

# Premiers contacts avec Matlab

## Exemples simples pour commencer à utiliser MATLAB en traitement du signal

### Contexte

Cet énoncé constitue le premier volet de séances de travaux pratiques de traitement du signal, dispensées dans le cadre de la L3 Informatique de La Rochelle université. Il s'inscrit dans le cours de traitement du signal, et consiste en une introduction au logiciel de calcul matriciel Matlab.

La vocation de cette première séance est de permettre à chaque étudiant de prendre en main le logiciel Matlab (connaissance de l'environnement, commandes usuelles, programmation de base), de se familiariser avec la génération de signaux numériques, et de réaliser ses premiers programmes qui constitueront les briques de bases nécessaires à la réalisation des autres TP.

Le programme des TP est en conséquence structuré comme suit :

1. Présentation de Matlab et du cadre de développement des applications du TP.
2. Traitements temporels des signaux : convolution, corrélation, filtrage

### Objectifs pratiques

Les objectifs sont triples : découvrir et assimiler les commandes usuelles de Matlab et le fonctionnement de Matlab-Help, rafraîchir et consolider les connaissances acquises en cours, et mettre en application l'ensemble de ces savoirs et savoir-faire via la réalisation de programmes. En conséquence, il sera demandé aux étudiants, excepté pour le premier TP :

- de rédiger en temps réel un court rapport illustré retraçant le TP en suivant la trame des questions posées dans l'énoncé. La clarté des graphiques, la pertinence des remarques, et la correction linguistique seront prises en compte dans l'évaluation. La concision des réponses sera d'autant plus appréciée qu'elle ira de pair avec leur précision ;
- d'écrire soigneusement les programmes demandés sous formes de M-Files commentées ;
- et enfin, lorsque cela vous sera demandé, de déposer une semaine après la fin la séance le rapport ainsi que l'ensemble des codes à l'adresse [michel.menard@univ-lr.fr](mailto:michel.menard@univ-lr.fr) avec pour objet « TPNo\_'votrenom' ».

### Caractéristiques :

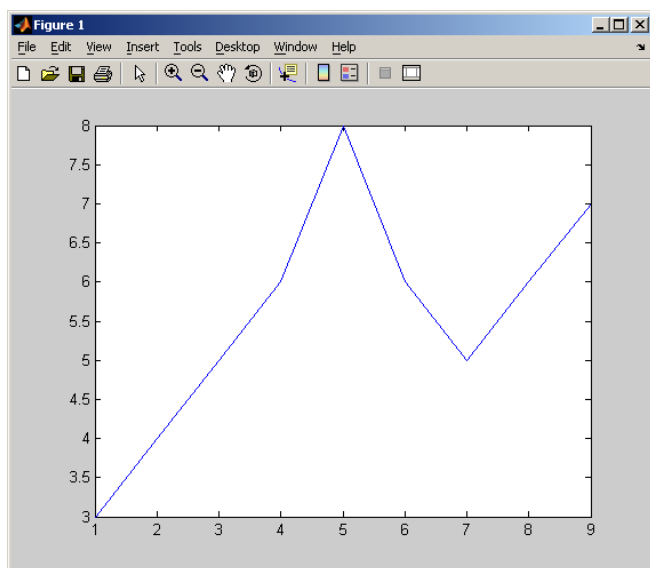
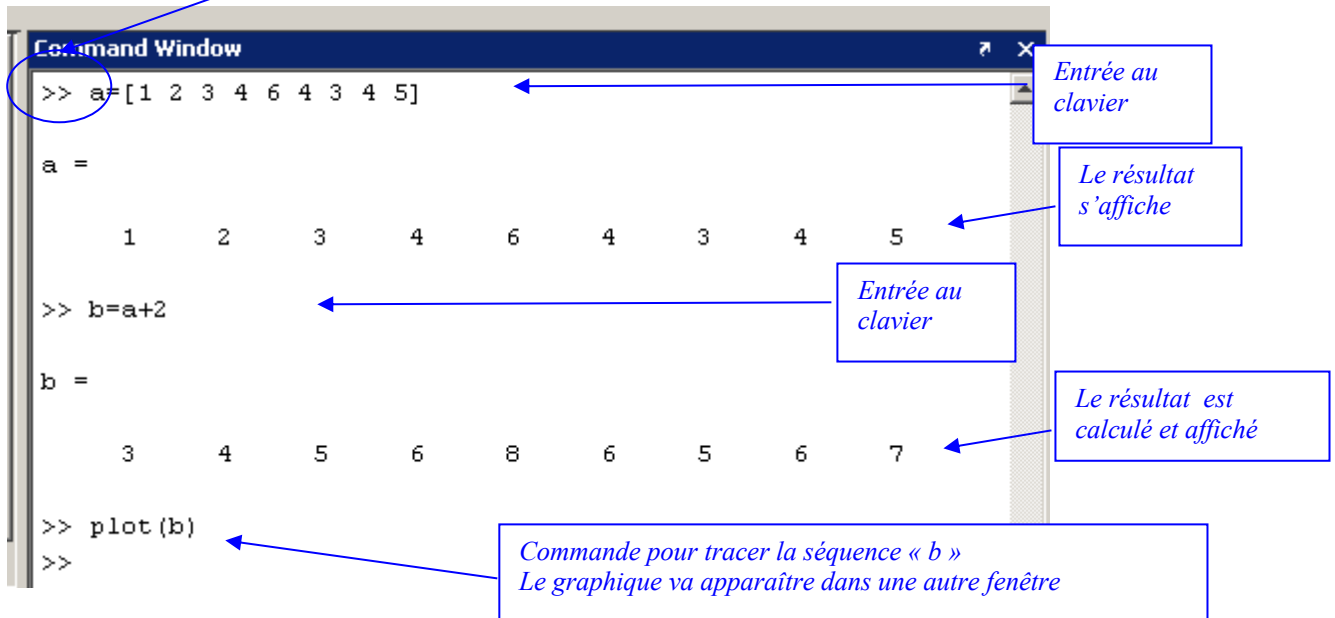
MATLAB : abréviation de « Matrix Laboratory ».

MATLAB a été conçu pour le calcul matriciel, d'où son intérêt en traitement du signal (produit scalaire). Ses atouts sont la simplicité et la portabilité (UNIX, Windows...). Il offre un langage de programmation avec de nombreuses fonctions pré-programmées. MATLAB est auto-documenté : les informations relatives à une instruction sont obtenues par la commande **help** suivie du nom de l'instruction.

# 1. Généralités

En mode commande, MATLAB affiche des chevrons >> et attend une commande :

**Mode interactif : indique que Matlab attend une commande**



*Affichage dans la nouvelle fenêtre des valeurs des éléments de la séquence b*

Si on frappe un point-virgule à la fin de la ligne de commande

```
>> b = a + 2 ;
```

et non

```
>>b=a+2
```

alors la commande est exécutée mais le résultat n'est pas affiché.

Autres exemples de commandes :

```
>> 1+3
```

```
>> 3^3
```

```
>> 2*sin(pi/4)
```

Le dernier résultat est stocké dans la variable **ans** qu'il est possible de réutiliser :

```
>> ans^2
```

```
>> ans-2
```

### Variables

Les résultats peuvent être stockés dans des variables :

```
>> x = pi/3
```

```
>> cos(x)
```

Pas de point virgule à la fin d'une expression : affichage du résultat.

```
>> y = sin(x)^2+cos(x)^2 ;
```

```
>> y
```

Un nom de variable commence par une lettre (dix neuf caractères max).

La commande **who** permet d'afficher la liste des variables utilisées. La commande **whos** donne plus d'informations sur ces variables.

Matlab fait la différence entre majuscules et minuscules.

Effacement des variables par **clear** nom\_variable.

Effacement de toutes les variables par **clear**.

### Variables complexes

Les notations **i** et **j** peuvent être utilisées pour désigner la racine de  $-1$ , et doivent être réservées à cet effet.

```
>> x=3; y=4.5*j ;
```

```
>> z=x+y
```

Il existe de nombreuses fonctions prédéfinies manipulant les nombres complexes :

```
>> real(z)
```

```
>> conj(z)
```

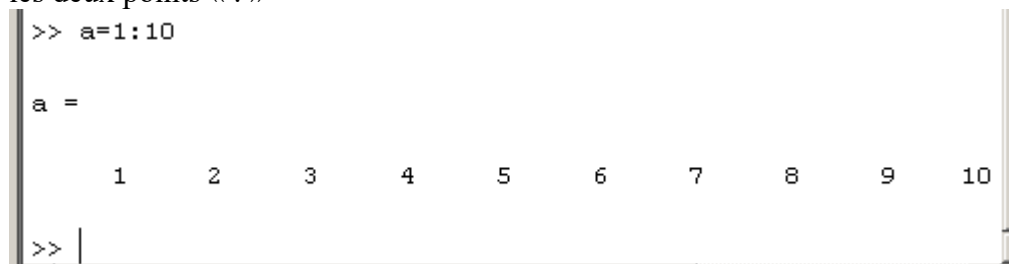
```
>> ro=abs(z)
```

```
>> phi=angle(z)
```

```
>> z=ro*exp(j*phi)
```

Attention : **i** et **j** utilisées comme autres variables, peuvent induire des erreurs.

Si on veut entrer une séquence longue d'intervalles réguliers sans entrer toutes les valeurs on utilise les deux points « : »



```
>> a=1:10

a =

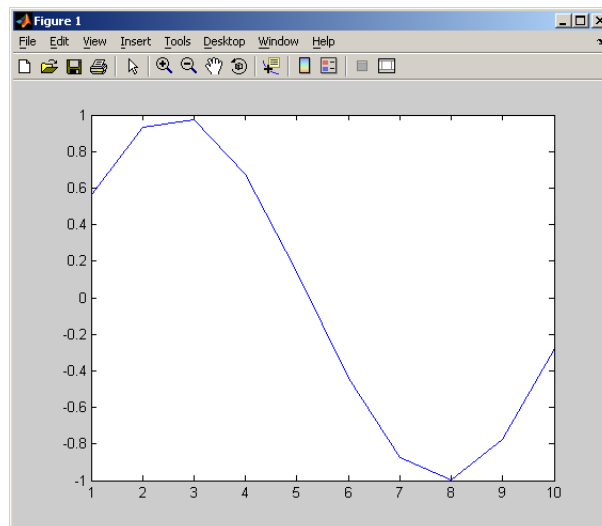
     1     2     3     4     5     6     7     8     9    10

>> |
```

Il est possible d'appliquer des fonctions sur la séquence « a » :

```
a =  
  
1      2      3      4      5      6      7      8      9      10  
  
>> plot(sin(3*a/5))  
>> |
```

La commande plot affiche le sinus de la séquence :



En abscisse il y a un simple numéro : la position de l'élément a dans le calcul (on notera que Matlab commence les numéros d'indice à 1 et non à zéro)

Pour se déplacer dans la liste des instructions, on peut utiliser les flèches du clavier



Si on frappe « Entrée », c'est la ligne sélectionnée qui est exécutée.

Analysez l'effet des autres flèches et des touches end, esc, del, backspace ou des combinaisons de touches

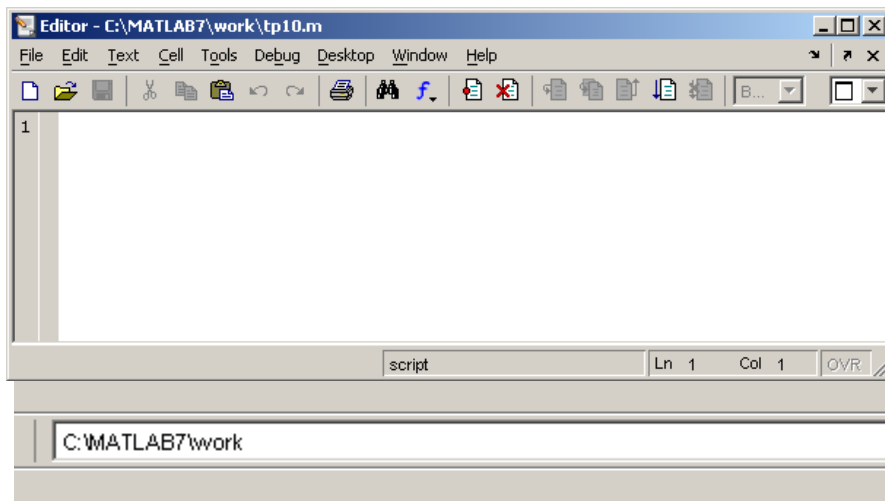
## 2. Création de programme dans un fichier

On se rend compte que la frappe directe d'instructions permet difficilement de générer sans erreur des séquences d'opérations très complexes ; pour y arriver il faut enregistrer les programmes dans un fichier dont on lancera l'exécution quand il sera prêt.

On clique sur « File » et (en gardant l'index appuyé, click gauche) dans le menu déroulant «New » « M-File». Ceci fait apparaître une nouvelle fenêtre ('Editor') dans laquelle on éditera le texte du programme

**Attention** : Il est important de s'habituer à la gestion des fichiers : il faut savoir dans quel dossier on range les fichiers pour pouvoir les retrouver, les modifier éventuellement et les exécuter par la suite. Une fois qu'on a écrit un programme il faut le sauvegarder : on choisit par exemple « save as » et le nom du fichier qui doit nécessairement avoir l'extension « .m » : Matlab reconnaitra par la suite cette extension lorsqu'on lui demandera l'exécution dans la fenêtre de commande ; par

exemple pour exécuté le programme enregistré dans le fichier « tp10.m » il faudra frapper dans la fenêtre « Command Window » >>tp10

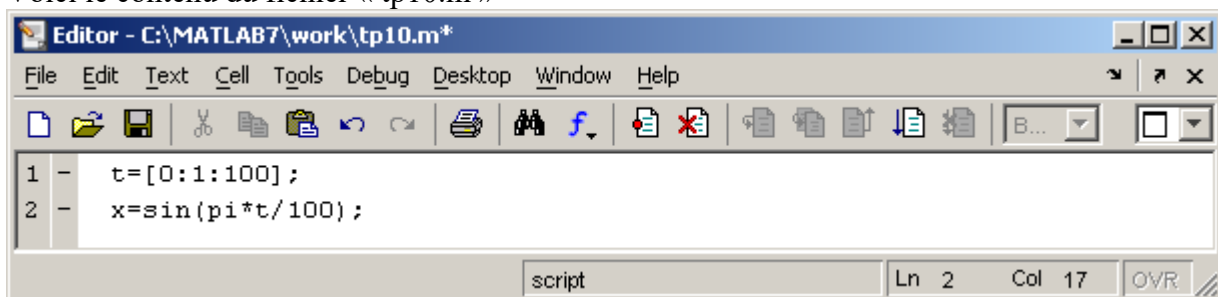


*Fenêtre où sera écrit le programme tp10.m*

*Dossier de travail courant  
En haut de la fenêtre Matlab*

Ici le fichier tp10.m » est rangé dans le dossier « work » de Matlab et donc peut s'exécuter.

Voici le contenu du fichier « tp10.m »

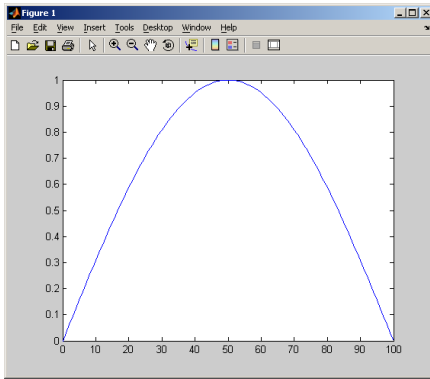


En frappant la commande « tp10 » dans la fenêtre de travail, on exécute le programme contenu dans le fichier « tp10.m ».

La deuxième commande « plot(t,x) » trace la séquence x en fonction de la séquence t: les instructions données dans la fenêtre des commandes (command window) reconnaissent les noms des variables donnés dans le fichier du programme: tout se passe comme si la séquence d'instructions du programme était entrée dans la fenêtre de commandes.

```
>> tp10
>> plot(t,x)
>>
```

*Fenêtre de travail où est  
frappée la commande*



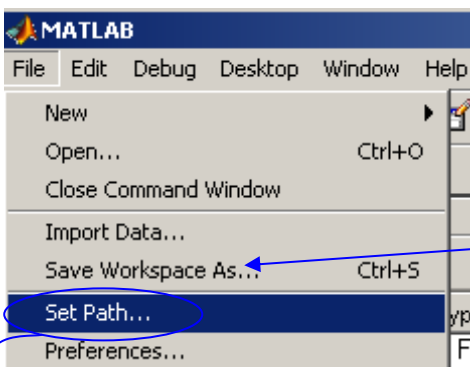
*Affichage de la courbe suite à la commande plot*

Autre exemple de fichier de commandes (avec l'extension .m – Rappel : fichier édité à partir de votre éditeur préféré, contenant une suite de commandes appelées en tapant le nom du fichier sans extension).

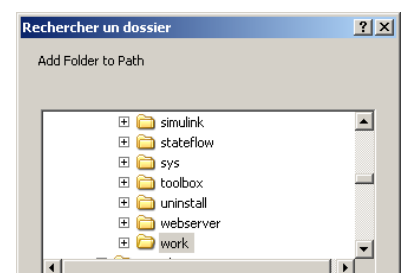
```
t = 1:100
y = cos(2*pi*t/50)
plot(t,y)
title('cosinus')
xlabel('temps')
ylabel('amplitude')
pause
Utilisation de boucles et contrôles for, while, if
    a = [0 -1 4 -6 9 -7] ; b = [ ] ; c = [ ]
for i = 1 :length(a),
    if a(i)<0
        b = [b a(i)] ;
    else
        c = [c a(i)] ;
    end ;
end ;
```

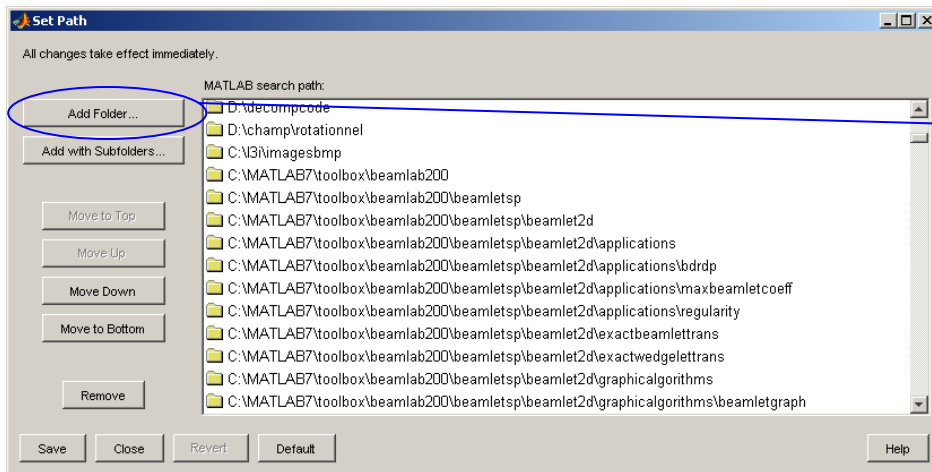
## Retrouver et exécuter des programmes Matlab écrits dans des fichiers « .m » et rangés dans un dossier

Lorsqu'on range les fichiers où sont écrits les programmes Matlab dans un dossier créé dans ce but, par exemple « d:\programMatlab », il faut donner à Matlab les informations pour qu'il puisse trouver le dossier en question ; ceci se fait en définissant le chemin d'accès (commande File/Set Path).



*Commande permettant de définir le chemin d'accès au nouveau dossier*





fichiers contenus dans le

Une nouvelle fenêtre s'ouvre on y écrit le texte du programme (voir ci-dessous).

*Remarque : dans un programme enregistré dans un fichier, on terminera une instruction par un point-virgule afin d'éviter d'afficher le résultat de l'instruction dans la fenêtre de commande.*

Et on l'enregistre (**save**) sous la forme d'un fichier 'tp11.m' ; l'extension '.m' indique que c'est un programme exécutable par Matlab (*ne pas oublier de l'enregistrer à nouveau lorsqu'on modifie le fichier*). L'enregistrement peut se faire dans le dossier que vous avez créé et dont vous avez inséré le path dans File/Set Path.

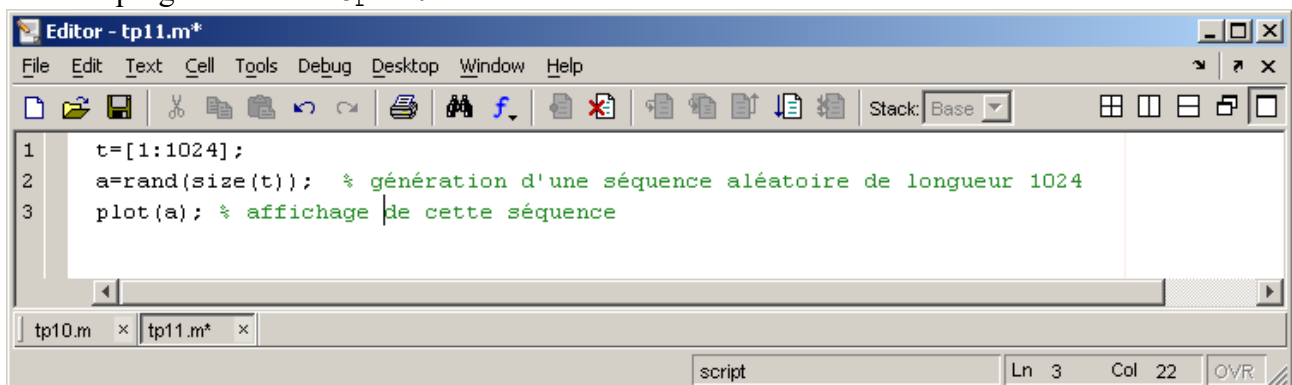
Pour l'exécuter dans la fenêtre Matlab

**>> tp11**

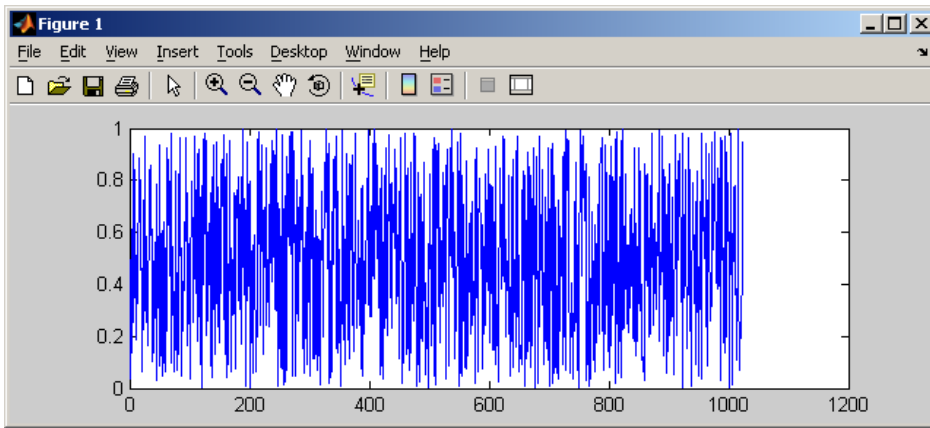
A la fin de l'exécution de Matlab on retrouve dans cette fenêtre le '>>'

Un **commentaire** commence par un signe % (en début de ligne où après une instruction)

Voici le programme tp11.m



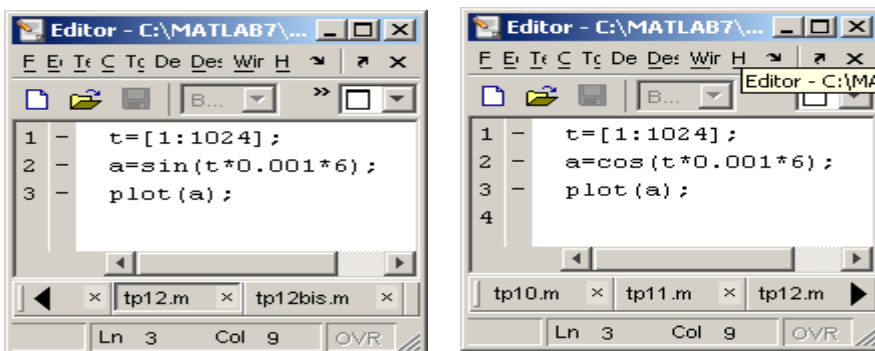
Qui engendre (fonction rand) une séquence pseudo aléatoire « a » dont la longueur est celle de la séquence « t » (soit ici 1024) et qui la trace. Son exécution sur la ligne de commande produit l'affichage :



Il est utile de bien jongler avec les fenêtres d'affichage des graphiques ; voici quelques opérations courantes (des possibilités supplémentaires seront décrites dans un paragraphe ultérieur ; une fois les bases acquises, il faudra se référer à la documentation Matlab pour utiliser les nombreuses possibilités offertes) ;

Si on veut superposer un deuxième graphique à un graphique qu'on vient de tracer, on utilise la commande 'hold on'

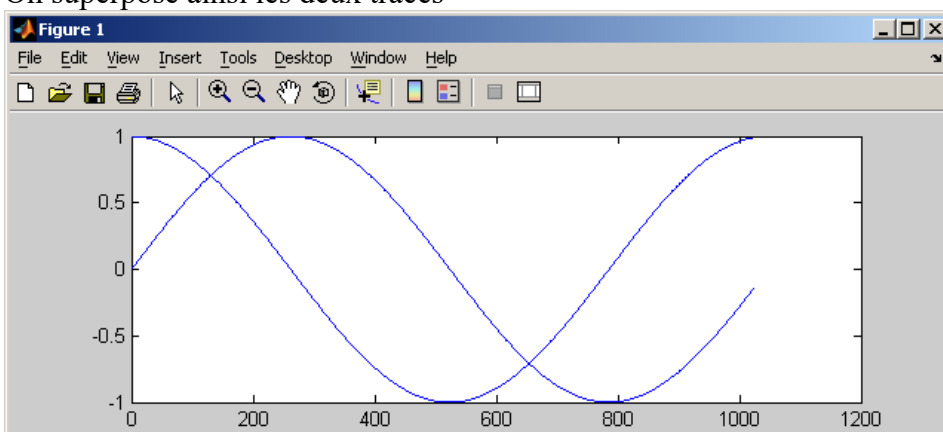
Voici deux programmes traçant un sinus et un cosinus



Si on exécute le premier programme, puis la commande « hold » et le deuxième programme :

```
>> tp12
>> hold on
>> tp12bis
>>
```

On superpose ainsi les deux tracés

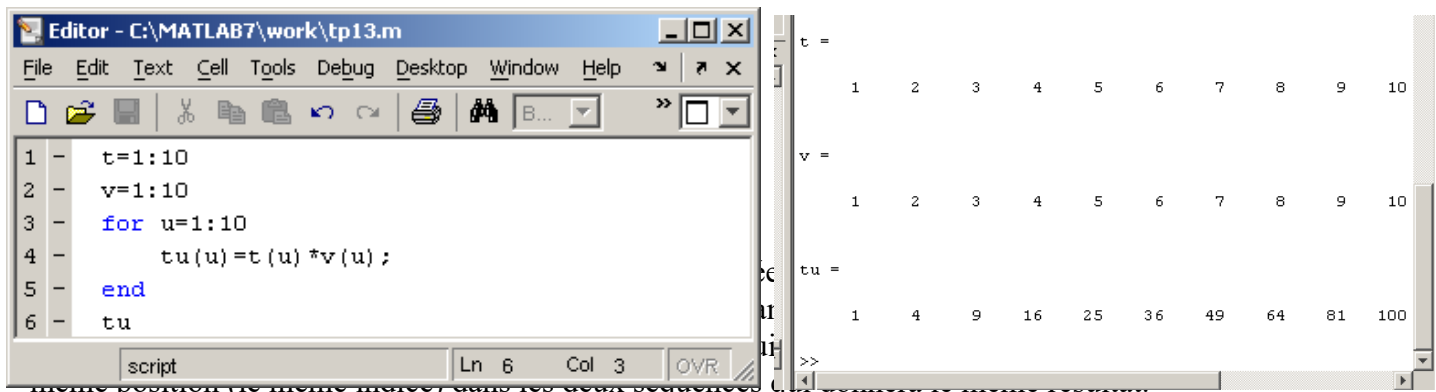




*Conseil : Si vous envisagez d'écrire des programmes compliqués, commencez par un cas simple que vous enrichirez petit à petit en prenant soin de vérifier avant une nouvelle modification que votre programme fonctionne correctement.*

### 3. Opérations en Matlab

Il arrive souvent qu'on applique la même opération à tous les éléments d'un tableau. On peut le faire en effectuant une boucle « for » ... « end » sur l'opération  
(On voit ici un exemple de boucle et de manipulation d'indice) ;



The screenshot shows the MATLAB Editor window with a script named 'tp13.m'. The script contains the following code:

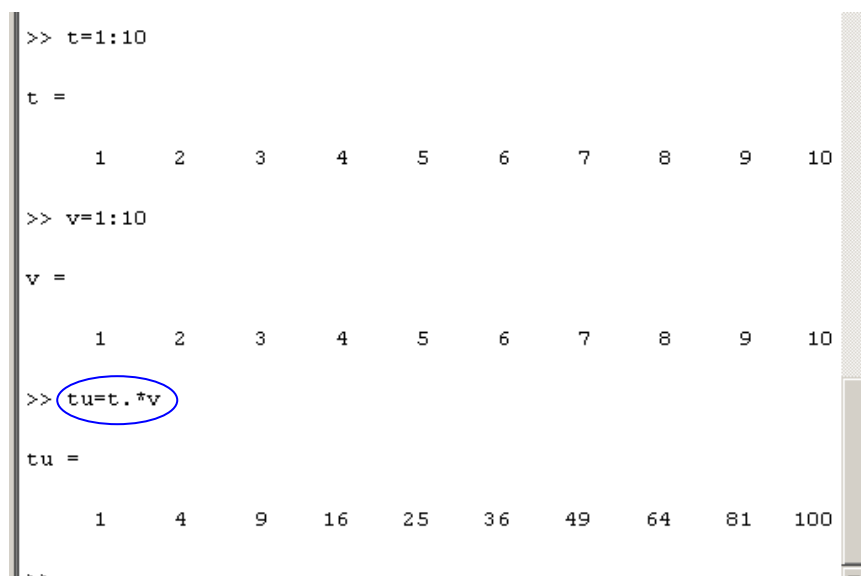
```
1 - t=1:10
2 - v=1:10
3 - for u=1:10
4 -     tu(u)=t(u)*v(u);
5 - end
6 - tu
```

The execution results are displayed on the right side of the window:

```
t =
     1     2     3     4     5     6     7     8     9    10

v =
     1     2     3     4     5     6     7     8     9    10

tu =
     1     4     9    16    25    36    49    64    81   100
```



The screenshot shows the MATLAB Command Window with the following commands and results:

```
>> t=1:10

t =
     1     2     3     4     5     6     7     8     9    10

>> v=1:10

v =
     1     2     3     4     5     6     7     8     9    10

>> tu=t.*v

tu =
     1     4     9    16    25    36    49    64    81   100

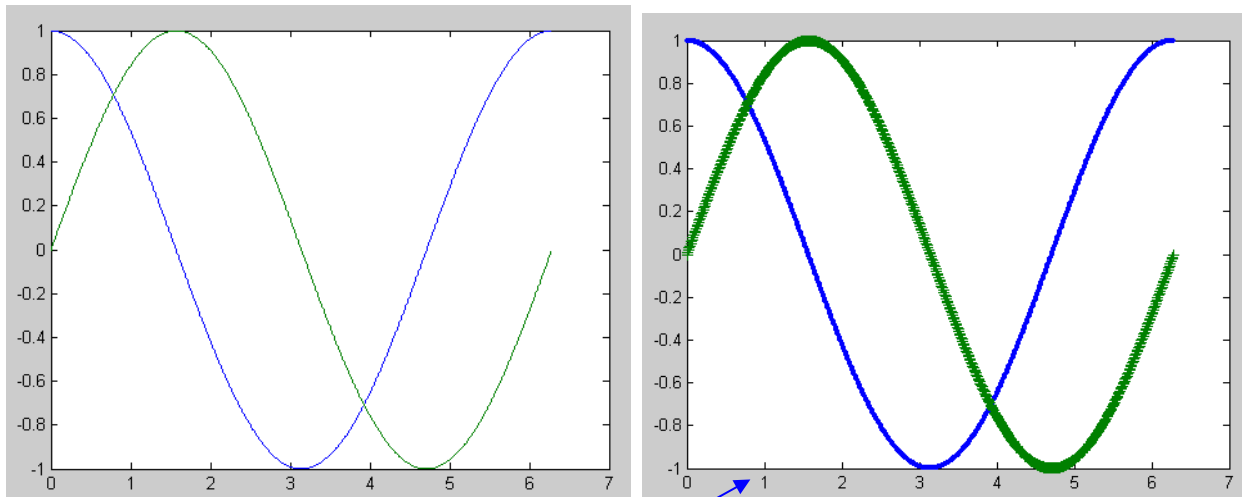
^^
```

### 4. Gestion des graphiques

Nous avons vu un exemple de tracé simple ; en voici un autre où on donne l'abscisse et l'ordonnée pour deux graphes

```
>> x = [0:0.01:2*pi];
>> plot(x,cos(x),x ,sin(x))
```



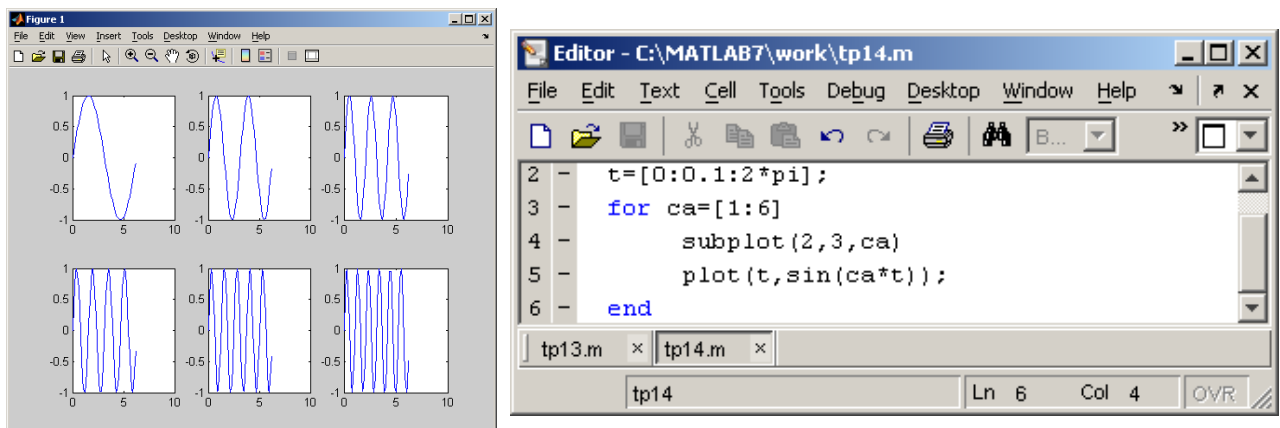


```
clear all
close all % ferme les anciennes figures
figure(1) ; % pour créer une nouvelle fenêtre de figure
x = [0:0.01:2*pi];
plot(x, cos(x),'.',x, sin(x),'+') % cos(x) en points, sin(x) en +
```

Pour rajouter un titre et une légende

```
title('sinus et cosinus'); xlabel('x'); ylabel('f(x)')
legend('cos(x)','sin(x)',0) % le « 0 » place la légende à côté des courbes
```

Il est courant de vouloir afficher plusieurs figures sur le même écran ce qui se fait avec la fonction subplot (2 lignes, 3 colonnes; le dernier paramètre est le numéro de la figure : 1 à 6).



## Autres exemples

Ex : représentation des variables complexes dans le plan complexe

```
>> z1 = 1+2*j; plot(z1,'x')
```

Représentation de plusieurs points ou courbes grâce à **hold** :

```
>> z1 = 1+2*j; plot(z1,'x')
```

```
>> hold on
```

```
>> z2 = -1+2*j; plot(z2,'x')
```

```
>> hold off
```

Ex. : visualiser les courbes :

```
>> plot(sin(pi*(0:0.2:10)))
```

```
>> plot(sin(pi*(0:0.2:10)),'g')
```

```
>> hold on
```

```
>> plot(sin(pi*(0:0.2:10)),'*r')
```

```
>> hold off
```

Utilisation de sous fenêtres :

```
>> subplot(2,1,1)
```

```
>> plot(sin(pi*(0:0.2:10)))
```

```
>> subplot(2,1,2)
```

```
>> plot(sin(pi*(0:0.2:10)),'g')
```

ex. : construisez une fenêtre à 4 quadrants avec différentes fonctions trigonométriques.

Effacement de la fenêtre par **clf**

Exemples de commandes expliquées dans le manuel Matlab

stem	grid	figure	disp
xlabel	ylabel	disp	
title	bar	step	

## 5. Conception de programmes un peu complexes

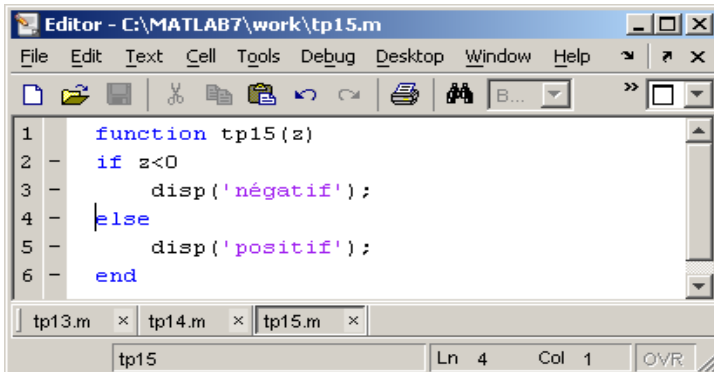
Un programme est une séquence d'instruction qui modifie des données, il est la plupart du temps nécessaire de modifier la séquence à effectuer en fonction d'un résultat de calcul ; on effectue alors un test sur le résultat

*Les tests logiques de base*

Opérateur	Description
Inversion ( <i>NOT</i> a) : ~a	retourne 1 si a est égal 0, 0 si a est égal à 1
Identité (double =) : a == b	retourne 1 si a égale b, 0 autrement
a < b	retourne 1 si a est plus petit que b, 0 autrement
a > b	retourne 1 si a est plus grand que b, 0 autrement
a <= b	retourne 1 si a est plus petit ou égal à b, 0 autrement
a >= b	retourne 1 si a est plus grand ou égal à b, 0 autrement
a ~=b	retourne 1 si a est différent de b, 0 autrement

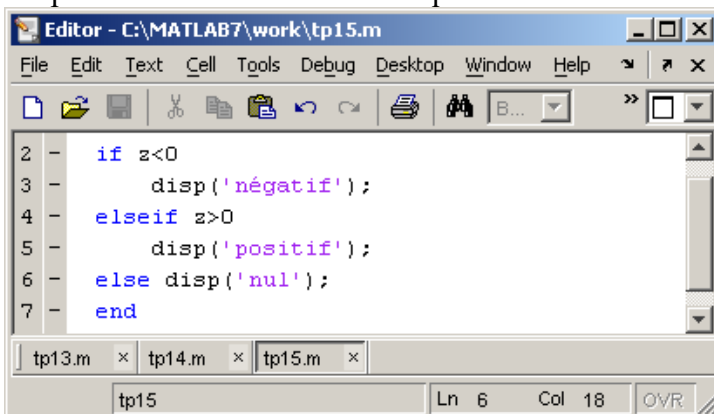
## 6. Utilisation d'un résultat de test pour modifier l'exécution d'un programme

On utilise couramment les tests logiques ci-dessus pour modifier la séquence d'instructions à exécuter : après le `if` on a la séquence d'instructions à exécuter si le résultat du test est « vrai » ; et après le `else` la séquence d'instructions à exécuter si le résultat du test est « faux ». La notion de fonction qui apparaît ici sera expliquée ultérieurement.



```
Editor - C:\MATLAB7\work\tp15.m
File Edit Text Cell Tools Debug Desktop Window Help
1 function tp15(z)
2 if z<0
3     disp('négatif');
4 else
5     disp('positif');
6 end
tp13.m x tp14.m x tp15.m x
tp15 Ln 4 Col 1 OVR
```

On peut combiner des tests en séquence :



```
Editor - C:\MATLAB7\work\tp15.m
File Edit Text Cell Tools Debug Desktop Window Help
2 if z<0
3     disp('négatif');
4 elseif z>0
5     disp('positif');
6 else disp('nul');
7 end
tp13.m x tp14.m x tp15.m x
tp15 Ln 6 Col 18 OVR
```

On peut effectuer des tests plus élaborés de ce type en utilisant l'instruction « **switch** » que nous ne développerons pas dans cette introduction

## 7. Un autre type d'opération courante est la répétition d'une séquence d'instructions (boucle)

Par exemple on répète le même calcul pour des valeurs successives d'un indice (tapez `tp16(1)` par exemple).

The image shows a MATLAB Editor window titled 'Editor - C:\MATLAB7\work\TP16.m'. The code defines a function `function [a]=somme(x)`. It initializes `a(1)=x` and `somme=0`, then enters a `for` loop with `ind=[2:8]`. Inside the loop, it calculates `somme=somme+a(ind-1);` and `a(ind)=somme;`. The function ends with `end`. The command window shows the output of the function: `ans =` followed by a row of values: 3, 3, 6, 12, 24, 48, 96, 192. The status bar indicates the cursor is at line 4, column 13.

```

1 function [a]=somme(x)
2 a(1)=x;
3 somme=0;
4 for ind=[2:8]
5     somme=somme+a(ind-1);
6     a(ind)=somme;
7 end

```

ans =

3 3 6 12 24 48 96 192

>>

Sur le même principe, on peut répéter une séquence tant qu'une condition est vérifiée avec l'instruction de boucle while

The image shows a MATLAB Editor window titled 'Editor - C:\MATLAB7\work\tp17.m'. The code defines a function `function [z]=tp17(frequence)`. It initializes `x=0` and `deltax=0.001`, then enters a `while` loop with the condition `sin(2*pi*frequence*x)>=0`. Inside the loop, it increments `x=x+deltax;`. The function ends with `end`. The status bar indicates the cursor is at line 3, column 1.

```

1 function [z]=tp17(frequence)
2 x=0;
3 deltax=0.001;
4 while(sin(2*pi*frequence*x)>=0)
5     x=x+deltax;
6 end
7 z=x;
8 end

```

## 8. Gestion des séquences (vecteurs, matrices) :

Matlab, comme son nom l'indique a d'abord été conçu pour optimiser et faciliter le calcul sur les matrices, dont les vecteurs sont un cas particulier ; pour donner les valeurs des composantes d'un vecteur ligne on écrit entre crochets (on peut séparer les éléments par des virgules ou par des blancs) :

```

>> vligne = [2, 4, 7];
>> vligne

vligne =

```

On utilise les « deux  
sous la forme suivante :

**[premier  
dernier élément]**

2 4 7

points » pour donner des intervalles

**élément : pas d'incrémentation :**

Par exemple si le premier élément est 0, on obtient :

```
>> w = [0 : 0.5 : 4];
>> w

w =

Columns 1 through 7
    0    0.5000    1.0000    1.5000    2.0000    2.5000
3.0000

Columns 8 through 9
```

Si le pas d'incrémentation est positif et que le premier élément est plus grand que le dernier, Matlab crée un vecteur vide.

```
>> v = [5:0.5:4];
>> v

v =

Empty matrix: 1-by-0
```

### Autres exemples de l'utilisation des deux points

```
>> x = 1:10
>> x = 3:0.2:5
>> x = 10:-1:1
```

Les deux points permettent donc de générer des vecteurs (remplis de valeurs équidistantes) : `a=1:0.25:2` génère le vecteur `a` (1 1.25 1.5 1.75 2).

Il est possible d'inclure le ':' dans une fonction MATLAB :

```
>> x = sin(2*pi*(0:0.25:2)/2)
>> x = [1:3,5:7,9:11]
>> x = [1:3;5:7;9:11]
```

On crée des vecteurs colonnes ou des matrices en séparant les lignes par des points virgules « ; »

```
>> matrice=[1,2,3;4,5,6;7,8,9]

matrice =

     1     2     3
     4     5     6
     7     8     9
```

Les opérations applicables aux matrices s'écrivent simplement. Par exemple on pourra faire le produit de deux matrices

```
>> matrice1=[1,2;3,4];
>> matrice2=[5,6;7,8];
>> matrice1*matrice2

ans =

    19    22
    43    50
```

On peut encore utiliser la notation condensée «.\*» qui effectue ici encore un produit terme à terme :

```
>> matrice1.*matrice2  
  
ans =  
  
     5     12  
    21     32
```

L'opération  $A*B$  est la multiplication matricielle des matrices A et B de dimensions respectives M.N et N.P, tandis que  $A.*B$  est la multiplication élément par élément de 2 matrices de dimensions identiques.

On accède à un élément du vecteur ou de la matrice en tapant

```
>> matrice1(2,1)  
  
ans =  
  
     3
```

On peut aussi sélectionner un sous-ensemble par exemple une ligne ou une colonne de la matrice en remplaçant un des indices par « : »

```
>> matrice1(2,:)   
  
ans =  
  
     3     4
```

```
>> matrice1(:,2)  
  
ans =  
  
     2  
     4
```

### Autres exemples

Définition de matrices :

```
>> A = [1 2 3 ; 4 5 6 ; 7 8 9]
```

```
>> B = [1, 2, 3 ; 4, 5, 6 ; 7, 8, 9]
```

Définition de vecteurs :

```
>> x = [1.1 2.2 3.3]
```

```
>> y = [3 ; 4 ; 5 ; 6 ; 7]
```

Modifications :

```
>> B(3,1) = 21;B
```

```
>> y(5) = 22;y
```

```
>> C = [A,B]
```

```
>> D = [A;B]
```

Extraction de vecteurs et sous-matrices :

```
>> x(:,2)
```

```
>> x(:,1:2)
```

Etudier le résultat des commandes suivantes, qui extraient des éléments différents de la matrice A définie par  $A=[1\ 2\ 3\ ;\ 4\ 5\ 6\ ;\ 7\ 8\ 9]$ .

```
>> D = A(2,1)
```

```
>> E = A([1,2],[2,3])
```

```
>> F = A(:,2)
```

```
>> G = A(2,:)
```

Que contiendra la matrice H suivante :  $H = A([3,1],:)$  ?

**ATTENTION** : Pas d'i

La taille des matrices et des vecteurs est donnée par **size**(nom\_variable)

Il est possible de définir des matrices complexes : `>> E=[1 2;3 4]+j*[5 6;7 8]`

La transposition des matrices s'obtient par `'`.

```
>> a = [1;2;3;4]; b = [5;6;7;8];
>> C = a'*b
>> D = a*b'
>> A2 = A.^2
>> AA = A^2
>> A.*B
>> A*B
X = A\B (équivalent à A-1*B) est solution de A*X=B
alors que X=A/B (B*A-1) est solution de X*A=B.
Prenez des exemples pour vérifier ces deux solutions
d'équations matricielles
```

On peut utiliser la multiplication de matrices pour effectuer des sommations de la forme

$$S = \sum_{n=0}^{N-1} a_n b_n = a \cdot b'$$

si  $a$  et  $b$  sont des vecteurs de dimensions  $N$ , de composantes  $a_n$  et  $b_n$

respectivement (notation très utilisée en traitement du signal).

Il existe de nombreuses fonctions prédéfinies manipulant les matrices : **inv**, **det**, **fliplr**, ...

**Réfléchissez à la résolution d'un système d'équations linéaires :**

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & -2 & 4 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$$

## 9. Fonctions

Pour éviter de rédiger des programmes trop longs et répétitifs, on peut écrire des séquences de lignes de programmation sous la forme de fonctions

Donnée à calculer = nom de la fonction (suite des variables auxquelles s'applique le calcul)

Pour effectuer le calcul on frappe le nom de la fonction avec comme arguments les valeurs des variables pour lesquelles on veut faire le calcul

Une fonction peut renvoyer plusieurs données : dans ce cas, la liste des noms de variables où ces données seront rangées, est donnée entre crochets

```
function [sortie1, sortie2, ...] = nom_de_fonction(entréel, entrée2, ...)
```



Pour créer une fonction, il faut utiliser le mot clé **function**. Attention le nom de la fonction doit être identique au nom du fichier Matlab (extension .m).

Exemple. fonction **fact** avec paramètre n en entrée, définie dans le fichier **fact.m** :

```
function f=fact(n)
    f=1;
    for i=1:n
        f=f*i;
    end
```

Toutes les variables définies dans la fonction sont locales à cette fonction. Ce n'est pas le cas des fichiers qui ne débutent pas par function.

Une fonction peut retourner plusieurs résultats :

```
function [x,y]=calcul(a,b)
x=a+b ;
y=a*b ;
```

ex. appel par [u,v]=calcul(2,3)

## Quelques fonctions utiles

Matlab intègre un grand nombre de fonctions dont la description est donnée dans l'aide ; voici quelques fonctions utiles.

Sur les matrices

```
>> V = [0:0.1:10]; % utilisation de length
- vecteur 1x101
>> n = length(V)

n =

    101
```

```
>> M = [1 2 3; 4 5 6]; % utilisation de size - matrice 2x3
>> [n,m] = size(M)

n =

     2

m =

     3
```

Notez que l'appel d'une fonction permet de renvoyer plusieurs résultats (arguments) dont la liste est donnée entre crochets : [n,m]

Ici la fonction size donne le nombre de lignes et de colonnes de la matrice

La fonction exponentielle s'écrit exp(x) ; toutefois l'écriture d'un exposant utilise l'accent circonflexe qui permet d'effectuer le calcul de puissance :

2 puissance 3 s'écrit :

```
>> 2^3  
ans =  
      8
```

Il existe aussi des fonctions écrites par d'autres programmeurs qu'on peut réutiliser en s'assurant tout de même que la fonction réalise bien ce qu'elle est censée faire ... Une des richesses de Matlab provient de ce qu'il est un outil utilisé par de très nombreux ingénieurs et est ainsi un outil permettant des échanges de programmes adaptés à différents types d'application des mathématiques dans le monde industriel ou de la recherche.

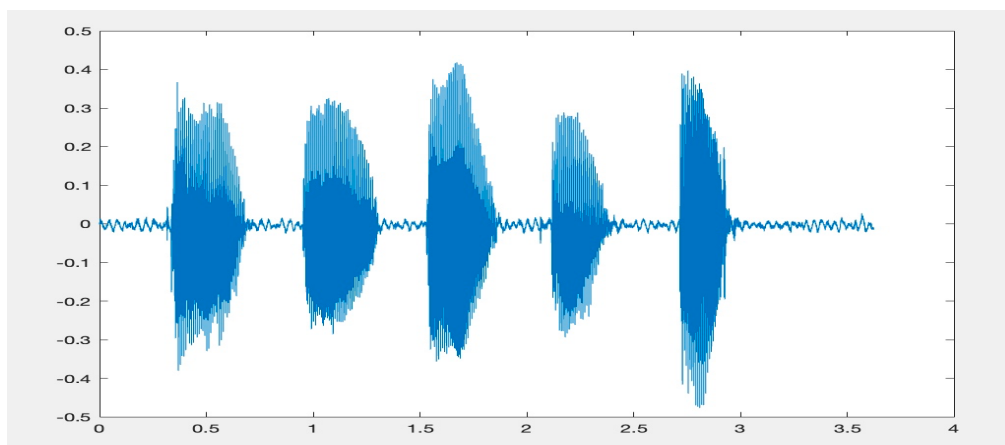
- Effectuer une recherche sous Google en tapant Matlab
- Visitez le site <http://www.mathworks.fr/>
- 

## 10. Entrées Sorties de sons et d'images :

On trouvera des explications détaillées dans l'aide de Matlab

### Lire des fichiers sons

Exemple de lecture d'un signal enregistré suivi de son tracé :



**Importez le fichier AIUEO.wav du dépôt moodle. Utilisez audioread à la place de wavread si votre version Matlab ne le permet plus, idem pour audiowrite à la place de wavwrite.**

**Affichez le signal sonore avec comme axe x, la bonne échelle de temps. Écoutez le signal sonore en utilisant la fréquence d'échantillonnage du signal.**

**Lire des fichiers d'images :** voir la fonction imread dans l'aide Matlab

**Générer des fichiers d'images :** voir la fonction imwrite dans l'aide Matlab

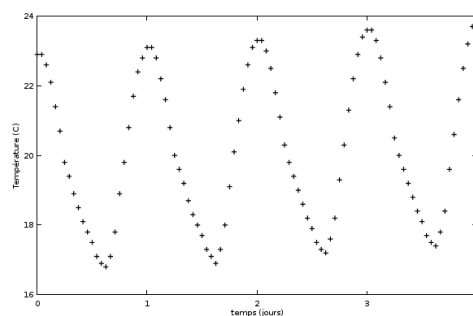
Importez des images du web ou des copies d'écran et visualisez-les.

# 11. Mise en pratique

## Première partie

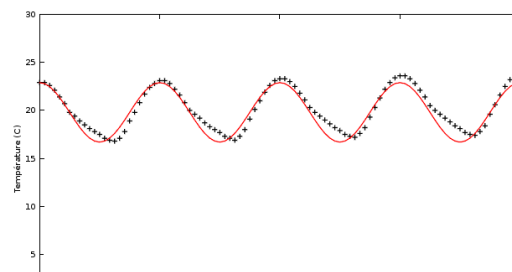
- Ecrire un script qui génère un signal sinusoïdal de fréquence 1kHz et d'amplitude 10 sur une durée de 2s.
- Ecrire un script qui génère un signal sinusoïdal de fréquence 5kHz et d'amplitude 10 sur une durée de 2s.
- Vous trouverez dans [le fichier data.dat](#) sur Moodle un relevé horaire de températures sur une durée de trois jours. Écrire un script qui :
  - importe les données sous Matlab,
  - visualise les données,
  - approxime le signal par la fonction  $x(t)=m + \cos(2\pi t / T)$  avec  $m=19.8$ ,  $a=3$ .  $T$  représente la période estimée,
  - approxime le signal par la fonction  $x(t)=(m+bt) + \cos(2\pi t / T)$  avec  $b=1$  degré/4 jours
  - calcule la distance euclidienne entre le signal expérimental et chacun des deux modèles

Ci-dessous, les résultats à obtenir :

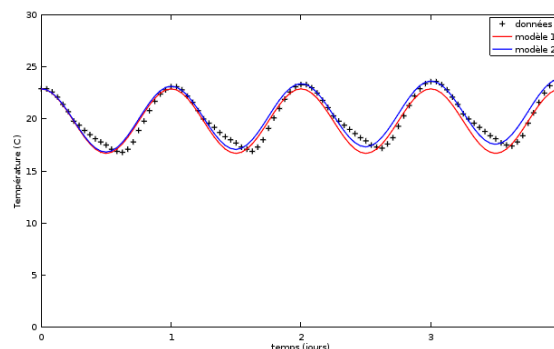


Représentation des données

$x(t)=m + \cos(2\pi t / T)$  avec  $m=20$ ,  $a=3$ . ( $d=7.22$ )



$x(t)=(m+bt) + \cos(2\pi t / T)$  avec  $b=1/4$ . ( $d=6.59$ )



- Lisez le fichier chf07.m présent sur moodle. Affichez plusieurs signaux. Combien de signaux sont labélisés N, r et V. Divisez l'ensemble de données en ensembles d'entraînement et de test. Dans l'ensemble d'entraînement, incluez 60 % des échantillons afin de maintenir la distribution naturelle des anomalies. Excluez les échantillons de la classe V de l'ensemble d'entraînement, mais incluez-les dans l'ensemble de test.

## Deuxième partie

Sur Moodle, vous trouverez les images `pmd_0_distance_2451.bmp`, et `pmd_1_amplitude_2709` que obtenues via une caméra PMD Technologies. Précisez la technologie employée. Donnez quelques exemples d'applications de ce type de technologie.



Nous souhaitons séparer les "objets" présents dans la scène, autrement dit obtenir 3 images (voir la figure ci-dessous) :

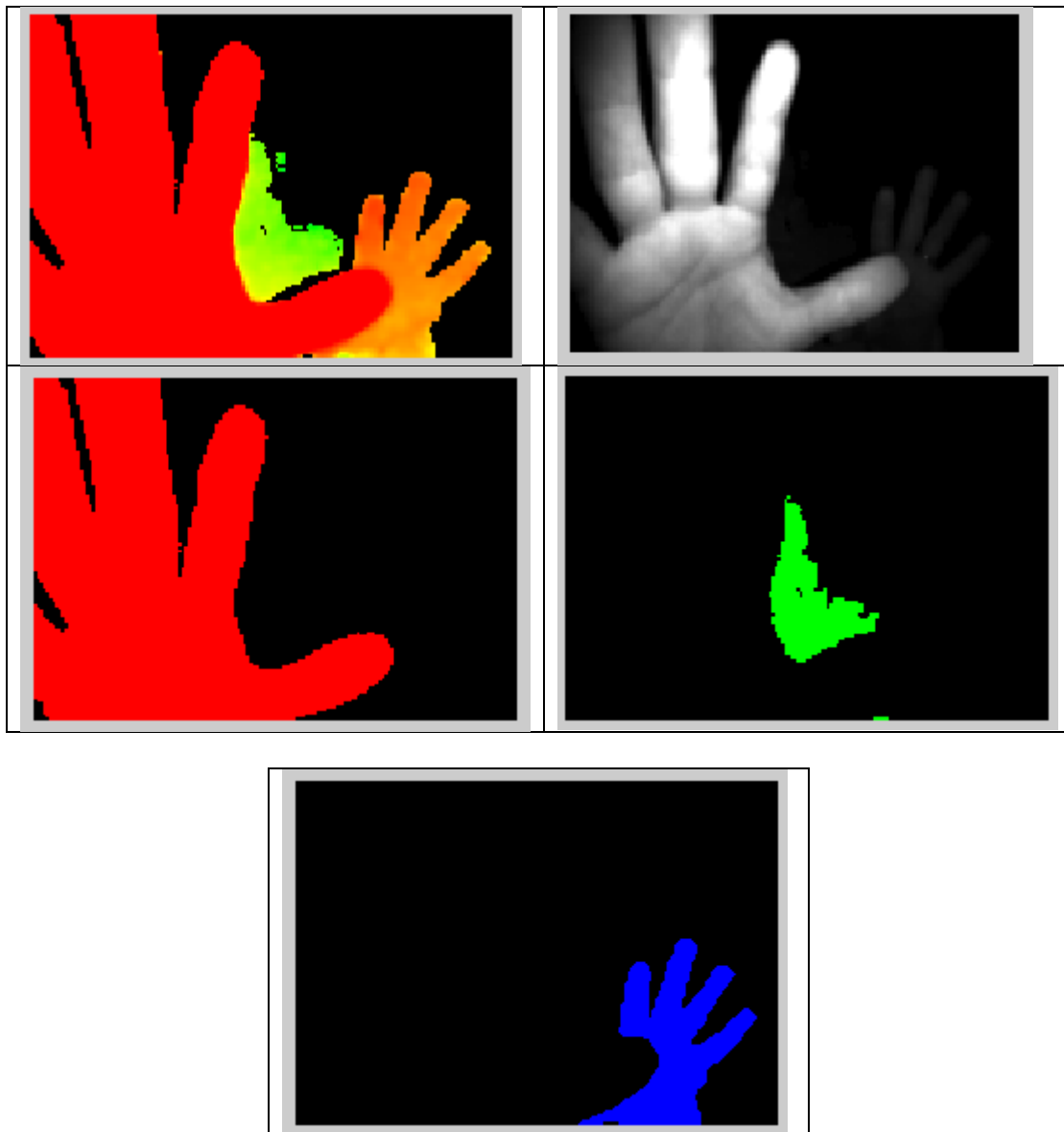
- la première représentant le fond,
- la seconde, la main droite,
- et enfin, la troisième la main gauche.

Recherchez un algorithme effectuant ce traitement.

Comment peut-il être optimisé en utilisant les performances de calcul vectoriel de Matlab ?

Ecrivez la nouvelle version.

Remarque : afin de supprimer quelques artefacts, vous pourrez utiliser la fonction matlab `imopen`.

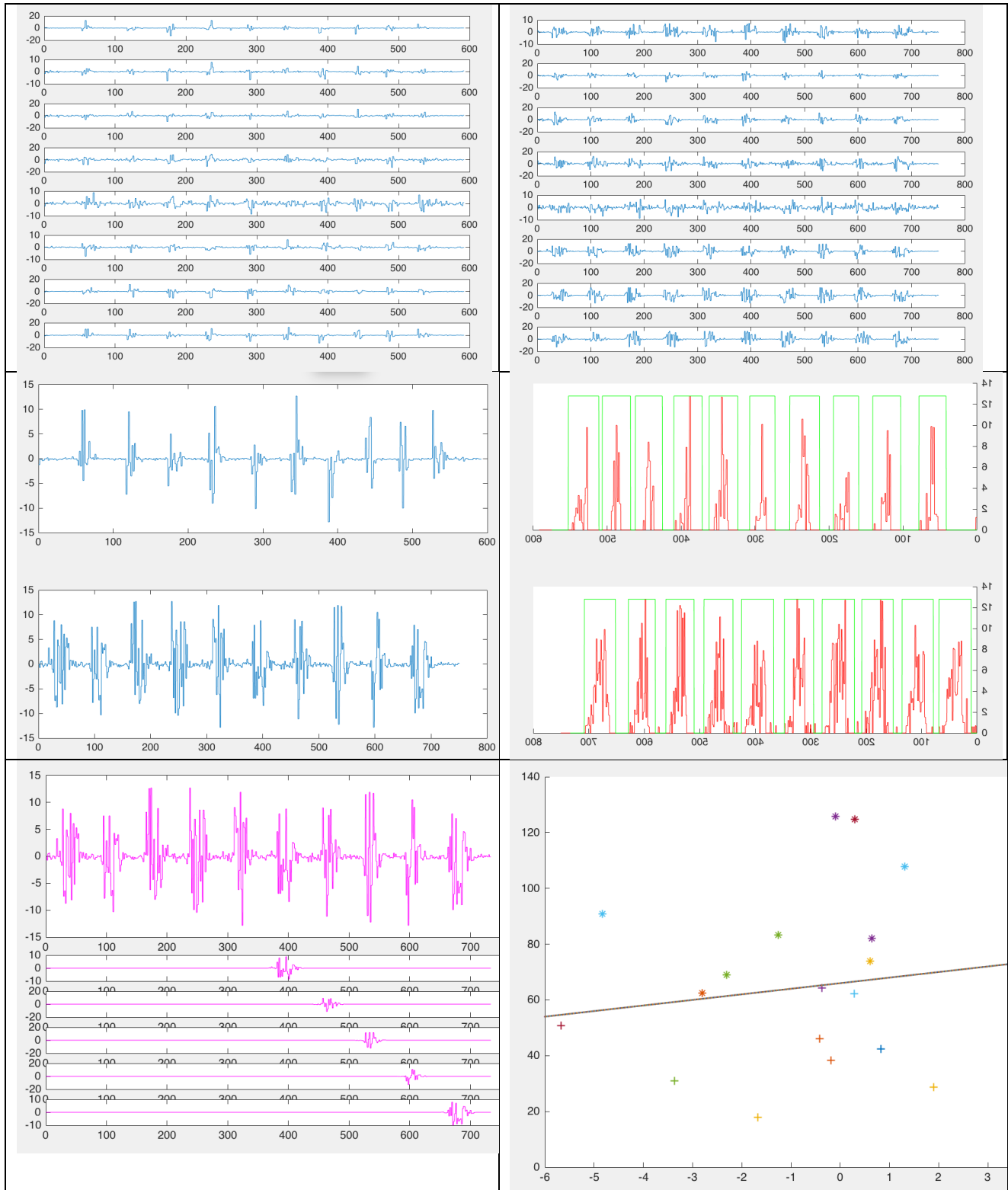


## Troisième partie

Consultez le site : <https://www.myo.com>

Qu'est-ce l'électromyographie de surface ?

A l'aide des signaux sauvegardés sur les fichiers double\_tapMat.dat et fistMat.txt, affichez les fenêtres graphiques suivantes (une présentation de ces signaux vous sera faite).



## 12. Accès au fichiers de commandes et fonctions – Aide en ligne

Depuis la ligne de commande Matlab, il est possible de modifier le répertoire courant à l'aide des commandes habituelles (**cd**, **pwd**, **ls**).

Lorsque vous demandez l'exécution d'un fichier de commande ou d'une fonction, Matlab recherche le fichier correspondant dans le répertoire courant et dans tous les répertoires définis dans le **path**. Pour mettre à jour le **path**, vous utiliserez la commande suivante :

```
>> path (path ; 'w:\repertoire1\repertoire2\');
```

Pour avoir l'aide sur une commande il suffit de saisir : **help** nom\_de\_la\_commande

## 13. Réalisations de quelques signaux

Une impulsion de Dirac  $d[n]$  de longueur  $N$  peut être générée en utilisant la commande Matlab :

```
d=[1 zeros(1,N-1)] ;
```

Cette séquence peut être retardée de  $M$  échantillons par la commande :

```
dr=[zeros(1,M) 1 zeros(1,N-M-1)] ;
```

La séquence échelon de pas unité de longueur  $N$  peut être générée en utilisant la commande Matlab :

```
e=[ones(1,N)] ;
```

1. Générez une séquence échelon retardée de  $M$  échantillons

Le programme suivant génère une impulsion de Dirac et affiche sous la forme d'un graphique la séquence des échantillons

```
% Programme P1_1
% Génération d'une impulsion de Dirac d'amplitude 1
clf ;
% Génération du signal de -10 à 20
n=-10 :20 ;
u=[zeros(1,10) 1 zeros(1,20)] ;
% Affichage du signal
stem(n,u) ;
xlabel('Index temporel n') ;ylabel('Amplitude') ;
title('Séquence impulsion unité') ;
axis([-10 20 0 1.2]) ;
```

2. Lancez le programme P1\_1
3. Quels sont les buts des commandes `clf`, `axis`, `title`, `xlabel`, `ylabel` ?
4. Modifiez le programme pour générer une impulsion unité retardée de 11 échantillons.
5. Modifiez le programme pour générer une séquence échelon avec une avance de 7 échantillons.

```
% Programme P1_2
% Génération d'une séquence temporelle exponentielle complexe
```

```

clf;
c = -(1/12)+(pi/6)*i;
K = 2;
n = 0:40;
x = K*exp(c*n);
subplot(2,1,1);
stem(n,real(x));
xlabel('index temporel n');ylabel('Amplitude');
title('Partie réelle ');
subplot(2,1,2);
stem(n,imag(x));
xlabel('index temporel n');ylabel('Amplitude');
title('Partie imaginaire');

```

```

% Programme P1_3
% Génération d'une séquence temporelle exponentielle réelle
clf;
n = 0:35; a = 1.2; K = 0.2;
x = K*a.^n;
stem(n,x);
xlabel('Index temporel n');ylabel('Amplitude');

```

6. Lancez le programme P1\_2 et générez la séquence temporelle à valeurs complexes.
7. Quel paramètre contrôle le taux de croissance ou de décroissance de cette séquence et celui de l'amplitude ?
8. Que se passe-t-il si le paramètre  $c$  devient  $(1/12)+(pi/6)*i$  ?
9. Quelles sont les buts des fonctions `real` et `imag` ?
10. Lancez le programme P1\_3 et générez la séquence temporelle à valeurs réelles.
11. Quel paramètre contrôle le taux de croissance ou de décroissance de cette séquence et celui de l'amplitude ?
12. Quel est la différence entre les opérateurs `^` et `.^` ?
13. Que se passe-t-il si  $a$  devient plus petit que 1 (ex :0.9) ?
14. Quelle est la longueur de la séquence et comment peut-elle être modifiée ?
15. Vous pouvez utiliser la commande `sum(s.*s)` pour calculer l'énergie de la séquence  $s$ .  
Évaluez l'énergie des séquences générées aux questions 10 et 14.

Un des signaux les plus utilisés en traitement du signal est la séquence sinusoïdale/co-sinusoïdale. Le programme P1\_4 est un exemple simple de génération d'une telle séquence.

```

% Programme P1_4
% Génération d'une séquence sinusoïdale
n = 0:40;
f = 0.1;
phase = 0;
A = 1.5;
arg = 2*pi*f*n - phase;
x = A*cos(arg);
clf; % Supprime les anciens graphes
stem(n,x); % Plot affiche la séquence générée
axis([0 40 -2 2]);
grid;

```

```

title(' Séquence sinusoidale');
xlabel('Index temporel n');
ylabel('Amplitude');
axis;

```

16. Lancez le programme P1\_4 et générez la séquence sinusoïdale.
17. Quelle est la fréquence du signal et comment peut-elle être modifiée ? Quels paramètres contrôlent :
  - la phase,
  - l'amplitude,
 Quelle est la période ?
18. Calculez la puissance moyenne de la séquence ?
19. Modifiez le programme pour générer un signal de fréquence égale à 1, puis 0.5, 0.9, et enfin 1.1 ? Comparer les différents signaux générés.
20. Modifier le programme pour générer une séquence de longueur 50, de fréquence 0.08, d'amplitude 2.5 et de phase 90.
21. Remplacez la commande stem par la commande plot, puis par la commande stairs ? Quelles sont les différences entre les différentes représentations ?

Un signal aléatoire de longueur N dont les échantillons sont uniformément distribués sur l'intervalle [0,1] peut être généré en utilisant la commande Matlab

```
x=rand(1,N) ;
```

Un signal aléatoire de longueur N dont les échantillons sont distribués selon une loi gaussienne de moyenne nulle et de variance unité, peut être générée en utilisant la commande Matlab

```
X=randn(1,N) ;
```

22. Ecrire un programme générant et affichant un signal aléatoire de longueur 100 dont les éléments sont uniformément distribués dans l'intervalle [-2 2] ;
23. Ecrire un programme générant et affichant un signal aléatoire de densité de probabilité gaussienne de 75 échantillons, de valeur moyenne nulle et de variance 3 ;
24. Ecrire un programme générant et affichant cinq séquences d'un signal aléatoire de longueur 31 et vérifiant
 
$$x(n) = A \cos(\omega_0 n + \varphi)$$
25. Où l'amplitude A et la phase  $\varphi$  sont des variables aléatoires statistiquement indépendantes de distribution uniforme dans l'intervalle [0, 4] pour l'amplitude et [0,  $2\pi$ ] pour la phase.