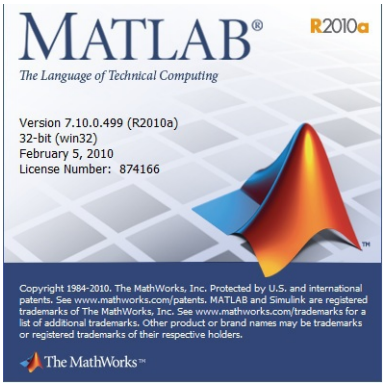
Chapitre 03 : Programmation sous Matlab

Université de M'sila



Dr. Chouder. R

M.C.B. université de M'sila

12 / 03/ 2020

Table des matières

Introduction

- Matlab peut être utilisé comme un langage de programmation évolué.
- On peut écrire :
    1. des scripts (fichiers de commandes)
    1. ou définir de nouvelles fonctions
- Ces scripts et fonctions peuvent utiliser les fonctions built-in de Matlab.

Les Scripts

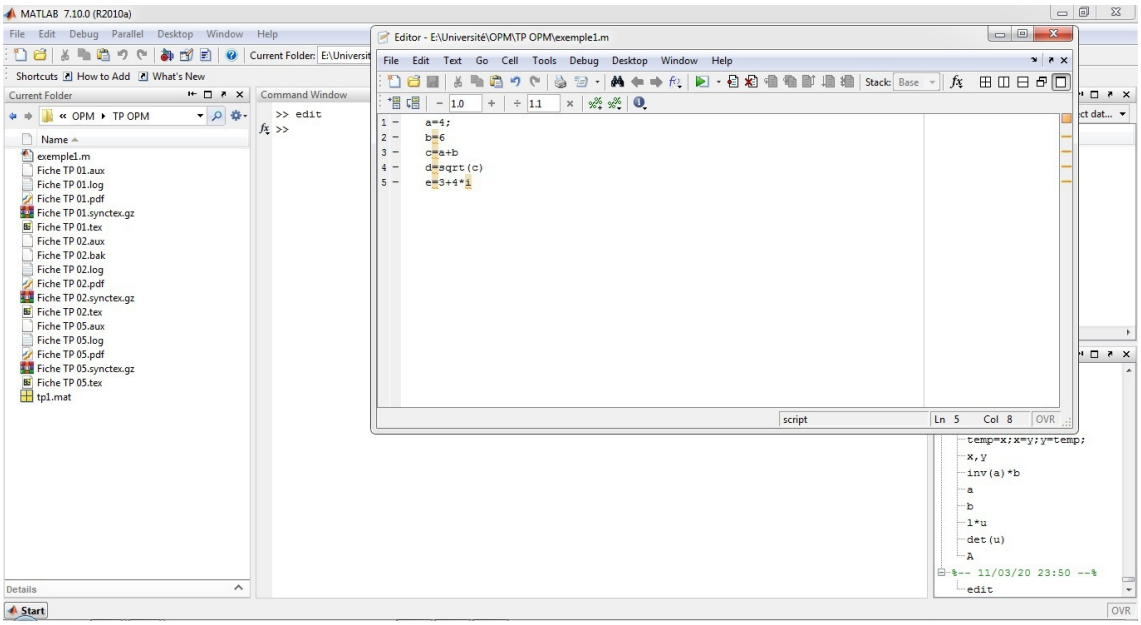
- Un script, ou un programme, est une suite d'instructions Matlab.
- Ces instructions sont écrites dans un fichier texte, enregistré avec l'extension .m
- Pour exécuter un script, il faut évoquer son nom de fichier dans la ligne de commande de Matlab.
- Les instructions du scripts s'exécutent l'une après l'autre comme si elles étaient saisies sur la ligne de commande de Matlab.
- Les variables définies dans le scripts restent dans la mémoire de Matlab après l'exécution du script.
- Le texte venant après le signe % est un commentaire, donc il n'est pas traité par Matlab.

Exemple : Exemple de script

- On appelle l'éditeur de Matlab avec la commande

  `>> edit`

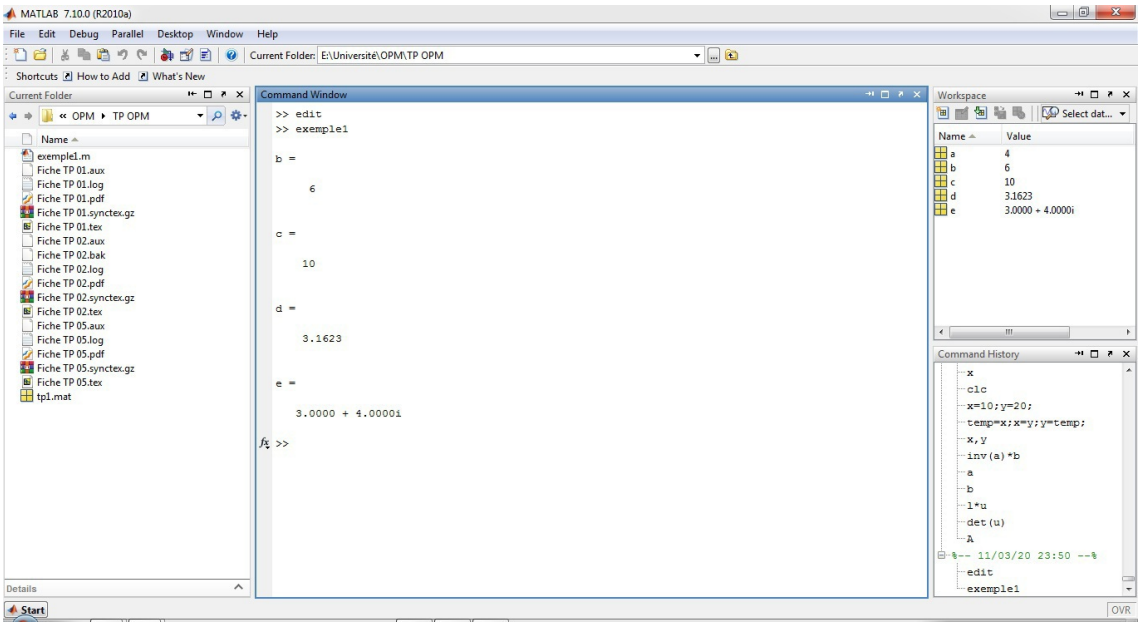
  ou bien, en utilisant le menu *File* → *New* → *Script*
- La fenêtre de l'éditeur s'affiche.
- On saisit les instructions, et on sauvegarde le fichier.



Pour exécuter le script *exemple1*, on revient à la fenêtre de commandes, et on saisit :

```
>> exemple1
```

On obtient ce qui suit :



Pour ouvrir le script *exemple1*, tapons la commande :

```
>> edit exemple1
```

Les structures de contrôle

II

Les structures de contrôle sous Matlab sont très proches de celles existant dans d'autres langages de programmation tels que le langage C. Elles sont utiles pour écrire des programmes Matlab.

1. Structure conditionnelle

Une structure conditionnelle permet d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée. Différentes formes de structures conditionnelles existent sous Matlab.

1.1. Forme simple

Syntaxe : Syntaxe:

if `condition`

`Instructions`

end

- `condition` : est une expression logique dont le résultat peut être vrai ou faux.
- `Instructions` : est une suite d'instructions.
- Si le résultat de l'évaluation de `condition` est vraie, on exécute `Instructions` , puis l'instruction qui suit le mot clé *end* .
- Si le résultat de l'évaluation de `condition` est faux on passe directement à l'instruction qui suit le mot clé *end*.

Exemple : Structure conditionnelle : Forme simple

```
somme   =  input('La somme de 5 et 4 est  :   ')
```

```
if (somme==9)
disp('Réponse correcte')
end
```

1.2. Forme alternative

Syntaxe : Syntaxe:

if `condition`

`Instructions_1`

else

`Instructions_2`

end

- `condition` est une expression logique dont le résultat peut être vrai ou faux.
- `Instructions_1` et `Instructions_2` sont deux suites d'instructions.
- Si le résultat de l'évaluation de `condition` est vraie, on exécute `Instructions_1`, puis l'instruction qui suit le mot clé *end*.
- Si le résultat de l'évaluation de `condition` est faux , on exécute `Instructions_2`, puis l'instruction qui suit le mot clé *end*.

Exemple : Structure conditionnelle : Forme alternative

```
if (x ~= 0)
y = 1/x ;
else
error('Division par zero');
end
```

1.3. Forme imbriquée

Syntaxe : Syntaxe:

if `condition_1`

`Instructions_1`

elseif `condition_2`

`Instructions_2`

elseif `condition_3`

`Instructions_3`

```
...
else
Instructions_n
end
```

- `condition_i` : est une expression logique dont le résultat peut être vrai ou faux.
- `Instructions _i :` est une suite d'instructions.
- Si le résultat de l'évaluation de `condition_i` est vraie, on exécute `Instructions _i` , puis l'instruction qui suit le mot clé *end*.
- Si aucune des expressions `condition_1, condition_2`,...,n'est vraie, on exécute l'instruction `Instructions_n` (suite d'instructions par défaut), puis l'instruction qui suit le mot clé *end*.
- Il n'est pas nécessaire de prévoir un cas par défaut (bien que cela soit préférable).
- S'il n'y a pas de cas par défaut, et si aucune des expressions `condition_1, condition_2` ,..., n'est vraie , alors on continue à la première instruction suivant le mot clé *end*.

Exemple : Structure conditionnelle : Forme imbriquée

```
age=input('Entrez votre âge : ');
if (age < 2)
disp('Vous êtes un bébé')
elseif (age < 13)
disp('Vous êtes un enfant')
elseif (age < 18)
disp ('Vous êtes un adolescent')
elseif (age < 60)
disp ('Vous êtes un adulte' )
else
disp ('Vous êtes un vieillard')
end
```

2. Exercice

Question

Écrire un programme qui trouve les racines d'une équation de second degré désigné par : $ax^2 + bx + c = 0.$

3. Choix multiple – instruction switch

Syntaxe : Syntaxe:

switch `var`

case `const_ 1,`

`instructions_1`

case `const_2,`

`instructions_2`

`...`

otherwise

`Instructions_n`

end

- `var` est une variable numérique ou une variable chaîne de caractères.
- `const_i` est une constante numérique ou des constantes chaînes de caractères de même type que `var`.
- `Instructions_i` est une suite d'instructions.
- si la variable `var` est égale à la constante `const_i` , on exécute la suite d'instructions correspondante (c'est -à -dire `Instructions_i` , puis l'instruction qui suit le mot clé *end*.
- si `var` n'est égale à aucune des constantes `const_1, const_2,...`, on exécute l'instruction `Instructions_n` (suite d'instructions par défaut) , puis l'instruction qui suit le mot clé *end*.
- Il n'est pas nécessaire de prévoir un cas par défaut (bien que cela soit préférable).
- S'il n'y a pas de cas par défaut, et si `var` n'est égale à aucune des constantes `const_i`, alors on continue à la première instruction suivant le mot clé *end*.

Exemple : Choix multiple – instruction switch

```
x = input ('Entrez un nombre : ') ;
switch x
case 0
disp('x = 0 ')
case 10
disp('x = 10 ')
case 100
disp('x = 100 ')
otherwise
```

```
disp('x   n''est pas 0 ou 10 ou 100 ')
end
```

4. Les boucles

Les boucles permettent d'exécuter une séquence d'instructions de manière répétée.

4.1. La boucle for

Syntaxe : Syntaxe:

```
for indice = inf: sup
Instructions
end
```

- `indice` : est une variable appelée l'indice de la boucle.
- `inf` (borne inférieure) et `sup` (borne supérieure) sont deux constantes réelles.
- `Instructions` est l a suite d'instructions à répéter (On parle du corps de la boucle) .
- Si *inf* ≤ *sup* , `Instructions` est exécutée (`supinf+1`) fois, pour les valeurs de la variable `indice` égales à `inf, inf+1,...,sup` , puis en passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (*end*).
- Si *inf* > *sup* , on passe directement à l'instruction qui suit immédiatement l'instruction de fin de boucle (*end*).
- L'indice de boucle ne prend pas nécessairement des valeurs entières.
- On peut naturellement imbriquer des boucles for les unes dans les autres.
- Il est possible d'utiliser un incrément (pas) autre que 1 (valeur par défaut). La syntaxe est alors :
  ```
  for indice = inf:pas:sup
  Instructions
  end
  ```
- Le pas peut être négatif.

Exemple : Calcul du factoriel

```
n=input('Entrée la valeur de l'entier n :') ;
fact = 1;
for k = 1:n
fact = fact*k;
end;
disp(fact)
```

4.2. La boucle while

Syntaxe : Syntaxe:

while `condition`

`Instructions`

end

- La boucle *while* permet de répéter une suite d'instruction tandis qu'une expression logique est vraie .
- `condition` : est une expression logique (ayant deux valeurs vrai ou faux).
- `Instructions` : est une suite d'instructions qui se répète tant que `condition` a la valeur vrai.
- Lorsque la valeur de `condition` devient faux, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (*end*).
- `condition` est en général le résultat d'un test (par exemple `i < 10`) ou le résultat d'une fonction logique (par exemple `~empty(x)`).

Exemple : Calcul du factoriel

```
n=input('Entrée la valeur de l'entier n :') ;
fact = 1;
k = 1 ;
while k <= n
fact = fact*k;
k = k+1;
end;
disp(fact)
```

Remarque : Interruption d'une boucle:

L'instruction *break* permet de sortir d'une boucle *for* ou d'une boucle *while*. L'exécution se poursuit alors séquentiellement à partir de l'instruction suivant le mot clé *end* fermant la boucle. En cas de boucles imbriquées, on interrompt seulement l'exécution de la boucle intérieure contenant l'instruction break.