TP1 - Prise en main de MATLAB

I. Introduction

Ce document a pour objectifs de présenter les fonctionnalités de base de MATLAB et de manipuler les commandes les plus utiles pour débuter.
Toutes les commandes qui suivent seront saisies dans la « fenêtre de commande » MATLAB. Chaque ligne de commande doit être validée par la touche «entrée » du clavier pour être prise en compte par le logiciel.

A. Création de variable

Création de la variable a :

```
>> a=1 (valider)

a =

   1
```

MATLAB fait la différence entre majuscule et minuscule **a** et **A** seront deux variables distinctes.
On remarque que MATLAB répond qu'il vient de créer la variable a. Cette nouvelle variable apparaît dans la fenêtre du **Workspace** et la commande que l'on vient de saisir apparaît dans la fenêtre **Command History**.
Création de la variable b :

```
>> b=5 ;
```

Le point-virgule à la fin de la ligne de commande indique que l'on ne souhaite pas de réponse à notre commande. La variable b apparaît dans le **Workspace**.

Il est facile de réaliser une opération entre deux variables, par exemple une multiplication.

```
>> a*b

ans =

   5
```

En appuyant sur les flèches haut et bas du clavier, il est possible de rappeler toutes les commandes de la fenêtre **Command History**.

B. Création de vecteur

Considérons la série de données du tableau de suivant.

x	0	10	20	30	40	50
y1	0	16	35	49	25	6
y2	0	12	21	36	49	56

tableau de données

Si l'on souhaite exploiter ces données il faut les saisir dans MATLAB sous la forme de vecteurs.

Le séparateur peut être un espace :

```
>> x=[0 10 20 30 40 50] ;
```

Le séparateur peut être une virgule :

```
>> y1=[0,16,35,49,25,6];
>> y2=[0 12 21 36 49 56];
```

Chaque composante i du vecteur x est stocké dans la variable x(i).

```
>> x(2)
ans =
   10
```

MATLAB propose également des fonctions de création automatique de vecteurs

```
>>u=[8 :0.1 :50]
```

Cette commande permet de créer le vecteur **u** qui aura comme composantes toutes les valeurs comprises entre **8** et **50** avec un pas de **0.1**. L'utilisation de cette commande permet de spécifier l'espacement entre deux composantes, le nombre de composantes étant calculé en fonction du pas et des bornes spécifiées.

Si le pas n'est pas spécifié, il prend par défaut la valeur de 1.

```
>>u=[8 :50]
```

Cette commande permet de créer le vecteur **u** qui aura comme composantes toutes les valeurs comprises entre **8** et **50** avec un pas de **1**.

Si au contraire, on souhaite spécifier le nombre exact de composantes, la fonction *linespace* peut être utilisée.

```
>>u=linspace(8,50,1000)
```

Cette commande permet de créer le vecteur u qui aura 1000 composantes régulièrement espacées entre les bornes 8 et 50.

C. Indexation des composantes d'un vecteur

Il est possible d'accéder facilement aux composantes d'un vecteur en utilisant l'indexation par ligne.
La variable **u(i)** contient la valeur de la $i^{ème}$ composante du vecteur **u**.

```
>>u(56)
ans=
10.313
```

D. Tracés de courbes

Deux vecteurs de même longueur peuvent être tracés à l'aide de la commande *plot* :

```
>> plot(x,y1)                    trace le vecteur y1 en fonction du vecteur x
```

Une fenêtre graphique « Figure 1 » s'ouvre avec la courbe demandée.

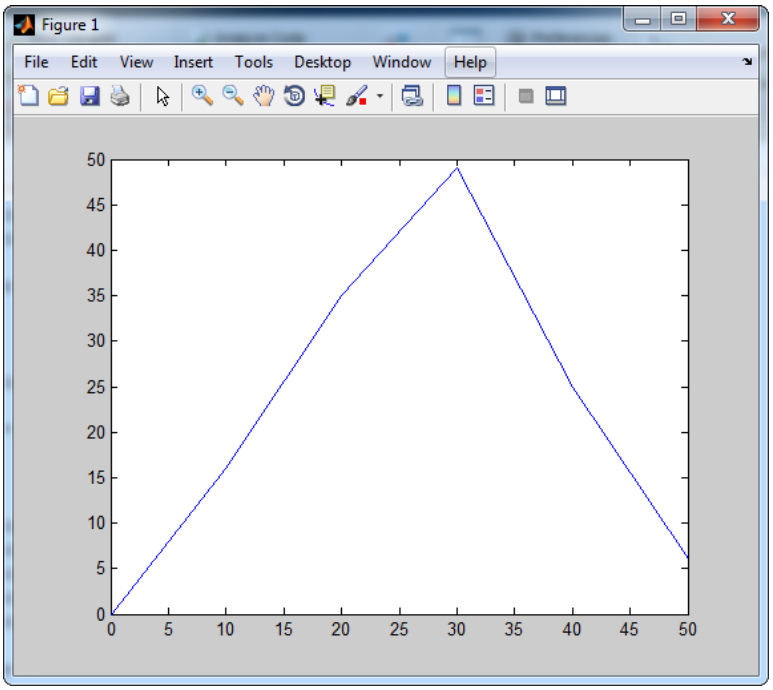


Figure 1 : tracé de deux vecteurs

On peut tracer le vecteur y2 en fonction du vecteur x

```
>> plot(x,y2)                    trace le vecteur y2 en fonction du vecteur x
```

On remarque que sur la fenêtre « Figure 1 », seule la courbe y2 en fonction de x est affichée. La courbe précédente a été supprimée.

Pour conserver les courbes dans une même fenêtre, on peut utiliser la commande *hold on*.

```
>> hold on
>> plot(x,y1)
```

La commande *hold off* annule la commande *hold on.*

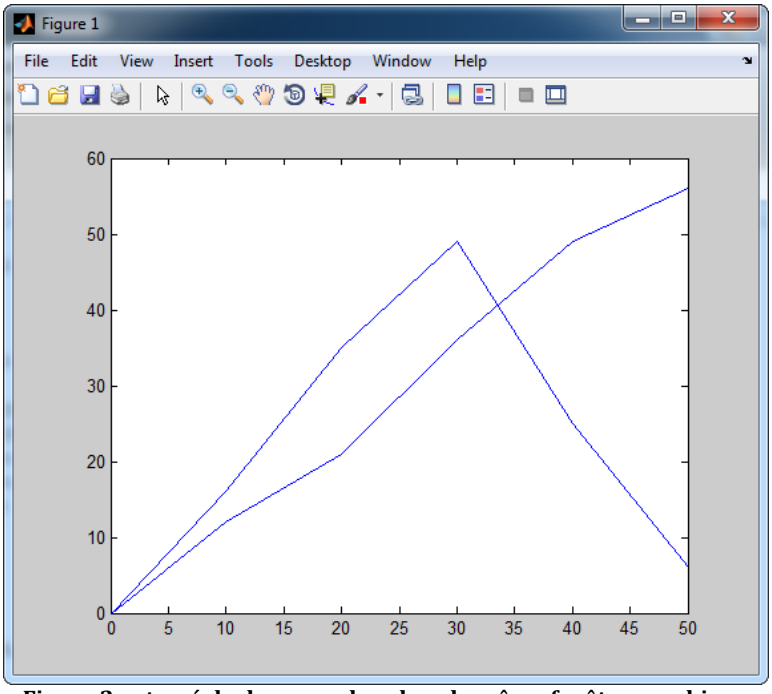


Figure 2 : tracé de deux courbes dans la même fenêtre graphique

La commande *grid on* permet d'afficher la grille

```
>>grid on
```

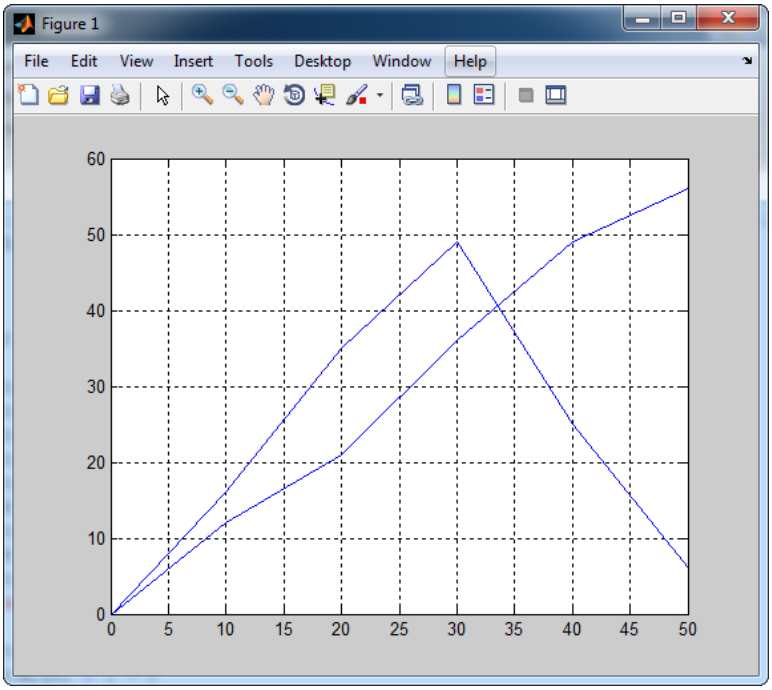


Figure 3 : affichage de la grille sur un graphique

La commande *grid off* permet de masquer la grille.

E. Mise en forme élémentaires des courbes

Des fonctions de mise en forme des courbes sont disponibles. La fonction plot accepte un troisième argument facultatif qui vous permet de spécifier une mise en forme pour les courbes.

Le tableau de la Figure 4 présente les codes que l'on peut saisir pour obtenir la mise en forme des courbes. Ces codes peuvent être combinés dans un ordre quelconque.

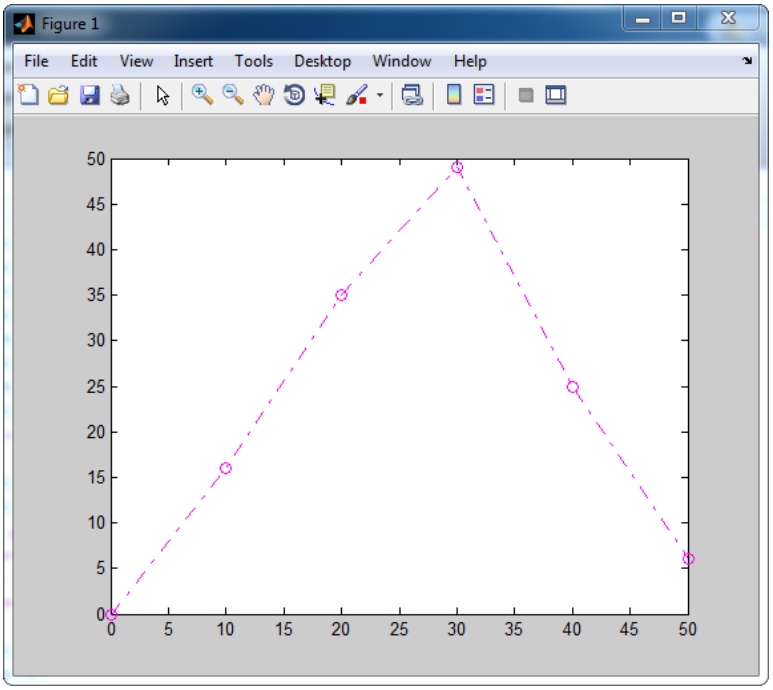
Couleur		Marqueur		Style de ligne	
b	Bleu	**.**	Point	**-**	Continu
g	Vert	**o**	Cercle	**--**	Trait
r	Rouge	**x**	Croix diagonale	**:**	Pointillés
c	Cyan	**+**	Croix verticale	**-.**	Trait-point
m	Magenta	*****	Etoile		
y	Jaune	**s**	Carré		**Remarque** : Si un style de marqueurs est spécifié sans aucun style de ligne aucune ligne ne sera tracée.
k	Noir	**d**	Losange		
w	blanc	**v**	Triangle vers le bas		
		^	Triangle vers le haut		
		<	Triangle à gauche		
		>	Triangle à droite		
		p	Pentagramme		
		h	hexagramme		

Figure 4 : codes de mise en forme des courbes

Vous pouvez faire quelques essais :

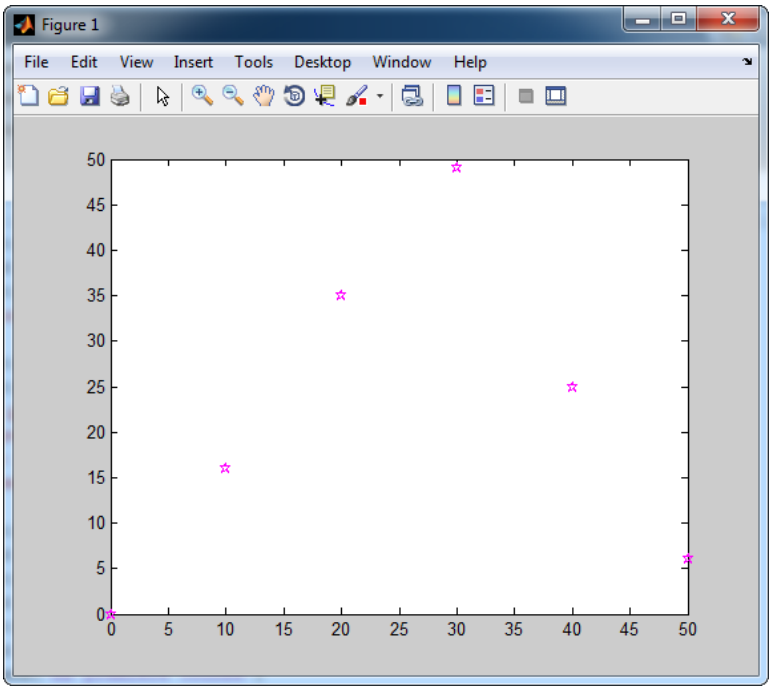
```
>>hold off
>>plot(x,y1,'r*:')
```

r pour rouge, * pour les marqueurs et **:** pour le style de ligne pointillés



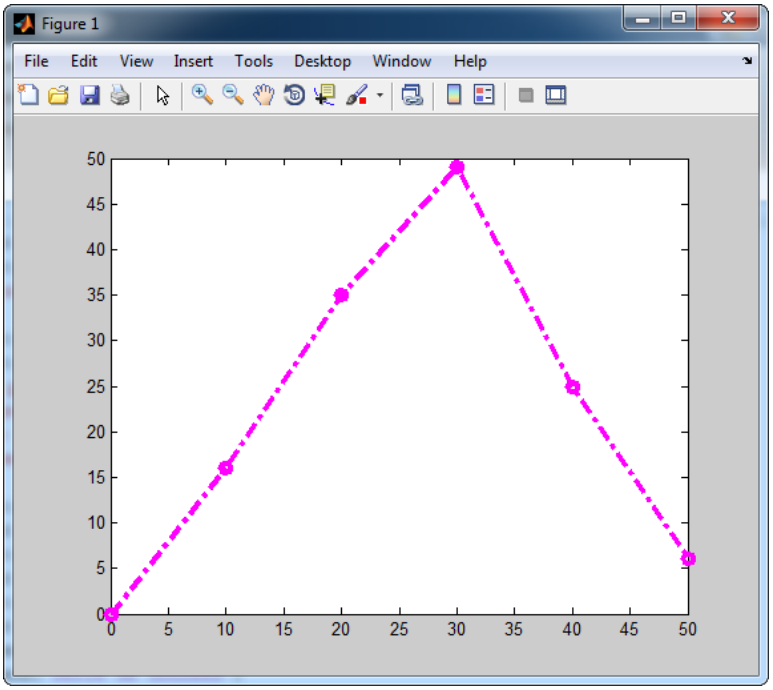
```
>>plot(x,y1,'mp')
```

Aucun style de ligne n'est spécifié, ce sont uniquement les marqueurs qui apparaissent.



Il est également possible de modifier l'épaisseur du trait avec l'argument *LineWidth*

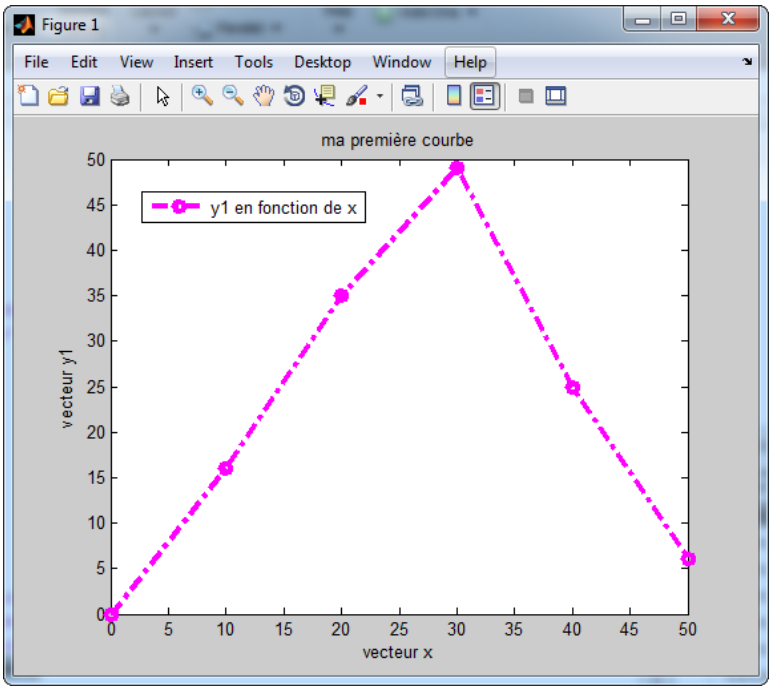
```
>> plot(x,y1,'mo-.','LineWidth',3)
```



F. Annotation des graphiques

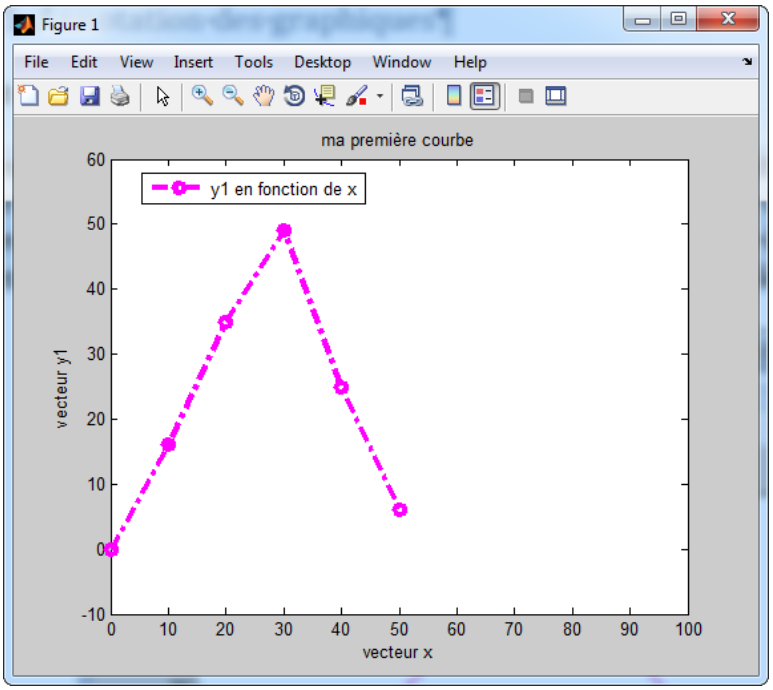
Il est également possible d'annoter le graphique:

```
>> title('ma première courbe')          % affiche un titre au graphique
>> xlabel('vecteur x')                  % étiquette de l'axe des abscisses
>> ylabel('vecteur y1')                 % étiquette de l'axe des ordonnées
>> legend('y1 en fonction de x')        % ajout d'une légende
```



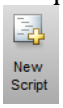
La commande *axis* permet de définir manuellement les échelles des axes.

```
axis([0 100 -10 60])
```



G. Créer un script élémentaire

Un script est un programme écrit en langage MATLAB. On comprendra aisément qu'il est très utile de regrouper une suite de lignes de commandes dans un fichier appelé script. L'intérêt est de pouvoir le sauvegarder et exécuter plusieurs fois la même séquence.

Pour créer un script, cliquer sur **New script** 

dans la barre de commande.

Une fenêtre Editor apparaît dans laquelle il est possible de saisir des lignes de commandes.

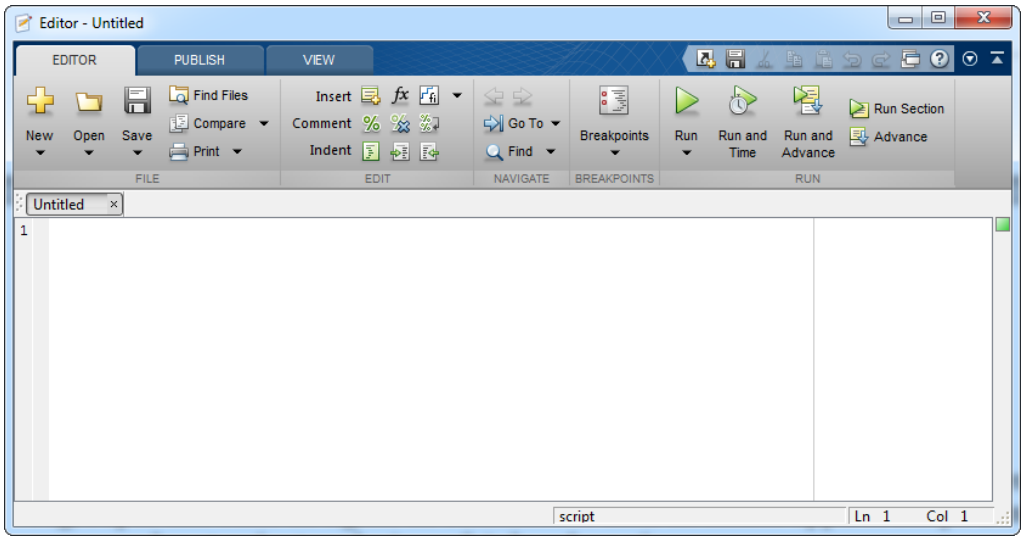


Figure 5 : fenêtre Editor d'édition des scripts

Saisir les lignes de commande et sauvegarder le fichier sous le nom ***plot_sinus_0.m*** (l'extension **.m** est l'extension des fichiers scripts de MATLAB).

Le signe **%** signal un commentaire dans un script.

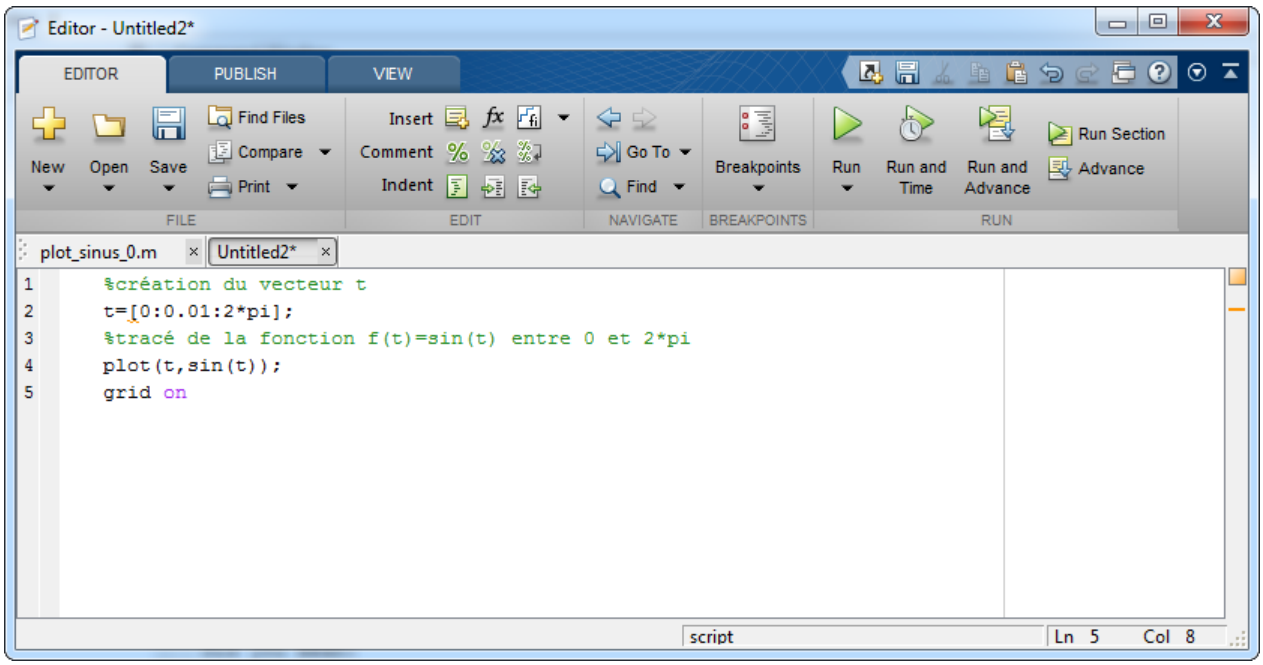


Figure 6 : premier script avec MATLAB

Exécuter le script en cliquant sur 

dans la barre de commande.

Si la fenêtre suivante apparaît, l'emplacement du fichier n'appartient pas au « **path** » de MATLAB et MATLAB ne peut par conséquent pas l'éxécuter, le logiciel vous propose :

- D'ajouter le dossier en question au « **path** » : **Add to Path** (recommandé)
- De déplacer ce fichier vers le dossier courant de travail de **MATLAB** : **Change FOLDER**

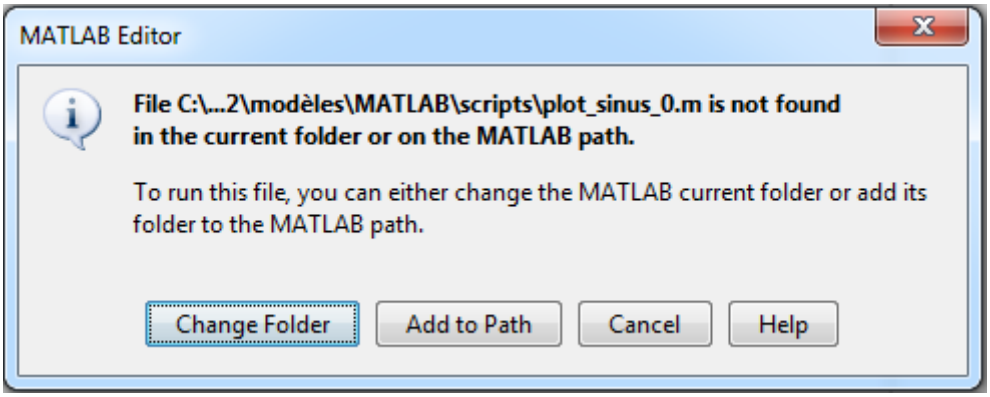
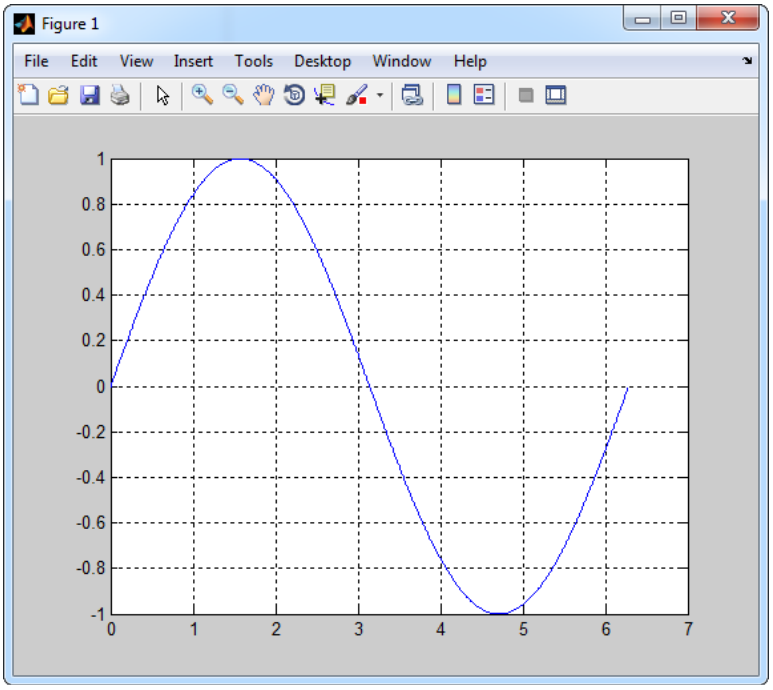


Figure 7 : fenêtre d'ajout automatique de dossier au « path » de MATLAB

On obtient la représentation de la fonction sinus en fonction du temps.



Il est possible d'utiliser des commandes de programmation classique comme une boucle **for**. Modifier votre script comme indiqué sur la Figure 82.

```
%création du vecteur t
t=[0:0.01:2*pi];
% la fonction hold all est identique à la fontion hold on mais permet
% de tracer chaque courbes avec une couleur différentes
hold all
% pour a prenant les valeurs comprises entre 1 et 4 avec un pas de 1

for a=[1:4]
%tracé de la fonction f(t)=sin(t) entre 0 et 2*pi
plot(t,sin(a*t),'LineWidth',2);
end
grid on
title('Fonction Sinus')
%MATLAB possède un interpréteur TeX de base, il est possible d'y faire
%figurer des caractères spéciaux ou des équations
xlabel('angle \theta en radians')
ylabel('f(\theta)=sin(\theta)')
axis([-1 7 -1.5 1.5])
```

Figure 8 : Tracé de plusieurs Sinus sur le même graphique

Exécuter le script et visualiser le résultat.

Si une erreur de syntaxe est détectée lors de l'exécution, le détail apparaît en rouge dans la fenêtre de commande.

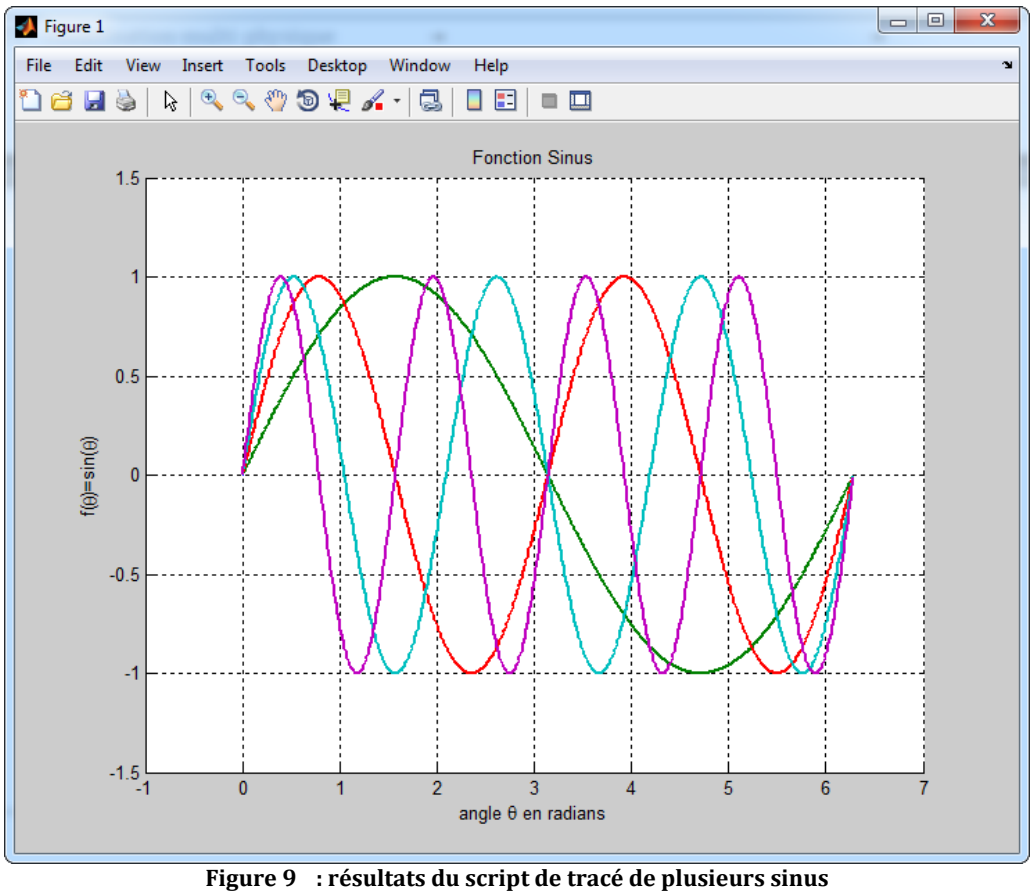


Figure 9 : résultats du script de tracé de plusieurs sinus

H. Les opérateurs de comparaison de MATLAB

Afin de réaliser des tests dans les scripts, **MATLAB** possède des opérateurs de comparaison et des opérateurs logiques.

Commandes utiles	
Opérateurs	Syntaxe MATLAB
Egal à	==
Différent de	~=
Supérieur à	>
Supérieur ou égal à	>=
Inférieur à	<
Inférieur ou égal à	>=
Negation(not)	~=
Ou(or)	|
Et(and)	&

Figure 10 : les opérateurs logiques et de comparaison de MATLAB

I. Les structure de boucles usuelles

Afin de réaliser des boucles dans les scripts, les trois solutions les plus utilisées sont :

- if – else if – else
- for
- while

1. Syntaxe de la boucle if – else if – else

```
if (test logique 1)
        action 1
elseif ( test logique 2)
        action 2
else
        action 3
```

Pour ce type de boucle, une seule action parmi *action1, action 2 et action 3* sera exécutée. Les tests sont effectués dans l'ordre et uniquement si cela est nécessaire. Ainsi si (*test logique 1*) est vrai, *action 1* sera exécutée et (test logique 2) ne sera pas évalué. Il peut y avoir un nombre quelconque d'instructions *elseif* ou aucune. Il peut y avoir une seule instruction *else* qui doit représenter la dernière condition.

2. Syntaxe de la boucle for

```
for variable=valeur :initiale :incrément :valeur_finale
        action
end
```

La boucle *for* exécute l'action un nombre défini de fois en faisant varier la valeur d'une variable.

3. Syntaxe de la boucle while

```
while condition
        action
end
```

L'action est exécutée tant que la condition évaluée est vraie. La sortie de la boucle s'opère lorsque la condition évaluée devient fausse.

II. Exemple d'exploitations

A. Interpolation d'une série de données

Lors de l'analyse de données expérimentales, il est souvent nécessaire d'interpoler une série de données. MATLAB possède des fonctionnalités qui permettent de réaliser des interpolations très simplement.
Considérons une série de points expérimentaux représentés par les deux vecteurs X=[0 10 20 30 40] et Y=[2 5 12 25 46].

L'objectif est de placer ces points sur un graphique et de tracer la courbe d'interpolation.

Dans la fenêtre de commande créer les deux vecteurs X et Y et visualiser l'allure de la répartition en traçant les points expérimentaux obtenus

```
>>X=[0 10 20 30 40];
>>Y=[2 5 12 25 46];
>>plot(X,Y,'r.','MarkerSize',25);
```

la spécification du marqueur sans spécifier le style de ligne affiche les marqueurs uniquement

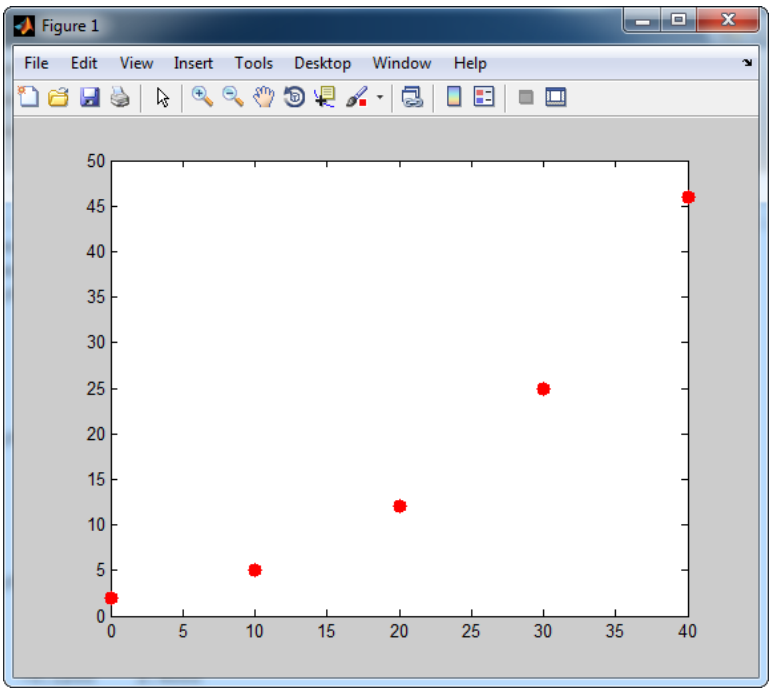


Figure 12 : repartition d'une série de points

L'allure de la répartition nous permet de permet de chercher une interpolation avec un polynôme de degré 2. La fonction *polyfit* donne automatique les coefficients du polynôme d'interpolation.

```
>> polyfit(X,Y,2)
ans =
   0.0300   -0.1200    2.4000
```

Le polynôme d'interpolation de degré 2 pour expression $p(x) = 0.03\,x^2 - 0.12\,x + 2.4$.

Tester le script de la Figure 13 pour comprendre l'intérêt d'utiliser un script pour automatiser ce type de tâche. Ce script permet de placer les différents points (ronds rouges) et de superposer la courbe d'interpolation.

```
1     % création des vecteurs X et Y
2 -   X=[0 10 20 30 40];
3 -   Y=[2 5 12 25 46];
4     %création du vecteur A qui contient les coefficients du polynôme
5     %d'interpolation de degré 2 qui interpole la série de donnée
6 -   A=polyfit(X,Y,2);
7     %Création d'un vecteur t contenant 100 composante prise entre X(0) et
8     %X(end), X(end) représente la dernière composante du vecteur X.
9 -   t=linspace(X(1),X(end),100);
10    %Création du vecteur U qui contient 100 éléments correspondants aux
11    %valeurs du polynôme d'interpolation pour les 100 éléments du vecteur t
12 -  U= polyval(A,t);
13    %tracé du polynôme d'interpolation
14 -  plot(t,U,'g','LineWidth',3);
15 -  hold on;
16    %tracé des points Y en fonction de X, la commande MarkerSize permet de
17    %spécifier la taille des marqueurs
18 -  plot(X,Y,'r.','MarkerSize', 25);
19 -  grid on;
```

Figure 13 : script d'interpolation d'une série de données

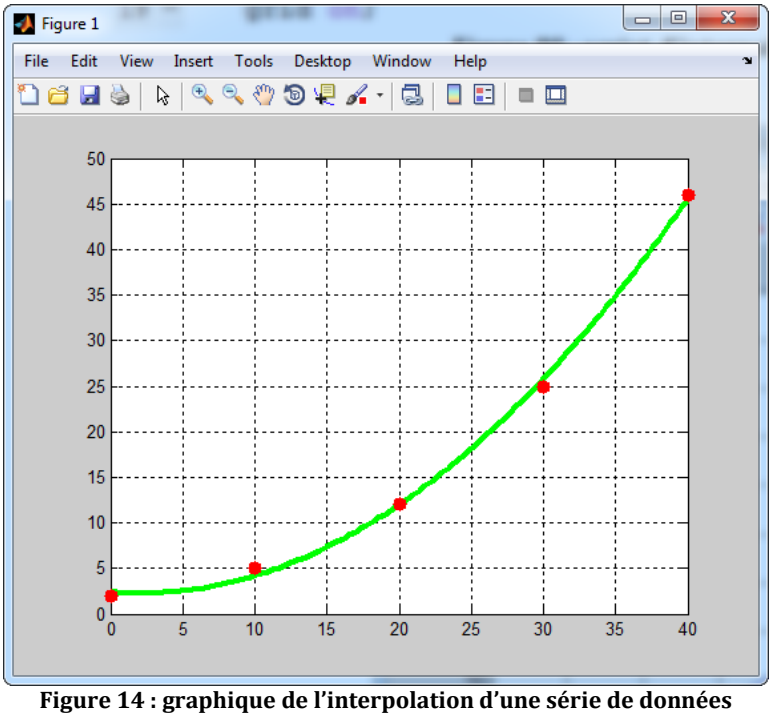


Figure 14 : graphique de l'interpolation d'une série de données