Prise en main Matlab/Octave*

Benjamin Monmege
benjamin.monmege@lsv.ens-cachan.fr

9 février 2012

1 Introduction

MATLAB [1] est à la fois un environnement pour faire du calcul numérique et un langage de programmation. C'est un logiciel commercial. Le nom MATLAB vient de « MATrix LABoratory ». Le logiciel permet de faire des manipulations sur des matrices, tracer des fonctions, visualiser des données, implémenter de nouveaux algorithmes, faire des interfaces graphiques, interfacer des programmes écrits dans d'autres langages de programmation. C'est un logiciel très utilisé dans le monde académique et industriel. Le succès de ce logiciel vient de sa simplicité de prise en main et d'utilisation.

Le langage MATLAB est *interprété* et non *compilé*, i.e., les instructions sont traduites en langage machine à la volée, et il n'y a donc pas besoin de faire appel à une étape de compilation (contrairement au C ou Java). Matlab est également un langage non-typé : nul besoin de déclarer le type des variables en amont, le logiciel adapte le typage nécessaire. Du coup, un algorithme codé en MATLAB sera généralement plus lent qu'un algorithme codé dans un langage comme C par exemple, sauf si l'algorithme peut exploiter des structures vectorielles et être automatiquement parallélisé par MATLAB, nous reviendrons dans la suite du cours sur ce point.

Octave [2] est un logiciel *open-source*, c'est-à-dire non seulement gratuit mais dont le code source est accessible et réutilisable (sauf en général pour des activités commerciales) « clone » de MATLAB : la plupart des commandes MATLAB sont disponibles avec Octave. Scilab est un autre logiciel *open-source* de calcul scientifique développé par l'INRIA (Institut National de Recherche en Informatique) [3] très similaire à MATLAB. Cependant les syntaxes de base de Scilab et MATLAB sont plus sensiblement différentes que celles d'Octave et MATLAB.

Les instructions, exercices et exemples donnés dans ce document sont *a priori* compatibles avec MATLAB et Octave– la plupart étant également compatible avec Scilab.

2 Prise en main de MATLAB ou Octave

La prise en main de MATLAB ou Octave dépend du système d'exploitation utilisé (Linux, Windows, MacOS). **Attention**, il existe des versions « console » et des versions « graphiques » de l'interface, qui doit ressembler à quelque chose comme celle de la Figure 1. Sous

*Document extrait de l'introduction à MATLAB rédigé par Anne Auger et Pierre Allegraud ({anne.auger,pierre.allegraud}@lri.fr). Ce script de cours reprend des éléments du document de Marie Postel, *Introduction au logiciel MATLAB* (http://www.ann.jussieu.fr/~postel/matlab/).

1. http://www.mathworks.com
2. http://www.gnu.org/software/octave/
3. http://www.scilab.org/

Linux, la commande `octave` lance le mode console, et `qtoctave` propose une interface graphique (si elle a été installée [4]).

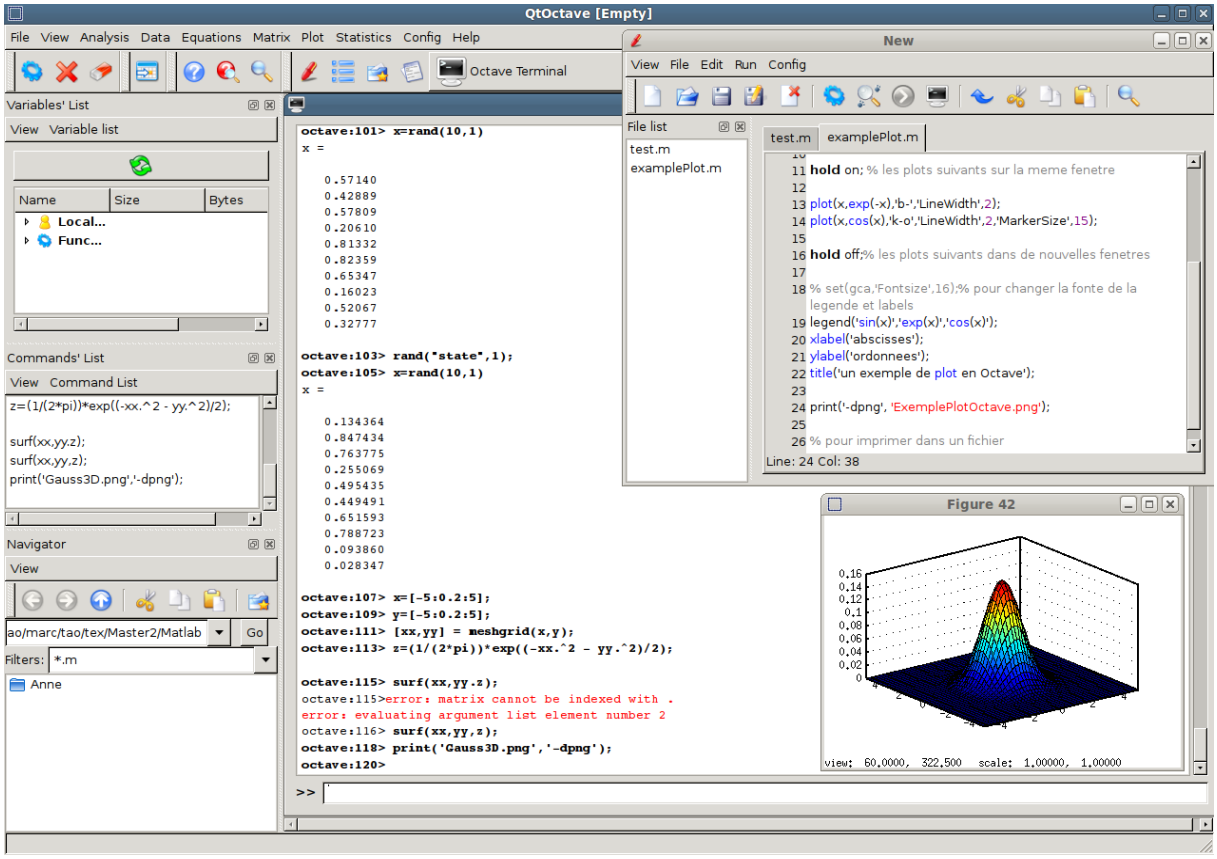


FIGURE 1 – Environnement graphique Octave (et Matlab, à quelques détails près). La fenêtre principale est partiellement recouverte par la fenêtre de l'éditeur (ici en haut à droite) et la fenêtre graphique (en bas à droite). L'entrée des commande se fait dans le champ suivant le 'prompt' >>.

Avant de lancer Octave, créer un répertoire `InitiationMatlab` et se mettre dans ce répertoire. Dorénavant les scripts et fonctions MATLAB que vous allez créer devront être sauvegardés dans ce répertoire (c'est le défaut si vous utilisez l'interface graphique). Quelques commandes utiles qui marchent sous linux ou dans la fenêtre de commande MATLAB ou Octave :

- pour créer un répertoire : `mkdir NomDuRepertoire`
- pour se déplacer dans un répertoire : `cd NomDuRepertoire`
- pour vérifier où l'on se trouve : `pwd`
- pour lister les fichiers du répertoire courant : `ls`

Les instructions seront à exécuter dans la fenêtre de commande après les `>>`. Toutes les instructions en ligne s'exécuteront après avoir tapé sur la touche `Entrée` ou `Return`. Pour commencer, tapez dans un premier temps

```
>> a=4
```

puis dans un deuxième temps

```
>> b=7;
```

Remarquez l'intérêt du point-virgule...

4. Cf. la page `http://qtoctave.wordpress.com/` pour la télécharger le cas échéant. À noter que ce logiciel n'est plus maintenu...

Remarque 1. Si vous oubliez le point-virgule, ou si l'exécution d'un programme est trop longue, vous pouvez arrêter l'exécution en faisant `Control C` dans la fenêtre de commande.

Pour afficher les variables en mémoire, taper `whos`, et pour effacer des variables, utiliser l'instruction `clear`. Dans l'interface graphique, ces données sont regroupées dans la fenêtre Workspace.

2.1 Fonction `help`

Etant donné le très grand nombre d'instructions utilisables, il est impossible de mémoriser chacune d'elle avec sa syntaxe correspondante. Il est donc essentiel d'utiliser l'aide. Pour trouver l'aide associée à une instruction, il vous suffit de taper `help NomInstruction`. Par exemple, taper

```
>> help rand
```

Dans MATLAB, vous pouvez également accéder à l'aide dans le menu du même nom.

2.2 Opérations arithmétiques, fonctions de base

Les opérations arithmétiques de base sont résumées dans le tableau suivant

Opération	Instruction MATLAB
$a+b$	`a+b`
$a-b$	`a-b`
$a \times b$	`a*b`
$\frac{a}{b}$	`a/b`
a^b	`a^b`

Par ailleurs toutes les fonctions de base $(\exp, \cos, \sin, \ln)$ sont déjà disponible avec une syntaxe intuitive (utiliser l'aide en cas de doute).

2.3 Afficher des données ; Commande `disp`

Si vous tapez dans la fenêtre de commande `a=2` vous allez observer ceci

```
>> a=2
a =
     2
```

Pour lire de nouveau la valeur de `a`, il suffit de taper à nouveau `a`. Il est également possible d'afficher les données proprement avec la commande `disp` que l'on peut également utiliser pour faire un affichage :

```
>> disp(['La valeur de a est ' num2str(a)])
La valeur de a est 2
```

3 Scripts, Fonctions et Graphiques

3.1 Écrire et exécuter un script

Il est généralement commode d'écrire l'ensemble des instructions que l'on veut exécuter dans un fichier que l'on appelle un *script* que l'on va sauvegarder. Le nom du fichier pour la sauvegarde devra avoir le suffixe `.m`. Une fois sauvegardé, on pourra exécuter le script en tapant le nom du fichier (sans le suffixe).

Les instructions `clear all`, `clc`, `close all` sont utiles pour tout remettre à zero. Vérifiez par vous-même la signification de ces commandes.

Remarquez que les variables à l'intérieur d'un script sont des variables globales auxquelles vous pouvez accéder dans la fenêtre de commande une fois le script exécuté.

3.2 Fonctions

Lorsque l'on fait appel à un même algorithme plusieurs fois en ne changeant que les variables d'entrée/sortie, il s'avère utile de pouvoir créer des fonctions. Pour créer une fonction, on ouvre un nouveau fichier dans lequel une fonction aura la syntaxe suivante

```
function [out1,out2,...]=nomfonction(input1,input2,...)
  statements
end
```

où `input1,input2` sont les variables d'entrées et `out1,out2` de sortie. Le fichier où se trouve la fonction doit porter le nom `nomfonction.m`. Regardons l'exemple proposé par l'aide MATLAB :

```
function [m,s] = stat(x)
  %STAT Interesting statistics.
  n = length(x);
  m = mean(x,n);
  s = sqrt(sum((x-m).^2)/n);
end
```

qui doit donc être sauvegardé dans un fichier appelé `stat.m`.

Les variables utilisées à l'intérieur d'une fonction sont des variables locales, c'est-à-dire qu'elles n'existent que lors de l'appel de fonction et sont inaccessibles à partir de l'espace principale. Si une variable de même nom existe déjà en mémoire, en tant que variable globale, il n'y aura pas d'interférence entre ces deux données. Notons que pour les scripts par contre, les variables restent dans l'espace principal.

3.3 Plot 2-D

Les représentations graphiques en MATLAB se basent sur une discrétisation des variables rangées dans des matrices ou des vecteurs colonnes.

Nous décrivons maintenant comment représenter une courbe du type $y = f(x)$ sur un intervalle $[a, b]$. On commence par créer un vecteur x contenant un ensemble de points $(x_i)_{1\leq i\leq N}$ ordonnés entre a et b et on crée un vecteur y, de même taille que le vecteur x contenant les valeurs $(f(x_i))_{1\leq i\leq N}$. On utilise ensuite l'instruction `plot(x,y)`. Le graphique va s'afficher dans une figure. Par exemple

```
x=linspace(0,2*pi,100);plot(x,sin(x));grid on;
```

va tracer la fonction sinus entre 0 et 2π en prenant 100 points répartis uniformément entre 0 et 2π.

Il est possible de rajouter plusieurs options à la commande `plot`, par exemple pour rajouter des marqueurs `'+'` aux points $(x_i, f(x_i))$ on utilise l'instruction

```
x=[0:0.1:2*pi];plot(x,sin(x),'+');grid on;
```

Pour changer la couleur, on spécifie entre `' '`, l'initiale anglaise de la couleur que l'on veut. Pour une courbe en rouge `x=[0:0.1:2*pi];plot(x,sin(x),'r');grid on;` avec des marqueurs rouges `x=[0:0.1:2*pi];plot(x,sin(x),'r+');grid on;` avec des marqueurs et un trait continu : `x=[0:0.1:2*pi];plot(x,sin(x),'r+-');grid on;`

Par défaut la fenêtre graphique est effacée avant chaque commande plot. Pour superposer des courbes, on utilise l'instruction `hold on`, et lorsque l'on veut arrêter de superposer des courbes on utilise l'instruction `hold off`. L'exemple présenté à la Table 1 illustre les instructions les plus utiles pour tracer une ou plusieurs courbes, ajuster la taille des fontes, couleurs, ajouter une légende... Le graphique correspondant est présenté à la Figure 2.

Il est souvent utile de tracer une courbe avec une échelle logarithmique soit pour l'axe des abscisses soit pour l'axe des ordonnées. La Table 2 donne les instructions pour les tracés en échelle logarithmique et résume les principales instructions graphiques.

```
figure(42) %specifie un index (42) pour la fenetre graphique

x=[0:0.1:3.5];

plot(x,sin(x),'r-+','LineWidth',2,'MarkerSize',15);
%'LineWidth',2: change la taille de la fonte  pour la courbe
%'MarkerSize',15: change la taille des marqueurs

grid on; % rajoute une grille

hold on; % les plots suivants sur la meme fenetre

plot(x,exp(-x),'b-','LineWidth',2);
plot(x,cos(x),'k-o','LineWidth',2,'MarkerSize',15);

hold off;% les plots suivants dans de nouvelles fenetres

% Matlab only set(gca,'Fontsize',16);
% pour changer la fonte de la legende et labels
legend('sin(x)','exp(x)','cos(x)'); % oups c'est exp(-x)
xlabel('abscisses');
ylabel('ordonnees');
title('un exemple de plot en Matlab');

print('-dpng', 'ExemplePlotOctave.png');
% pour imprimer dans le fichier ExemplePlotOctave.png
```

TABLE 1 – Un exemple d'utilisation de l'instruction `plot` MATLAB, ainsi que des instructions les plus utiles pour ajuster la taille des fontes, les couleurs, les marqueurs, rajouter un titre, une légende. Le graphe correspondant à ce script se trouve Fig 2.

3.4 Plot 3-D

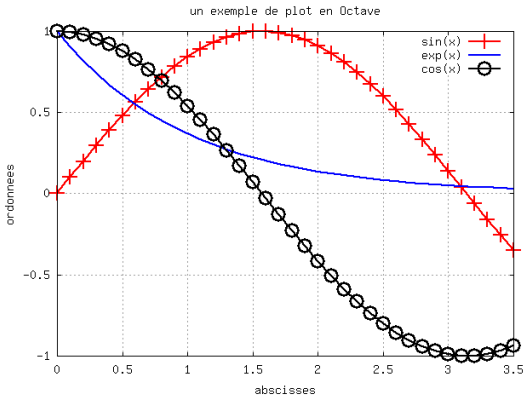
MATLAB permet également de visualiser des fonctions de 2-variables. Considérons la fonction

$$z = f(x, y) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x^2 + y^2)}{2}\right) \ ,$$

que nous allons représenter pour $(x, y) \in [-5, 5] \times [-5, 5]$. A partir d'un maillage des abscisses et des ordonnées défini par `x=-5:0.2:5;y=-5:0.2:5;` l'instruction `[xx,yy] = meshgrid(x,y)` va créer deux tableaux bi-dimensionnels `xx` et `yy` correspondant aux coordonnées des abscisses et ordonnées de la grille $x \times y$. Il reste maintenant à calculer la valeur de la fonction f en chaque point de la grille

```
z=(1/(2*pi))*exp((-xx.^2 - yy.^2)/2);
```

et à faire appel à l'instruction `surf` : `surf(xx,yy,z)`. Nous obtenons la figure présentée Fig 2. Il existe d'autres fonctions pour visualiser des fonctions de deux variables que vous pouvez découvrir dans l'aide MATLAB, citons `contour` et `ezcontour` pour tracer les lignes de niveau de la fonction.



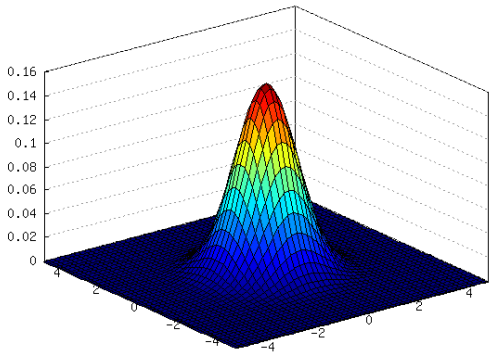


FIGURE 2 – Gauche : Exemple de plot 2-D Matlab, le script utilisé pour obtenir ce graphe est présenté dans la Table 1. Droite : plot 3-D Matlab de la fonction $f(x,y) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x^2+y^2)}{2}\right)$.

Instruction	Description
`plot(x,y)`	tracé de la courbe passant par les points (x,y)
`semilogy(x,y)`	idem avec échelle logarithmique sur l'axe des ordonnées
`semilogx(x,y)`	idem avec échelle logarithmique sur l'axe des abscisses
`loglog(x,y)`	idem avec échelle logarithmique sur les deux axes
`xlabel('label')`	label pour l'axe des abscisses
`ylabel('label')`	label pour l'axe des ordonnées
`title('title-of-figure')`	titre au dessus du graphique
`legend('str1','str2',...)`	légende avec une chaîne de caractères pour chaque courbe
`x,y,'label'`	chaîne de caractères à la position (x,y)
`plot3(x,y,z)`	tracé de la surface passant par les points (x,y,z)
`hold on, hold off`	active / désactive la conservation de la fenêtre graphique
`grid on`	rajoute une grille

TABLE 2 – Résumé des principales instructions graphiques. Voir également le script Table 1.

4 Vecteurs et Matrices

4.1 Vecteurs

4.1.1 Vecteur colonne et ligne

Vecteur colonne

```
>> v=[2;3;7.3]
v =
    2.0000
    3.0000
    7.3000
```

Vecteur ligne

Deux syntaxes possibles pour un vecteur ligne :

```
>> v=[2,-3,7]
v =
     2    -3     7

>> v=[2 -3 7]
v =
     2    -3     7
```

4.1.2 Transposée d'un vecteur

La transposée d'un vecteur ligne s'obtient é l'aide de l'apostrophe ' :

```
>> v=[1, 5, 4]
v =
     1     5     4
```

```
>> v'
ans =
     1
     5
     4
```

4.1.3 Somme de deux vecteurs

```
>> w=[-5 3 10]
w =
    -5     3    10
```

```
>> w+v
ans =
    -4     8    14
```

Remarque 2. Vous ne pouvez additionner que des vecteurs de même taille :

```
>> w=[-5 3 10];v=[1 ; 5; 4];v+w
??? Error using ==> plus
Matrix dimensions must agree.
```

4.1.4 Produit scalaire euclidien

Nous rappelons qu'étant donnés deux vecteurs lignes réels de taille n, $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$ leur produit scalaire $\langle x|y\rangle$ est donné par $\langle x|y\rangle = \sum_{i=1}^{n} x_i y_i$ mais est aussi égal à la multiplication matricielle du vecteur ligne x par le vecteur colonne y^T, i.e. $\langle x|y\rangle = x\, y^T$.

4.1.5 Opérations terme à terme

Produit des composantes terme à terme

```
>> v=[1 5 -3];w=[2 2 1];
>> v.*w
ans =
     2    10    -3
```

Division des composantes terme à terme

```
>> a=[4 3 10];b=[2 1 5];
>> a./b
ans =
     2     3     2
```

Mise à la puissance terme à terme

```
>> a=[4 3 10];a.^3
ans =
     64     27   1000
```

4.1.6 Vecteurs spéciaux

Vecteurs à incrément constant

```
>> v=[0:0.25:0.75]
v =
    0  0.2500  0.5000  0.7500
```

```
>> v=[1:2:10]
v =
    1    3    5    7    9
```

Pour créer un vecteur à incrément constant mais en spécifiant le nombre de point au lieu du pas de discrétisation on peut utiliser la fonction `linspace`.

```
>> linspace(-1,1,9)
ans =
-1.0000 -0.7500 -0.5000 -0.2500
0 0.2500 0.5000 0.7500 1.0000
```

Ones : vecteurs avec que des 1

```
>> ones(1,4)
ans =
     1     1     1     1
```

Zeros : vecteurs avec que des 0

```
>> zeros(1,4)
ans =
     0     0     0     0
```

4.2 Matrices

Les matrices suivent la même syntaxe que les vecteurs. Les composantes des lignes sont séparées par des virgules ou des espaces et une ligne est séparée de la suivante par un point virgule. Par exemple voici une matrice 3×3 :

```
>> A=[1 0 3 ; 2 1 4 ; 6 -1 2]
A =
     1     0     3
     2     1     4
     6    -1     2
```

Taille d'une matrice

```
>> size(A)
ans =
     3     3
```

Extraction de lignes ou colonnes

```
>> A(:,1)
ans =
       1
       2
       6
```

```
>> A(2,:)
ans =
     2     1     4
```

Accès aux éléments

```
>> A(2,3)
ans =
     4
```

4.2.1 Matrices spéciales

Matrice identité

```
>> eye(3,3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Les commandes `zeros(n,p)` et `ones(n,p)` sont les mêmes pour les matrices que pour les vecteurs.

4.2.2 Multiplication de matrices

Elle est identique à la multiplication de vecteurs.

4.2.3 Résumé des différentes fonctions sur les matrices

Notons tout d'abord que les fonctions scalaires comme `exp`, `sqrt` sont définies sur des matrices. Par exemple :

```
>> x=[0:pi/4:pi]
x =
0 0.7854 1.5708 2.3562 3.1416
```

```
>> sin(x)
ans =
0 0.7071 1.0000 0.7071 0.0000
```

Les fonctions principales sur les matrices sont résumées dans la Table 3.

`ones(n,p)` `eye(n,p)` `zeros(n,p)` `size(A)`	
`diag(u)`	crée une matrice carrée avec le vecteur u sur la diagonale et zero ailleurs
`diag(U)`	extrais la diagonale de la matrice U
`triu(A)`	renvoie la partie supérieure de A
`tril(A)`	renvoie la partie inférieure de A
`A\b`	résolution du système linéaire $Ax = b$
`det(A)`	déterminant d'une matrice
`rank(A)`	rang d'une matrice
`inv(A)`	inverse d'une matrice
`[V,D] = eig(A)`	diagonalisation de A : D matrice diagonale contenant les valeurs propres et B les vecteurs propres, i.e. $A * V = V * D$

TABLE 3 – Résumé des fonctions et opérations de base sur les matrices

5 Les boucles et tests

5.1 Boucle `for`

5.1.1 Syntaxe

La syntaxe générale pour une boucle for est l'une des suivantes

```
for index=j:k
  statements
end
```

```
for index=j:m:k
  statements
end
```

Il est possible d'utiliser la boucle `for` sur une seule ligne (pratique pour utiliser en ligne de commande). Dans ce cas, la syntaxe est la suivante

```
for index=j:m:k, statement; end
```

5.1.2 Eviter la boucle `for` en vectorisant

Du fait de sa nature interprétée, les grande boucles sont très pénalisantes pour MATLAB et Octave. Mais il est parfois (souvent) possible d'éviter une boucle `for` en utilisant la vectorisation naturelle de MATLAB. Par exemple, supposons que nous voulons évaluer $\sum_{i=1}^{10^7} 1/i^2$. Comparer les deux solutions suivantes :

```
tic
s=0;
for i=1:10^7, s=s+1/i^2; end
disp(['s=' num2str(s)]);
toc % display time CPU till tic
```

```
tic
i=1:10^7;
s=sum(1./i.^2);
disp(['s=' num2str(s)]);
toc
```

5.2 Boucle `while`

La syntaxe pour une boucle `while` est la suivante

```
while condition
   statements
end
```

et la syntaxe pour une boucle `while` en ligne est la suivante

```
while condition statements; end
```

5.3 `if-else`, `elseif`

La syntaxe pour l'utilisation de `if-else` est :

```
if condition
   statementsA
else
   statementsB
end
```

Il est possible d'utiliser la commande `if` en ligne, par exemple

```
r=rand(1,1);
if r> 0.5 disp('r > 0.5'); end
```

5.4 Opérateurs de relation $<, =, \ldots$

Les opérateurs de relation sont résumés dans la table suivante :

Opérat.	Signification
$a < b$	teste : a est strictement plus petit que b
$a <= b$	teste : a est plus petit ou égal à b
$a == b$	teste : a est égal à b
$a \tilde{} = b$	teste : a est différent de b
$a > b$	teste : a est strictement plus grand que b
$a >= b$	teste : a plus grand ou égal à b

5.5 Opérateurs logiques (AND, OR, NOT)

Les trois opérateurs logiques AND, OR et NOT s'écrivent en MATLAB :

Operateurs	Signification
~	NOT
&	AND
|	OR

6 Exercices

Exercice 1. Considérons la matrice

$$A = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$$

1. Calculer AA^T et A^{-1}.
2. Soit $u = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$ et $v = \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$. Calculer Au et Av, interpréter géométriquement le résultat.

Exercice 2. 1. Résoudre par deux méthodes différentes le système

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & -2 & 4 \\ 0 & -2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 3 \end{pmatrix}$$

2. Créer de manière aléatoire des matrices et des seconds membres de plus en plus grands, et appliquer les 2 méthodes en les chronométrant. Que constate-t-on ?
3. Tracer les courbes des temps de calcul de chacune des méthodes en fonction de la taille des matrices.

Exercice 3. **(i)** Écrire une fonction qui prend comme argument un vecteur x et retourne

$$f_{\text{sphere}}(x) = \sum_{i=1}^{n} x_i^2$$

où n est la taille du vecteur.
(ii) Supposons que $n = 2$, tracer les lignes d'iso-valeur de la fonction f_{sphere}.
(iii) Faites de même avec la fonction qui prend comme argument un vecteur x et retourne

$$f_{\text{ellipsoid}}(x) = \sum_{i=1}^{n} 100^{\frac{i-1}{n-1}} x_i^2$$

où n est la taille du vecteur.