République algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de JIJEL
Faculté des Sciences Exactes et Informatique
Département d'informatique

Les Outils de Programmation pour les Mathématiques (OPM)

L1 Informatique/Mathématiques/ MI

Introduction

Jusqu'ici, nous avons toujours considéré que l'on travaillait dans la fenêtre de commandes, en créant des variables dans l'espace mémoire de MATLAB (le *workspace*), et en utilisant différentes fonctions prédéfinies.

Cette approche est très pratique pour démarrer, ou pour réaliser de petits calculs, qui ne nécessitent pas trop de lignes de commandes. Mais elle n'est pas du tout appropriée dès que l'on s'attaque à des problèmes plus complexes, qui demandent des manipulations de commandes plus structurées ou plus nombreuses.

Nous allons voir dans ce qui suit, comment utiliser MATLAB comme un véritable langage de programmation, en passant des fichiers de commandes (que l'on peut sauvegarder et donc réutiliser), en écrivant nos propres fonctions et en utilisant des structures de contrôle.

Scripts, fichiers de commandes

Un Script est un fichier de commandes MATLAB portant une extension '.m'.

Ce type de fichier est un simple fichier texte, qui comporte des commandes MATLAB, identiques à celles que l'on peut employer directement dans la fenêtre de commandes de MATLAB.

L'avantage évident est que ces commandes sont stockées dans un fichier, il est donc aisé de les rappeler pour les réexécuter, même dans une nouvelle session de MATLAB (c'est-à-dire après fermeture et réouverture de MATLAB).

Scripts, fichiers de commandes

- Les fichiers script Servent à définir des:

✓ commandes qui se répètent très fréquemment

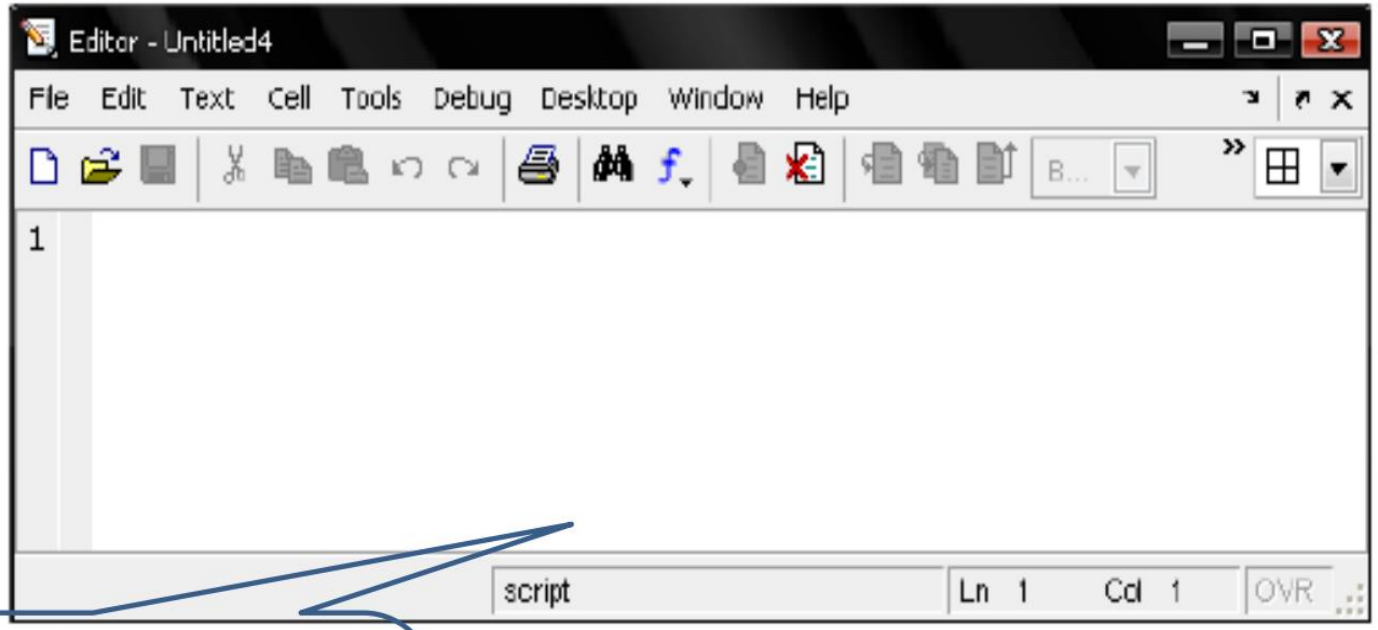
✓ ou des matrices de grande taille.

- Les variables définies dans un fichier script

✓ sont globales

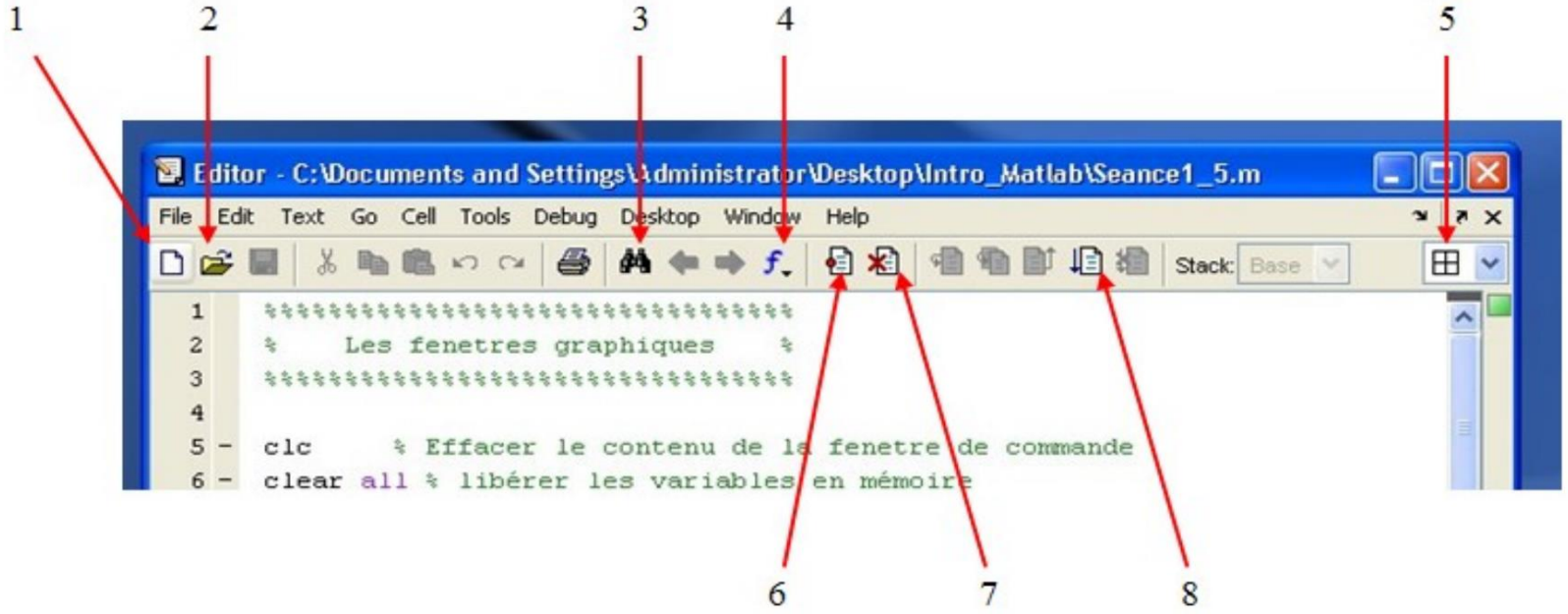
✓ et peuvent écraser les variables qui portent les mêmes noms dans le workspace.

- M-file: Ce sont des programmes en langage MATLAB écrits par l'utilisateur
- Pour créer un M-file : Aller dans le menu *File→New→ M - File* ou cliquez sur le bouton de la page blanche  (L'enregistrement se fait normalement dans le répertoire courant).
- Une fenêtre d'édition comme celle-ci va apparaitre



Ecrire votre programme dans cette fenêtre, puis l'enregistrer avec un nom (par exemple : '**tp.m**').

Les principales commandes d'un M-File :



(1) Créer un nouveau M-File
(2) Ouvrir un M-file existant
(3) Chercher-Remplacer dans le M-file ouvert
(4) Chercher une fonction dans le M-file
(5) Modifier l'apparence de la fenêtre « Editor »
(6) Placer un 'breakpoint' dans le M-File (nécessaire pour faire tourner le débogueur)
(7) Enlever tous les 'breakpoint' du M-File
(8) Exécuter le M-File

Scripts, fichiers de commandes

- Pour exécuter un fichier, on utilise la commande :

>> *nomdefichier* (*nomdefichier* est le nom du fichier).

- Pour retourner à la fenêtre d'édition (après l'avoir fermer) il suffit de saisir la commande :

>> edit *nomdefichier*

Les commentaires

En pratique, un commentaire peut servir à apporter des informations permettant une meilleure compréhension des lignes de commandes pour soi, ou pour ses collègues.

Tout ce qui se trouve après le symbole % sera considéré comme un commentaire. Il sera donc ignoré lors de l'exécution du script.

On peut aussi mettre en commentaire une ligne de commande que l'on souhaite ignorer dans l'exécution.

Structures de contrôle de flux

Les structures de contrôle de flux sont des instructions permettant de définir et de manipuler l'ordre d'exécution des tâches dans un programme.

MATLAB compte huit structures de contrôle de flux à savoir :

-If
- break
- switch
- try - catch
- for
- return
- while
-continue

Le mot clé if - exécuter du code sous condition

Le mot clé if

Le mot clé if (qui est bien un mot clé et non pas une fonction) permet d’exécuter du code si une condition est vraie.

Ce mot clé est utilisé dans la plupart des langages de programmation existant aujourd’hui : il est donc nécessaire de comprendre ce qu' il fait, car il n'est pas limité qu’au langage Matlab.

La syntaxe du if peut légèrement varier d’un langage à un autre, mais son sens reste le même.

Syntaxe du if

Voici la syntaxe générale de ce que l'on appelle « un bloc if » :

```
//Bloc de code 0

if <condition 1>
      //Bloc de Code 1
end

//Bloc de Code 2
```

Tout le **Bloc de code 0** est lu et exécuté. A l'issue de son exécution, la **condition 1** est évaluée.

Si elle est vraie, le **Bloc de code 1** est exécuté, puis le **Bloc de code 2.** Sinon, c'est uniquement le **Bloc de code 2** qui est exécuté.

Un premier programme avec if

Ala fin de ce programme, que vaudra la variable y ?

```
x=input('Entrez une valeur svp');
y=2;

if x<4
    y=0;
end
```

A la fin du programme, la variable y :

- sera mise à 0 si la valeur de x est inferieure à 4
- gardera sa valeur initiale sinon

Un exemple

Quelle fonction mathématique ce code permet-il de réaliser ?

```
x = input('Entrez une valeur : ');

if x<0
    x = -x;
end

disp(x)
```

Ce code permet de réaliser la valeur absolue :

Si x est négatif, il est multiplié par -1,

Puis, quoiqu'il soit arrivé auparavant, x est affiché

L' écriture de conditions

On peut utiliser différents symboles pour construire une condition.

Symbole	Description	Exemple
&&	Combine deux conditions avec un "et logique".	(A > 3) & (A<8)
||	Combine deux conditions avec un "ou logique".	(A > 6) | (A < 3)
~	Inverse une condition	~((A > 3) & (A < 4))

Exemple :

```
x = input('Entrez une valeur : ');
y = input('Entrez une autre valeur : ');

if x>0 && y>0
    disp('Les deux valeur entrées sont positives')
end
```

Le mot clef “elseif”

Le mot clef **elseif,** qui accompagne toujours un **if,** permet d’exécuter du code si les conditions du **if** ou des autres **elseif** situés auparavant étaient fausses, et que sa propre condition est vraie.

Comme pour le **if,** ce mot clef est présent dans la plupart des langages de programmation modernes, même si sa syntaxe peut différer quelque peu selon le langage.

Syntaxe du if/elseif

Voici la syntaxe générale de ce que l'on appelle « un bloc if » avec **elseif** :

```
//Bloc de code 0

if <condition 1>
    //Bloc de Code 1
elseif <condition 2>
    //Bloc de Code 2
elseif <condition 3>
    //Bloc de Code 3
elseif ...
    //...
end

//Bloc de Code n
```

Tout le **Bloc de code 0** est exécuté. Ensuite, la **condition 1** est évaluée.

Si elle est vraie, le **Bloc de code 1** est exécuté, puis le **Bloc de code n.**

Sinon, si la condition 2 est vraie, le **Bloc de code 2** est exécuté, puis le **Bloc de code n.**

Sinon, si la condition 3 est vraie, le **Bloc de code 3** est exécuté, puis le **Bloc de code n.**

...

Sinon, **si toutes les conditions sont fausses**, c'est **uniquement le Bloc de code n** qui est exécuté.

Syntaxe du if/elseif

Questions

```
//Bloc de code 0

if <condition 1>
    //Bloc de Code 1
elseif <condition 2>
    //Bloc de Code 2
elseif <condition 3>
    //Bloc de Code 3
elseif <condition 4>
    //Bloc de Code 4
end

//Bloc de Code n
```

A quelles conditions le bloc de code 4 s'exécutera ?

Si les conditions 1, 2 et 3 sont fausses, et que la condition 4 est vraie

Est-ce que les blocs 3 et 4 peuvent être exécutés l'un après l'autre ?

Non, c'est le principe du elseif : soit le 3, soit le 4 sera exécuté.

Syntaxe du if/elseif

Questions

```
a=input('Entrez votre note de contrôle Matlab');

if a<14
    disp('Vous n'êtes pas très bon');
elseif a>=15
     disp('Pas mal');
elseif a>18
     disp('Vous pourrez passer en deuxième année');
elseif a<4
     disp('Il y a des places de libres en MACS...');
```

A quelles conditions le programme invitera l'utilisateur à aller en MACS ?

Si on n'a pas a<14, ni a>=15, ni a>18, et que l'on a a<4.

C'est impossible, ça n'arrivera jamais…

Que se passe-t-il si a vaut 14.5 ?

Le programme n'affiche rien, car aucune condition n'est satisfaite.

Autre chose ?

Oui, il manque le mot clef **end** à la fin du bloc if ! Matlab affichera une erreur !

Un premier programme avec elseif

A votre avis, à la fin de ce programme, que vaudra la variable y ?

```
x=input('Entrez une valeur svp');
y=2;

if x<4
    y=0;
elseif x>9
    y=1;
end
```

A la fin du programme, la variable y

- sera mise à 0 si la valeur de x est inférieure à 4
- sera mise à 1 si la valeur de x est supérieure à 9
- gardera sa valeur initiale sinon

Un second programme avec elseif

II est tout a fait possible, après un if, de placer plusieurs elseif. Qu'est ce que ce programme affichera ?

```
a=input('Entrez votre age');

if a>=100
    disp('Vous êtes assez âgé');
elseif a>=18
     disp('Vous êtes majeur');
elseif a>=16
     disp('Vous pouvez conduire (accompagné)');
end
```

Ici, ce programme affichera

- Que la personne est âgée si elle a 100 ans ou plus.
- Sinon, que la personne est majeure si elle a 18 ans ou plus.
- Sinon, que la personne peut conduire si elle a 16 ans ou plus.

Un second programme avec elseif

Quelle est la différence entre ces deux programmes ?

```
a=input('Entrez votre age');

if a>=100
    disp('Vous êtes assez âgé');
elseif a>=18
     disp('Vous êtes majeur');
elseif a>=16
     disp('Vous pouvez conduire');
end
```

```
a=input('Entrez votre age');

if a>=100
    disp('Vous êtes assez âgé');
end

if a>=18
     disp('Vous êtes majeur');
end

if a>=16
     disp('Vous pouvez conduire');
end
```

Dans le programme de gauche, le **programme n'affichera toujours qu'une seule phrase** ou rien du tout. A droite, il peut afficher zéro, une, deux ou trois phrases (ex : a=110).

Se passer du elseif

Question : Pouvez-vous réécrire le code en bas à gauche afin de ne plus utiliser de **elseif** mais plusieurs **if** ?

```
a = input('Entrez votre age : ');

if a>=100
    disp('Vous êtes assez âgé');
elseif a>=18
     disp('Vous êtes majeur');
elseif a>=16
     disp('Vous pouvez conduire');
end
```

```
a = input('Entrez votre age : ');

if a>=100
    disp('Vous êtes assez âgé');
end

if a>=18
     disp('Vous êtes majeur');
end

if a>=16
     disp('Vous pouvez conduire');
end
```

Ne fonctionne pas : si age vaut 110, le programme de gauche affichera un seul message tandis que le programme de droite en affichera trois !

Se passer du elseif

Question : Pouvez-vous réécrire le code en bas à gauche afin de ne plus utiliser de **elseif** mais plusieurs **if** ?

```
a = input('Entrez votre age : ');

if a>=100
    disp('Vous êtes assez âgé');
elseif a>=18
     disp('Vous êtes majeur');
elseif a>=16
     disp('Vous pouvez conduire');
end
```

```
a = input('Entrez votre age : ');

if a>=100
    disp('Vous êtes assez âgé');
end

if ~(a>=100) && a>=18
     disp('Vous êtes majeur');
end

if ~(a>=100) && ~(a>=18) && a>=16
     disp('Vous pouvez conduire');
end
```

Là, c'est bon !

Le mot clef else

Le mot clef **else,** qui accompagne toujours un **if,** permet d'éxecuter du code si les conditions du **if** ou des autres **elseif** situés auparavant étaient fausses. Le **else** ne possède aucune condition l'accompagnant : il est exécuté si tous les autres choix ont été rejetés.

Comme pour le **if,** ce mot clef est présent dans la plupart des langages de programmation modernes, même si sa syntaxe peut différer quelque peu selon le langage.

Syntaxe du if/elseif/else

Voici la syntaxe générale de ce que l'on appelle « un bloc if » avec **toutes les options possibles :**

```
//Bloc de code 0

if <condition 1>
    //Bloc de Code 1
elseif <condition 2>
    //Bloc de Code 2
elseif <condition 3>
    //Bloc de Code 3
elseif ...
    //...
else ...
    //Bloc de code n

end

//Bloc de Code n+1
```

Tout le **Bloc de code 0** est exécuté. Ensuite, la **condition 1** est évaluée.

Si elle est vraie, le **Bloc de code 1** est exécuté, puis le **Bloc de code n+1.**

Sinon, si la condition 2 est vraie, le **Bloc de code 2** est exécuté, puis le **Bloc de code n+1.**

Sinon, si la condition 3 est vraie, le **Bloc de code 3** est exécuté, puis le **Bloc de code n+1.**

Sinon, **si toutes les conditions sont fausses**, c'est **le Bloc de code n** puis **le Bloc de code n+1** qui sont exécutés.

Syntaxe du if/elseif/else

Il y a quelques règles à suivre lorsqu'on écrit un **bloc if** :

```
//Bloc de code 0

if <condition 1>
    //Bloc de Code 1
elseif <condition 2>
    //Bloc de Code 2
elseif <condition 3>
    //Bloc de Code 3
elseif ...
    //...
else ...
    //Bloc de code n

end

//Bloc de Code n+1
```

On **commence toujours par écrire le mot clef if,** suivi d'une condition est d'un bloc de code.

Ensuite, on peut mettre **zéro, un ou plusieurs mots clefs elseif**, chacun suivis d'une condition et d'un bloc de code.

Enfin, on peut mettre **zéro ou un mot clef else,** suivi d'un bloc de code.

On termine le bloc if avec le mot clef **end**

Un exemple

Voici un code affichant un menu permettant d'additionner, de multiplier, de soustraire ou de diviser deux valeurs selon le choix d'un utilisateur :

```
x = input('Entrez une valeur : ');
y = input('Entrez une valeur : ');
z = input('Entrez votre choix d operation : ');

if z==1
    disp(x+y);
elseif z==2
    disp(x*y);
elseif z==3
    disp(x-y);
elseif z==4
    disp(x/y);
end
```

Que se passe-t-il si l'utilisateur se trompe et entre une valeur qui n'a rien à voir avec le menu (comme 7) ?

Le programme n'affichera rien (aucun message d'erreur).

Un exemple

Le mot clef else va nous aider à gérer la situation d'un mauvais choix. Avec ce mot clef, on dira à Matlab d'exécuter un code à la condition qu'aucune ligne de code des if /elseif précédents n'ait été exécutée.

```
x = input('Entrez une valeur : ');
y = input('Entrez une valeur : ');
z = input('Entrez votre choix d operation : ');

if z==1
    disp(x+y);
elseif z==2
    disp(x*y);
elseif z==3
    disp(x-y);
elseif z==4
    disp(x/y);
else
    disp('Mauvais choix');
end
```

De cette façon, si l'utilisateur entre un mauvais choix, on le lui dira…

Un second exemple avec else

Le mot clef else permet de mettre, dans un bloc if, du code qui ne sera exécuté que si aucune des conditions précédentes du bloc if n' étaient vraie.

```
a=input('Entrez votre age');

if a>=100
    disp('Vous êtes assez âgé');
elseif a>=18
     disp('Vous êtes majeur');
elseif a>=16
     disp('Vous pouvez conduire (accompagné)');
else
    disp('Vous êtes jeune');
end
```

Ici, ce programme affichera

- Que la personne est âgée si elle a 100 ans ou plus
- Sinon, que la personne est majeure si elle a 18 ans ou plus
- Sinon, que la personne peut conduire si elle a 16 ans ou plus
- Sinon, que la personne est jeune

Exercice 1

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');

if x==0
    disp('x est nul');
elseif x>0
    disp('x est positif');
else x<0
    disp('x est négatif');
end
```

Ce programme possède une erreur de syntaxe : il ne doit y avoir aucune condition après le mot clef else.

Exercice 2

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');

if x==0
    disp('x est nul');
else
    disp('x est négatif');
elseif x>0
    disp('x est positif');
end
```

Ce programme possède une erreur de syntaxe : après un **else,** le bloc if est terminé et il faut nécessairement mettre le mot clef **end.** Il faut mettre le else à la fin !

Exercice 3

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');

if x==0
    disp('x est nul');
else
    disp('x est négatif');
else
    disp('x est positif');
end
```

Ce programme possède une erreur de syntaxe : il ne peut y avoir qu'un seul mot clef **else** dans un bloc **if.**

Exercice 4

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');

if x==0
    disp('x est nul');
elseif x<0
    disp('x est négatif');
elseif
    disp('x est positif');
end
```

Ce programme possède une erreur de syntaxe : le dernier **elseif** n'a pas de condition. II faut lui en rajouter une ou le remplacer par **else.**

Exercice 5

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');

if x==0
    disp('x est nul');
if x<0
    disp('x est négatif');
if x>0
    disp('x est positif');
end
```

Ce programme possède une erreur de syntaxe : il y a plusieurs **if** mais un seul **end.** Soit on veut faire plusieurs blocs **if** et il faut terminer chacun d'entre eux par un **end,** soit on veut un seul bloc **if,** et il faut remplacer les **if** par des **elseif.**

Exercice 6

Que pensez-vous de ce programme ?

```
x = input('Entrez une valeur ');
y = input('Entrez une valeur ');

if x>=0
    if y >=0
        disp('x et y sont positifs');
    else
        disp('x est positif et y est négatif');
    end
else
    if y >=0
        disp('x est négatif et y est positif');
    else
        disp('x et y sont négatifs');
    end
end
```

II n'y a aucune erreur de syntaxe dans ce programme : il est tout à fait autorisé dans un if, un bloc de code qui contient lui-même un if.

Exercice 7

Ecrire le programme qui trouve les racines d'une équation de second degré désigné par :

$$ax^2+bx+c=0$$

Solution

Voici le M-File qui contient le programme qui est enregistré avec le nom : Equation.m

Dans cette solution on suppose que la valeur de **a** introduite par l'utilisateur est différente de 0

```
% Programme de résolution de l'équation a*x^2+b*x+c=0
a = input ('Entrez la valeur de a : ');                % lire a
b = input ('Entrez la valeur de b : ');                % lire b
c = input ('Entrez la valeur de c : ');                % lire c
delta = b^2-4*a*c ;                                    % Calculer delta
if delta<0
        disp('Pas de solution')                        % Pas de solution
elseif delta==0
        disp('Solution double : ')                     % Solution double
        x=-b/(2*a)
else
        disp('Deux solutions distinctes: ')            % Deux solutions
       x1=(-b+sqrt(delta))/(2*a)
       x2=(-b-sqrt(delta))/(2*a)
end
```

Exercice 7(suite)

Pour l'exécution du programme, il suffit de taper le nom du programme :

```
>> Equation
Entrez la valeur de a : -3
 Entrez la valeur de b : 2
Entrez la valeur de c : 1
'Deux solutions distinctes :
x1 =-1/3
x2 =1
```

Remarque : Il existe la fonction solve prédéfinie en MATLAB pour trouver les racines d'une équation.

```
>> solve('3*x^2+4*x+2=0','x')
ans =
-2/6
 1
```

La fonction modulo

La fonction modulo, qui s' écrit **mod,** permet de récupérer le reste de la division euclidienne d'un nombre par un autre.

Par exemple, ce code affichera le reste de la division euclidienne de 23 par 7 :

```
>> a = mod(23,7);
>> disp(a)
     2
```

Le résultat est 2 car on a bien $23 = 7*3 + 2$ ← Le reste

Le quotient

Pourquoi parler de la fonction modulo maintenant ?

La fonction modulo

Le modulo est très pratique pour tester si un nombre est divisible par un autre : dans ce cas, le reste de la division euclidienne est 0. On l'utilise souvent dans une condition pour tester la divisibilité d'un nombre par un autre

Par exemple, ce code permet de tester si le nombre **a** est divisible par 7 :

```
a = input('Entrez une valeur : ');

if mod(a,7) == 0
    disp('a est divisible par 7');
else
    disp('a n est pas divisible par 7');
end
```

La fonction modulo

Que fait ce code ?

```
a = input('Entrez une valeur : ');

if mod(a,2) == 0
    disp('a est un nombre pair');
else
    disp('a est un nombre impair');
end
```

Ce code teste la parité d'un nombre.

L'instruction switch

```
switch (expression)
      case valeur_1
            Groupe d'instructions 1
      case valeur_2
            Groupe d'instructions 2
            . . .
      case valeur_n
            Groupe d'instructions n
      otherwise
            Groupe d'instructions si tous les case ont échoué
end
```

L’instruction switch

Exemple:

```
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
```

Son exécution donne :

Entrez un nombre : 30

x n'est pas 0 ou 10 ou 100

Les boucles

Les boucles

Les boucles permettent de répéter plusieurs fois un même bloc de code, permettant de faire à la machine des milliards d'opérations alors que l'on a soi même écrit qu' une dizaine de lignes de code.

Il y a deux types de boucles que l'on va voir : le **for** et le **while**.

Syntaxe de la boucle for

Voici les règles à suivre pour écrire une **boucle for** :

```
for variable = expression_vecteur
        Groupe d'instructions
end
```

L'expression_vecteur correspond à la définition d'un vecteur :

début : pas : fin ou début : fin

La variable va parcourir tous les éléments du vecteur et pour chacun, il va exécuter le groupe d'instructions

La boucle for

Exemple:

L'instruction for	`for i = 1 : 4`
	`  j=i*2 ;`
	`  disp(j)`
	`end`
		`for i = 1 : 2 : 4`
		`  j=i*2 ;`
		`  disp(j)`
		`end`
			`for i = [1,4,7]`
			`  j=i*2 ;`
			`  disp(j)`
			`end`

Le résultat de l'exécution	2 4 6 8	2 6	2 8 14

Syntaxe du while

L'autre boucle à connaitre (et, dans Matlab, en général plus utile que la boucle **for)** est la **boucle while :**

```
//Bloc de code 0

while <condition 1>
    //Bloc de Code 1
end

//Bloc de Code 2
```

Il est impératif que le **bloc de code 1** fasse évoluer la **condition 1**.

Si ce n'est pas le cas et que le programme « entre » dans la boucle, il n'en sortira jamais (ce n'est pas souhaitable).

Première utilisation de la boucle while

Voici un premier programme avec une boucle **while.** Que fait ce programme ?

```
a = input('Entrez une valeur entiere : ');

s=1;
while a>1
    s = s*a;
    a = a-1;
end
disp(s);
```

Ce programme calcule, dans la variable s, le produit de toutes les valeurs entre 1 et a.

Il calcule donc la factorielle de a.

Seconde utilisation de la boucle while

Voici un second programme avec une boucle **while.** Que fait ce programme ?

```
a = input('Entrez une valeur positive : ');

while a<0
    a = input('Entrez une valeur positive : ');
end

disp(a);
```

Ce programme demande une valeur positive à l'utilisateur et lui redemande tant qu'il ne le fait pas...

Troisième utilisation de la boucle while

Voici un troisième programme avec une boucle while. Que fait-il ?

```
a = input('Entrez une valeur : ');
b = input('Entrez une valeur : ');

n = min(a,b);
while ~(mod(a,n)==0 && mod(b,n)==0)
    n=n-1;
end

disp(n)
```

Ce programme cherche un diviseur commun à a et b, en partant de la plus petite de ces deux valeurs : il calcule pgcd(a,b).

Quelle boucle utiliser ?

Vaut-il mieux utiliser une boucle **for** ou **while** dans un programme ?

En général, si on sait à l'avance combien de fois la boucle doit se répéter, on utilisera une boucle **for.** Sinon, on utilisera un while.

Parmi les trois programmes que l'on vient de voir en exemple du while, lesquels devraient utiliser une boucle **for ?**

Programme	for ou while ?	Explications
factorielle(a)	**for/vectorisé**	La boucle se répétera (a-1) fois
nombre positif	**while**	On ne sait pas combien de fois l'utilisateur va saisir un nombre négatif
pgcd(a,b)	Les deux	Soit on teste tous les entiers entre 1 et min(a,b), soit on teste ceux entre a et 1 jusqu'à en trouver un qui divise.

Factorielle avec for

Pour réaliser une factorielle avec une boucle **for,** on peut faire ainsi

```
a = input('Entrez une valeur entiere : ');

s=1;
for i = 1:a
    s = s*i;
end
disp(s);
```

On peut évidemment vectoriser ce code et éviter la boucle **for**

```
a = input('Entrez une valeur : ');
s = prod(1:a);
disp(s);
```

On peut aussi utiliser la fonction factorielle de Matlab : **factorial**

Le mot clef break

Le mot clef **break** permet d'interrompre une boucle **for** ou **while** et de poursuivre l'exécution du code après la boucle.

II est parfois considéré comme une mauvaise pratique de programmation, car il est souvent possible de modifier les conditions du **while** afin de l' éviter.
De plus, on ne peut pas l'utiliser dans le cas d'une vectorisation de code.

Nous n'examinerons pas en détail ce mot clef: sachez qu'il existe, mais évitez de trop souvent l'utiliser.

Le mot clef continue

Le mot clef **continue** permet de passer directement à l'itération suivante de la boucle

Exemple

```
a = 10;
while a < 20
  if a == 15
    a = a + 1;
    continue;
  end
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

Exercice

1. Ecrire un programme Matlab qui permet de calculer la somme des éléments d'un vecteur
2. Ecrire un programme Matlab qui permet de calculer le produit des éléments d'un vecteur
3. Ecrire un programme Matlab qui permet de calculer la moyenne des éléments d'un vecteur
4. Ecrire un programme Matlab qui permet de créer une matrice M ayant le vecteur V dans le diagonal, et 0 ailleurs
5. Ecrire un programme Matlab qui permet d 'ordonner les éléments du vecteur V par ordre croissant

Solution Exercice

La fonction	Description	Le programme qui la simule

sum (V)	La somme des éléments d'un vecteur V	`V=input('Donnez les éléments du vecteur V'); n = length(V);`
		`somme = 0 ;`
		`for i = 1 : n`
		`  somme=somme+V(i) ;`
		`end`
		`disp(somme)`

prod (V)	Le produit des éléments d'un vecteur V	`V=input('Donnez les éléments du vecteur V'); n = length(V);`
		`produit = 1 ;`
		`for i = 1 : n`
		`  produit=produit*V(i) ;`
		`end`
		`disp(produit)`

mean (V)	La moyenne des éléments d'un vecteur V	`V=input('Donnez les éléments du vecteur V'); n = length(V);`
		`moyenne = 0 ;`
		`for i = 1 : n`
		`  moyenne = moyenne+V(i) ;`
		`end`
		`moyenne = moyenne / n`

Solution Exercice

diag (V)	Créer une matrice ayant le vecteur V dans le diagonale, et 0 ailleurs	`V=input('Donnez les éléments du vecteur V'); n = length(V);`
		`A = zeros(n) ;`
		`for i = 1 : n`
		`  A(i,i)=V(i) ;`
		`end`
		`disp(A)`

sort (V)	Ordonne les éléments du vecteur V par ordre croissant	`V=input('Donnez les éléments du vecteur V'); n = length(V);`
		`for i = 1 : n-1`
		`  for j = i+1 : n`
		`  if V(i) > V(j)`
		`  tmp = V(i) ;`
		`  V(i) = V(j) ;`
		`  V(j) = tmp ;`
		`  end`
		`  end`
		`end`
		`disp(V)`

Conclusion

Les boucles étendent les possibilités d'un programme, en **répétant plusieurs fois un même comportement.**

Ceci permet de réaliser un nombre important d'opérations sans pour autant avoir à coder chacune d'entre elles.

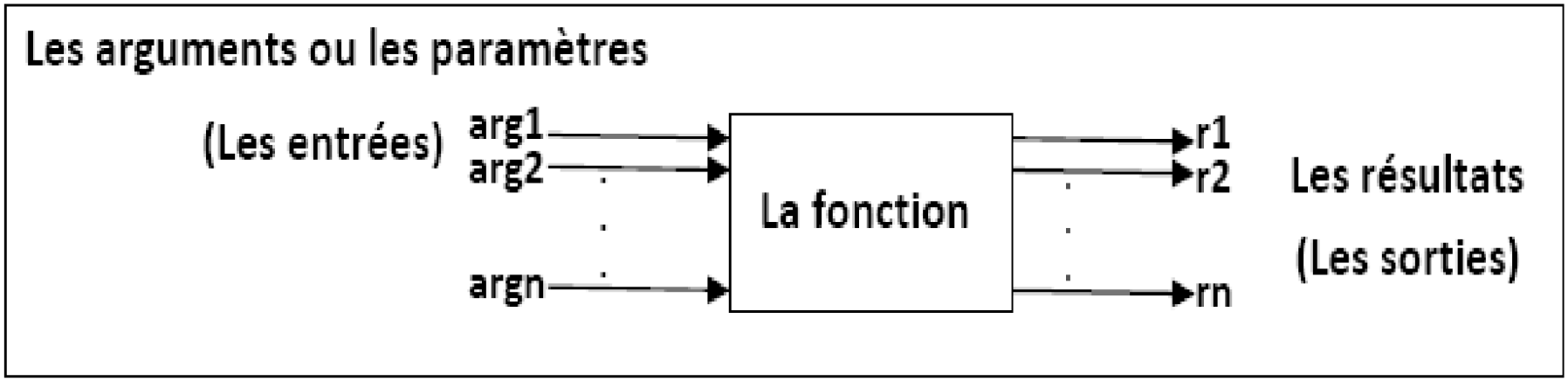
Pour pouvoir profiter des boucles, il faut être capable de penser un programme comme une série d'actions similaires.

Pour conserver un programme rapide, il est préférable **d' éviter les boucles si possible, et d'utiliser les fonctions de Matlab agissant sur les matrices :** cela s'appelle la vectorisation de code.

Avant d'optimiser un code, il est recommandé de l'écrire avec des boucles afin de tester son fonctionnement, puis de tenter de le vectoriser.

Fonctions sous Matlab

Les fonctions sont les pièces constituant un programme. On leur donne des paramètres en entrée et elles exécutent une action ou retournent une valeur.



Fonctions sous Matlab

A la différence des scripts, ce qui se passe en interne d'une fonction est (presque) indépendant du workspace. On parle de portée des variables. Les variables sont de 3 types :

- Les variables d'entrée de la fonction.
- Les variables de sortie de la fonction, auxquelles vous devez attribuer une valeur.
- Les variables locales, qui sont temporaires et qui disparaissent quand on quitte la fonction. Elles ne sont donc pas accessibles à l'extérieur de la fonction.

Fonctions sous Matlab

Habituellement, on utilise les fonctions afin de :

- Programmer des opérations répétitives
- réutilisation, recyclage
- Limiter le nombre de variables dans l'invite Matlab
- encapsulation
- Diviser le programme (problème) de manière claire

Fonctions sous Matlab

Création d'une fonction

MATLAB contient un grand nombre de fonctions prédéfinies comme sin, cos, sqrt, sum, … etc. Il est possible de créer nos propres fonctions en écrivant leurs codes source dans des fichiers M-Files

Fonctions sous Matlab

Création d'une fonction

Pour écrire une fonction dans Matlab, la première règle à respecter est de donner au fichier .m (M-File) le même nom que la fonction que l'on est en train d'´écrire.

Pour définir une fonction dans Matlab, on doit d'abord donner les noms des valeurs en sortie générées par la fonction, puis le nom de la fonction, et enfin les noms des paramètres en entrée de la fonction:

```
function [r1, r2, …, rn] = nom_fonction (arg1, arg2, …, argn)

      % le corps de la fonction
      . . .
      r1 = . . .  % la valeur retournée pour r1
      r2 = . . .  % la valeur retournée pour
      r2 . . .
      rn = . . .  % la valeur retournée pour rn

end               % le end est facultatif
```

Exemple:

Par exemple, si on souhaite faire une fonction produit, qui calcule et renvoie en sortie le produit de deux scalaires passés en paramètre, on écrira dans le fichier produit.m :

```
function res = produit(x, y)
res = x * y;
End
```

Pour tester cette fonction, placez-vous (à l'aide de l'explorateur de fichiers Matlab) dans le répertoire qui contient le fichier produit.m et on écrira dans la fenêtre de commande :

```
e = produit(2,4);
```

Une fois cette commande exécutée, on récupère bien dans « e » la valeur de sortie de la fonction, c'est `a dire le produit des deux paramètres en entrée. On remarquera que l'on ne voit pas, dans la zone des variables de Matlab, une variable nommée res. En effet, la variable res est interne à la fonction produit, et ne vit que pendant l'appel de cette fonction. Une fois la fonction terminée, la variable est effacée. De plus, la variable res n'existe que pour la fonction produit : elle ne peut pas être lue par une autre fonction ou par une commande directement dans Matlab. On récupère la valeur de res en la déclarant comme une valeur de sortie de la fonction, et en récupérant cette sortie dans la variable e

Exercice: écrire une fonction qui calcule la racine carrée d'un nombre par la méthode de Newton.

Solution

```
function r = racine(nombre)
 r = nombre/2;
precision = 6;
for i = 1: precision
r = (r + nombre ./ r) / 2;
end
```

Exécution :

```
>> x = racine(9)
       x = 3
>> x = racine(196)
      x = 14.0000
>> x = racine([16,144,9,5])
         x = 4.0000 12.0000 3.0000 2.2361
```

Remarque : Contrairement à un programme (un script), une fonction peut être utilisée dans une expression par exemple : 2*racine(9)-1.

Dans l'exemple suivant le programme est enregistré sous le nom 'racine'

Un programme	Une fonction

`a = input('Entrez un nombre positif: ');`
`x = a/2;`
`precision = 6;`
`for i = 1:precision`
`  x = (x + a ./ x) / 2;`
`end`
`disp(x)`
	`function r = racine(nombre)`
	`  r = nombre/2;`
	`  precision = 6;`
	`  for i = 1:precision`
	`  r = (r + nombre ./ r) / 2;`
	`  end`

L'exécution :
`>> racine` ↵
`Entrez un nombre positif: 16` ↵
`4`
	L'exécution :
	`>> racine(16)` ↵
	`ans =`
	`4`

on ne peut pas écrire des expressions tel que : `>> 2 * racine + 4` ✗	on peut écrire sans problème des expressions comme : `>> 2 * racine(x) + 4` ✓

Fonction récursive :

Dans la définition d'une fonction f, il arrive que l'on désire rappeler la même fonction f pour des valeurs différentes. L'exemple typique étant le cas de la définition de la fonction factorielle sur les entiers

$$\text{factoriel}(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * \text{factoriel}(n-1) & \text{si } n > 0 \end{cases}$$

Ceci est possible en MatLab, comme dans bien des langages de programmation, mais il faut être attentif au fait que cet outil amène souvent des boucles sans fin, si la condition d'arrêt n'est pas bien pensée.

Voilà l'exemple de la factorielle programmée en MatLab.

```
function fact =factoriel(n) %calcule le factoriel de n.
if n==0
    fact=1;
else
    fact= n*factoriel(n-1);
end
end
```

La fonction inline

Lorsque le corps de la fonction se résume à une expression relativement simple, on peut créer la fonction directement dans l'espace de travail courant, sans utiliser un m-file auxiliaire.

Syntaxe : La syntaxe des fonctions inline est simple :

$$nom\text{-}de\text{-}fonction = \text{inline}('expression', 'var1', 'var2', \ldots)$$

Exemple:

```
>> f = inline('x.^2 + x.*y', 'x', 'y')

f =

     Inline function:
     f(x,y) = x.^2 + x.*y

>> f(1,2)

ans =

     3
```

La fonction inline

L'exemple suivant met en évidence le mécanisme de déclaration implicite.

Exemple :

La définition

```
>> f = inline('x.^2')

f =

     Inline function:
     f(x) = x.^2
```

est équivalente à :

>> f = inline('x.^2', 'x')

Ce mécanisme est ambigu dès qu'il y a plusieurs variables

La définition

>> f = inline('x.^2 + x.*y') est équivalente à : >> f = inline('x.^2 + x.*y', 'x', 'y')

alors que

>> f = inline('x.^2 + x.*t') est équivalente à : >> f = inline('x.^2 + x.*t', 't', 'x')

Fonctions anonymes

Ce mode de définition de fonctions utilise comme pour les fonctions inline l'espace de travail courant. La syntaxe minimale est peu explicite, mais les fonctions ainsi définies seraient plus efficaces que les fonctions inline.

Syntaxe :

nom-de-fonction = @(var1, var2, ...) expression

- Contrairement aux fonctions inline l'expression mathématique qui constitue le corps de la fonction ainsi que les variables ne doivent pas être tapées entre apostrophes.
- Il est préférable que les fonctions soient vectorisées comme le sont les fonctions prédéfinie

Exemple :

```
>> g = @(x, y) x.^2 + x.*y
g =
@(x, y) x.^2 + x.*y
>> g(1, 2)
    ans =
    3.0000
```

Polynômes

Pour MATLAB, un polynôme est une liste : la liste des coefficients ordonnés par ordre décroissant :

Exemple :

Le polynôme $p(x) = 1 - 2x + 4x^3$ est représenté par le liste :

```
>> p = [4 0 -2 1]
p =
4 0 -2 1
```

Polynômes

Les fonctions usuelles du calcul polynomial sont les suivantes

Fonction	Arguments	Résultat
polyval	un polynôme p et un nombre a	calcul de $p(a)$
roots	un polynôme p	la liste des racines de p
conv	deux polynômes p et q	le polynôme produit $p \times q$
deconv	deux polynômes p et q	le quotient et le reste de la division euclidienne de p par q
polyder	un polynôme p	le polynôme-dérivée de p