CMI, Université de Provence **Licence de Mathématiques**
Année 03-04

Initiation à *Matlab*

Florence Hubert

Table des matières

1 *Matlab* ou *Maple* ?

Matlab est un programme de calcul numérique, tandis que *Maple* est un programme de calcul formel et symbolique qui permet d'obtenir des solutions algébriques ou analytiques de nombreux problèmes mathématiques. Autrement dit,

- si on a besoin d'inverser une matrice comportant des inconnues ou des paramètres, si on doit calculer le discriminant d'un polynôme dont les coefficients sont des paramètres, si on veut obtenir l'expression explicite d'une primitive d'une fonction, on utilisera *Maple* .
- Si au contraire, on doit inverser une matrice composée de nombres, si on veut résoudre une équation aux dérivées partielles, utiliser un schéma numérique d'intégration ou de recherche de minimum, en résumé si la réponse est un nombre décimal et non une expression formelle, on utilisera *Matlab* .

Des *boites à outils* respectivement *Matlab* ou *Maple* , permettent d'utiliser conjointement les capacités des deux logiciels. La *boite à outil Maple* n'est pas disponible dans la version de *Matlab* proposée à l'agrégation et l'experiance montre néanmoins que *Matlab* reste plus efficace pour tous ce qui est cacul numérique conséquent.

2 Quelques rappels de *Matlab*

Matlab est l'abrévation en anglais de "matrix laboratory", tous les objets en jeu sont des matrices, y compris les scalaires (matrices 1×1). Nous proposons ici des rappels non exhaustifs des commandes utiles en calcul scientifique. Nous renvoyons le lecteur aux livres suivants pour des compléments [**?**],[**?**],[**?**]. Les programmes qui suivent ont été effectués avec *Matlab* 6.

2.1 Première utilisation de *Matlab* , l'aide en ligne

Une fois *Matlab* lancé, les instructions de *Matlab* doivent suivre le sigle `>>`.

Par exemple :

```
>>1+1
ans =
       2
>>pi
ans =
       3.1416
>>eps
ans =
       2.2204e-16
>>i
ans =
       0+ 1.0000i
>>j
ans =
       0+ 1.0000i
```

```
>>t=1+1
t =
       2
>>t=1+1;
>>t
t=
       2
>>u=sin(t)
u =
       0.9093
>>v=exp(u)
v =
       2.4826
>>format long
>>v
v =
       2.48257772801500
>>format short
>>v
v =
       2.4826
>>who
Your variables are:
ans       t         u         v
>>whos
Name      Size         Bytes  Class
ans       1x1              8  double array
t         1x1              8  double array
u         1x1              8  double array
v         1x1              8  double array
Grand total is 4 elements using 32 bytes
leaving 14998496 bytes of memory free.
>>exit
```

On remarque que :

- par défaut, tout calcul est affecté dans la variable *ans*
- les variables *pi, eps, i, j* sont déja affectées, les variables *i, j* peuvent être réaffectées
- l'affectation se fait grâce au sigle =
- le résultat d'une affectation est imprimé sauf si cette affectation est suivie du sigle *;*
- la commande *format* permet de modifier l'affichage du format des différentes variables
- les commandes *who, whos* permettent de lister l'ensemble des variables utilisées
- la commande *clear* efface le contenu de tous les variables utilisées

Attention : il très fortement déconseillé d'utiliser des noms de variables déja utilisées par *Matlab* .

Par exemple, pour sortir de *Matlab* , il suffit d'utiliser les commandes `quit` ou `exit`. Lorque l'on veut interrompre un programme, la commande `CTRL-c` permet de récupérer la main.

Une façon efficace de découvrir *Matlab* d'utiliser **son aide en ligne**.

- `help` : "help" tout seul donne la liste des aides générales possibles.
- `helpwin` : ouvre une fenêtre et donne accès à une aide détaillée, en gros à tout le manuel de *Matlab* .
- `help` *nom de commande* : exemple, `help plot` indique la syntaxe des graphes en 2D.
- `lookfor` *nom de commande* : exemple, `lookfor plot` donne une liste de toutes les commandes qui ont un rapport avec `plot`.
- `demo` : lance une démo générale de *Matlab* .
- `help demos` : donne une liste des demos existantes.

Attention, il n'est pas possible de sauvegarder la *fenêtre de commande* de *Matlab* . Pour sauvegarder un travail, il est conseillé de travailler dans une fenêtre d'édition. On peut utiliser l'éditeur de *Matlab* et ouvrir ou créer un fichier en cliquant sur `fichier` puis `open` ou `new`. Tout autre éditeur de texte (Emacs, Nedit,...) convient également. Pour exécuter les commandes d'un fichier qui s'appelle `toto.m`, il suffit alors de taper `toto` dans la fenêtre de commandes de *Matlab*.

2.2 Les matrices dans *Matlab*

2.2.1 Création de matrices

- $A = [a_{11} \dots a_{1m}; \dots; a_{n1} \dots a_{nm}]$
  Exemple :

```
>>A=[1 2 3 4;2 3 4 1;3 4 1 2;4 1 2 3]

A =

     1     2     3     4
     2     3     4     1
     3     4     1     2
     4     1     2     3
```

- quelques matrices prédéfinies

```
>>B=zeros(2,3)
B =
     0     0     0
     0     0     0
```

```
>>C=ones(3,2)
C =
     1     1
     1     1
     1     1
>>Id=eye(2,3)
Id =
     1     0     0
     0     1     0
>>D=rand(4,4)
D =
     0.8913    0.8214    0.9218    0.9355
     0.7621    0.4447    0.7382    0.9169
     0.4565    0.6154    0.1763    0.4103
     0.0185    0.7919    0.4057    0.8936
>>E=randn(3,4)
E =
    -0.4326    0.2877    1.1892    0.1746
    -1.6656   -1.1465   -0.0376   -0.1867
     0.1253    1.1909    0.3273    0.7258
```

zeros(n,m) construit une matrice nulle de taille $n \times m$,

ones(n,m) construit une matrice dont tous les éléments sont égaux à un et de taille $n \times m$,

eye(n,m) construit une matrice de taille $n \times m$ dont tous les éléments diagonaux sont égaux à un,

rand(n,m) construit une matrice de taille $n \times m$ dont tous les éléments sont choisis aléatoirement avec la loi uniforme sur $[0, 1]$,

randn(n,m) construit une matrice de taille $n \times m$ dont tous les éléments sont choisis aléatoirement avec la loi normale.

- Construction élément par élément

  Exemple :

```
>>for i=1:3,for j=1:4, F(i,j)=i+(j-1)*3;end;end;
>>F
F =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

- la commande *size* permet d'obtenir la taille de la matrice :

```
>>size(A)
```

```
ans =
       4     4
>>size(F)
ans =
       3     4
>>size(F,1)
ans =
       3
>>size(F,2)
ans =
       4
```

2.2.2 Matrices et vecteurs

Les vecteurs dans *Matlab* sont écrits en ligne :

```
>>for i=1:3, V(i)=i;end;
>>V
V =
       1     2     3
>>size(V)
ans =
       1     3
```

Une matrice $n \times m$ est stockée colonnes après colonnes dans *Matlab* comme un vecteur de taille $n * m$:

```
>>for i=1:3,for j=1:4, F(i,j)=i+(j-1)*3;end;end;
>>F
F =
       1     4     7    10
       2     5     8    11
       3     6     9    12
>>F(4)
ans =
       4
```

De même, on peut affecter directement les éléments d'une matrice, après avoir défini sa taille :

```
>>F=zeros(3,4);
>>for i=1:12,F(i)=i;end
>>F
F =
       1     4     7    10
       2     5     8    11
       3     6     9    12
```

2.2.3 Les premières opérations sur les matrices

- Transposition :

```
>>F'
ans =
        1     2     3
        4     5     6
        7     8     9
       10    11    12
```

- La diagonale, la trace, le rang, le déterminant

```
>>diag(A)
ans =
        1
        3
        1
        3
 >>trace(A)
 ans =
        8
 >>rank(A)
 ans =
        4
 >>det(A)
 ans =
        160
```

- Addition ou soustraction

```
>>F+E
ans =
        0.5674    4.2877    8.1892   10.1746
        0.3344    3.8535    7.9624   10.8133
        3.1253    7.1909    9.3273   12.7258
```

- Multiplication, la puissance $n^{ième}$

```
>>F*A
ans =
        70    52    46    52
        80    62    56    62
        90    72    66    72
```

```
>>A^3
ans =
        240   244   256   260
        244   256   260   240
        256   260   240   244
        260   240   244   256
```

- L'inverse

```
>>inv(A)
ans =
       -0.2250    0.0250    0.0250    0.2750
        0.0250    0.0250    0.2750   -0.2250
        0.0250    0.2750   -0.2250    0.0250
        0.2750   -0.2250    0.0250    0.0250
```

2.2.4 Matrices et tableaux

Une matrice peut être considérée élément par élément, c'est alors un tableau. Les opérations élémentaires sur les tableaux sont alors les suivantes :

- Addition ou soustraction : pas de changements
- Multiplication, la puissance $n^{ième}$

```
>>D.*A
ans =
        0.8913    1.6428    2.7654    3.7419
        1.5242    1.3341    2.9528    0.9169
        1.3694    2.4617    0.1763    0.8205
        0.0740    0.7919    0.8114    2.6809
 >>A.^3
 ans =
          1     8    27    64
          8    27    64     1
         27    64     1     8
         64     1     8    27
```

Remarque 2.1
Si les tailles des tableaux ou matrices ne sont pas adaptées, *Matlab* donne des messages d'erreurs de la forme :

```
>>F.*A
??? Error using ==> .*
Matrix dimensions must agree.
```

2.3 Le calcul vectoriel

Il est souvent intéressant de travailler sur les lignes ou les colonnes d'une matrice avec *Matlab* .

- `A(i, :)` désigne la $i^{ième}$ ligne de la matrice A
- `A(2:3, :)` désigne la sous matrice formées des $2^{ième}$ et $3^{ième}$ lignes de la matrice A
- `A(:, j)` désigne la $j^{ième}$ colonne de la matrice A
- `A(:, 1:2:m)` désigne la sous matrice formée des colonnes impaires de la matrice A

Remarque 2.2

- `xmin : dx : xmax` désigne l'ensemble des points de la forme xmin$+i*dx$, $i \in \mathbb{N}$ compris entre xmin et xmax, l'extrémité étant xmin étant toujours incluse. Par exemple, `1:3:7` ou `1:3:9` désignent le vecteur `1,4,7`
- `xmin : xmax` idem avec par défaut dx=1.

2.4 Programmation en *Matlab*

2.4.1 Les fonctions élémentaires

Un certain nombre de fonctions élémentaires sont prédéfinies par `exp, sin, cos, abs,...` La plupart de ces fonctions agissent également sur des matrices ou vecteurs :

```
>>x=[0:0.5:pi]
x =
0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
>>sin(x)
ans =
      0   0.4794   0.8415   0.9975   0.9093   0.5985   0.1411
```

2.4.2 Créer une fonction ou une procédure

On va créer un fichier "texte" que l'on appelera *nomdefonction*`.m` ou *nomdeprocedure*`.m`. Vous pouvez utiliser l'éditeur de *Matlab* (cliquer en haut à droite sur file -¿new ou file-¿open) mais aussi, n'importe quel éditeur de texte. **emacs, nedit, Text editor, ...**

On va décrire la syntaxe de ces fichiers à l'aide de deux exemples.

⋆ **Les fonctions**

On va créer une fonction $f : (x, l) \to \frac{1}{x^2+l}$. Le premier exemple est édité sous le nom `f1.m`.

```
function [y]=f1(x,l)
y=1/(l+x^2);
% on peut agrémenter tous ces fichiers par des commentaires
% qui seront précédés du sigle %
```

Une autre possibilité permet de travailler avec des variables x qui seront des tableaux (fichier *f2.m*):

```
function [y]=f2(x,l)
y=1./(l+x.^2);
% Les opérateurs agissant sur les éléments des tableaux
% sont les opérateurs classiques précédés d'un point
```

Remarque 2.3
On peut déclarer une fonction à l'intérieur d'une fonction de deux façons différentes : soit à l'aide de la commande ***function*** précédemment vue soit à l'aide de la commande ***inline*** comme le montre les exemples qui suivent.

```
function [y]=f3(x,l)
f=inline('1./z')
y=f(l+x.^2);
```

⋆ Les procédures

On se donne une discrétisation de l'intervalle $[0, 1]$ de pas h, et on crée une procédure qui calcule les vecteurs X et F de composantes $X(i) = (i-1) * h, F(i) = f((i-1) * h, l)$.

Le premier programme est sauvegardé sous le nom ***essai1.m*** et fait appel à la fonction ***f1.m*** qui doit se trouver dans le même répertoire.

```
%essai1.m
l=1;
h=0.1;
N=floor(1/h); % partie entière
for i=1:N+1, X(i)=(i-1)*h;end
for i=1:N+1, F(i)=f1((i-1)*h,l);end
X % affichage de X
F % affichage de F
```

Le deuxième programme est sauvegardé sous le nom ***essai2.m***, fait appel à la fonction ***f2.m*** qui doit se trouver dans le même répertoire et utilise le calcul vectoriel, la variable h est laissée comme paramètre à choisir ultérieurement.

```
%essai2.m
l=1;
X=[0:h:1];
F=f2(X,l);
X
F
```

Remarque 2.4

- les commentaires situés juste après la première ligne d'une fonction *lambda.m* ou qui sont en tête d'une procédure *lambda.m* sont accessibles dans *Matlab* par la commande *help lambda*.
- On peut accéder directement au listing d'une fonction ou procédure *lambda.m* en utilisant la commande *type lambda*.
- On ne peut pas déclarer une fonction à l'intérieur d'une procédure à l'aide de la syntaxe *function*, il faut utiliser la syntaxe *inline* vue précèdemment.

```
        %essai3.m
        l=1;
        X=[0:h:1];
f=inline('1./(l+x.^2)','x','l');
        F=f(X,l);
        X
        F
```

⋆ **Exécuter une procédure par *Matlab***

Il suffit de taper le nom du fichier après avoir défini les paramètres qu'il utilise.
Exemple :

```
>>essai1
X =
    Columns 1 through 7
    0   0.1000   0.2000   0.3000   0.4000   0.5000   0.6000
    Columns 8 through 11
    0.7000   0.8000   0.9000   1.0000
    F =
    Columns 1 through 7
    1.0000   0.9901   0.9615   0.9174   0.8621   0.8000   0.7353
    Columns 8 through 11
    0.6711   0.6098   0.5525   0.5000
```

ou encore, comme h est laissé comme paramètre dans le programme *essai2.m* :

```
>>h=0.2;
>>essai2
X =
    0    0.2000    0.4000    0.6000    0.8000    1.0000
F =
    1.0000    0.9615    0.8621    0.7353    0.6098   0.5000
```

Examinons maintenant la syntaxe des boucles qui permettrons de créer les fonctions et les procédures.

⋆ **Syntaxe des opérations logiques**

- == égalité
- >= supérieur ou égal
- ~= différent
- *&* "et" logique
- */* "ou" logique
- *not* la négation

⋆ **Syntaxe des boucles**

- *for* i=imin : imax,
  < commandes >;
  end
  ou
  for i=imin : h : imax,
  < commandes >;
  end
  Exemples :

```
>>for i=1:2:10,X(i)=i;end
>>X
X =
    1     0     3     0     5     0     7     0     9
>>for i=10:-1:1,Y(i)=10-i;end
>>Y
Y =
    9     8     7     6     5     4     3     2     1     0
```

- *while* < test >,
  < commandes >
  end
- *if* < test >,
  < commandes >
  elseif < test >,
  < commandes >
  else < commandes >,
  end

2.4.3 Conseils de programmation

Pour gagner du temps en *Matlab* , il faut éviter le plus possible les boucles dans les programmes.
Illustrons ce fait sur un exemple :

```
%test.m
a=1000*ones(20);z=zeros(20);
tic % initialise le temps
for i=1:20, for j=1:20, z(i,j)=a(i,j);end;end;
toc % affiche le temps écoulé
tic % initialise le temps
z=a;
toc % affiche le temps écoulé

>>test
       elapsed_time =
                     0.0501
       elapsed_time =
                     9.1100e-04
```

2.4.4 Enregistrer ou lire des données créées par *Matlab*

- *save a* :

  sauve toutes les variables existantes dans le fichier a.mat

- *save a 'F' 'X'*

  sauve les variables F et X dans le fichier a.mat

- *nomdufichier=['ah=' num2str(h)'.mat'];save(nomdufichier,'F','X')*

  sauve les variables F et X dans le fichier nomdufichier.mat, la commande ***num2str*** permet de transformer un nombre en une chaîne de caractères.

- **load a** ou **load nomdufichier**

  charge les données qui sont dans chacun de ses fichiers (*Matlab* affecte à nouveau automatiquement les variables F et X).

- *save resultat.dat -ascii -double*

  sauve toutes les variables existantes sous forme ascii dans le fichier resultat.dat

- *clear*

  efface toutes les variables et données existantes.

- **load resultat.dat**

  charge les données qui sont stockées dans resultat.dat (*Matlab* ne connait plus dans ce cas les variables F et X).

2.5 Graphisme dans *Matlab*

2.5.1 Graphe de points en 2D

Soient Y et Z deux vecteurs de taille n.

- Définir une courbe, première possibilité :

  `plot(Z)` relie les points de coordonnées $(i, Z(i))$

- Définir une courbe, deuxième possibilité :

  `plot(Y,Z)` relie les points de coordonnées $(Y(i), Z(i))$

- Définir deux ou plusieurs courbes, première possibilité :

  `plot(Y,Z,Y,cos(Z))`, graphes des points de coordonnées $(Y(i), Z(i))$ et de coordonnées $(Y(i), \cos Z(i))$

- Définir deux ou plusieurs courbes, deuxième possibilité :

  `W=[Z',cos(Z)']; plot(Y,W)`

- Définir le style des courbes :

  `plot(Y,Z,'-r',Y,cos(Z),':')` on peut pour chacune des courbes imposer un style de ligne, un style de point, une couleur.

  - Quelques style de lignes :
    `-` ligne pleine, `:` pointillés, `none` pas de ligne entre les points, ...
  - Quelques couleurs :
    `y` jaune, `r` rouge, `b` bleu, `g` vert, `w` blanc, `k` noir, ...
  - Quelques styles de points :
    `+`,`o`,`.` , `square`, `diamond`, `none`, ...

- Contrôler les axes des courbes : la taille des axes est géré automatiquement par *Matlab* . Pour imposer la taille des axes, il suffit d'utiliser :

  `axis([xmin,xmax,ymin,ymax])`.

  Pour obtenir la même échelle sur les deux axes, taper `axis square`.

- Rajouter des annotations sur le graphe :

  - `title` ajoute un titre au graphe,
  - `xlabel` ajoute une légende à l'axe horizontal du graphe,
  - `ylabel` ajoute une légende à l'axe vertical du graphe,
  - `text` ajoute un texte à l'emplacement précisé.
  - `legend` ajoute des légendes aux différentes courbes.

Pour plus de précision utiliser la commande `help plot`.
Exemple :

```
>> Y=[0:0.05:1];Z1=sin(2*pi*Y);Z2=cos(2*pi*Y);
>> plot(Y,Z1,':b',Y,Z2,'+k');
>> title('Exemple de courbes');
>> xlabel('Y');ylabel('Z');
>> legend('sin','cos');
```

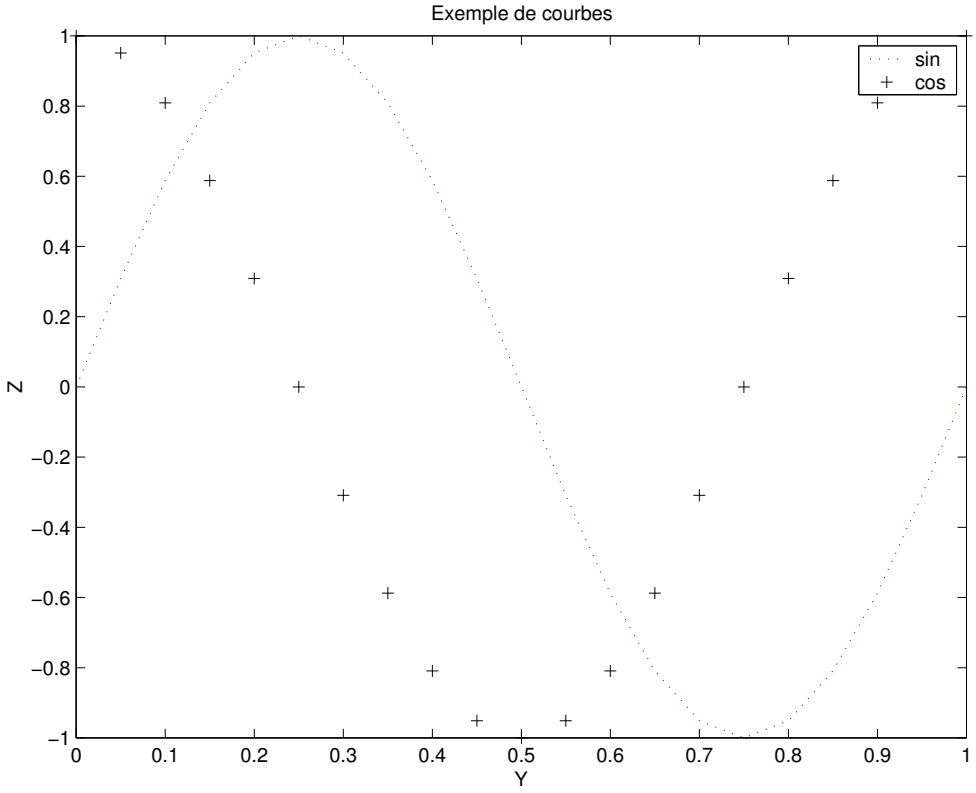

Figure 1:

2.5.2 Graphe d'une fonction

Le graphe d'une fonction sur un intervalle I se ramène au graphe d'un vecteur, en choisissant une discrétisation de cet intervalle I.

Exemple :

```
>>x=[0:0.5:pi];
>>plot(sin(x));
>>plot(x,sin(x));
>>plot(f2(x,1));
```

2.5.3 Graphe d'une courbe dans l'espace

Soient X, Y et Z trois vecteurs de taille n.
`plot3(X,Y,Z)` graphe des points $(X(i), Y(i), Z(i))$

2.5.4 Graphe d'une surface

Soient X et Y deux vecteurs de tailles n et m, et Z une matrice de taille $n \times m$ (resp. soit X, Y et Z trois matrices de taille $n \times m$).

La génération des surfaces se fait à l'aide des deux commandes `surf` et `mesh`.

- `surf(Z)` trace une surface "pleine" passant par les points $(i, j, Z(i, j))$
- `surf(X,Y,Z)` trace une surface "pleine" passant par les points
- `mesh(Z)` trace un maillage ("fil de fer") passant par les points $(i, j, Z(i, j))$
- `mesh(X,Y,Z)` trace un maillage ("fil de fer") passant par les points $(X(i), Y(j), Z(i, j))$ (resp. par les points $(X(i, j), Y(i, j), Z(i, j))$)
- `contour(Z)` courbes de niveaux de la surface passant par les points $(i, j, Z(i, j))$
- autres possibilités :`surfc`, `meshc`, `meshz`, `pcolor`, `surfl`,... Voir l'aide en ligne.

Remarque 2.5
La gestion des axes et des annotations graphiques est identique à celle du graphisme 2-D.

2.5.5 Graphe d'une fonction de deux variables réelles ou d'une surface paramétrée

De même que pour le graphisme 2D, on se ramène au graphe d'une courbe ou de matrices :

- Courbe dans l'espace : $t \to (x(t), y(t), z(t))$ pour $t \in I$. On se donne une discrétisation de I : T vecteur de taille n et on se ramène au graphe des vecteurs $(x(T), y(T), z(T))$.
- Surface dans l'espace donnée par $(x, y) \to z(x, y)$ pour $x \in I, y \in J$. On se donne une discrétisation de I et de J : I_x vecteur de taille n, J_y vecteur de taille m et on se ramène au graphe de la matrice $z(I_x, J_y)$.
- Surface paramétrée donnée par $(u, v) \to (x(u, v), y(u, v), z(u, v))$ pour $u \in I, v \in J$. On se donne une discrétisation de I et de J : I_u vecteur de taille n, J_v vecteur de taille m et on se ramène au graphe des matrices $(x(I_u, J_v), y(I_u, J_v), z(I_u, J_v))$.

2.5.6 Gestion des figures

⋆ Ouvrir une figure

Toute commande graphique (`plot`, `surf`,...) ouvre une fenêtre graphique si aucune n'est encore ouverte. On peut également ouvrir une fenêtre graphique à l'aide de la commande `figure`.

⋆ Ouvrir plusieurs figures

Toute nouvelle figure doit être ouverte à l'aide de la commande `figure`. Les commandes graphiques sont effectuées dans la dernière fenêtre activée.

⋆ **La commande** *hold*

Toute commande graphique (*plot*, *surf*,...) efface la commande précédente. Pour conserver plusieurs courbes ou surfaces sur la même figure, on peut utiliser la commande *hold on*.

⋆ **Créer plusieurs graphes sur une figure**

- Deux graphes côte à côte

```
>>subplot(1,2,1)
>>commande graphique 1
>>subplot(1,2,2)
>>commande graphique 2
```

- Deux graphes l'un au dessous de l'autre

```
>>subplot(2,1,1)
>>commande graphique 1
>>subplot(2,1,2)
>>commande graphique 2
```

- Quatre graphes

```
>>subplot(2,2,1)
>>commande graphique 1 (graphe situé en haut à gauche)
>>subplot(2,2,2)
>>commande graphique 2 (graphe situé en haut à droite)
>>subplot(2,2,3)
>>commande graphique 3 (graphe situé en bas à gauche)
>>subplot(2,2,4)
>>commande graphique 4 (graphe situé en bas à droite)
```

⋆ **Effacer tout le contenu d'une figure**

Utiliser la commande *clf*.

⋆ **Impression et sauvegarde postcript d'une figure**

La sortie postcript ou imprimante s'effectue directement à partir de la fenêtre graphique (file – print) ou en cliquant sur le bouton de l'imprimante. Sinon, on peut utiliser la commande **print**. Pour plus de détails **help print**.

2.5.7 Animation de graphisme

On peut créer une animation de deux façons différentes :

- soit en sauvegardant un certain nombre de figures et en les faisant défiler ultérieurement comme un film,
- soit en effaçant et redessinant au fur et à mesure les objects à l'écran.

⋆ **Créer un film à l'aide des commandes *movie* et *getframe***

Exemple :

```
>>for i=1:5
>>plot(sin(i*pi*[0:0.025:2]));
>>M(:,i)=getframe;
>>end;
>>movie(M);
```

⋆ **Créer une animation à l'aide des commandes *hold on* et *pause***

- Lorsque deux commandes ***plot*** se succédent, la deuxième efface sur la figure la première. La commande ***hold on*** permet de garder dans la figure le résultat des deux commandes ***plot***.
- La commande ***pause*** permet d'effectuer un arrêt dans un programme, en particulier entre deux commandes ***plot***. Pour relancer le programmer, il suffit d'appuyer sur n'importe quelle touche du clavier.
- La commande ***pause(x)*** permet d'effectuer un arrêt d'une durée fixée par ***x*** dans un programme. Plus ***x*** est grand, plus la pause est longue !

Exemples :

```
>>y=0:0.1:2;
>>plot(y,sin(y*pi),'-r');
>>hold on;
>>for x=0:0.1:2
    plot(x,sin(x*pi),'*');
    pause(1);
  end;
```

```
>>y=0:0.1:2;
 >>for x=0:0.1:2
   plot(y,sin(y*pi),'-r',x,sin(x*pi),'*');
    pause(0.5);
  end;
```

2.6 Quelques commandes utiles

Voici quelques noms de commandes qui peuvent être utiles. Nous renvoyons le lecteur à l'aide de *Matlab* pour plus de détails sur ces différentes fonctions.

2.6.1 Opérations sur les coefficients d'une matrice

- *max* Coefficient maximal
- *min* Coefficient minimal
- *mean* Moyenne des coefficients
- *median* Valeur médianne des coefficients
- *sort* Tri par ordre croissant des coefficients
- *prod* Produit des coefficients
- *sum* Somme des coefficients

2.6.2 Propriétés élémentaires d'une matrice

- *det* Déterminant
- *eig, eigs, svds* Valeurs propres et vecteurs propres
- *expm* Exponentielle de matrice
- *inv* Matrice inverse
- *null* Noyau
- *orth* Image
- *rank* Rang
- *sqrtm* Racine carrée d'une matrice
- *trace* Trace

2.6.3 Norme et conditionnement d'une matrice

- *cond, condest, rcond* Conditionnement d'une matrice
- *condeig* Conditionnement par rapport aux valeurs propres
- *norm, normest* Norme de matrice

2.6.4 Réduction classiques pour les matrices

- *chol* Factorisation de Choleski
- *hess* Forme de Hessenberg
- *lu* Factorisation LU
- *qr* Décomposition QR
- *rref* Forme échelonnée
- *schur* Décomposition de Schur

2.6.5 Interpolation numérique

- *interp1* Méthodes d'interpolation 1-D
- *interpFT* Méthodes d'interpolation 1-D, utilisant FFT
- *interp2* Méthodes d'interpolation 2-D
- *interp3* Méthodes d'interpolation 3-D
- *polyfit* Interpolation de Lagrange
- *spline* Interpolation par spline cubique

2.6.6 Intégration numérique

- *quad, quad8* Méthodes de quadratures numériques
- *trapz* Méthode des trapèzes

2.6.7 Equation différentielle ordinaire

- *ode45, ode23, ode1113* Résolution d'EDO par des méthodes de Runge Kutta
- *ode15s,ode23s* Résolution d'EDO raides
- *odefile* Définition d'un problème d'EDO
- *odeset,odeget* paramètre d'un fichier d'EDO
- *vdpode* Exemple d'odefile : le système de Van Der Pol

2.6.8 Equations aux dérivées partielles

- *del2* Approximation du laplacien en 2D
- *diff* Différences
- *gradient* Approximation du gradient

2.6.9 Algorithme de minimisation

- *bicg* Méthode de Gradient BiConjugué
- *bicgstab* Méthode de Gradient BiConjugué Stabilisé
- *fminbnd* Minimisation d'une fonction à une variable
- *fminsearch* Minimisation d'une fonction à plusieurs variables

2.6.10 Fourrier

- *fft* Transformation de Fourier rapide en 1-D
- *ifft* Inverse de la transformation de Fourrier rapide en 1-D

2.6.11 Théorie de nombres

- *factor* Facteur premier d'un entier
- *gcd,lcm* PGCD, PPCM de deux entiers
- *primes* Nombres premiers

2.6.12 Zéros de fonctions

- *fzero* Zéros d'une fonction réelle
- *poly* Transforme un ensemble de racines en un polynôme
- *polyval* Evaluation d'un polynôme en un point
- *roots* Racines d'un polynôme