Chapitre 3

Ecriture de programmes Matlab

3.1 Les structures conditionnelles

3.1.1 Le teste conditionnel `if`

Ce test s'emploie, souvent, dans la plupart des programmes, il permet de réaliser une suite d'instructions si sa condition est satisfaite.

Le test `if` a la forme générale suivante : `if-elseif-else-end`.

```
if <condition 1>
 <instruction 1.1>
 <instruction 1.2>
 ...
elseif <condition 2>
 <instruction 2.1>
 <instruction 2.2>
 ...
...
else
 <instruction n.1>
 <instruction n.2>
 ...
end
```

où <condition 1>, <condition 2>, ... représentent des ensembles de conditions logiques, dont la valeur est `vrai` ou `faux`. La première condition ayant la valeur 1 entraîne l'exécution de l'instruction correspondante.

Si toutes les conditions sont fausses, les instructions <instruction n.1>,<instruction n.2>, ... sont exécutées.

Si la valeur de <condition k> est zéro, les instructions <instruction k.1>, <instruction k.2>, ...ne sont pas exécutées et l'interpréteur passe à la suite.

Exemple 1

```
>> V=268.0826;
>> R=4;
>> if V>150, surface=pi*R^2, end
```

On a la valeur de V >150 (268.0826>150) Alors l'exécution on a :

```
surface =
         50.2655
```

Exemple 2 Pour calculer les racines d'un trinôme de second degré, $ax^2 + bx + c$, on met les instructions suivantes :

```
if a ~= 0
    sq = sqrt(b*b - 4*a*c);
    x(1) = 0.5*(-b + sq)/a;
    x(2) = 0.5*(-b - sq)/a;
elseif b ~= 0
    x(1) = -c/b;
elseif c ~= 0
    disp('Equation impossible');
else
    disp(' L''equation est une egalite');
end
```

Remarques

- La commande disp(' ') affiche simplement ce qui est écrit entre crochets.
- La double apostrophe sert à représenter une apostrophe dans une chaîne de caractères. Ceci est nécessaire car une simple apostrophe est une commande Matlab.

3.1.2 La clause 'switch'

Une clause `switch` exécute un block d'instructions parmi plusieurs choix si sa condition est satisfaite. A chaque choix est attribué un `case`.

```
switch < switch expression>
  case <case expression>
       <statements>
  case <case expression>
       <statements>
          ...
          ...
  otherwise
     <statements>
 end
```

Exemple :

```
grade = 'A';
switch(grade)
    case 'A'
        disp('Excellent' );
    case 'B'
        disp('Trés bien\n' );
    case 'C'
        disp('Bien\n' );
    case 'D'
        disp('Passable\n' );
    case 'F'
        disp('Essaye encore\n' );
    otherwise
        disp('Invalid grade\n' );
end
```

Imbrication de switch

Une switch peut faire partie d'autres switch :

```
switch(ch1)
    case 'A' fprintf('Cet A appartient à switch extérieur');
        switch(ch2)
            case 'A' fprintf('Cet A appartient à switch intérieur' );
            case 'B' fprintf('Cet B appartient à switch intérieur' );
        end
    case 'B' fprintf('Cet B appartient à switch extérieur');
end
```

Exemple : Créer un script avec le code suivant :

```
a = 100;
b = 200;
switch(a)
    case 10
        fprintf('This is part of outer switch %d\n', a );
    switch(b)
        case 200
            fprintf('This is part of inner switch %d\n', a );
    end
end
fprintf('Exact value of a is : %d\n', a );
fprintf('Exact value of b is : %d\n', b );
```

L'execution de ce programme donne :

```
This is part of outer switch 100
This is part of inner switch 100
Exact value of a is : 100
Exact value of b is : 200
```

3.2 Les boucles

3.2.1 La boucle for

Une boucle `for` répète des instructions pendant que le compteur de la boucle balaie les valeurs rangées dans un vecteur ligne.

Exemple Pour calculer les 6 premiers termes d'une suite de Fibonacci $f_i = f_{i-1} + f_{i-2}$,avec $f_1 = 0$ et $f_2 = 1$, on peut utiliser les instructions suivantes :

```
>> f(1) = 0; f(2) = 1;
>> for i = 3:6
f(i) = f(i-1) + f(i-2);
end
>> f

f =

     0     1     1     2     3     5
```

Remarque
- L'utilisation du point-virgule (;) permet de séparer plusieurs instructions Matlab entrées sur une même ligne.
- On pourrait remplacer la seconde instruction par : `» for i = [3 4 5 6]`.
- Matlab n'exécute l'ensemble du bloc de commandes qu'une fois tapé `end`.

3.2.2 La boucle 'while'

La boucle `while` répète un bloc d'instructions tant qu'une condition donnée est vraie.

Exemple Les instructions suivantes ont le même effet que les précédentes :

```
>> f(1) = 0; f(2) = 1; k = 3;
>> while k <= 6
   f(k) = f(k-1) + f(k-2); k = k + 1;
   end
```

Remarque
- Le compteur 'k' est ajouté ici, ce compteur doit être initialisé et incrémenté pour assurer la condition d'arrêt de la boucle while.

3.3 Les scripts

Un nouveau programme Matlab doit être placé dans un fichier, appelé m-fichier, dont le nom comporte l'extension .m . Les programmes Matlab peuvent êtres des scripts ou des fonctions. Un script est simplement une collection de commandes Matlab, placée dans un m-fichier et pouvant êtres utilisée interactivement. Un script n'accepte pas des entrée, et ne retourn aucune sortie. Les fonctions sont aussi stockées dans des m-fichiers, les fonctions peuvent recevoir des entrées et retourner des sorties. Ils peuvent aussi avoir des variables locales.

Création et exécution de scripts

Pour créer un un fichier script, on utilise l'editeur de texte de Matlab, comme on peut utiliser n'importe quel editeur de texte :
- Utilisant la ligne de commande prompt >>.
- Utilisant l'interface graphique IDE(Integrated Development Environment).

Si vous utilisez la ligne de commande, taper `edit` dans ligne de commande, l'editeur de commande de MATLAB s'ouvre. Comme on peut taper le nom de fichier directement apres `edit`.

```
>> edit
ou
>> edit <file_name>
```

La commande précédente permet de créer le script dans le répertoire MATLAB par défaut. Or, dans le cas où on veut stocker tous les programmes dans un dossier spécifier on doit donner le chemin.

Exemple : Pour créer un dossier "progs".

```
>> mkdir progs % creer un dossier progs
>> chdir progs % changer le dossier courant vers progs
>> edit prog1.m % creer un m file nommee prog1.m
```

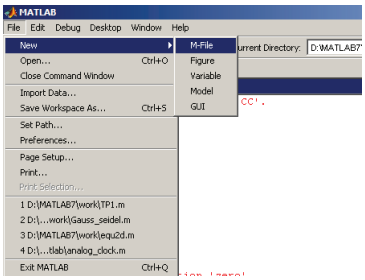

FIGURE 3.1 – Création de script avec l'interface graphique IDE

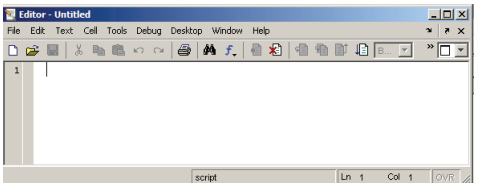

FIGURE 3.2 – Editeur de texte Matlab

Après la création la saisie des instructions et la sauvegarde du fichier, on peut l'éxécuter avec :

1. Avec l'IDE en cliquant sur le boutton `Run` dans la fenêtre de l'editeur.
2. Avec la ligne de commande prompt, en tapant le nom de fichier (sans l'extension), exemple :

```
>> tp1
```

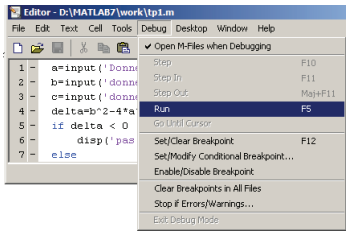

FIGURE 3.3 – Execution de script avec l'IDE

Exemple :
Pour le polynôme : $g(x) = \frac{2x^3+7x^2+3x+1}{x^2-3x+5.e^{-x}}$
On peut écrire dans un script, qu'on choisit de nommer TP, comme suit :

```
>>edit TP
```

ensuite on tape sur le script :

```
 x=3 ;
g=(2*x^3+7*x^2+3*x-1)/(x^2-3*x+5*exp(-x));
```

ce script est enregistré dans le fichier TP.m.
Pour le lancer, on écrit simplement l'instruction TP après le prompt >> Matlab.

```
>> TP
g=
   502.1384
```

3.4 Les fonctions

Une fonction est aussi définie dans un m-fichier qui commence par une ligne de la forme : `function [out1,... ,outn]=name(in1 ,... ,inm)` .
Ou :

- `out1,...,outn` sont les variables de sortie sur lesquels les résultats de la fonction sont retournés ;
- `in1,...,inm` sont les variables d'entrée, qui sont nécessaire à la fonction pour accomplir ses calculs.

Le nom du fichier script doit être le même nom de la fonction.

Exemple 1 On définit une fonction `determ` , qui calcule le déterminant d'une matrice carrée d'ordre 2, on va créer un fichier `determ.m`, avec le code suivant :

```
function det=determ(A)
%Cette fonction calcule le determinant
%seulement d'une matrice caree 2*2
[n,m]=size(A);
if n==m
  if n==2
    det = A(1 ,1)*A(2,2)-A(2 ,1)*A(1 ,2);
  else
    disp(' Seulement des matrices 2x2 ');
  end
else
   disp(' Seulement des matrices carr\'ees ');
end
return
```

La première ligne de la fonction commence avec le mot clé `function`. Ensuite on a une variable de sortie `det` et une variable d'entrée A . La ligne suivante est un commentaire, qui permet d'afficher un aide sur cette fonction, comme suit :

```
 help determ
```

Et MATLAB retourne :

```
 cette fonction calcule le determinant seulement d'une matrice caree 2*2
```

Pour appeler la fonction, on doit entrer en premier une matrice A avec la ligne de commande :

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

A =

     1     2
     3     4
```

Ensuite La fonction `determ` est appelée depuis la ligne de commande ainsi :

```
>> det=determ(A)

det =

    -2
```

Exemple 2 La fonction suivante permet de calculer les solutions d'une équation de 2^{eme} degrée :

```
function [x1,x2]= degre2(a,b,c)
delta=b^2-4*a*c;
if delta < 0
    disp ('Pas de solution ...')
else
        x1= (-b-sqrt(delta))/(2*a);
        x2= (-b+sqrt(delta))/(2*a);
end
end
```

Cette fonction doit etre enregistrer sur le fichier : degre2.m .
Voici deux exécutions en ligne de commande :

```
>> [r1,r2]=degre2(1,3,1)
r1 =
   -2.6180
r2 =
   -0.3820
>> [r1,r2]= degre2 (1,1,1)
Pas de solution ...
```

3.4.1 Fonction principale et sous fonctions

Chaque fichier de fontion doit contenir une fonction principale, et n'importe quel nombre de sous fonctions, qui figurent après la fonction principale, quelle les utilise. Les fonctions principales peuvent êtres appelés de l'extèrieur du fichier dans lequel ils sont définies, soit depuis la ligne de commande ou depuis d'autres fonctions, bien que les sous fonctions ne peuvent êtres appelés qu'à partir du fichier de la fonction. Les sous fonctions sont visibles seulement pour la fonction principale, et les autres sous fonctions dans le même fichier où il sont définies.

Example : Soit la fonction `equa2` qui calcul les racines d'une équation d'ordre 2. Le fichier du fonction equa2.m, va contenir la fonction principale `quadratic` et une sous fonction `dics` qui calcule le discriminant, comme suit :

```
function [x1,x2]= equa2(a,b,c)
%cette fonction retourn les racines
% d'une equation de 2 eme degree.
d = disc(a,b,c);
 if d<0
  disp('Pas de solutions');
 else
  x1 =(-b + sqrt(d))/(2*a);
  x2 =(-b - sqrt(d))/(2*a);
end
```

```
end   %end de la fonction principale

function dis = disc(a,b,c)
%fonction qui calculate le discriminant
 dis = b^2-4*a*c;
end  %end de la sous fonction
```

L'appel de la fonction depuis la ligne de commande :

```
[x1,x2]= equa2(2,4,-4)
```

MATLAB exécute les intructions précédengtes et retourne le résultat :

```
x1 =
    0.7321
x2 =
   -2.7321
```

3.4.2 Fonctions imbriquées

Les fonctions imbriquées sont des fonctions définies dans le corps d'autres fonctions.

Exemple : reprennat la fonction `equa2` précédente :

```
function[x1,x2]= equa2(a,b,c)
 function disc % fonction imbriquée
  d = sqrt(b^2-4*a*c);
 end    %end de la fonction disc
disc;
x1 =(-b + d)/(2*a); x2 =(-b - d)/(2*a);
end     %end of fonction principale equa2
```

Cette fois-ci la fonction `disc` est à l'intérieur de `equa2`.