Introduction à Matlab

1 Introduction

Matlab *(MATrix LABoratory)* est un logiciel pour le calcul scientifique, orienté vers le vecteurs et les listes de données. Matlab est un langage interprété, chaque ligne d'un programme Matlab est lue, interprétée et exécutée.

Pour entrer dans Matlab cliquez sur Start/Programmes/Matlab sous l'environnement Windows, ou tapez matlab au prompt de la shell sous Unix. Matlab se présente avec un environnement interactif et un prompt (généralement >>) dans lequel on peut introduire des *commandes*. Par exemple, pour sortir de Matlab tapez la commande:

```
>> quit
```

2 Aide en ligne

Matlab offre une aide en ligne très complète. Tapez:

```
>> help commande
```

pour une explication de l'utilisation de la commande Matlab `commande`

```
>> lookfor sujet
```

pour une liste des commandes Matlab qui concernent le sujet `sujet`

```
>> helpwin
```

pour ouvrir un guide des commandes Matlab sur différents sujets.

3 Déclaration des variables

Matlab permet de créer et d'initialiser des variables. La déclaration des variables en Matlab suit les règles suivantes:

- toutes les variables sont des *matrices*
- pas de déclaration de type

`>> a=5`	*variable scalaire*	(1×1)
`>> b=[4  6]`	*vecteur ligne*	(1×2)
`>> c=[-5; 2]`	*vecteur colonne*	(2×1)
`>> d=[2 3; -1 7]`	*matrice carrée*	(2×2)

Dans ces exemples on a utilisé les opérateurs suivants:

- séparateur de ligne: point virgule ou return
- séparateur de colonne: virgule ou espace blanc

Il faut noter que Matlab montre les nombres avec quatre chiffres après la virgule, tandis que la représentation interne est faite avec 16 chiffres. Pour changer la façon de montrer les nombres en Matlab, on utilise la commande `format`. Par exemple, si avant de taper `>> pi` (en Matlab la variable `pi` est une approximation de π), nous écrivons

`>> format long`	nous obtiendrons	`3.1416`;
`>> format short`	nous obtiendrons	`3.14159265358979`;
`>> format long e`	nous obtiendrons	`3.141592653589793e+00`;
`>> format short e`	nous obtiendrons	`3.1416e+00`.

4 Workspace

Matlab permet de connaître plusieurs informations sur les variables déclarées. Tapez:

`>> who`	pour afficher toutes les variables.
`>> whos`	pour afficher toutes les variables avec indication sur leur taille.
`>> size(a)`	pour afficher les dimensions de la matrice `a`.
`>> clear var`	pour effacer la variable `var`.
`>> clear` (ou `>> clear all`)	pour effacer toutes les variables.

Pour ces opérations vous pouvez aussi utiliser l'interface graphique de Matlab.

5 Opérations fondamentales

Matlab peut effectuer plusieurs opérations entre matrices. Les opérations fondamentales peuvent être partagées en deux catégories.

Opérations matricielles

Les opérations matricielles usuelles sont définies par `+, -, *, /, ^`

`>> C = A + B`	est la somme matricielle, $C_{ij} = A_{ij} + B_{ij}$
`>> C = A * B`	est le produit matriciel, $C_{ij} = \sum_k A_{ik} B_{kj}$
`>> C = A / B`	est la division matricielle, $C = AB^{-1}$
`>> C = A^3`	est la troisième puissance matricielle (`C = A*A*A`)

Il faut remarquer que cettes opérations sont bien définies seulement si les matrices ont des *dimensions cohérentes*. Pour l'addition `A+B`, `A` et `B` doivent avoir la même taille; pour le produit `A*B`, le nombre des colonnes de `A` et le nombre des lignes de `B` doivent être égaux; les opérations `A/B` et `B^3` demandent une matrice `B` carrée. Par exemple:

```
>> A = [1 2 3; 4 5 6]; B = [7 8 9; 10 11 12]; C = [13 14; 15 16; 17 18];
>> A + B
ans =

     8    10    12
    14    16    18

>> A + C
```

```
??? Error using ==> +
Matrix dimensions must agree.

>> A * C
ans =

    94   100
   229   244

>> A * B
??? Error using ==> *
Inner matrix dimensions must agree.
```

Notez que Matlab renvoie un message d'erreur si le dimensions des matrices ne s'accordent pas avec l'opération commandée.

Opérations élément par élément

Pour exécuter des opérations entre matrices élément par élément il faut faire précéder l'opérateur d'un point. Les opérateurs élément par élément sont donc `.* ./ .^`

`>> C = A .* B`	est le produit élément par élément, $C_{ij} = A_{ij}B_{ij}$
`>> C = A ./ B`	est la division élément par élément, $C_{ij} = \frac{A_{ij}}{B_{ij}}$
`>> C = A.^3`	est la troisième puissance élément par élément, $C_{ij} = A_{ij}^3$

6 Manipulation des matrices

Après avoir défini la matrice `A` (par exemple la matrice carrée `A` de taille `n`), Matlab permet les opérations suivantes sur les *sous-matrices* de `A`.

Extraction de sous-matrices

`>> A(2,3)`	extraction de l' élément A_{23}
`>> A(:,5)`	extraction de la colonne $[A_{13}; \ldots ; A_{n3}]$
`>> A(1:4,3)`	extraction de la sous-colonne $[A_{13}; \ldots ; A_{43}]$
`>> A(1,:)`	extraction de la ligne $[A_{1j}, \ldots , A_{1n}]$
`>> diag(A)`	extraction de la diagonale $[A_{11}; \ldots ; A_{nn}]$

Construction de matrices particulières

Matlab permet, en plus, de définir des matrices particulières (une fois définis les entiers `n`, `m` et le vecteur `v`).

`>> A = eye(n)`	matrice identité $n \times n$
`>> A = diag(v)`	matrice diagonale avec `v` comme diagonale
`>> A = zeros(n,m)`	matrice de zéros avec `n` lignes et `m` colonnes
`>> A = ones(n,m)`	matrice de un avec `n` lignes et `m` colonnes
`>> A = rand(n,m)`	matrice aléatoire avec `n` lignes et `m` colonnes
`>> x = [but:pas:fin]`	vecteur ligne de points équidistribués avec pas `pas` entre `but` et `fin`

Fonctions matricielles

Soit `A` une matrice, Matlab permet d'effectuer directement les opérations suivantes.

`>> C = A'`	transposée de `A`, $C_{ij} = A_{ji}$
`>> C = inv(A)`	inverse de `A` (matrice carrée), $C = A^{-1}$
`>> d = det(A)`	déterminant de `A` (matrice carrée)
`>> r = rank(A)`	rang de `A`
`>> nrm = norm(A)`	norme 2 de `A`
`>> v = eig(A)`	valeurs propres de `A` (matrice carrée)

Soit `A` une matrice carrée de taille `n` et soit `b` un vecteur de taille `n`, alors le vecteur `x`, solution du système linéaire `Ax = b` est défini par

```
>> x = A^-1 *  b
```

Cette commande est théoriquement correcte, cependant si on est intéressé seulement à la solution `x` et pas à l'inverse `A^-1`, il est préférable utiliser un *backslash*

```
>> x = A \ b
```

Cette commande résout le système linéaire avec des algorithmes fiables et optimales.

7 Graphisme 2D

Matlab offre plusieurs possibilités pour tracer un graphe en 2D. On va en présenter deux: la commande `plot` et la commande `fplot`.
Avant d'expliquer ces deux commandes en détail, on souligne qu'avec `plot` on doit toujours utiliser des vecteurs, alors qu'avec `fplot` non.
On considère maintenant la fonction $f(x) = x^3 - 2\sin x + 1$ et on voit comment on peut tracer son graphe dans l'intervalle $[-1, 1]$, en utilisant les commandes `plot` et `fplot`.

Commande `plot`

Pour tracer le graphe de $f(x)$ il faut passer par les étapes suivantes :

- Définir la fonction $f(x)$:

  ```
  >> f = 'x.^3 - 2*sin(x) + 1';
  ```

  le mot `f` contient la définition de la fonction choisie (les guillemets font partie de la définition dans la syntaxe Matlab).

- Définir un *vecteur de points* dans l'intervalle donné:

  ```
  >> x = [-1:0.1:+1];
  ```

  on a défini un vecteur de 21 points equidistribués dans l'intervalle donné, avec un pas de 0.1.

- Évaluer la fonction `f` dans l'intervalle $[-1, 1]$, en utilisant la commande `eval`:

  ```
  >> y = eval(f);
  ```

  on note que dans la définition de la fonction f on a dû écrire `.^` (puissance élément par élément) et pas `^` (puissance matricielle), parce qu'il faut évaluer la fonction en chaque élément du *vecteur* `x`. C'est-à-dire:

  $$y(i) = x(i)^3 - 2\sin(x(i)) + 1, \quad \text{pour } i = 1, \ldots, 21;$$

  La variable `y` est donc un *vecteur* qui contient les valeurs de `f` pour chaque entrée du vecteur `x`.

- Tracer le graphe :

  ```
  >> plot(x,y); grid;
  ```

  cette commande ouvre une fenêtre avec le graphe. La commande `grid` dessine une grille de repère.

Le résultat de ces commandes est représenté dans la Figure 1.

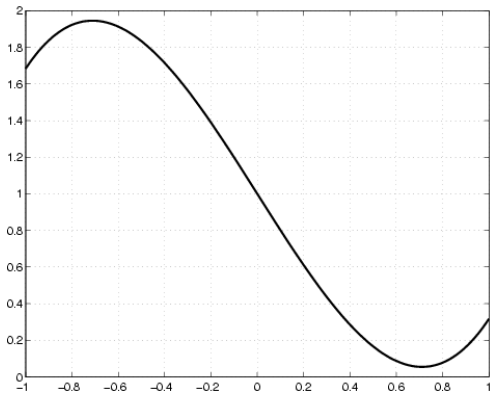

Figure 1: Graphe de la fonction $f(x) = x^3 - 2\sin(x) + 1$

Commande `fplot`

Pour tracer le graphe avec la commande `fplot` il faut passer par les étapes suivantes :

- Définir la fonction $f(x)$:

```
>> f = 'x^3 - 2*sin(x) + 1';
```

- Tracer le graphe :

```
>> fplot(f,[-1,1]); grid;
```

  La commande `fplot` demande la définition de la fonction `f` et de l'intervalle où il faut la tracer. La définition des vecteurs `x` et `y` *n'est plus nécessaire* et on note que dans la définition de `f` il suffit d'écrire `f='x^3 - ...'` au lieu de `f = x.^3 - ...'`, comme on avait fait pour utiliser `plot`. Le graphe obtenu est encore montré dans la Figure 1.

Exemple Tracer le graphe des fonctions $f_1(x) = \sin(2x) - 1 + x$ et $f_2(x) = x^3 - 2\sin(x)$ dans l'intervalle $[-1, 1]$ sur la même fenêtre.

En utilisant la commande `fplot`, il faut procéder de la façon suivante:

- Définir les fonctions:

```
>> f1 = 'sin(2*x) - 1 + x';
>> f2 = 'x^3 - 2*sin(x)';
```

- Tracer le graphe:

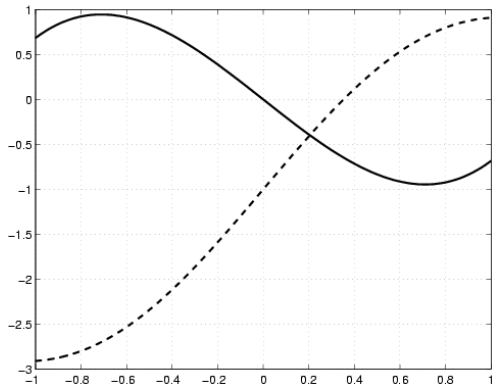

Figure 2: Résultat de l'exemple

```
>> fplot(f1,[-1,1]);  grid;  hold on;
```

La commande `hold on` affiche tous les prochains graphes sur la fenêtre qu'on vient d'ouvrir. Donc le graphe:

```
>> fplot(f2,[-1,1]);
```

est affiché sur la même fenêtre, comme on peut le voir dans la Figure 2.
La commande `hold off` arrête cet effet.

Pour plus de détails sur les fonctions `plot` et `fplot` (par exemple comment changer la couleur du graphe) tapez `help plot` ou `help fplot`.

8 Boucles de contrôle

Matlab offre, comme plusieurs autres langages, quelques boucles de contrôle. Cettes commandes sont spécialement utiles pour écrire des programmes Matlab.

Boucle `for`

Si l'on veut exécuter des instructions pour chaque valeur

$$i = m,\ m+1,\ \ldots,\ n$$

d'une certaine variable i entre deux bornes assignés m et n, on utilise `for`. Par exemple pour calculer le produit scalaire `ps` entre deux vecteurs `x` et `y` de taille `n` on utilise:

```
>> ps = 0;
>> for i = 1:n;
>>    ps = ps + x(i)*y(i);
>> end;
```

Cette boucle est équivalente au le produit matriciel entre le vecteur transposé de `x` et le vecteur `y`, en supposant que les facteurs soient deux vecteurs colonne:

```
>> ps = x'*y;
```

Boucle `while`

Si l'on veut exécuter plusieurs fois des instructions tandis qu'une expression logique est vraie, on utilise `while`. Par exemple, le même calcul qu'on a considéré avec la boucle `for` peut être exécuté avec

```
>> ps = 0;
>> i = 0;
>> while (i < n);
>>    i = i + 1;
>>    ps = ps + x(i)*y(i);
>> end;
```

Instruction conditionelle `if`

Si l'on veut exécuter des instructions seulement si une expression logique est vraie, on utilise `if`. Par exemple, si l'on veut calculer la racine carrée d'une variable scalaire `r` seulement si `r` n'est pas négative:

```
>> if (r >= 0)
>>   racine = sqrt(r);
>> end;
```

On a plusieurs *opérateurs logiques* à disposition:

Opérateur	Action logique
`&`	*and*
`|`	*or*
`~`	*not*
`==`	*equal to*

Taper par exemple `help &` pour une explication de la liste des opérateurs.

9 Scripts et fonctions

Matlab permet d'écrire des programmes. Les outils principaux sont les script files et les fonctions.

Script files

Un script file est une suite de commandes Matlab. Les noms des script files doivent avoir l'extension "`.m`". Pour exécuter un script file tapez son nom, sans extension, dans le prompt Matlab.

Fonctions

Une fonction Matlab est une suite de commandes qui nécessite un input pour être exécutée et qui renvoie un output. La déclaration des fonctions en Matlab suit les règles suivantes.

- Une fonction est contenue dans un fichier "`.m`" avec le même nom que la fonction.
- Le fichier qui définit une fonction doit commencer par:
  `function` [*output arguments*] = `nom_fonction(`*input arguments*`)`
  Par exemple, l'en-tête d'une fonction pour la méthode de dichotomie pourrait être:

```
function [zero,res,niter]=bisection(fun,a,b,tol,nmax,varargin)
%BISECTION Find function zeros.
%   ZERO=BISECTION(FUN,A,B,TOL,NMAX) tries to find a zero ZERO
%   of the continuous function FUN in the interval [A,B]
%   using the bisection method.
            .
            .
            .  % istructions
            .
 zero = ...; res = ...; niter = ...;
            .
            .  % istructions
            .
            .

return        % fin du corps de la fonction
```

- Les lignes de commentaires qui éventuellement suivent cet en-tête constituent le help de la fonction. En Matlab les lignes de commentaire sont introduites par `%`.
- Toutes les variables internes à la fonction sont locales.

Par exemple, supposons qu'on aie écrit sur le fichier `my_function.m` le lignes suivantes:

```
function f = my_function(x);
f = x.^3 - 2*sin(x) + 1;
return;
```

Il est possible d'utiliser la fonction `my_function.m` de la même façon que les autres commandes Matlab. Donc les commandes

```
>> x = 0;
>> y = x.^3 - 2*sin(x) + 1
```

et

```
>> x = 0;
>> y = my_function(x)
```

sont équivalentes et renvoient le même résultat:

```
y =
     1
```

Fonctions de plusieurs variables

Pour définir des fonctions dépendantes d'un ou plusieurs paramètres nous pouvons utiliser les fonctions *inline*. Pour définir une fonction *inline* `f` qui dépend de la variable `x` on écrit:

```
>> f = inline('log(x) - 1','x');
```

Si `f` dépend de de plusieurs variables, nous écrirons par example

```
>> f = inline('log(x) - p','x','p');
```

Pour évaluer une fonction `f` définie grâce à *inline*, il faut utiliser la commande `feval` en donnant les valeurs des paramètres comme montré dans l'exemple suivant:

```
>> f = inline('x.^3 + a.*x','x','a');
>> xval = 1; aval = 2;
>> fval = feval(f, xval, yval)
val =
     3
```

10 Polynômes en Matlab

Considérons un polynôme de degré n:

$$f(x) = a_n x^n \ + \ + a_{n-1} x^{n-1} \ + \ \ldots \ + a_1 x \ + \ a_0.$$

Matlab répresente les polynômes de degré n sous la forme d'un vecteur $p = [a_n, a_{n-1}, \ldots, a_0]$ de taille $n+1$ qui contient les coefficients en ordre décroissante par rapport au degré associé. Par exemple, le vecteur associé au polynôme $f(x) = 3x^3 - 4x^2 + x$ est

```
>> p = [3, -4, 1, 0];
```

Mois	Heure de son réveil (moyenne)
1	8.0
2	7.5
3	7.5
4	7.5
5	7.6
6	7.8

Mois	Heure de son réveil (moyenne)
7	8.0
8	10.0
9	8.0
10	7.5
11	7.5
12	7.7

Table 1: Heure de son réveil (de 0 à 24 avec décimes d'heure)

Pour évaluer ce polynôme on utilise la commande `polyval`:

```
>> x = [0:.1:1];
>> y = polyval(p, x);
```

Dans ce cas, les éléments du vecteur `y` sont les valeurs du polynôme calculées pour chaque élément du vecteur `x`.
Pour obtenir le vecteur associé au polynôme de degré `n` approximant un jeu de données on utilise `polyfit`. Si la taille des données est plus grande de `n+1` on a le polynôme approximant au sens des moindres carrés; si elle est égal à `n+1` on a le polynôme interpolant. Par exemple, si on veut calculer le polynôme de degré 8 qui approxime le données de la Table 10 (au sens des moindre carrées) et en tracer le graphe, il faut utiliser les commandes qui suivent;

```
x = [1 2 3 4 5 6 7 8 9 10 11 12];
y = [8 7.5 7.5 7.5 7.6 7.8 8 10 8 7.5 7.5 7.7];
p = polyfit(x,y,8);   % Calculons le polynome interpolant (degre=8)
plot(x, y, 'or');
axis([1 12 6 12])
hold on;
f = inline('polyval(p,x)','x','p');
fplot(f, [1 12], [], [], [], p);  % Trace le graphe de f(x,p)
                                  % pour x entre 0 et 12, en donnant
                                  % p comme parametre de f.
```

Il faut noter que `f` est déclarée comme fonction *inline* avec les arguments `x` et `p` (le vecteur associé au polynôme interpolant); la commande `fplot` (voir `help fplot`) accepte le paramètre `p` à donner à `f`.

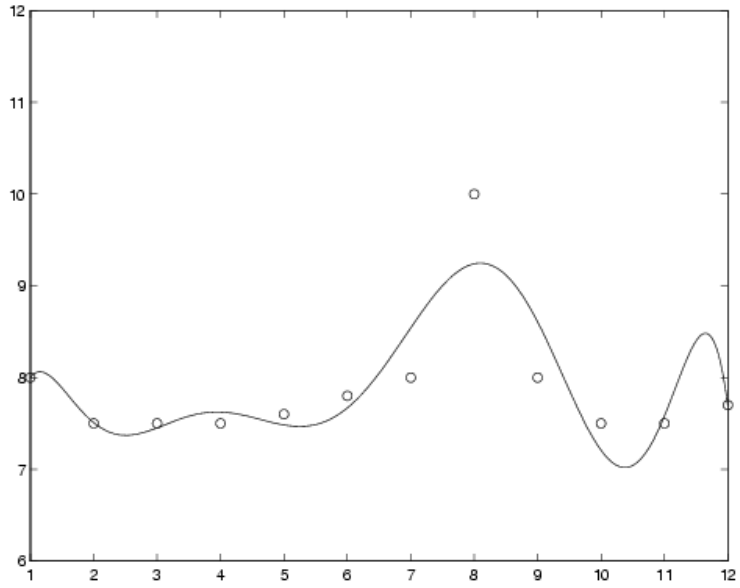

Figure 3: Données et polynôme d'interpolation