Chapitre 4: Programmer sous Matlab

4.1 Opérateurs de comparaison et opérateurs logiques

a) Les opérateurs de comparaison sont:

= = : égale à (X= = Y)

> : Strictement plus grand que (X > Y)

< : Strictement plus petit que (X< Y)

>= : plus grand ou égale à (X>= Y)

<= : plus petit ou égale à (X<= Y)

~= : différent de (X~=Y)

b) Les opérateurs logiques sont:

& : et (X &Y)

| : ou (or) (X | Y)

- : Non X (-X)

4.2 Instructions de contrôle

4.2.1 Boucle for (parcours d'un intervalle)

Sa syntaxe est:

for indice = borne_inf : pas : borne_sup

Séquence d'instructions

end

Exemple faire un programme Matlab qui calcule la somme suivante $\sum_{i=3}^{n} i$

Solution:

```
n=input('donner la valeur de n');
s=0;
for i=3:n
   s=s+i;
end
disp('la somme s est:   '),s

%disp(['la somme s est: ',num2str(s)]
```

L'exécution:

```
>> ex1_matlab
donner la valeur de n6
la somme s est:
s =
   18
```

Exemple faire un programme Matlab qui calcule la somme suivante 1+1.2+1.4+1.6+1.8+2

Solution:

```
clear all
s=0;
for i=1:0.2:2
   s=s+i
end
disp('la somme s est:   '),s
```

L'exécution:

```
>> ex2_matlab
s =
     1
s =
   2.2000
s =
   3.6000
s =
   5.2000
s =
     7
s =
     9
la Somme s est:
s =
     9
```

4.2.2 Boucle While (tant que)

Sa syntaxe est:

while expression logique

Séquence d'instructions

end

Exemple: faire un programme sous matlab qui calcule la somme suivante:

S=1+2/2! +3/3!+… on arrête le calcule quand S>2.5

Solution:

```
clear all
s=1;i=1,f=1;
while s<=2.5
   i=i+1
   f=f*i;
   s=s+i/f
end
```

L' execution:

```
>> ex3_matlab
i =
     1
i =
     2
s =
     2
i =
     3
s =
    2.5000
i =
     4
s =
    2.6667
```

4.2.3 L'instruction if (si)

1er cas : Sa syntaxe est:

if *expression logique*

Séquence d'instructions

end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

1. y = x s x<0
2. $y=x^2$ si x>0
3. y=10 si x=0

Solution

```
clear all
x=input('introduire la valeur de x    ');
if x<0
   y=x;
end
if x>0
   y=x^2;
end
if x==0
   y=10;
end
disp('la valeur de y est:    '),y
```

L'exécution:

```
>> ex4_matlab
introduire la valeur de x    5
la valeur de y est:
y =
    25
```

2er cas :Sa syntaxe est:

if expression logique

Séquence d'instructions

else

Séquence d'instructions

end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

1. y = x s x<0
2. $y=x^2$ si x≥0

Solution

```
clear all
x=input('introduire la valeur de x    ');
if x<0
     y=x;
else
      y=x^2;
end
disp('la valeur de y est:    '),y
```

L'exécution:

```
>> ex4_matlab
introduire la valeur de x    5
la valeur de y est:
y =
   25
```

3er cas :Sa syntaxe est:

if *expression logique*
Séquence d'instructions
elseif expression logique
Séquence d'instructions
elseif expression logique
Séquence d'instructions
.
.
else
Séquence d'instructions
end

Exemple:

Faire un programme sous Matlab qui résout le problème suivant:

Y=x si x<0
Y=10 si x=0
Y=sqrt(x) si 0<x<20
$Y=x^2$ si x =20
$Y=x^3$ si x>20

Solution

```
clear all
x=input('introduire la valeur de x    ');
if x<0
   y=x;
elseif x==0
   y=10;
elseif x==20
   y=x^2;
elseif x>0 & x<20
   y=sqrt(x);
else
   y=x^3;
end
disp('la valeur de y est:    '),y
```

L'exécution:

```
>> ex5_matlab
introduire la valeur de x    60
la valeur de y est:
y =
     216000
```

4.2.4 L'instruction switch

Sa syntaxe est:

```
Switch  var
case cst-1
   Séquence d'instructions-1
case cst-2
   Séquence d'instructions-2
.
.
.
case cst-N
   Séquence d'instructions-N
otherwise
   Séquence d'instructions par défaut
end
```

var: est une variable numérique ou chaîne de caractère.
cst-1, cst-2….cst-N: sont des constantes numérique ou chaîne de caractères.
Si l'instruction à exécuter est la même pour un ensemble de cas alors la syntaxe est :
Case {cst-1, cst-2,…}

Exemples:

```
jj=input('donner le jour  1 :7  ');
switch jj
   case 1
      disp('samedi')
   case 2
      disp('dimanche')
   case 3
      disp('lundi')
   case 4
      disp('mardi')
   case 5
      disp('mercredi')
   case 6
      disp('jeudi')
   case 7
      disp('vendredi')
end
%
```

```
mm=input('donner le mois: ');
switch mm
   case 'Ja'
      disp('Janvier')
   case 'F'
      disp('Février')
   case 'M'
      disp('Mars')
   case 'Av'
      disp('Avril')
   otherwise
      disp('autre')
end
```

A l'exécution:

```
>> exp_switch
  donner le jour: 4
   mardi
  donner le mois: 'Av'
  Avril

>>
```

4.3 Instructions d'interruption d'une boucle

Il est possible de provoquer une sortie prématurée d'une boucle de contrôle.

- L'instruction **break** permet de sortir d'une *boucle for* ou d'une *boucle while*. En cas de boucles imbriquées, on interrompt seulement l'exécution de la boucle intérieure contenant l'instruction *break*.

- L'instruction **return** provoque un retour au programme appelant (ou au clavier). Les instructions suivant le *return* ne sont donc pas exécutées. L'instruction *return* est souvent utilisée conjointement avec une instruction conditionnée par exemple pour tester dans le corps d'une fonction si les paramètres d'entrée ont les valeurs attendues.

- L'instruction ***error*** permet d'arrêter un programme et d'afficher un message d'erreur. La syntaxe est : error(' message d''erreur ').

- L'instruction **warning** permet d'afficher un message de mise en garde sans suspendre l'exécution du programme. La syntaxe est warning(' message de mise en garde '). Il est possible d'indiquer à MATLAB de ne pas afficher les messages de mise en garde d'un programme en tapant warning off dans la fenêtre de commandes. On rétablit l'affichage en tapant warning on.

- *pause* : interrompt l'exécution jusqu'à ce que l'utilisateur tape un return

- *pause(n)* : interrompt l'exécution pendant *n* secondes.

- *pause off* : indique que les ***pause*** rencontrées ultérieurement doivent être ignorées, ce qui permet de faire tourner tous seuls des scripts requérant normalement l'intervention de l'utilisateur.

Exemples:

a- Commande break

```
S=0;
for i=1:10
    i
    S=S+i^2
    if S > 15
        break
    end
end
```

A l'exécution:

```
i =
     1
S =
     1
i =
     2
S =
     5
i =
     3
S =
    14
i =
     4
S =
    30
>>
```

b- Commande return

```
function somme(n)
%
if n < 0
  return
end
S=0;
for i=1:n
   S=S+i^2
end
```

A l'exécution:

```
>> somme(2)
S =
     1
S =
     5
>> somme(-2)

>>
```

→ sortie de la fonction

c- Commande *error*

```
clear all
s=2;
for i=1:5
   i
   s=s+i^2
   if s>5
      error('s est superieur à 5')
   end
end
```

A l'exécution:

```
i =
     1
s =
     3
i =
     2
s =
     7
??? Error using ==> myfile
s est superieur à 5
>>
```

d- Commande *warning*

```
clear all
% warning off/on        → afficher  ou  masquer warning
s=2;
for i=1:5
   i
   s=s+i^2
   if s>5
       warning('s est superieur à 5')
   end
end
```

A l'exécution:

```
>>
i =
    1
s =
    3
i =
    2
s =
    7
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
i =
    3
s =
   16
Warning: s est superieur à 5

> In C:\MATLAB6p5\work\exp.m at line 8
i =
    4
s =
   32
Warning: s est superieur à 5
> In C:\MATLAB6p5\work\exp.m at line 8
i =
    5
s =
   57
Warning: s est superieur à 5
> In C:\MATLAB6p5\work\exp.m at line 8
>>
```

e- Commande *pause*

```
clear all
s=2;
for i=1:5
   i
   s=s+i^2
   if s>5
       pause
   end
end
```

A l'exécution:

```
>>
i =
    1
s =
    3
i =
    2
s =
    7
|  <-------  Le curseur clignote ⇒ tapez Entrée
```

4.4 La programmation vectorielle

4.4.1 Manipulation des vecteurs

Les éléments d'un vecteur peuvent être manipulés grâce à leur indice dans le tableau. Le k-ieme élément du vecteur x est désignée par x(k). Le premier élément d'un vecteur a obligatoirement pour indice 1.

Il est possible de manipuler plusieurs éléments d'un vecteur simultanément. Ainsi les éléments *k* à *l* du vecteur x sont désignés par x(k:l). On peut également manipuler facilement les éléments d'un vecteur dont les indices sont en progression arithmétique. Ainsi si l'on souhaite extraire les éléments *k, k+p, k+2p,..., k+Np = l,* on écrira x(k:p:l). Plus généralement, si K est un vecteur de valeurs entières, X(K) retourne les éléments du vecteur X dont les indices sont les éléments du vecteur K.

Exemple :

```
>> X =[ 1     2     3     4     5     6     7     8     9    10] ;
>> X(5)
ans =
     5
>> X(4:10)
ans =
     4     5     6     7     8     9    10

>> X(2:2:10)
ans =
     2     4     6     8    10

>> K = [1 3 4 6]; X(K)
ans =
     1     3     4     6
>>
```

Il est très facile de définir un vecteur dont les composantes forment une suite arithmétique. Pour définir un vecteur x dont les composantes forment une suite arithmétique de raison *h*, de premier terme *a* et de dernier terme *b*, on écrira x = a:h:b. Si *a-b* n'est pas un multiple de *h*, le dernier élément du vecteur x sera *a + ent((a-b)/h) h* où e*nt* est la fonction partie entière.

```
>> x = 1.1:0.1:1.9
x =
  Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
  Columns 8 through 9
    1.8000    1.9000

>> x = 1.1:0.4:2
x =
    1.1000    1.5000    1.9000
```

La commande ***linspace*** permet de définir un vecteur x de longueur *N* dont les composantes forment une suite arithmétique de premier terme *a* et de dernier terme *b* (donc de pas *h*=(*b-a*)/(*N-1)*). Les composantes du vecteur sont donc *lin*éairement e*spacé*es.

La syntaxe est : `x = linspace(a,b,N)`.

```
>> x = linspace(1.1,1.9,9)
ans =
  Columns 1 through 7
    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000
  Columns 8 through 9
    1.8000    1.9000
```

- linspace(a,b) $\Rightarrow$ N=100 par défaut.

Vecteurs spéciaux

Les commandes ones, zeros et rand permettent de définir des vecteurs dont les éléments ont respectivement pour valeurs 0, 1 et des nombres générés de manière aléatoire.

ones(1,n) : vecteur ligne de longueur n dont tous les éléments valent 1

ones(m,1) : vecteur colonne de longueur m dont tous les éléments valent 1

zeros(1,n) : vecteur ligne de longueur n dont tous les éléments valent 0

zeros(m,1) : vecteur colonne de longueur m dont tous les éléments valent 0

rand(1,n) : vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1

rand(m,1) : vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1

4.4.2 Manipulation des matrices

Définir une matrice

On a déjà vu que l'on définissait la matrice : $A = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$ en tapant A = [1 3; 4 2]. D'une façon générale donc, on définit une matrice en donnant la liste de ses éléments entre crochets.

On peut construire très simplement une matrice << par blocs >>. Si A, B, C, D désignent 4 matrices (aux dimensions compatibles), on définit la matrice bloc :

$$K = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

par l'instruction K = [A B ; C D].

Voici un exemple de construction par blocs de la matrice

```
>> A11 = [35 1 6; 3 32 7; 31 9 2];
>> A12 = [26 19; 21 23; 22 27];
>> A21 = [ 8 28 33; 30 5 34];
>> A22 = [17 10; 12 14];
```

$$B = \left(\begin{array}{ccc|cc|c} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ \hline 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ \hline 4 & 36 & 29 & 13 & 18 & 11 \end{array}\right)$$

```
>> B11 = [ A11 A12; A21 A22 ]
B11 =
   35    1    6   26   19
    3   32    7   21   23
   31    9    2   22   27
    8   28   33   17   10
   30    5   34   12   14
>> B12 = [ 24   25   20   15   16]';
>> B = [ B11  B12];
>> B21 = [ 4   36   29   13   18   11];
>> B = [B ; B21]
B =
   35    1    6   26   19   24
    3   32    7   21   23   25
   31    9    2   22   27   20
    8   28   33   17   10   15
   30    5   34   12   14   16
    4   36   29   13   18   11
>>
```

Un élément d'une matrice est référencé par ses numéros de ligne et de colonne. A(i,j) désigne le $i^{ème}$ élément de la $j^{ème}$ ligne de la matrice A. Ainsi A(2,1) désigne le premier élément de la deuxième ligne de A,

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

Matrices spéciales

Certaines matrices se construisent très simplement grâce à des commandes dédiées. Citons les plus utilisées:

eye(n) : la matrice identité de dimension n

ones(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 1

zeros(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 0

rand(m,n) : une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1

magic(n) : permet d'obtenir une matrice magique de dimension *n*.

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

>> ones(3,2)
ans =
   1   1
   1   1
   1   1

>> zeros(2)
ans =
   0   0
   0   0
```

```
>> rand(2,3)
ans =
   0.4565   0.8214   0.6154
   0.0185   0.4447   0.7919

>> magic(3)
ans =
   8   1   6          la → somme
= 15
   3   5   7
   4   9   2
```

Le symbole deux-points (**:**) permet d'extraire simplement des lignes ou des colonnes d'une matrice.
Le $j^{ème}$ vecteur colonne de la matrice A est désigné par A(:,j). C'est simple, il suffit de traduire le symbole deux-points (:) par << tout >>. Ainsi A(:,j) désigne toutes les lignes et la $j^{ème}$ colonne de la matrice A. Bien entendu, la $i^{ème}$ ligne de la matrice A est désignée par A(i,:).

```
>> A = magic(5)
A =
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9
```

```
>>  A(1,:)
ans =
   17   24    1    8
15
>>  A(:,2)
ans =
   24
    5
    6
   12
   18
```

Exercice :
Comment échanger les colonnes 2 et 3 de la matrice A ?

```
>> v = A(:,2); A(:,2) = A(:,3); A(:,3) = v;
A =
   17    1   24    8   15
   23    7    5   14   16
    4   13    6   20   22
   10   19   12   21    3
   11   25   18    2    9
```

On peut également extraire plusieurs lignes ou colonnes simultanément.
Si J est un vecteur d'entiers, A(:,J) est la matrice issue de A dont les colonnes sont les colonnes de la matrice A d'indices contenus dans le vecteur J. De même A(J,:)est la matrice issue de A dont les lignes sont les lignes de la matrice A d'indices contenus dans le vecteur J. D'une façon plus générale, il est possible de n'extraire qu'une partie des éléments des lignes et colonnes d'une matrice. Si L et C sont deux vecteurs d'indices, A(L,C) désigne la matrice issue de la matrice A dont les éléments sont les A(i,j) tels que i soit dans L et j soit dans C.

```
>> A = magic(5)
A =
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9

>> L = [1 3 5]; C = [3 4];
```

```
>> A(L,C)
ans =
    1    8
   13   20
   25    2
>> A(1:2:5,3:4)
ans =
    1    8
   13   20
   25    2
>>
```

Il existe des commandes MATLAB permettant de manipuler globalement des matrices.

La commande diag permet d'extraire la diagonale d'une matrice: si A est une matrice, v=diag(A) est le vecteur composé des éléments diagonaux de A. Elle permet aussi de créer une matrice de diagonale fixée: si v est un vecteur de dimension n, A=diag(v) est la matrice diagonale dont la diagonale est v.

```
>> A=eye(3)
A =
    1    0    0
    0    1    0
    0    0    1

>> diag(A)
ans =
    1
    1
    1
```

```
>> v=[1:3]
v =
    1 2 3

>> diag(v)
 ans =
        1    0    0
        0    2    0
        0    0    3
>>
```

On n'est pas obligé de se limiter à la diagonale principale. La commande **diag** admet un second paramètre *k* pour désigner la *k* sur-diagonale (si *k*>0) ou la *k* sous-diagonale (si *k*<0).

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

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

```
>> diag(A,1)
ans =
    5
    5
    5
>> diag(A,-2)
ans =
    2
    2
>>
```

On dispose également de la commande **tril** permet d'obtenir la partie triangulaire inférieure (l pour lower) d'une matrice. La commande **triu** permet d'obtenir la partie triangulaire supérieure (u pour upper) d'une matrice.

```
>> A = [ 2   1   1 ; -1  2   1 ; -1  -1   2]
A =
    2    1    1
   -1    2    1
   -1   -1    2
```

```
>> triu(A)
ans =
    2    1    1
    0    2    1
    0    0    2
>> tril(A)
ans =
    2    0    0
   -1    2    0
   -1   -1    2
>>
```

Autres fonctions :

inv(A) : renvoie l'inverse de la matrice carrée A
det(A) : renvoie le déterminant de la matrice carrée A
A' : renvoie la transposée de la matrice A à coefficients réels
nnz(A) : renvoie le nombre d'éléments non nuls de la matrice A.

```
>> A = [3 1 2; -1 4 1; -2 -1 7]
A =
    3    1    2
   -1    4    1
   -2   -1    7
>> det(A)
ans =
  110
>> inv(A)
ans =
   0.2636   -0.0818   -0.0636
   0.0455    0.2273   -0.0455
   0.0818    0.0091    0.1182
>> A'
ans =
    3   -1   -2
    1    4   -1
    2    1    7
>> nnz(A)
ans =
    9
>>
```

La structure *sparse*

On appelle *matrice creuse* (*sparse matrix* en anglais) une matrice comportant une forte proportion de coefficients nuls. De nombreux problèmes issus de la physique conduisent à l'analyse de systèmes linéaires à matrice creuse. L'intérêt de telles matrices résulte non seulement de la réduction de la place mémoire (on ne stocke pas les zéros) mais aussi de la réduction du nombre d'opérations (on n'effectuera pas les opérations portant sur les zéros). Par défaut dans MATLAB une matrice est considérée comme *pleine* (ou *full* en anglais), c'est-à-dire que tous ses coefficients sont mémorisés.

Si *M* est une matrice, la commande **sparse(M)** permet d'obtenir la même matrice mais stockée sous la forme *sparse*. Si l'on a une matrice stockée sous la forme *sparse*, on peut obtenir la même matrice stockée sous la forme ordinaire par la commande **full**. Il est possible de visualiser graphiquement la structure d'une matrice grâce à la commande **spy**. Si *M* est une matrice, la commande **spy(M)** ouvre une fenêtre graphique et affiche sous forme de croix les éléments non-nuls de la matrice. Le numéro des lignes est porté sur l'axe des ordonnées, celui des colonnes en abscisse. On obtient également le nombre d'éléments non-nuls de la matrice.

Exemple :

```
>> N=5;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>> nnz(A)
ans =
    13
>> whos
  Name      Size            Bytes  Class
  A         5x5               200  double array
  B         5x5               180  sparse array
  N         1x1                 8  double array
Grand total is 39 elements using 388 bytes

>> spy(A)
```

```
>> B = sparse(A)
B =
   (1,1)       2
   (2,1)      -1
   (1,2)      -1
   (2,2)       2
   (3,2)      -1
   (2,3)      -1
   (3,3)       2
   (4,3)      -1
   (3,4)      -1
   (4,4)       2
   (5,4)      -1
   (4,5)      -1
   (5,5)       2
```