Modélisation et résolutions numérique et symbolique de problèmes via les logiciels Maple et MATLAB (MODEL)

Cours n°6 : Introduction à Maple

Stef Graillat

Université Pierre et Marie Curie (Paris 6)

Résumé du cours précédent

Méthode de Monte Carlo :

- Statistique de base : nombre aléatoire et génération
- Méthode de Monte Carlo et calcul d'intégrales
- Méthode de Monte Carlo et optimisation
- Méthode de Monte Carlo et comptage
- Introduction aux produits dérivés en finance
- Calcul du prix d'une option

Objectifs

1. Maîtriser les concepts et commandes de base de Maple
2. Connaître un logiciel de calcul formel

Plan du cours

1. Introduction à Maple

Bibliographie

- Maple, Règles et fonctions essentielles, N. Puech, Springer, 2009
- Maple Sugar : Une initiation progressive à Maple, G. Le Bris, Cassini, 1999
- Introduction to Maple, A. Heck, 3e édition, Springer, 2003
- Essential Maple 7 : An Introduction for Scientific Programmers, R. Corless, Springer, 2002
- Maple and Mathematica : A Problem Solving Approach for Mathematics, I. Shingareva et C. Lizarraga-Celaya, Springer, 2007
- Maple - Son bon usage en mathématiques, Ph. Dumas et X. Gourdon, Springer, 1997

Présentation de Maple

- Outil très utilisé dans l'enseignement, les centres de recherche et l'industrie
- Maple est un acronyme pour MAthematical PLEasure, et signifie également érable en anglais
- Maple est originaire de l'Université de Waterloo, Canada. C'est maintenant un produit de Waterloo Maple Software Corporation.

Présentation de Maple

Avantages :

- interactivité/interface pratique :
  - programmation rapide et simple, facilité pour tester et modifier
  - oubli du bas niveau ; on se consacre pleinement à l'aspect algorithmique
- riche en fonctionnalités
- prises en compte des notions mathématiques formelles
- retourne des résultats sous la forme d'objets mathématiques

Inconvénients :

- pas de notion de compilation, peu d'entrées/sorties
- sémantique du langage pas toujours claire

Maple : un système de calcul formel

Maple est un système de calcul formel (on dit aussi système de calcul symbolique)

À la différence d'un système de calcul numérique (comme MATLAB par exemple) qui ne peut manipuler que des expressions numériques, un système de calcul formel peut aussi manipuler des expressions symboliques, c'est-à-dire des expressions ne comportant pas de paramètres numériques.

Maple : un système de calcul formel

```
> dsolve({a*diff(y(x),x)+b*y(x)=0,y(0)=C},{y(x)}) ;
```

$$y(x) = Ce^{\left(-\frac{bx}{a}\right)}$$

La traduction de cette commande est la suivante :

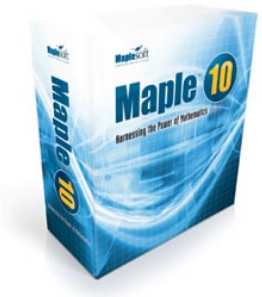
- `diff(y(x),x)` correspond à $y'(x)$;
- `dsolve({...},{y(x)})` est la commande pour résoudre l'équation différentielle (donnée entre accolades et ici symbolisée par ...) par rapport à la fonction inconnue y et qui a pour condition initiale $y(0) = C$.

Lancement de Maple

Le nom du fichier exécutable est

- `maple`, en mode texte
- `xmaple`, avec interface graphique
  plus légère avec l'option `-cw` (classic worksheet).

Sur les machines de l'ARI, il s'agit de Maple 10. Le programme est situe dans le répertoire `/usr/local/maple10/bin`



Quelques points de syntaxe

- L'instruction `quit` permet de quitter une session Maple
- L'aide sur une fonction, un type, . . . s'appelle par l'instruction ? suivi du mot concerné. Par exemple `?string`
- Une instruction se termine par un point-virgule ; (sauf l'aide et quit) ou un double point :
  Le point-virgule ; permet d'exécuter la commande et d'afficher le résultat. Si l'on veut simplement exécuter les commandes sans voir le résultat s'afficher, il faut remplacer le point-virgule par deux points :
- Touche `Entrée` $\longrightarrow$ Exécution
- Commentaires : après un dièse #
- La commande `restart` remet tout dans l'état initial (annule toutes les définitions). Cela devrait être la première instruction de tout programme Maple

Quelques points de syntaxe

Le symbole > représente le prompt. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un bloc d'instructions. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche `Entrée`). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper `Maj` puis `Entrée`

Il faut noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules.

Quelques points de syntaxe

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.

Le pourcentage % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.

Par exemple,

```
>1+1 ;
2
>sin(%*x) ;
sin(2x)
>cos(%%*x) ;
cos(2x)
```


Quelques points de syntaxe

On peut obtenir facilement des lettres grecques en tapant leur nom directement (alpha pour α, beta pour β et ainsi de suite).

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple.

Par exemple :

```
> 1+2; # Calcule la somme de 1 et de 2
```

Variables

- Une variable informatique est un nom dont la première occurrence apparaît dans le membre gauche d'une affectation.
- Une variable mathématique est un nom dont la première occurrence apparaît dans le membre droit d'une affectation.
- On n'affecte jamais une variable mathématique.
- On peut désaffecter une variable

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


Opérateurs

- Opérateurs arithmétiques : + (addition), - (soustraction), * (multiplication), / (division), ^ (puissance), `mod` (reste dans une division entière) ;
- Opérateurs de comparaison : < (inférieur), <= (inférieur ou égal), > (supérieur), >= (supérieur ou égal), = (égalité), <> (différent de) ;
- Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
- Opérateurs logiques : `and` (et logique), `or` (ou logique) ;
- Opérateur d'affectation : :=

Opérateurs

- Fonctions trigonométriques : `sin(x)`,`cos(x)`,`tan(x)`,`arcsin(x)`,`arcos(x)` et `arctan(x)` ;
- Fonctions trigonométriques hyperboliques : `sinh(x)`,`cosh(x)`,`tanh(x)`,`arcsinh(x)`, `arccosh(x)`,`arctanh(x)` ;
- Logarithme naturel : `ln(x)` ;
- Racine carrée : `sqrt(x)` ;
- Valeur absolue (et module complexe) : `abs(x)` ;
- Factorielle : `n!` ;
- Coefficients binomiaux C_n^k : `binomial(n,k)` ;
- Quotient de la division euclidienne de a par b : `iquo(a,b)` ;
- Reste de la division euclidienne de a par b : `irem(a,b)` ;
- Max et min d'un ensemble : `max(E)` et `min(E)`.

Alternative

```
if cond1 then
   instructions1
else
   instructions2
fi
```

Si plusieurs alternatives imbriquées

```
if cond1 then
   instructions1
elif cond2 then
   instructions2
else
   instructions3
fi
```

Boucle for

```
for var from expIni to expFin do
   instructions
od;
```

Changer le pas de la variable de boucle

```
for var from expIni to expFin by pas do
   instructions
od;
```


Boucle Tant que

```
while expr do
   instructions
od;
```

Fonctions et procédures

Opérateur flèche pour des fonctions dont le résultat se réduit à une instruction

Syntaxe : `param -> result`
où

- `param` est le paramètre (ou le n-uplet de paramètres)
- `result` est l'instruction donnant la valeur calculée en fonction des paramètres

Exemple :

```
x -> 3*x+1;
```

Nommage :

```
f := (x,y) -> 3*x*y+1;
```


Fonctions et procédures

Définition par le mot-clef `proc`

Syntaxe :
Le mot-clef `proc` permet de définir une fonction en respectant la syntaxe suivante :

```
proc( param )
   déclarations optionnelles
   corps de la fonction
end
```

Fonctions et procédures

Valeur de retour :

- Comportement par défaut : dernière valeur calculée
  Par défaut, la valeur de retour d'une fonction lors d'un appel est la dernière valeur calculée dans l'exécution du corps de la fonction, où les arguments effectifs remplacent les paramètres formels.
- Retour explicite de valeur par instruction `RETURN`
  L'instruction `RETURN(val)` arrête l'exécution de la fonction et fournit val comme valeur de retour de l'appel de fonction

Visibilité des variables

Variable globale : variable présente dans l'environnement d'exécution jusqu'à la fin de la session

- Déclaration explicite : Mot clef `global`
- Déclaration explicite d'une variable globale à l'intérieur d'une fonction : dans la partie "déclarations optionnelles", par

  ```
  global séquence_des_variables_globales
  ```

Variable locale : variable qui n'est accessible qu'à l'intérieur d'une fonction. Elle ne vit que le temps d'exécution de cette fonction.

L'espace mémoire qui lui est alloué est libéré à la fin de l'exécution de la fonction.

- Déclaration explicite : mot clef `local`
- Déclaration explicite par

  ```
  local séquence_variables_locales
  ```

Passage de paramètre

- Par valeur (comportement par défaut)
  le paramètre formel représente la valeur de l'expression qui est passée en argument à l'appel de la fonction.
- Par adresse (ou référence)
  le paramètre formel représente l'adresse de la variable qui sera passée en argument à l'appel de la fonction.

  Pour passer l'adresse d'une variable en argument de la fonction, il suffit que son nom soit passé en valeur. On passe donc l'argument désévalué (c-a-d encadrée par des quotes).

Exemple de fonction

Algorithme d'Euclide

```
euclide := proc(x,y,z)
  local a,b,r ;     # liste des variables locales la procédure
  a:=x; b:=y;
  while b<>0 do     # tant que b différent de 0 ...
     r:=rem(a,b,z); # reste division euclidienne de a par b
     a:=b;
     b:=r
  od;               # fin de la boucle while
  a                 # résultat renvoyé par la procédure
end:
```

Entiers, rationnels, flottants, complexes

- Maple peut gérer des entiers jusqu'à 500000 chiffres décimaux

```
> N := 100!;
N := 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

- Maple simplifie systématiquement les rationnels

```
> x := 1/21 + 3/35;
                        x := 2/15
```

- La précision des nombres flottants est contrôlée par la variable `Digits`. Par défaut, `Digits` vaut 10

```
> 1.2^20;
                       38.33759992

> Digits:=20: 1.2^20;
                   38.337599924474751222
```


Entiers, rationnels, flottants, complexes (suite)

- On peut évaluer une expression en flottant par la commande `evalf`

```
> Digits:=30 : x:=exp(Pi*sqrt(163)); evalf(x);
                                 1/2
                    x := exp(Pi 163   )

                                                        18
                    0.262537412640768744000000000024 10
```

- Le nombre complexe $i = \sqrt{-1}$ est représenté par `I` en Maple

```
> z:=(1+2*I)^2;
                       z := -3 + 4 I
```

- Pour mettre un complexe sous forme cartésienne, on utilise `evalc`

```
> z^(1/2);
                              1/2
                      (-3 + 4 I)
> evalc(%);
                       1 + 2 I
```

Entiers, rationnels, flottants, complexes (suite)

`iquo(m,n)`	quotient de la division euclidienne de m par n
`irem(m,n)`	reste de la division euclidienne de m par n
`ifactor(n)`	factorisation de l'entier n
`numer, denom`	numérateur et dénominateur d'une fraction
`trunc, round, frac, floor, ceil`	parties entières ou fractionnaire
`evalf`	évaluation numérique en flottant
`evalc`	évaluation d'un complexe

Fonctions définies par des expressions

- Les fonctions mathématiques sont représentés par des expressions

```
> f:=x^2*sin(y)+cos(x^4)*y^2;
                          2                  4  2
                 f := x  sin(y) + cos(x )y
```

- Pour dériver f par rapport à x

```
> g:=diff(f,x);
                                          4    3  2
                 g := 2 x sin(y) - 4 sin(x ) x  y
```

- Pour avoir la valeur de g en $(x, y) = (1, 1)$

```
> subs(x=1,y=1,g);
                          -2 sin(1)
```

Fonctions définies par des expressions (suite)

`subs(x=u,f)`	substitution de x par u dans l'expression f
`diff(f,x)`	dérivée de f par rapport à x
`int(f,x)`	primitive de f par rapport à x
`int(f,x=a..b)`	intégrale de f entre a et b
`series(f,x=a,n)`	développement de Taylor de f en x au voisinage de a à l'ordre n

Polynômes, fractions rationnelles

- On peut obtenir le coefficient devant $x^{10}y^{10}$ dans $(1+x+y)^{30}$ par

```
> p:=expand((1+x+y)^30):coeff(coeff(p,x,10),y,10);
                          5550996791340
```

- Les racines d'un polynôme sont obtenues avec la commande `solve`

```
> solve(x^2-x-1,x);
                                  1/2          1/2
                              5            5
                     1/2 + ----,  1/2 - ----
                              2            2
```

- En général, les équations ne sont pas solubles par radicaux

```
> p:=x^5+2*x+1: solve(p,x);
                                   5
                          RootOf(_Z  + 2 _Z + 1)
```

Polynômes, fractions rationnelles (suite)

- Pour obtenir une approximation numérique des racines réelles

```
> p:=x^5+2*x+1: fsolve(p,x);
                        -0.4863890359
```

- Pour obtenir une approximation numérique des racines réelles et complexes

```
> p:=x^5+2*x+1: fsolve(p,x,complex);
-0.7018735689 - 0.8796971979 I, -0.7018735689 +
0.8796971979 I, -0.4863890359, 0.9450680868 -
0.8545175144 I, 0.9450680868 + 0.8545175144 I
```


Polynômes, fractions rationnelles (suite)

- Fonctions rationnelles

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

- Pour obtenir la forme réduite

```
> f:=normal(f);
                              x (2 x + 1)
                    f := --------------------
                           2            2
                         (x  + x + 1) (x  - 1)
```

Polynômes, fractions rationnelles (suite)

`degree(p,x)`	degré du polynôme p en la variable x
`coeff(p,x,i)`	coefficient de x^i dans le polynôme p
`collect(p,x)`	regroupement des termes de p suivant les puissances de x
`expand(p)`	développement du polynôme p
`rem`	reste dans la division euclidienne
`quo`	quotient dans la division euclidienne
`factor`	factorisation dans le corps des coefficients
`gcd`	pcgd de deux polynômes
`solve`	procédure de résolution algébrique d'équations
`fsolve`	procédure de résolution numérique d'équations
`evala`	calcul sur des quantités algébriques
`normal`	réduction d'une fraction
`denom, numer`	dénominateur et numérateur d'une fraction
`convert(f,parfrac,x)`	décomposition de la fraction f en éléments simples

Algèbre linéaire

- Définition d'une matrice

```
> A := matrix(2,2,[1,2,3,4]); B := matrix(2,2,[4,2,1,4]):
                                  [1    2]
                            A := [       ]
                                  [3    4]
```

- Évaluation d'une matrice

```
> evalm(A+B);
                                 [5    4]
                                 [       ]
                                 [4    8]
```

- Accès aux éléments d'une matrice

```
> A[2,1];
                                 3
```

Algèbre linéaire (suite)

Les premières commandes font partie du package `linalg`. Les secondes demandent l'utilisation de la fonction `evalm`

`matrix`	définition d'une matrice
`add(A,B)`	somme des matrices A et B
`add(A,B,lambda,mu)`	combinaison linéaire $A + B$
`multiply(A,B)`	produit des matrices A et B
`rank(A)`	rang de la matrice A
`det(A)`	déterminant de A
`array(1..n,1..n,identity)`	matrice identité d'ordre n
`inverse(A)`	inverse de A
`transpose(A)`	matrice transposée de A
`nullspace(A)`	noyau de A
`charpoly(A,x)`	polynôme caractéristique de A
`eigenvals(A)`	valeurs propres de A
`eigenvects(A)`	vecteurs propres de A
`A+B`	somme des matrices A et B
`alpha*A`	produit de la matrice A par le scalaire
`A &* B`	produit des matrices A et B
`A^k`	A^k
`&*()`	matrice identité
`A^(-1)`	A^{-1}

Types de bases

- produit (`*`)
- somme (`+`)
- tableau (`array`)
- booléen (`boolean`)
- nombre complexe (`complex`)
- équation (`equation`)
- nombre flottant (`float`)
- nombre entier (`integer`)
- matrice (`matrix`)
- polynôme (`polynom`)
- intervalle (`range`)
- nombre rationnel (`rational`)
- chaîne de caractère (`string`)
- table (`table`)
- etc.

On obtient le type d'une variable avec la commande `whattype`.

Bibliothèques

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées packages. Pour pouvoir utiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package) ;
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg) ;
[BlockDiagonal; GramSchmidt; JordanBlock; LUdecomp; ...]
```

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les :

Conclusion

- Maple comporte plus de 2500 procédures
- Il ne faut pas hésiter à lire les pages d'aide