des primitives... En algèbre linéaire, vous pourrez obtenir des réductions de matrices, des polynômes caractéristiques... On dispose également de nombreuses possibilités en arithmétique et géométrie.

```
>taylor(sin(x),x=0,6),asympt(1+x/(1+x^2),x),
 dsolve(diff(y(x),x)+y=0,y(x)),ithprime(10000),isprime(10000),
 factor(X^4+1,sqrt(2)); # factorisation dans un sur-corps de Q
```

$$x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^7)\ ,\ 1 + \frac{1}{x} - \frac{1}{x^3} + \frac{1}{x^5} + O(\frac{1}{x^7}),$$

$$y(x) = e^{-x}C_1\ ,\ 104729,\ \texttt{true},\ (X^2 + \sqrt{2}X + 1)(X^2 - \sqrt{2} + 1)$$

2.2 Calculs numériques

Vous pouvez évaluer des intégrales ou des sommes de séries. Comme d'habitude, MAPLE vous donnera un résultat exact s'il connaît un algorithme de calcul ; sinon, il vous retournera l'expression que vous avez entrée ! Pour avoir une évaluation d'une série ou intégrale que l'on ne sait pas calculer, il faut utiliser `evalf` ; MAPLE vous signalera cependant s'il n'a pas réussi à calculer le résultat exact.

```
>int(exp(-x^2),x=-infinity..infinity),sum(1/n^30,n=1..infinity),
>sum(1/(n^2+n^(1/4)),n=1..infinity),evalf(sum(1/(n^2+n^(1/4)),n=1
..infinity));
```

$$\sqrt{\pi}, \frac{6892673020804}{566087880466908267407001562 5}\pi^{30}, \sum_{n=1}^{\infty} \frac{1}{n^2 + n^{1/4}}, \text{FAIL}(1.062926930)$$

2.3 Graphiques

On peut tracer de nombreuses courbes 2D ou 3D : des courbes simples, paramétrées, implicites. On peut également représenter des surfaces, éventuellement paramétrées ou implicites... Vous pouvez vous amuser en réalisant les tracés suivants :

```
>plot(x*sin(x),x=-3*Pi..3*Pi);
>plots[polarplot](2*t);
>plot3d([x*sin(x),x*cos(y),x*sin(y)],x=0..2*Pi,y=0..Pi);
```

Testez l'interface permettant de "manipuler" des graphiques 3D.

3 Programmation en MAPLE

3.1 Syntaxe générale

La syntaxe d'un programme est :

```
proc(<inputs>)
local <varlocales>;
    <instruction 1>;
    ...
    <instruction n-1>;
    <instruction n>
end;
```

1. Prendre garde aux ";" : ils peuvent être remplacés par des " :" (voir 1.4), mais on doit en mettre ni trop ni trop peu.
2. `<inputs>` désigne les entrées du programme **sans le type !** Exemple : `proc(N)` `proc(N,x,y)`.
3. `<varlocales>` désigne les variables locales. Exemple : `local x,n;`
4. Les instructions peuvent être des sous-programmes, des appels de programmes, des assignations, des tests ou des itérations...

3.2 Variables locales

Ce sont des variables que l'on utilise dans la procédure **localement** : une variable globale portant le même nom qu'une variable locale ne verra pas sa valeur affectée si on change la valeur de la variable locale :

```
>x:=‘itchy‘:proc() local x; x:=‘scratchy‘ end:x;
                          itchy
```

Le programme suivant réalise une multiplication par 2 :

```
> fois_deux:=proc(x) local z;
  z:=x; RETURN(x+z);
  z:=3; end:
>fois_deux(4);
                                   8
> x,z;
                                  x,z
> x:=4:z:=12:
> x,z;
                                  4,12
> fois_deux(17);
                                   34
> x,z;
                                  4,12
```

Le lecteur est **vivement** invité à méditer ce qui précède...

3.3 Tests

Le programme suivant prend un entier, et retourne le minimum de cet entier et de 1000 :

```
>tres_grand:=proc(x)
if x>1000 then RETURN(1000) else RETURN(x) fi
end:
> tres_grand(200),tres_grand(2000);
                               200, 1000
```

Remarques :

1. S'il n'y a pas de RETURN, le résultat rendu est celui de la dernière instruction : on aurait donc pû écrire : `if x>1000 then 1000 else x fi`.
2. Il est cependant conseiller de mettre TOUJOURS l'instruction `RETURN` (avec les majuscules OBLIGATOIRES), pour être certain de ce que Maple va retourner (c'est à vous de la choisir, et non au hasard...).
3. S'il n'y a qu'une seule instruction dans la procédure et pas de variable locale, on peut utiliser la syntaxe réduite suivante pour les fonctions : `tres_grand:=x->if x>1000 then 1000 else x fi;`
4. Encore mieux : `tres_grand:=x->min(x,1000);`

La syntaxe des tests est : `if <cond> then <pr1> else <pr2> fi ;`

1. `cond` est une expression dont l'évaluation est un booléen.
2. `pr`i ($i = 1, 2$) est une suite d'instructions séparées par des " ;" ou des " :" (sauf la dernière instruction avant le `else` et le `fi`).
3. Ne pas oublier le `fi`.
4. On peut imbriquer imbriquer plusieurs tests de façon élégante en utilisant `elif`, qui est un raccourci de `else if`, et qui ne demande pas de "refermer" un deuxième `if`.
5. Un exemple illustrant la remarque précédente : on détermine lequel de deux couples de réels est le plus grand au sens lexicographique (ordre du dictionnaire : c'est le premier élément qui compte, puis, en cas d'égalité, le second) :

```
> lexico:=(x,y)->if x[1]>y[1] then x
       elif x[1]=y[1] and x[2]>=y[2] then x else y fi:
> lexico([4,2],[3,6]),lexico([4,2],[4,1]);
                              [4, 2], [4, 2],
> lexico([4,2],[4,3]),lexico([4,2],[5,3]);
                              [4, 3], [5, 3]
```

3.4 Itérations

On dispose de `while` et `for`, comme en Pascal, C ou Basic... La syntaxe (où presque tout est facultatif) est :

`for <var> from <x1> to <x2> by <step> while <cond> do <calc> od;`

1. `calc` est une suite d'instructions séparées par des ; (sauf la dernière avant `od;`). Ces instructions sont effectuées en boucle pour `var` allant de `x1` à `x2` avec un pas `step`, tant que l'évaluation de la condition `cond` donne comme résultat `true`.
2. Si on ne met pas `from...`, cela signifie par défaut : `from 1`. Même chose pour `by`
3. Deux exemples à méditer :

```
>l:=[]:for i from 1 to 10 by 2 do l:=[op(l),i] od:l;
        [1,3,5,7,9]
>l:=[]:for i from 1 to 10 by 2 while i<=5 do l:=[op(l),i] od:l;
          [1,3,5]
```

3.5 Les options

On peut inclure certaines options (essentiellement `remember`) à un programme en déclarant celles-ci après les variables locales : L'option `remember` dans une procédure récursive fait que les résultats des appels à la procédure sont stockés, évitant ainsi des appels inutiles. L'exemple canonique où cette option est utile est le calcul du $n$ème terme de la suite de Fibonacci (voir l'exemple dans le paragraphe suivant).

3.6 Quelques petits exemples

3.6.1 Suite de Fibonacci

```
fibo:=proc(N) option remember;
  if N<=1 then N else fibo(N-1)+fibo(N-2) fi
end;
```

Pour comprendre ce qui se passe lorsque l'on exécute `fibo(4)` par exemple, il est conseillé de *tracer* la fonction : taper `trace(fibo) ;` puis `fibo(4) ;`

3.6.2 Détection d'un palindrome

```
palindrome:=proc(st)   #  aabcbaa est un palindrome ...
local l,rep,i;
  rep:=true; l:=length(st);
  for i from 1 to trunc((l+1)/2) while rep do
      rep:=evalb(substring(st,i..i)=substring(st,l+1-i..l+1-i))
     od; rep; end;
```

En exercice, vous pouvez essayer de faire un programme **récursif** détectant les palindromes...

3.6.3 Racines d'un trinôme

```
prg_inutile:=proc(a,b,c)   # racines de aX^2+bX+c
local delta;
  delta:=b^2-4*a*c;
  evalc({(-b-delta^(1/2))/2/a,(-b+delta^(1/2))/2/a})
end;
```

4 Divers

4.1 Sauvegardes

On peut sauvegarder toute la session MAPLE (à la souris), ou bien seulement certaines variables grâce à `save(...)`.

Il est conseillé de sauvegarder TRES régulièrement son travail (toutes les 10 minutes par exemple).

4.2 Désassignation

Pour *désassigner* une variable (lui faire "oublier" sa valeur), on peut taper : `>x:='x';` Pour désassigner toutes les variables (donc en fait réinitialiser la session MAPLE), on exécute "restart" (à la souris). Attention, dans ce dernier cas, **tout ce qui précède** est perdu.

4.3 Librairie et "packages"

MAPLE dispose d'un très grand nombre de fonctions/procédures préprogrammées. Cependant, une petite partie seulement de ces fonctions est accessible directement. Certaines fonctions sont stockées en librairie. Pour les exécuter, il faut d'abord aller les lire en librairie : "`>readlib(toto);`". On peut alors exécuter la fonction `toto` autant de fois que l'on veut, sans devoir aller la relire à chaque fois.

De plus, MAPLE dispose d'une vingtaine d'ensembles de fonctions spécialisées. Ces ensembles sont des "packages". On peut soit aller lire l'intégralité des fonctions d'un package *pack* en tapant `>with(pack) ;` soit appeler directement la fonction *prg* du package *pack* par : `>pack[prg](...);` . Lorsque l'on charge intégralement un package, MAPLE signale de quelles nouvelles fonctions on dispose, avec en particulier celles "écrasant" des définitions précédentes (voir l'annexe **??**).

4.4 Monsieur, ça marche pas

On donne ici une série de questions à vous poser *avant* d'annoncer que votre version de Maple ne fonctionne pas :

1. La salle est-elle sous tension ?
2. L'ordinateur est-il branché ?
3. L'ordinateur est-il allumé ?
4. Maple est-il lancé ?
5. As-tu *double-cliqué* plutôt que *cliqué deux fois mollement* ?
6. As-tu mis le point-virgule ?
7. As-tu tapé `Digits` et pas `digits` ?
8. La fonction (ou variable) qui ne fait (vaut) pas ce qu'il faut a-t-elle été définie avec un `:=` et pas un `=` ?
9. Et si tu allais regarder les exemples donnés dans l'aide de cette fonction ?
10. Qu'est-ce que ta procédure est censé faire ?
11. Pourrais-tu envisager de ne pas présenter ta procédure n'importe comment (passages à la ligne, indentation...) ?
12. Et si tu écrivais en Maple plutôt qu'en C ou en Pascal ?
13. C'est quoi ce `and` à la place d'un point virgule ?
14. Ta procédure est-elle compréhensible par une autre forme d'intelligence que ton propre cerveau ?
15. As-tu enfin apporté ton poly distribué en début d'année, dans lequel on voit la syntaxe précise d'un test ?
16. As-tu enfin apporté ton poly distribué en début d'année, dans lequel on voit la syntaxe précise d'une boucle ?
17. Quand tu exécutes `f(17) ;` quelles sont les 10 premières instructions exécutées par le programme `f` ?
18. As-tu testé la fonction précédente que tu utilises ici ?
19. As-tu testé *de façon pertinente* (c'est-à-dire sur des exemples tordus) la fonction précédente que tu utilises ici ?
20. As-tu écouté ce que j'ai hurlé à toute la classe (après avoir lourdement réclamé l'attention) il y a 45 secondes ?
21. Tu te fiches de qui ?

5 Quelques fonctions utiles

Nous donnons, dans quelques domaines, le nom de certaines fonctions qui sont programmées dans le noyau de MAPLE ; pour découvrir la syntaxe exacte et les options de ces fonctions, il y a deux possibilités : les essais multiples, ou l'utilisation de l'aide. A vous de voir...

5.1 Différenciation et intégration

1. `diff` : dérivation d'une expression. On peut dériver selon plusieurs variables.
2. `taylor,asympt` : développements de Taylor ou asymptotiques d'une expression.
3. `int` : calcul de l'intégrale ou de la primitive d'une fonction.
4. `student[...]` : package contenant des commandes permettant des calculs ou étude de fonctions "à la main" : `changevar,showtangent...`

5.2 Arithmétique/dénombrement

Les packages `numtheory` et `combinat` fournissent un grand nombre de possibilités. Un package `GaussInt` contient des fonctions plus spécialisées sur les entiers de Gauss. Comme fonctions accessibles directement, on a :

1. `bernoulli,euler` : fonctions de Bernoulli et Euler.
2. `ifactor,ifactors` : facteurs premiers d'un entier.
3. `chrem` : restes chinois...

Les fonctions suivantes se passent de description : `lcm,gcd,isprime,issqr, ithprime,nextprime`.

5.3 Simplification d'expressions

Les résultats donnés par Maple ne le sont pas toujours sous une forme très sympathique : il faut parfois "retoucher" le résultat. On dispose pour cela entre autres :

1. `convert` : conversion d'une expression en somme (par exemple) de ses composantes, ou d'un cosinus en exponentielle...
2. `normal,expand,simplify` : pour normaliser, développer ou appliquer certaines règles de simplification à une expression.
3. `factor,collect` : factorisation et développement de polynômes.

5.4 Algèbre linéaire

Pour faire des multiplications ou additions de matrices/vecteurs, vous pouvez utiliser `evalm`, mais dès que vous voulez faire quelque chose de plus évolué, il faut charger le package `linalg` ; voir `??` pour la liste des fonctions disponibles.

5.5 Graphiques

Les deux premières instructions sont accessibles directement, mais toutes les suivantes sont dans le package `plots` ; il est donc conseillé de charger ce package si vous devez faire des graphiques à haute dose.

1. `plot` : tracé de divers graphiques. L'utilisation canonique étant :
   `>plot(f(x),x=a..b);` Pour les nombreuses autres possibilités existant, consulter l'aide...
2. `plot3d` : c'est l'analogue 3d de `plot`.
3. `display,display3d` : permet de superposer plusieurs graphiques.
4. `implicitplot,implicitplot3d` : tracé d'une courbe définie par une relation implicite, du type $x^2 + 2y^2 = 1$.
5. `polarplot` : tracé d'une courbe en polaire.
6. `polygonplot,polygonplot3d` : tracé d'une ligne polygonale en deux ou trois dimensions.

5.6 Diverses résolutions

MAPLE sait résoudre un certain nombre d'équations.

1. `solve,isolve,msolve,fsolve` : solveur (de systèmes) d'équations. Les préfixes "i", "m" et "f" indiquent que l'on souhaite une résolution entière, modulaire, ou décimale.
2. `dsolve` : résolution (éventuellement numérique) d'équations différentielles. On a souvent intérêt à utiliser les fonctions accessibles dans le package `DEtools`, pour des résolutions "guidées".
3. `rsolve` : calcul de certaines suites définies par des relations de récurrence.