# Démarrer avec Matlab

Jean-Paul Chehab
Laboratoire de Mathématiques Paul Painlevé, CNRS UMR 8524
Equipe AN-EDP
Université de Lille 1

27 janvier 2005

**Résumé**

On présente ici quelques commandes de base de MATLAB, avec en perspective la résolution de problèmes académiques d'analyse numérique des EDP. On cherche en particulier à utiliser ce logiciel dans une démarche expérimentale où, d'une part, les calculs sont programmés en vue d'une illustration de l'analyse mathématique du problème, et, d'autre part, les résultats sont visualisés et interprétés.
Le présent document a simplement pour but de mettre le pied à l'étrier et il a été donc conçu comme un *vademecum* pour débutants ; on trouvera par exemple sur le net des cours complets de Matlab.

## Table des matières

# 1 Le logiciel numérique Matlab

MATLAB (MATrix LABoratory) est un logiciel conçu pour effectuer efficacement des calculs en algèbre linéaire numérique [1]. L'environnement de MATLAB comporte

- un langage de programmation
  dans lequel les matrices peuvent être manipulées formellement ('+, *, inverse') rendant ainsi la mise en œuvre de méthodes numériques proche du formalisme algorithmique
- une interface graphique
  elle permet de visualiser simplement les résultats de calculs au cours ou après l'execution de ceux-ci ; des animations sont possibles.
- un ensemble de fonctions et d'opérations d'algèbre linéaire préprogrammées
  Il s'agit du calcul numérique de valeurs propres, de valeurs singulières, déterminant, exponentielle d'une matrice, l'inverse d'une matrice etc ...
- Des librairies spécialisées (toolboxes)
  pour la résolution de famille de problèmes spécifiques, tels que EDP, optimisation, contrôle, finances, statistiques, analyse du signal etc.

Cet ensemble de caractéristiques font de MATLAB un outil efficace (simplicité, rapidité, lisibilité) utilisé aussi bien par les chercheurs que les étudiants se destinant aux mathématiques appliquées. Enfin, signalons que, d'une part, MATLAB (comme MAPLE) est un des logiciels auquel on peut avoir recours à la troisième épreuve de l'agrégation et que, d'autre part, il est utilisé dans l'industrie, dans certaines sociétés d'ingéniérie.

Ajoutons que MATLAB est un langage interpété, ce qui permet de l'utiliser de manière interactive (pas de compilation). De plus, nombre d'opérations usuelles et de fonctions d'algèbre linéaire [2] y sont implentées avec une efficacité optimisée, ce qui rend ce logiciel adapté à la modélisation et au développement d'algorithmes; la programmation de ces derniers est en particulier proche de la définition mathématique de ceux-ci.
En contrepartie, ce logiciel ne permet pas de simuler efficacement de grands systèmes (plus de $10^6$ inconnues) ; dans ce dernier cas il faudra avoir recours à une programmation en FORTRAN ou en C, par exemple.

**Remarque 1** *Signalons l'existence d'autres logiciels du même type : SCILAB est un logiciel libre (gratuit) développé par l'INRIA* [3].

Pour lancer MATLAB sous unix, il suffit de taper `matlab` puis d'appuyer sur la touche *entrée* ou *return*.

# 2 Tour rapide des commandes et help

La commande `demo` lance une selection des possibilités de MATLAB ; les lignes de programmation de chaque illustration sont visibles.

Mais la meilleure façon de prendre contact avec un logiciel n'est-elle pas de se jetter à l'eau? Comme la plupart des logiciels, MATLAB comporte un nombre important de commandes, qu'il ne s'agit pas de connaître dans leur ensemble. Il est par contre important de savoir dans quel famille de commandes chercher ce que l'on cherche et de pouvoir accéder à sa description, sa syntaxe : c'est la fonction du `help`. Taper `help` affiche à l'écran la liste des familles de commande. La recherche s'affine en tapant `help` suivit du nom d'une commande ou de celui d'un groupe de commandes.

1. l'algèbre linéaire numérique est l'étude et la mise en œuvre de méthodes d'algèbre linéaire en vue de résoudre, à l'aide de moyens de calculs, des systèmes linéaires
2. produit de matrices, recherche approchée de valeurs propores etc
3. On peut télécharger ce logiciel depuis l'adresse `http://www-rocq.inria.fr/scilab`

# 3 Les variables scalaires et les variables logiques

## 3.1 Commentaires et affichage ou non des résultats

Tout comme en MAPLE, les commandes MATLAB ont un mode affichage et non affichage à l'écran selon qu'on termine la commande par un `,` (ou rien du tout) ou par un `;`, respectivement. Ces caractères font donc office de séparateurs, ce qui permet d'écrire plusieurs commandes par ligne. Par exemple

```
>> 5;
```

ne retournera rien alors que

```
>> 5
5
```

affiche le résultat.

Enfin, et c'est primordial pour rendre lisible les programmes MATLAB, on peut rendre inactive dans une ligne tout ce qui suit un %. Et donc écrire des phrases de commentaires après ce signe.

## 3.2 Les variables scalaires

### 3.2.1 Les constantes

| | |
|---|---|
| `pi` | `3.1416` |
| `i` | `0 + 1.0000i` |
| `j` | `0 + 1.0000i` |
| `eps` | `2.2204e-016` |
| `realmin` | `2.2251e-308` |
| `realmax` | `1.7977e+308` |
| `Inf` | `infini` |
| `NaN` | `Not a Number` |

### 3.2.2 Les réels

Les calculs sont effectués en double précision, c'est à dire avec 16 chiffres significatifs ; l'epsilon machine, c'est à dire le plus petit réel représenté tel que '`1 +`$\epsilon$`=1`' est `eps` (cf tableau ci-dessus).

### 3.2.3 Les complexes

En MATLAB on définit `i` (ou `j`, voir le tableau des constantes) comme la racine de $-1$ ; il faudra être vigilant au fait que `i` pourra désigner une toute autre variable suivant le contexte.

Les opérations algébriques usuelles entre nombres complexes s'effectuent comme formellement. Par exemple

```
>>z1=3+5*i
```

```
z1=

3.0000 + 5.0000i

>>z2=4-2*i

z2=

4.0000 - 2.0000i

>>» z1+z2,z1*z2,z1/z2

ans =

7.0000 + 3.0000i

ans =

22.0000 +14.0000i

ans =

0.1000 + 1.3000i
```

Ici `ans` est l'abréviation de *answer*.

### 3.2.4 Opérations, opérateurs et fonctions incorporées

| | |
|---|---|
| `abs` | valeur absolue ou module complexe |
| `sqrt` | racine carrée |
| `real` | partie réelle |
| `imag` | partie imaginaire |
| `conj` | complexe conjugué |
| `round` | arrondi à l'entier le plus proche |
| `floor` | arrondi à l'entier inférieur |
| `ceil` | arrondi à l'entier supérieur |
| `sign` | signe |

Sans oublier les fonctions usuelles, cf Annexe.

## 3.3 Les variables logiques

Les variables logiques seront principalement utilisées dans les stuctures de contrôles qui apparaissent lors de la programmation d'algorithmes.

### 3.3.1 Opérateurs de relation et opérateurs logiques

| | | | |
|---|---|---|---|
| $<$ | $>$ | & | et |
| $>=$ | $<=$ | \| | ou |
| $==$ | ~= | ~ | non |

# 4 Les vecteurs et les matrices

Etant données les applications visées (résolution numérique d'EDP elliptiques), on se concentrera plus particulièrement sur

- la construction de systèmes linéaires (matrices et vecteurs)
- la résolution de ceux-ci
- la visualisation des résultats

## 4.1 Construction

### 4.1.1 Vecteurs et matrices construits "à la main"

On construit un vecteur ligne comme ci dessous :

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

Le vecteur colonne s'obtient en séparant chacune des entrées par un ;

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

Les vecteurs à incréments constants. Le vecteur

```
v=1:5:34
```

a pour composantes

```
v=
   1  6  11  16  21  26  31
```

Les matrices se construisent alors logiquement comme empilement de vecteurs lignes, soit en séparant chaque ligne de la suivante par un ;
Par exemple,

```
A=[ 8 1 6 ; 3 5 7 ; 4 9 2]
```

est la matrice magique d'ordre 3 ; on retrouve cette matrice avec l'instruction `magic(3)` :

```
      8  1  6
A=    3  5  7
      4  9  2
```

L'élément $A_{2,3}$ est donné (tout simplement) par `A(2,3)`.

### 4.1.2 Matrices et vecteurs particuliers

| | |
|---|---|
| `eye(n,m)` | matrice identité d'ordre $n \times m$ |
| `zeros(n,m)` | matrice nulle $n \times m$ |
| `ones(n,m)` | matrice de coefficients tous égaux à 1 $n \times m$ |
| `rand(n,m)` | matrice aléatoire $n \times m$ |
| `hilb(n)` | matrice de Hilbert d'ordre $n$ |
| `magic(n)` | matrice magique d'ordre $n$ |
| `toeplitz(v)` | matrice de Toeplitz construite à partir du vecteur `v` |

Exemple Si $v$ est le vecteur `v=[2 -1 0 0 0]`, alors `A=toeplitz(v)` sera la matrice

```
      2  -1   0   0   0
     -1   2  -1   0   0
A=    0  -1   2  -1   0
      0   0  -1   2  -1
      0   0   0  -1   2
```

### 4.1.3 Matrices par bloc et sous-matrices

`MATLAB` permet d'extraire facilement des sous-matrices à partir d'une matrice donnée. En reprenant l'exemple précédent, on a

`A(1,:)= 2 -1 0 0 0 % la première ligne de la matrice` ,

la commande `A(:,3)` retourne la troisième colonne. Il s'ensuit logiquement que `A(2:4,3:5)` est la sous-matrice $A_{i,j}, i = 2, \cdots 4, j = 3, \cdots, 5$.

L'assemblage de matrices à partir de sous-matrices s'effectue aussi facilement. Par exemple soit `B=ones(5)`. Alors la matrice `M` définie par

`M(1:5,1:5)=B,M(1:5,6:10)=A`

sera

```
1  1  1  1  1   2  -1   0   0   0
1  1  1  1  1  -1   2  -1   0   0
1  1  1  1  1   0  -1   2  -1   0
1  1  1  1  1   0   0  -1   2  -1
1  1  1  1  1   0   0   0  -1   2
```

### 4.1.4 Les matrices creuses

Les matrices creuses (creux se dit *sparse* en anglais) sont des matrices dont la plupart des coefficients sont nuls, c'est par exemple le cas de la matrice `A` construite plus haut. Ces matrices jouent un rôle important en calcul scientifique : d'une part, du fait de leur spécificité, les produits matrice vecteurs sont peu couteux, ce qui rend efficace en temps de calcul la résolution de systèmes linéaires sous-jacents par des méthodes de type gradient, et, d'autre part, ces matrices apparaissent lors de la discrétisation d'opérateurs différentiels, en différences finies comme en éléments finis, c'est à dire dans nombre d'approximation de problèmes issus de la physique.

Dans MATLAB, il est possible de définir de telles matrices avec la commande `sparse` que l'on consultera dans le `help`.
Précisons que MATLAB stocke les indices des matrices de type *sparse* d'une manière différente de celles des matrices pleines et que, par exemple, les produits entre matrices creuses sont effectués suivant des schémas spécifiques. Nous avons représentés ci-dessous les éléments non nuls des matrices du laplacien en dimension 1 et 2, lorsque la grille est composé de 31 points de discrétisation dans chaque direction du domaine. La visualisation s'effectue à l'aide de la commande `spy`.

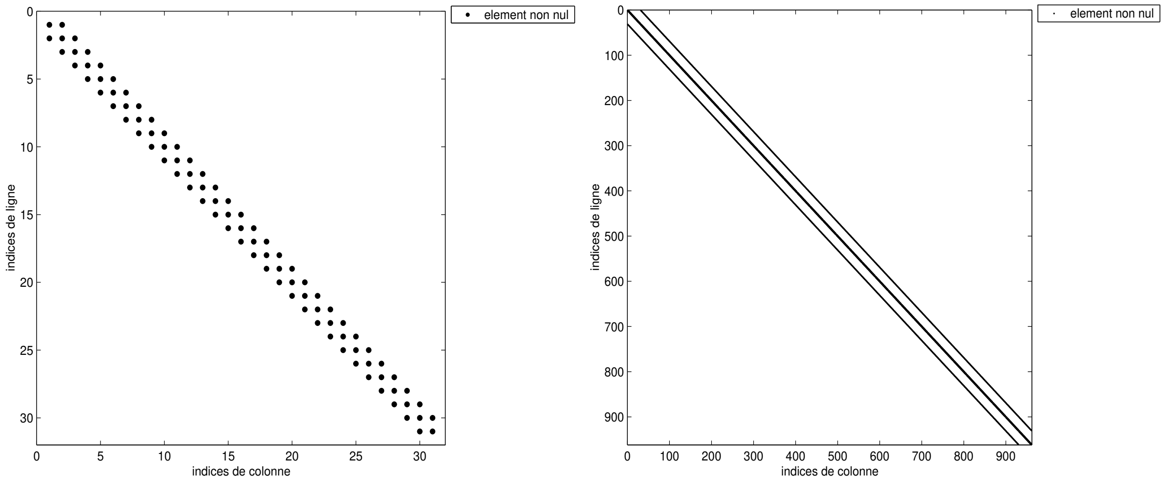


FIG. 1 – *Représentation des éléments non nuls des matrices de l'opérateur de Laplace en dimension 1* (0.0947%) *et en dimension 2* (0.0051%)

## 4.2 Opérations sur les matrices et les vecteurs

Les opérateurs suivants agissent aussi bien sur les scalaires que sur les matrices, avec la même écriture, ce qui premet de présenter la mise en œuvre des méthodes matricielles sous une forme très proche de celle, formelle, des algorithmes.

### 4.2.1 Les opérateurs

| | |
|---|---|
| Addition | + |
| Soustraction | - |
| Multiplication | * |
| Multiplication (composante par composante) | *. |
| Division | / |
| Division (à gauche) | \ (pour les matrices) |
| Puissance | ^ |
| Conjugaison, transposition | ' |

Plus précisément, concernant la division entre matrices : Si $x$ et $b$ sont deux vecteur colonnes de même dimension $n$ et $A$ une matrice carrée inversible de dimension $n$, alors

$$\begin{array}{ll} \mathtt{x} = \mathtt{A}``b & \text{est la solution de } \mathtt{A} * \mathtt{x} = \mathtt{b}, \\ \mathtt{x} = \mathtt{b}``A & \text{est la solution de } \mathtt{x} * \mathtt{A} = \mathtt{b} \end{array}$$

La division à gauche des matrices est effectuée par Matlab en utilisant une factorisation de Gauss.

### 4.2.2 Fonctions sur les tableaux

| | |
|---|---|
| `sum` | somme des éléments d'un tableau |
| `prod` | produit des éléments d'un tableau |
| `sort` | trie par ordre croissant les éléments d'un tableau |
| `mean` | moyenne arithmétique des éléments d'un tableau |
| `any` | vaut 1 si tous les éléments d'un tableau sont non nuls, vaut 0 sinon |

### 4.2.3 Image d'un tableau par une fonction scalaire

L'image d'un tableau `A` par une fonction scalaire `f` est le tableau `B` dont les composantes sont les images des composantes de `A` par `f`. Par exemple,

```
>> A= 1:4;
>>B=sin(A)

B =

0.8415 0.9093 0.1411 -0.7568
```

### 4.2.4 Fonctions matricielles intégrées

| | |
|---|---|
| `det` | déterminant |
| `poly` | polynôme caractéristique |
| `norm` | norme (1, 2, infinie et de Frobenius) |
| `cond` | conditionnement |
| `rank` | rang |
| `size` | taille |
| `eig` | vecteurs et valeurs propres |
| `svd` | (Singular vValues Decomposition) décomposition en valeurs singulières |
| `chol` | factorisation de Cholesky |
| `lu` | factorisation LU |
| `qr` | factorisation QR |
| `expm` | exponentielle d'une matrice<br>(à ne pas confondre avec `exp`) |
| `kron` | produit tensoriel<br>`C=kron(A,B)` retourne la matrice par blocs $C_{i,j} = A_{i,j}B$ |

Exemple L'instruction **kron** permet de construire en une ligne la matrice de discrétisation de $-\Delta$ en dimension 2

```
>>n=3;
>> T=toeplitz([2,-1,ones(1,n-2)]);
>> Id=eye(n,n);
>> A=kron(T,Id)+kron(Id,T))
>>A=

     4  -1   0  -1   0   0   0   0   0
    -1   4  -1   0  -1   0   0   0   0
     0  -1   4   0   0  -1   0   0   0
    -1   0   0   4  -1   0  -1   0   0
     0  -1   0  -1   4  -1   0  -1   0
     0   0  -1   0  -1   4   0   0  -1
     0   0   0  -1   0   0   4  -1   0
     0   0   0   0  -1   0  -1   4  -1
     0   0   0   0   0  -1   0  -1   4
```

#### 4.2.5 Fonctions de recherche d'éléments

| | |
|---|---|
| `max(A)` | recherche le plus petit élément du tableau `A` |
| `min(A)` | recherche le plus grand élément du tableau `A` |
| `sort(A)` | trie les éléments du tableau `A` |
| `find(A >0)` | renvoie les indicies des composantes de `A` strictement positives |
| `find(A==0)` | renvoie les indicies des composantes de `A` égales à zéro |

# 5 Programmation (structure de contrôle)

## 5.1 Les boucles

Boucle for

```
for i=1:n
instructions
end
```

Boucle while

```
while proposition logique
instructions
end
```

## 5.2 Les tests

Test simple

```
if proposition logique
instructions
end
```

Tests multiples

```
if proposition logique 1
instructions 1
elseif proposition logique 2
instructions 2
else
instructions 3
end
```

# 6 Programmes et sous-programmes

## 6.1 Ecrire un programme

Dès que la tâche à effectuer n'est plus triviale, il devient impératif d'organiser la suite de commandes à executer. On suivra la démarche suivante :

- Regrouper dans un *programme principal* le plan de la tâche à effectuer
- mettre chaque étape (non triviale) de ce plan sous forme de sous-programme

Dans chacune des unités de programme, décrire la tâche à effectuer, les variables en entrée, celles en sortie. Commenter le plus clairement possible les différentes articulations. Un programme mal ou pas commenté est facilement illisible par une autre personne que l'auteur et même par l'auteur lui-même si suffisamment d'eau a coulé dans la Deule.

## 6.2 Le fichier "point m" (ou script)

Une succession de commandes peut être écrite dans un fichier (à l'aide d'un éditeur quelconque). Ce fichier devra avoir une extension `.m` et son exécution s'effectuera simplement en tapant son nom (sans extension).
Voici par exemple le programme du calcul approché d'une intégrale par deux formules composites différentes (la programmation n'est pas optimale volontairement)

```
%
% Calcul approché de Pi par integration approchee de
%
% 4*sqrt(1-x^2) entre 0 et 1
%
%par la methode des rectangles a gauche et par celle des trapezes
%
nr=10;%nombre de points pour la methode des rectangles
nt=10;%nombre de points pour la methode des trapezes
%1) calcul par les rectangles
hr=1/(nr);%nombre de subdivisions
xr=0:hr:1;%suite des points d'integration
int_r=0;%initialisation du compteur
for i=1:nr-1
int_r=int_r+sqrt(1-xr(i)^2);
end
int_r=4*hr*int_r;
err_r=pi-int_r;%erreur
%
%2) calcul par les trapezes
ht=1/(nt);%nombre de subdivisions
xt=0:hr:1;%suite des points d'integration
int_t=0;%initialisation du compteur
for i=1:nt+1
int_t=int_t+0.5*(sqrt(1-xt(i)^2) +sqrt(1-xt(i+1)^2));
end
int_t=4*ht*int_t;
err_t=pi-int_t;%erreur

%Affichage des resultats
fprintf('Methode des rectangles avec %f \n points %f \n erreur %f \n',nr,valeur_r,err_r)
fprintf('Methode des trapezes avec %f \n points %f \n erreur %f \n', nt,valeur_t,err_t)
```

## 6.3 Fonctions `inline`

C'est le moyen qu'offre `MATLAB` pour manipuler fonctions simples.

```
>>f=inline('sqrt(1-x^2')
```

On peut alors manier `f` comme toute fonction usuelle telle que `sin, cos`.

### 6.4 Le fichier `function`

Lorsqu'une série d'instructions doit être executée plusieurs fois dans un même programme, on a recours aux fonctions externes ou `function`. Elles ont la structure de fonctions, c'est à dire qu'elle ont une suite d'arguments et retournent un suite de résultats. La syntaxe générale est :

function [y1,y2,···,yp]=funktion(x1,x2,···,xq)

Par exemple la fonction trapèzes :

```
%
function intt=trapezes(nt,f) ht=1/(nt);%nombre de subdivisions
xt=0:hr:1;%suite des points d'integration
intt=0;%initialisation du compteur
for i=1:nt
intt=intt+0.5*(f(xt(i))+f(xt(i+1))
end
intt=4*ht*intt;
```

La valeur du résultat est sockée dans la variable de sortie **intt** ; cette valeur est accessible depuis un script par
`>>valeur=trapezes(nt,f);`

# 7 Le graphisme

L'interface graphique de MATLAB est particulièrement efficace : elle permet de visualiser, à l'aide de commandes simples, des résultats au cours ou à la fin de l'exécution d'un programme. On présente ici quelques commandes de base pour en savoir davantage tapez **help plot** et/ou visulaliser les démos graphiques.

## 7.1 Graphique simple

### 7.1.1 Les courbes en 1d

Soient `x` et `u` deux tableau à un indice de `n` éléments
`>> plot(x,u)`
repésentera `u` avec le tableau `x` pour abscisses.

La structure de commande pour représenter plusieurs courbes sur un même graphique est
`plot(x1,u1,x2,u2,···xp,up)`

Exemple
```
>>h=0.1;
>>x=0:h:1;
>>u1=sin(pi*x);u2=cos(pi*x);
>>plot(x,u1,x,u2)
```

Ici les courbes sont toutes représentées de la même manière, à savoir à l'aide d'un trait continu. Il est possible de "forcer le trait" comme suit
`>>plot(x,u1,'+-b',x,u2,'v--r')`

**Remarque 2** *Si le tableau des abscisses n'est pas précisé les données seront représentées comme ordonnées avec le tableau des indices pour abscisses, c'est à dire que par défaut* `x=1:n`.

**Remarque 3** *Consulter le descriptif de la commande* `hold on` *...*

### 7.1.2 Les surfaces

La commande `mesh` permet de représenter des surfaces $z = f(x, y)$. En MATLAB, `x` et `y` seront des tableaux à un indice si bien que `z` sera représenté sous forme de matrice ; (z) représentera donc graphiquement la matrice `z`.

<u>Exemple</u>

```
>>hx=0.1;
>>hy=0.2;
>>x=0:hx:1;
>>y=0:hy:2;
>>u1=sin(pi*x)*cos(pi*y);
>>mesh(x,y,u1)% ou bien surf(x,y,u1)
```

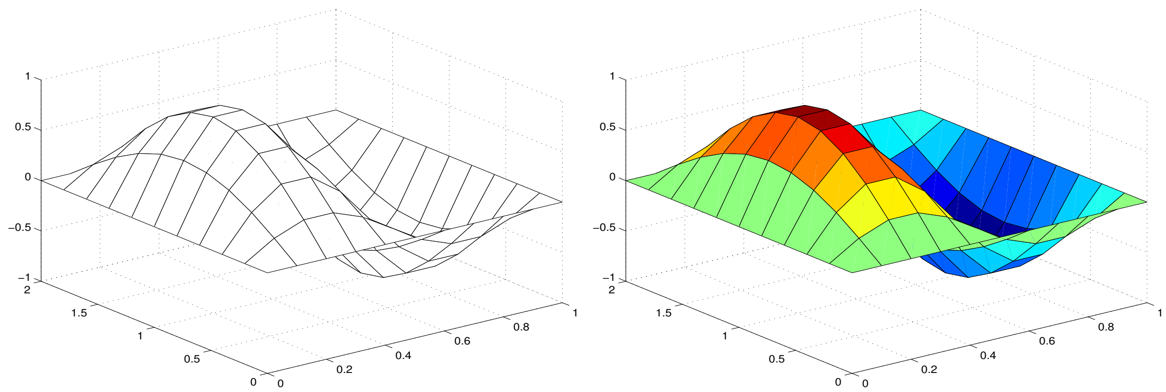


FIG. 2 – *Surfaces avec* `mesh` *(à gauche) et* `surf` *(à droite)*

## 7.2 Plusieurs graphiques sur une même fenêtre

Là encore la structure de tableau est sous-jacente : la fenêtre graphique est organisée comme un tableau rectangulaire de (cases) graphiques. Il faut donc d'une part construire le tableau (nombre de lignes, nombres de colonnes) et d'autre part pouvoir placer un graphique donné dans une case donnée. La commande consacrée est `subplot`. Par exemple

```
>>h=0.1;
>>x=0:h:1;
>>u1=sin(pi*x);u2=cos(pi*x);
>>subplot(2,1,1),plot(x,u1)
>>subplot(2,1,2),plot(x,u1)
```

représentera les courbes `plot(x,u1)` et `plot(x,u2)` dans un tableau de graphiques de 2 lignes, un colonne ; `plot(x,u1)` est le premier et `plot(x,u2)` selon l'ordre lexicographique de haut en bas et de gauche à droite.

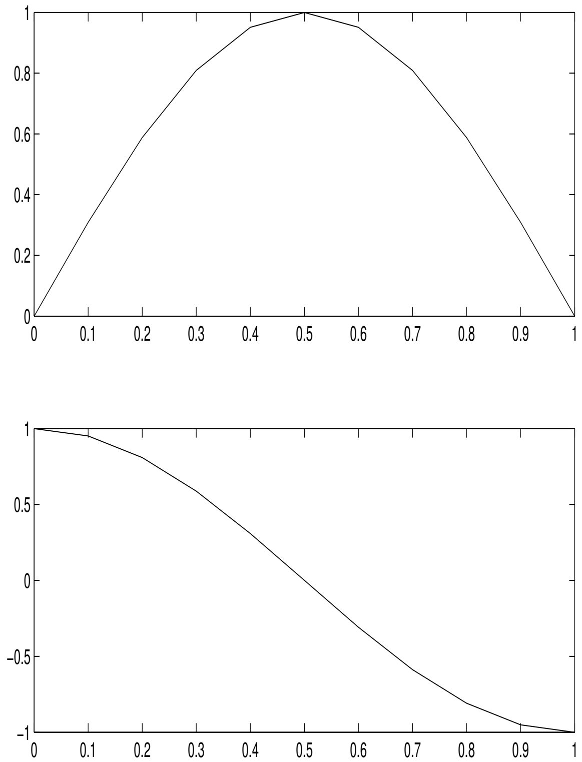


FIG. 3 –

## 7.3 Légendes et titres

Les instructions relatives aux titres légendes et label se placent après l'ordre `plot` (ou autres) du graphique auxquels ils correspondent. En voici quelques uns à travers l'exemple suivant :

```
>>plot(x,u1)
>>title('Le graphe de f(x)=sin(pi*x) sur [0,1]')
>>xlabel('x')
>>ylabel('y')
```

Lorsque plusieurs courbes sont représentées il est naturel d'associer chaque courbe à un type de trait et de signaler explicitement cette correspondance dans la fenêtre graphique. La commande `legend` effectue cette tâche.

```
>>plot(x,u1,'+-b',x,u2,'v--r')
>>title('Les graphes de f(x)=sin(pi*x) et de g(x)=cos(pi*x) sur [0,1]') >>xlabel('x')
>>ylabel('y')
>>legend('f(x)','g(x)')
```

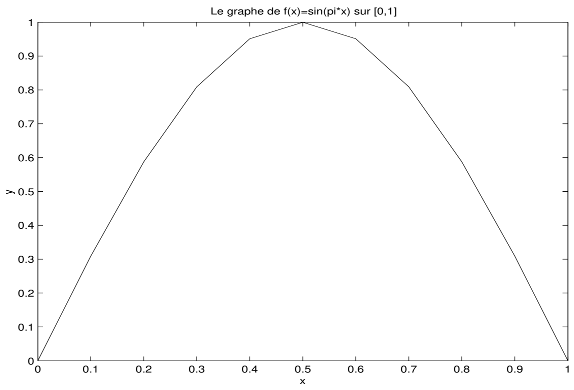


FIG. 4 –

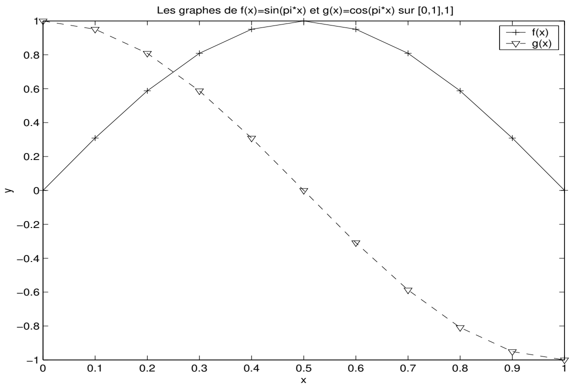


FIG. 5 –

## 7.4 Séquence de figures (animation naive)

La manière la plus simple de visualiser l'évolution d'un type de résultats numériques est d'utiliser la commande `drawnow`. Par exemple

```
>>h=0.1;
>>dt=0.1;
>>x=0:h:1;
>>nitertemps=100;
>>for i=1:nitertemps
>>u=sin(pi*x*sin(i*dt));
>>plot(x,u)
>>drawnow;
>>end
```

affichera successivement les représentations graphiques de $u(x,t) = \sin(\pi x \sin(t))$ pour les valeurs discrètes de $t$, $i \cdot dt, i = 1, \cdots,$`nitertemps`.

# 8 Annexe : quelques commandes (liste non exhaustive)

## 8.1 Commandes générales

| Commande | fonction |
|---|---|
| `help` | aide en ligne |
| `demo` | execution des démonstrations |
| `who` | liste des variables en mémoire |
| `whos` | liste des variables en mémoire avec leur taille |
| `clear` | efface l'espace mémoire |
| `quit` | pour quitter MATLAB |

## 8.2 Fonctions usuelles

| Commande | fonction |
|---|---|
| `abs` | valeur absolue |
| `sqrt` | racine carrée |
| `real` | partie réelle |
| `imag` | partie imaginaire |
| `conj` | complexe conjugué |
| `exp` | exponentielle |
| `log` | logarithme néperien |
| `log10` | logarithme en base 10 |
| `sin` | sinus |
| `cos` | cosinus |
| `tan` | tangente |
| `asin` | arcsinus |
| `acos` | arccosinus |
| `atan` | arctangente |
| `sinh` | sinus hyperbolique |
| `cosh` | cosinus hyperbolique |
| `tanh` | tangente hyperbolique |
| `asinh` | arcsinus hyperbolique |
| `acosh` | arccosinus hyperbolique |
| `atanh` | arctangente hyperbolique |

## 8.3 Commandes de tableaux et commandes matricielles

| Commande | fonction |
|---|---|
| `tril` | partie triangulaire inférieure d'une matrice |
| `triu` | partie triangulaire supérieure d'une matrice |
| `diag` | matrice diagonale |
| `hilb` | matrice de Hilbert |
| `invhilb` | inverse de la matrice de Hilbert |
| `hadamard` | matrice de Hadamard |
| `pascal` | matrice de Pascal |
| `vander` | matrice de vandermonde |

### 8.4 Commandes graphiques

| Commande | fonction |
|---|---|
| `loglog` | échelle log en x et en y |
| `semilogy` | échelle log en y seulement |
| `semilogx` | échelle log en x seulement |
| `polar` | dessine en polaires |
| `grid` | dessine une grille |
| `gtext` | place un texte à l'aide de la souris |
| `axis` | ajustement des axes |
| `hold` | fige un graphique sur l'écran |
| `clg` | efface la fenêtre graphique |
| `print` | impression d'une graphique ou génération d'un fichier postscript |