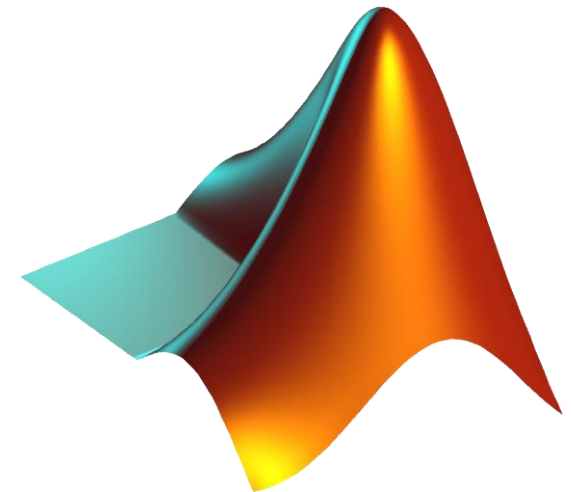
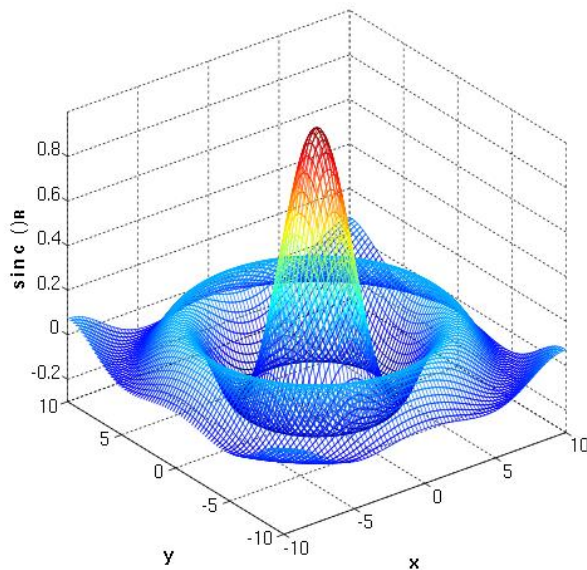


OUTILS NUMERIQUES POUR L'INGENIEUR I

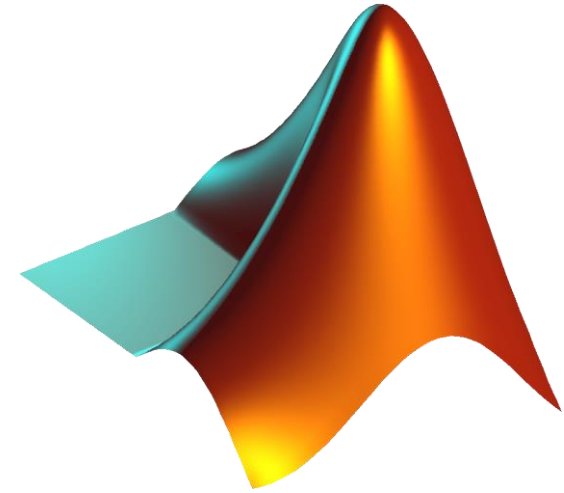
MATLAB



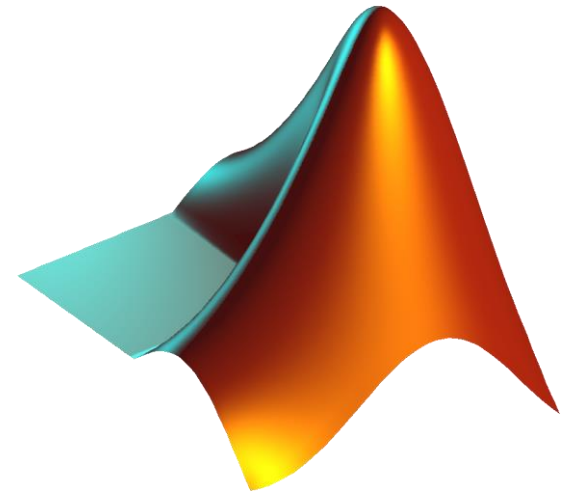
B. KADOCH et C. MARIANI

Plan du cours

- 1) Présentation
- 2) Environnement
- 3) Commandes principales
- 4) M-files: Scripts, Fonctions
- 5) Utilisation de Matlab pour le calcul scientifique:
Quelques applications
- 6) Lire et écrire des données dans différents formats de fichiers



Présentation de Matlab



Présentation de Matlab

Logiciel commercial de calcul et de développement conçu pour des utilisateurs scientifiques (Industrie + Recherche)

- Mathématiques et calculs
- Développement d'algorithmes
- Modélisation, simulation et prototypes
- Analyse, exploration et visualisation de données
- Opérations avec des matrices

Toolboxes (boîtes à outils)

Traitement du signal, d'images

Optimisation, contrôle, équations différentielle ...

Présentation de Matlab

Langage interprété:

lors de l'exécution d'un programme, le code est traduit automatiquement en langage machine

- Matlab traite des données numériques et non pas des équations formelles
- particulièrement adapté au calcul matriciel

Alternatives open source

Octave: <https://www.gnu.org/software/octave/>

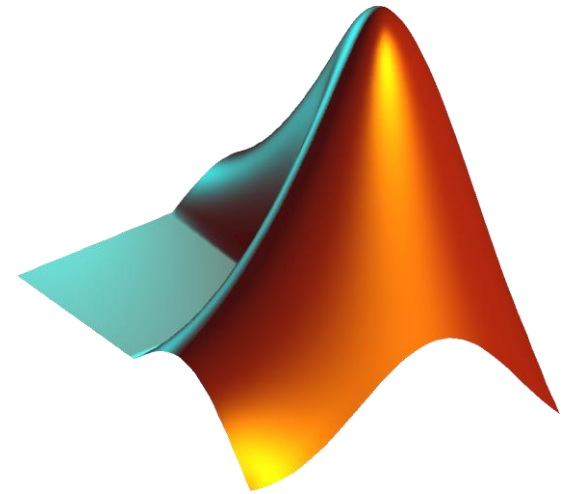
Scilab: <http://www.scilab.org/fr/>

Présentation de Matlab

AVANTAGES

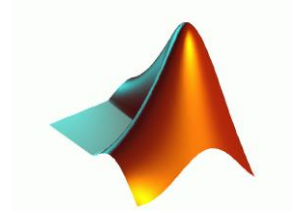
- Le langage matlab est simple.
- Matlab a été conçu pour scientifiques et des ingénieurs.
- L'environnement est interactif.
- L'intégration des outils graphiques et mathématiques.
- Toolboxes et Simulink amènent une puissance et une souplesse.
- Les outils de calcul symbolique amènent une dimension qui ne se trouve pas dans les langages comme Fortran, C++, Java,...

Environnement



Environnement : Démarrer Matlab

- Sous Windows : double-clique sur l'icône►
ou passer par le menu démarrer
- Sous Unix ou Linux : taper matlab dans la console



Un environnement de développement intégré ([IDE](#)) s'ouvre :

- ✓ fenêtre de commandes ([Command Window](#))
- ✓ fenêtre de l'historique des commandes ([Command History](#))
- ✓ fenêtre du répertoire courant ([Current Directory](#))
- ✓ fenêtre des variables définies ([Workspace](#))
- ✓ fenêtre [graphique](#)

Environnement

Aide
contextuelle

Répertoire de travail

Parcourir les
répertoires

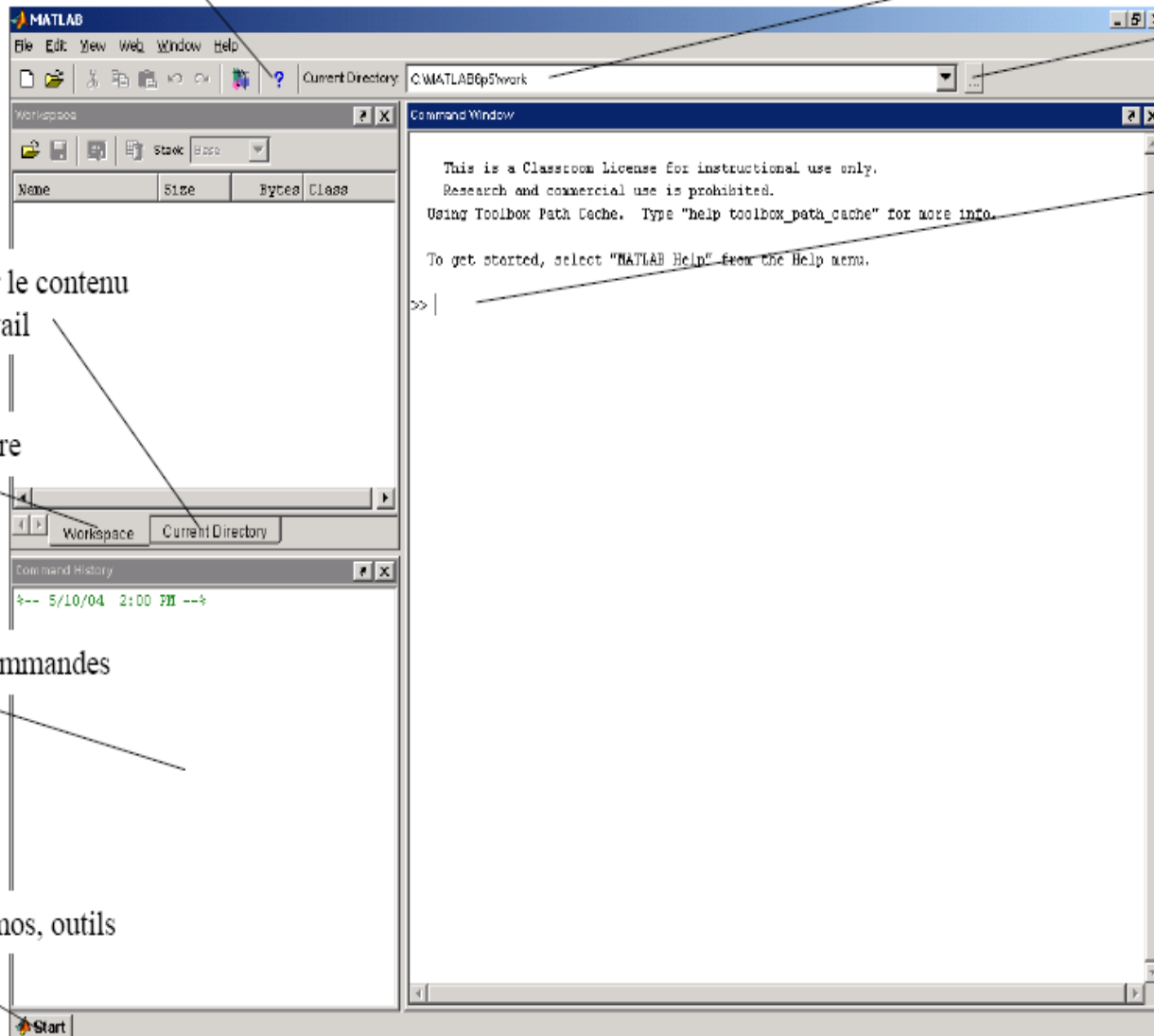
Ligne de
commande

Permet de visualiser le contenu
du répertoire de travail

Variables en mémoire

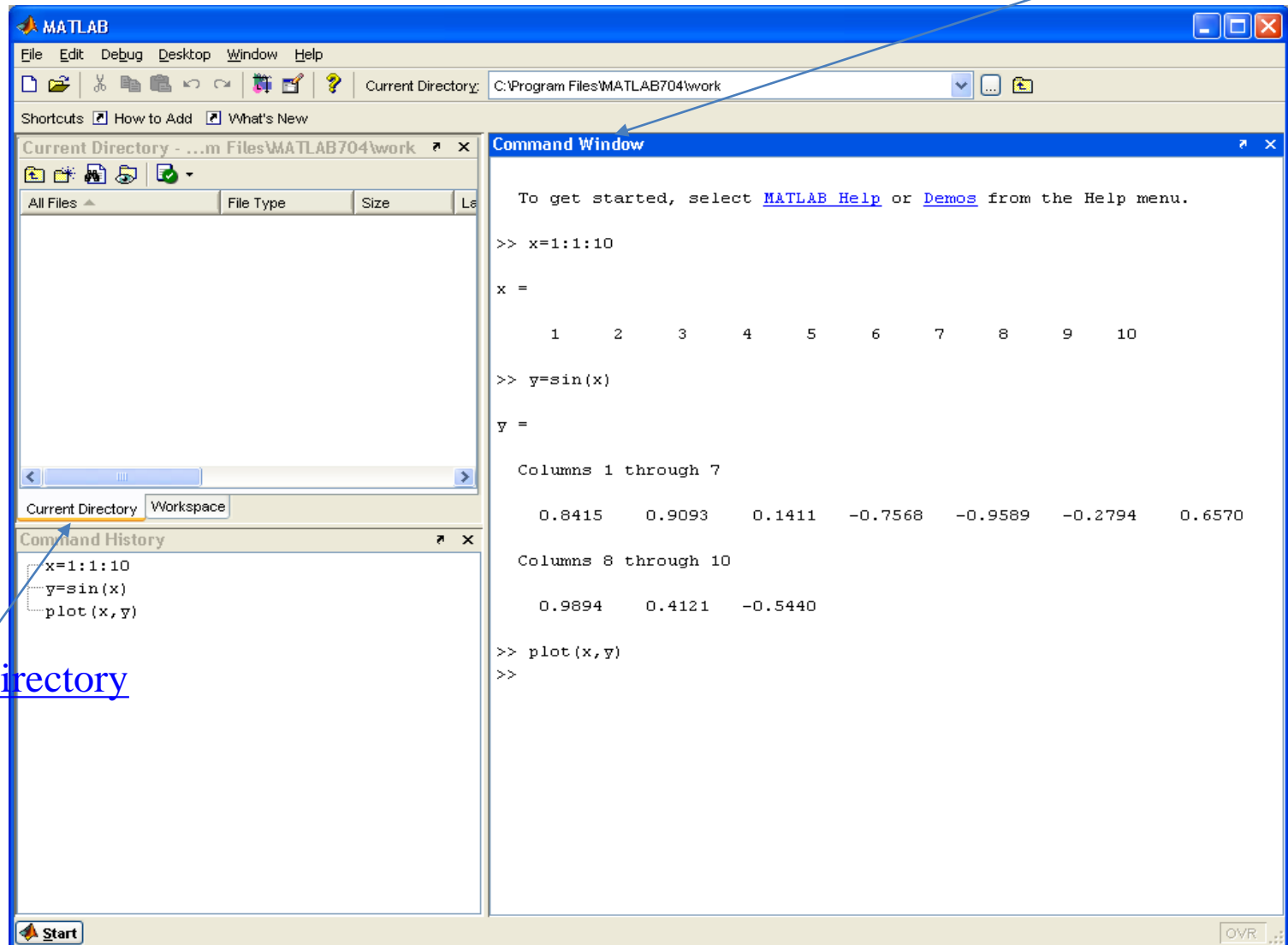
Historique des commandes

Documentation, démos, outils



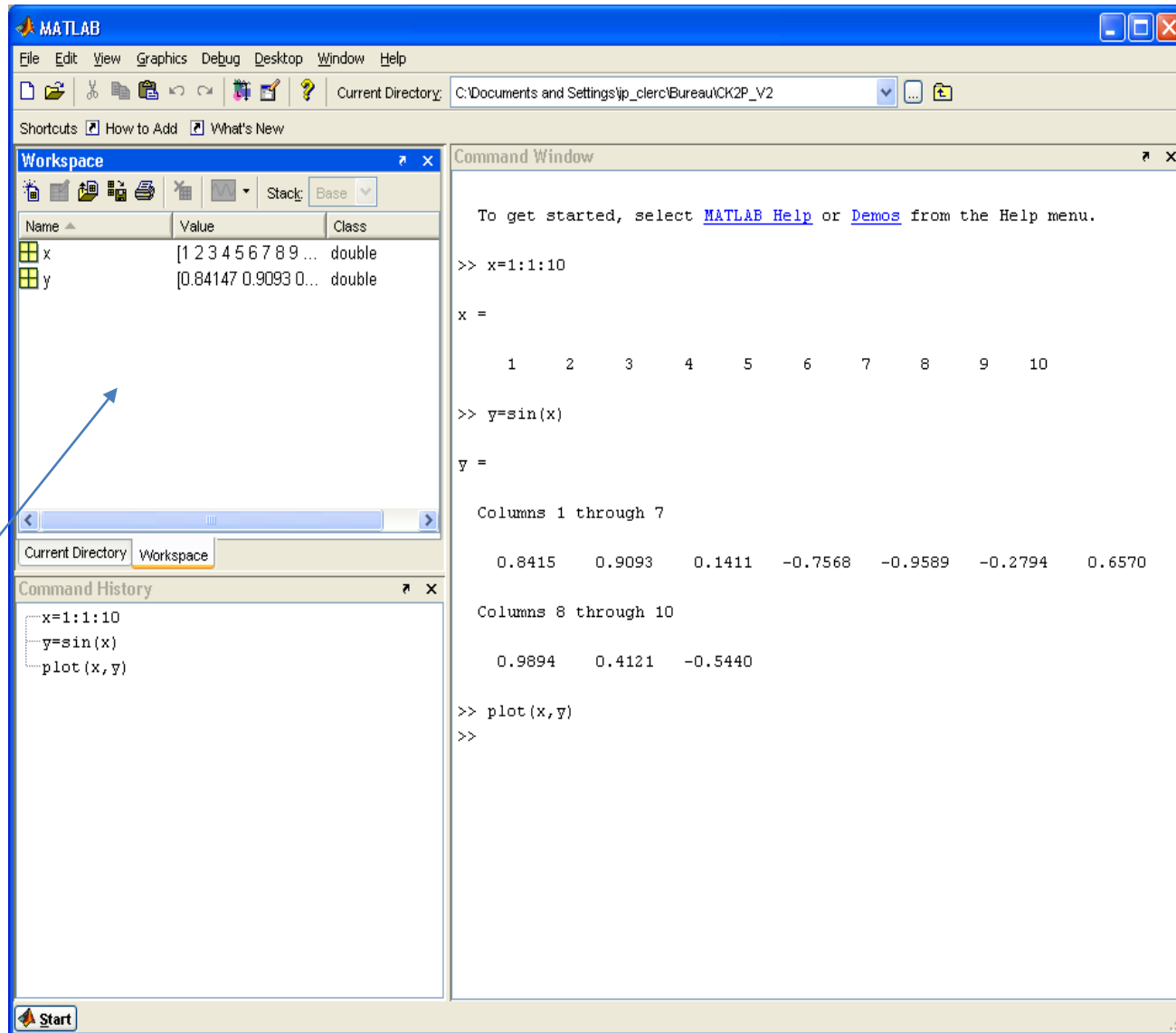
Environnement

Command Window



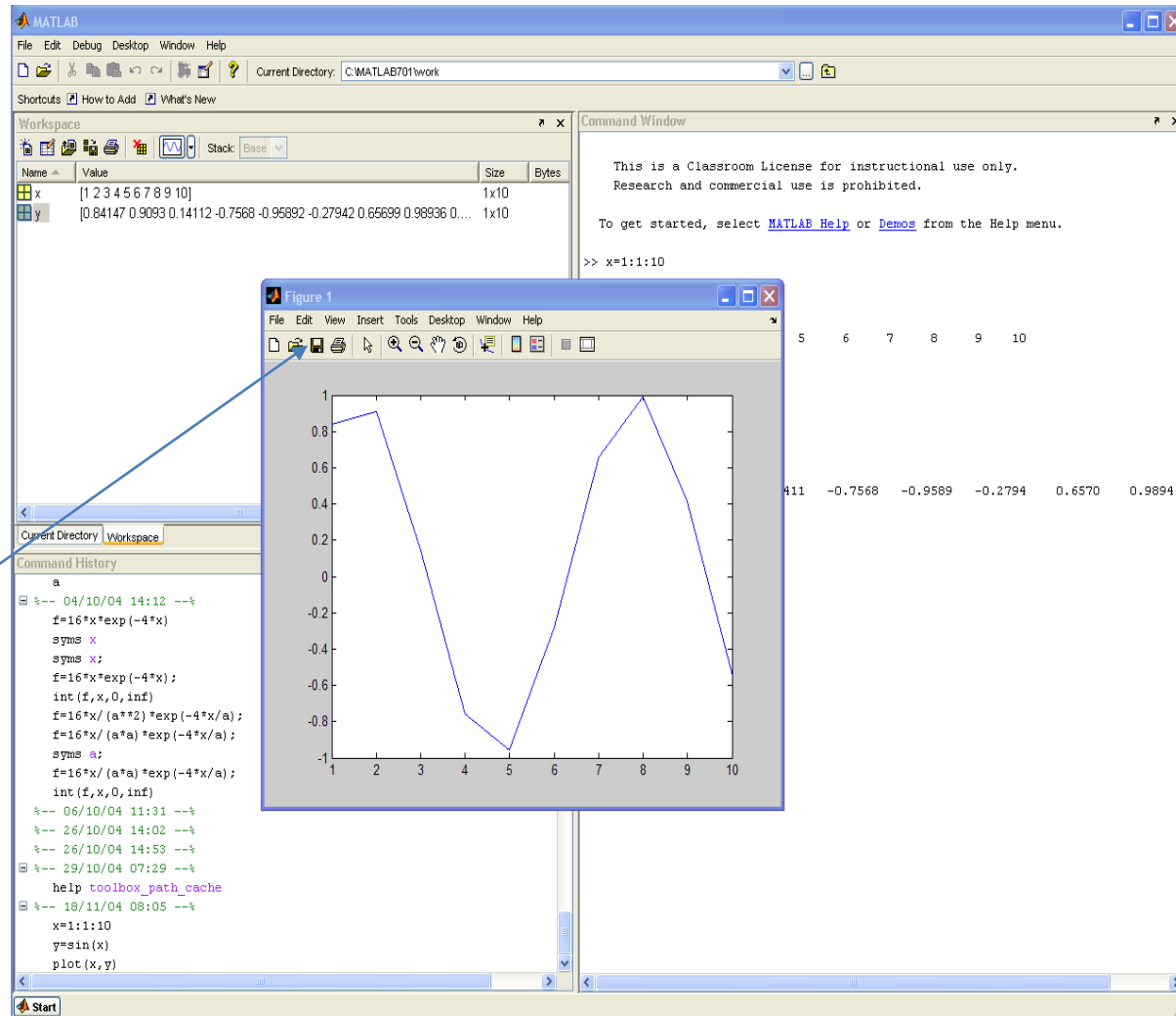
Current Directory

Environnement



Workspace

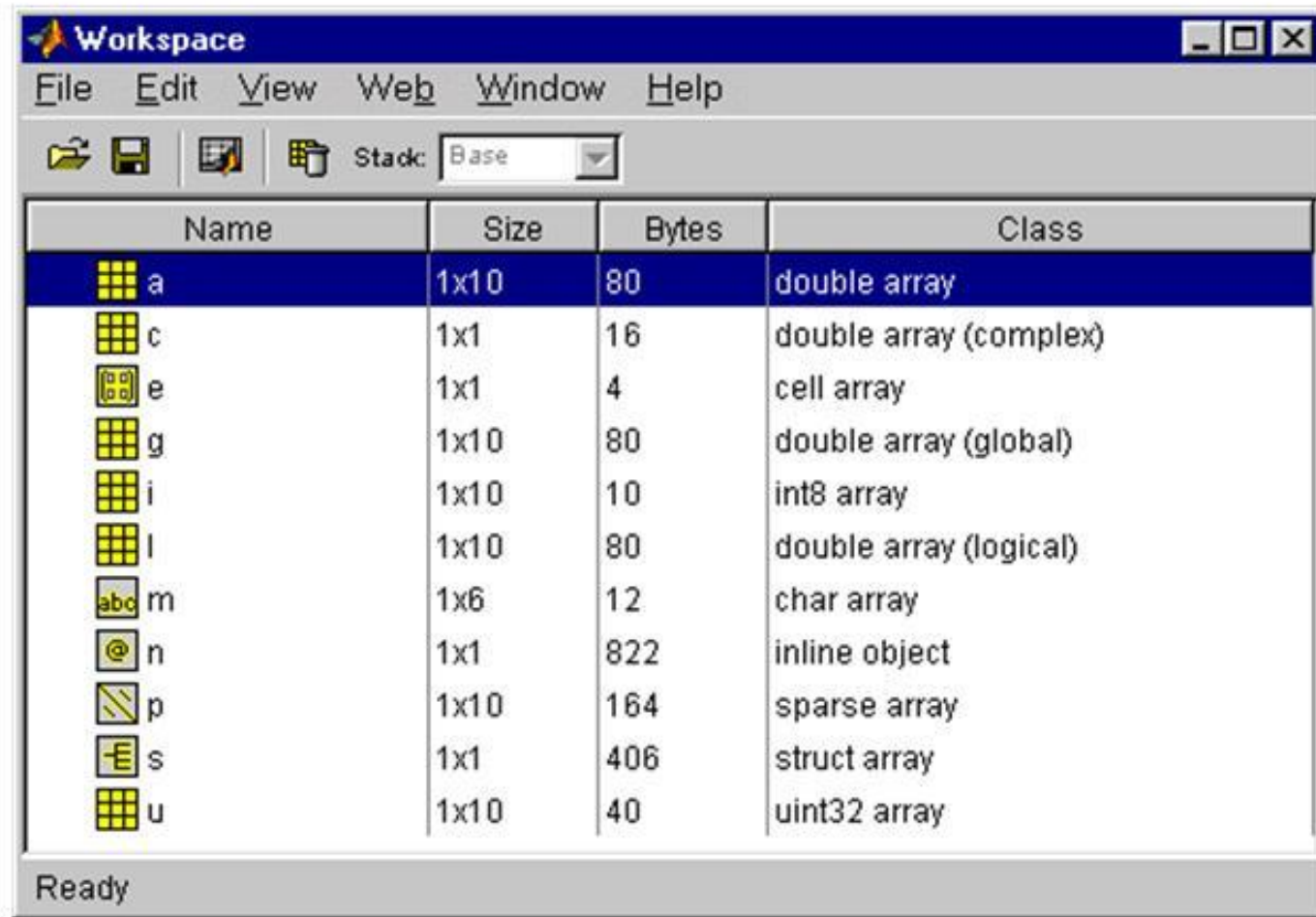
Environnement



Fenêtre
graphique

Environnement:

Navigateur workspace



The screenshot shows the MATLAB Workspace window. The title bar is 'Workspace'. The menu bar includes 'File', 'Edit', 'View', 'Web', 'Window', and 'Help'. The toolbar has icons for opening, saving, and running, along with a 'Stack' dropdown menu set to 'Base'. The main area is a table with four columns: 'Name', 'Size', 'Bytes', and 'Class'. The table lists several variables: 'a' (1x10 double array), 'c' (1x1 double array (complex)), 'e' (1x1 cell array), 'g' (1x10 double array (global)), 'i' (1x10 int8 array), 'l' (1x10 double array (logical)), 'm' (1x6 char array), 'n' (1x1 inline object), 'p' (1x10 sparse array), 's' (1x1 struct array), and 'u' (1x10 uint32 array). The status bar at the bottom says 'Ready'.

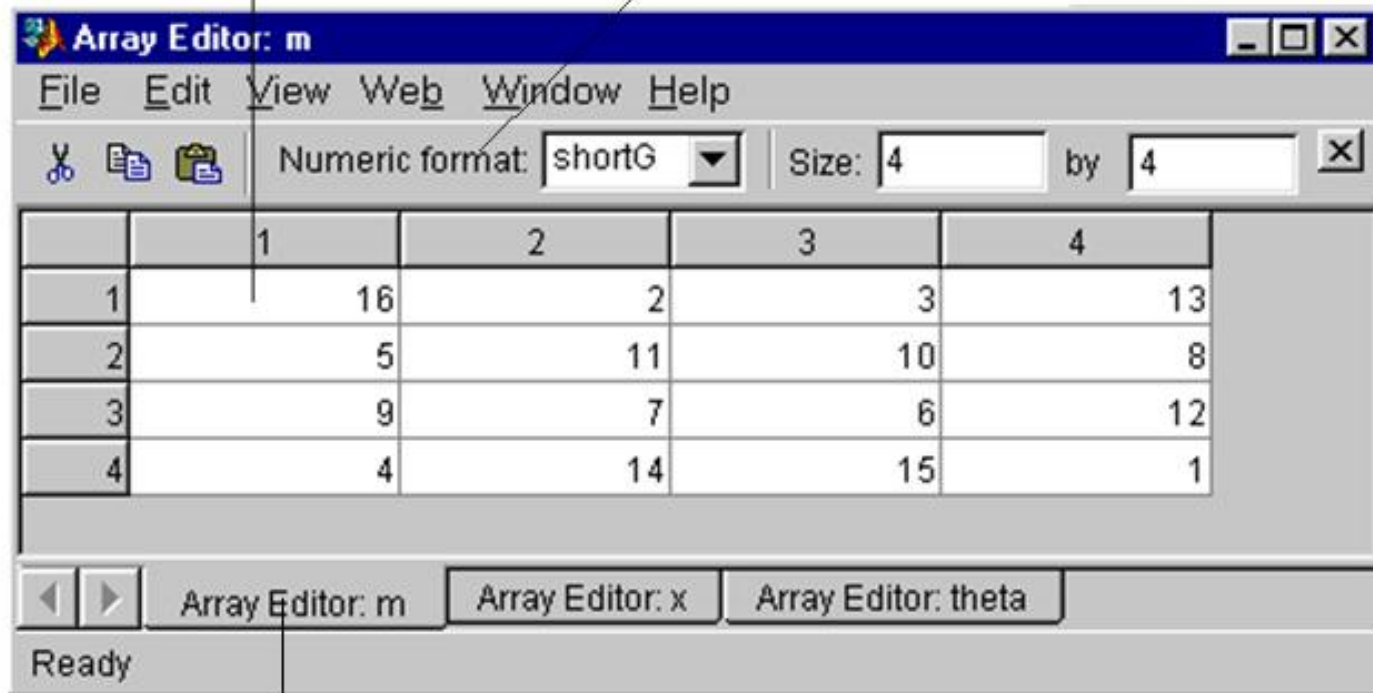
Name	Size	Bytes	Class
a	1x10	80	double array
c	1x1	16	double array (complex)
e	1x1	4	cell array
g	1x10	80	double array (global)
i	1x10	10	int8 array
l	1x10	80	double array (logical)
m	1x6	12	char array
n	1x1	822	inline object
p	1x10	164	sparse array
s	1x1	406	struct array
u	1x10	40	uint32 array

Environnement:

Array Editeur

Changer les valeurs d'un élément

Changer le format d'affichage



Utiliser les onglets pour voir les différentes variables ouvertes

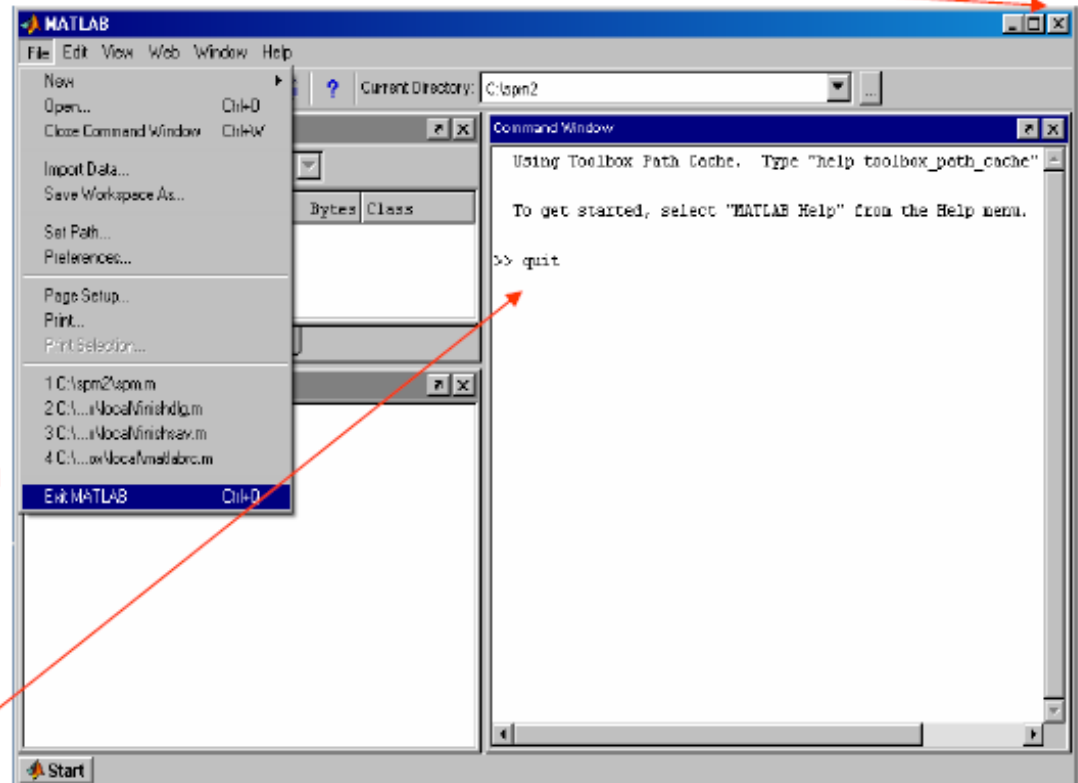
Environnement:

Quitter Matlab

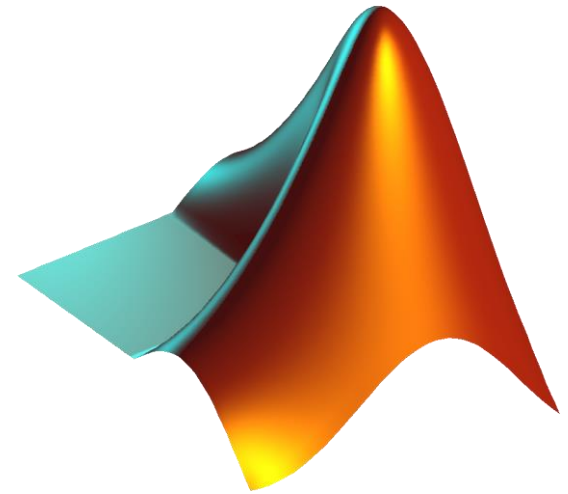
- Cliquer sur la croix

- Menu File : Exit MATLAB

- Command Window : quit



Commandes principales



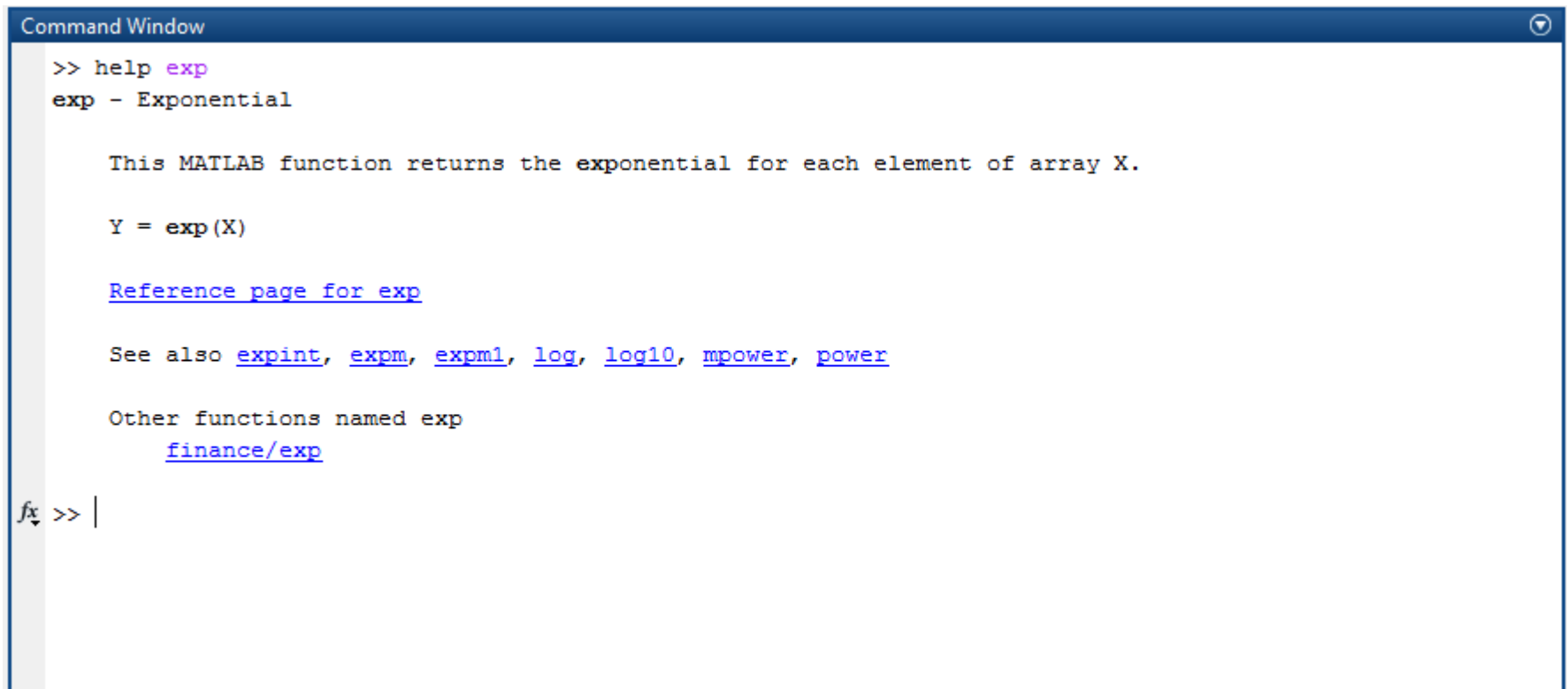
Aide de Matlab

(information la plus importante du cours)

- L'aide de Matlab est accessible via la commande `doc`.
- Pour avoir des informations sur une fonction particulière, on utilise la commande `help nom_de_la_fonction` ou `doc nom_de_la_fonction`.
- On peut utiliser la commande `lookfor` avec un mot clé pour rechercher dans l'aide.
- Documentation en ligne: <https://fr.mathworks.com/help/matlab/>
- Google ou autres moteurs de recherche sont aussi très efficaces: beaucoup d'information et exemples peuvent trouver sur internet

Aide de Matlab

(information la plus importante du cours)



```
Command Window
>> help exp
exp - Exponential

This MATLAB function returns the exponential for each element of array X.

Y = exp(X)

Reference page for exp

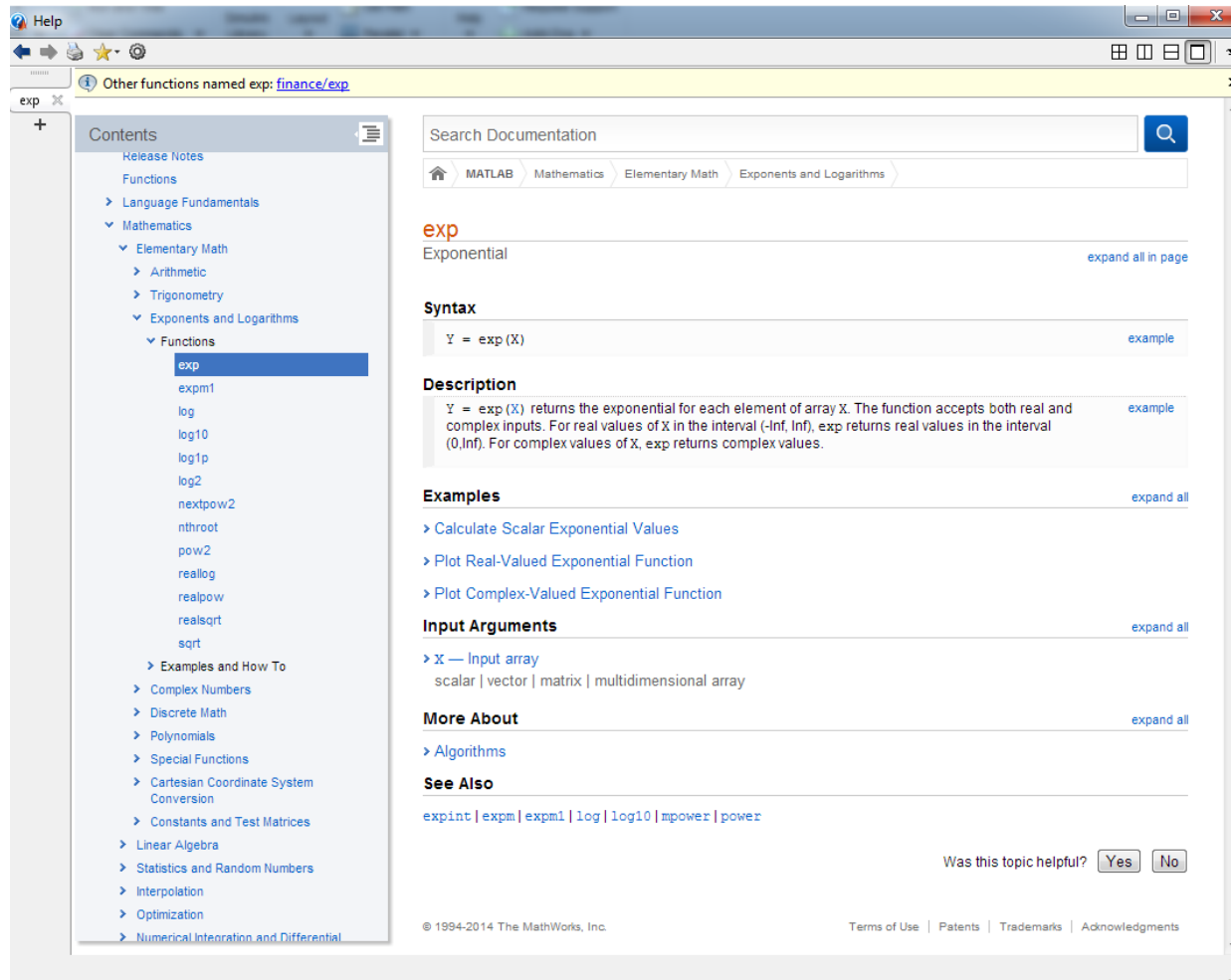
See also expint, expm, expm1, log, log10, mpower, power

Other functions named exp
    finance/exp

fx >> |
```

Aide de Matlab

(information la plus importante du cours)



Autres commandes utiles de Matlab

- **helpwin** aide en ligne dans une fenêtre séparée
- **which** localise fonctions et fichiers
- **what** liste des fichiers matlab dans le répertoire courant
- **who, whos** liste des variables dans le workspace

Type de données et de variables

- Objets : tableaux rectangulaires de nombres
 - Un scalaire est considéré comme un tableau à une ligne et une colonne
 - Un vecteur comme un tableau à une ligne ou une colonne
 - Une matrice comme un tableau à plusieurs lignes et plusieurs colonnes

——→ Simplicité d'écriture des principales opérations arithmétiques
- Type et dimension d'une variable :
 - Détermination automatique à partir de l'expression mathématique ou de la valeur affectée
 - Si la variable existe déjà, le contenu est écrasé par une nouvelle valeur affectée à cette variable
- Type de données :
 - Type réel
 - Type complexe
 - Type chaîne de caractères
 - Type logique

Vecteurs

- Vecteur = liste de ses éléments entre [] :
 - Vecteur ligne : $x = [1\ 2\ 3]$ ou $x = [1,2,3]$
 - Vecteur colonne : $x = [1;2;3]$
- Longueur d'un vecteur :
 - Commande `length(x)`
- Type et dimensions :
 - Commandes `who` / `whos`

```
>> x=[1 2 3]
```

```
x =
```

```
    1    2    3
```

```
>> length(x)
```

```
ans =
```

```
    3
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
ans	1x1	8	double	
x	1x3	24	double	

Vecteurs

- Éléments d'un vecteur :
 - $x(k)$ = $k^{\text{ème}}$ élément du vecteur x
 - $x(k:l)$ = éléments k à l du vecteur x
- Suite de nombres dans un vecteur :
 - $x = a:h:b$ = suite de a à b de h
 - Par défaut, $a:b$ est une suite de a à b de pas $h=1$
- Concaténer des vecteurs :
 - $x_3 = [x_1 \ x_2]$ ou $x_3 = [x_1; x_2]$ concatène les vecteurs x_1 et x_2 dans x_3

```
>> x1=[1 2 3];  
>> x2=[4 5 6];  
>> x3=[x1 x2]  
  
x3 =  
  
     1     2     3     4     5     6  
  
>> x4=[x1;x2]  
  
x4 =  
  
     1     2     3  
     4     5     6
```

Vecteurs spéciaux

- Commande `ones` = vecteur dont tous les éléments valent 1
 - `ones(1,n)` = vecteur ligne de n éléments
 - `ones(m,1)` = vecteur colonne de m éléments
- Commande `zeros` = vecteur dont tous les éléments valent 0
 - `zeros(1,n)` = vecteur ligne de n éléments
 - `zeros(m,1)` = vecteur colonne de m éléments
- Commande `rand` = vecteur dont les éléments sont générés aléatoirement entre 0 et 1
 - `rand(1,n)` = vecteur ligne de n éléments
 - `rand(m,1)` = vecteur colonne de m éléments

Matrices

- Matrice : liste de ses éléments entre []
 - Ligne séparées par un point virgule ou par un retour chariot
 - Éléments d'une même ligne séparée par un blanc ou une virguleEx : $A = [1 \ 2; 3 \ 4]$ ou
 $A = [1,2;3,4]$ ou
 $A = [1,2$
 $3,4]$
- Taille d'une matrice :
 - Commande `size(A)` = taille de la matrice
 - Commande `size(A,1)` = nombre de lignes
 - Commande `size(A,2)` = nombre de colonnes
- Type et dimension :
 - Commande `who` / `whos`

Matrices

- Extraire des lignes ou des colonnes :
 - $A(:,j)$ = extraire tous les éléments de la $j^{\text{ème}}$ colonne
 - $A(i,:)$ = extraire tous les éléments de la $i^{\text{ème}}$ ligne
 - $A(:,\text{end})$ = extraire tous les éléments de la dernière colonne
- Extraire un bloc de la matrice
 - $A(I,J)$ = extraire le bloc défini par les vecteurs d'indice I et J
 - I = vecteur d'indices des lignes à extraire
 - J = vecteur d'indices des colonnes à extraire
- Extraire les diagonales
 - Commande $\text{diag}(A)$: diagonale principale de la matrice A
 - Commande $\text{diag}(A,k)$: sous diagonale de position k par rapport à la diagonale principale

Matrices

- Extraire les parties triangulaires :
 - Commande `tril(A)` = partie triangulaire inférieure
 - Commande `tril(A,k)` = partie triangulaire inférieure à partir de la $k^{\text{ème}}$ diagonale
 - Commande `triu(A)` = partie triangulaire supérieure
 - Commande `triu(A,k)` = partie triangulaire supérieure à partir de la $k^{\text{ème}}$ diagonale

Opérations élémentaires

- Opérations sur :
 - les scalaires
 - les vecteurs
 - les matrices
- Constantes :
 - $\pi = 3.141592\dots$
 - i, j = nombres complexes (racine carrée de -1)
 - eps = précision numérique relative $2.2204\text{e-}016$
 - realmin = plus petit nombre à virgule flottante ($2.2251\text{e-}308$)
 - realmax = plus grand nombre à virgule flottante ($1.7977\text{e+}308$)
 - NaN = « not-a-number » résultat d'une opération non définie (ex : résultat de l'opération $0/0$)

Opérations sur les scalaires

- Opérations usuelles :
 - + addition
 - soustraction
 - * multiplication
 - / division
 - ^ puissance
- Opérations spéciales :
 - Commande `rem(m,n)` : reste de la division entière de deux entiers m/n
 - Commandes `lcm(m,n)` et `gcd(m,n)` : plus petit multiple commun et plus grand diviseur commun à deux entiers m et n
 - ...

Opérations élémentaires

Dans la fenêtre de commande, on peut taper des instructions qui seront interprétées par le logiciel:

- Les opérateurs $+$ $-$ $*$ $/$ $^$ sont inclus.
- Les parenthèses s'utilisent de manière usuelle.

```
>> 2+3  
  
ans =  
  
5  
  
>> 2*3  
  
ans =  
  
6  
  
>> 2/3  
  
ans =  
  
0.6667  
  
>> 2^3  
  
ans =  
  
8  
  
fx >> |
```

La racine carrée s'utilise de la manière suivante:

```
Command Window  
  
>> sqrt(2)  
  
ans =  
  
1.4142  
  
>> 2^0.5  
  
ans =  
  
1.4142  
  
fx >> |
```

Opérations élémentaires

Définir et utilisation de variables:

```
Command Window
>> x=3
x =
    3
>> x^2
ans =
    9
fx >> |
```

Le « ; » à la fin d'une commande permet de ne pas afficher le résultat et sert aussi à terminer une opération

```
Command Window
>> x=3;
>> x^2
ans =
    9
>> a=1; a+1
ans =
    2
fx >> |
```

Fonctions mathématiques

- Fonctions mathématiques :
 - Logarithme népérien: *log(x)*
 - Logarithme en base 10 : *log10(x)*
 - Exponentielle : *exp(x)*
 - Racine carrée : *sqrt(x)*
 - Valeur absolue : *abs(x)*
 - Cosinus : *cos(x)*
 - Sinus : *sin(x)*
 - Tangente : *tan(x)*
 - ...
- Fonctions d'arrondis :
 - Entier le plus proche : *round(x)*
 - Arrondi par défaut : *floor(x)*
 - Arrondi par excès : *ceil(x)*
 - ...

Types

➤ Type numérique : réel et complexe

Exemple : calcul de e , $e^{i\pi/4}$, i^2

`exp(1)` `exp(i*pi/4)` `j*j`

➤ Type booléen : 0 pour faux et 1 pour vrai

Exemple : deux expressions booléennes

`3 > 8` `3 < 8`

➤ Type caractère

```
Command Window
>> exp(1)

ans =

    2.7183

>> exp(i*pi/4)

ans =

    0.7071 + 0.7071i

>> j*j

ans =

    -1
```

```
Command Window
>> 3>8

ans =

     0

>> 3<8

ans =

     1
```

Nombres complexes

```
Command Window
>> c1=1-2*i

c1 =

    1.0000 - 2.0000i

>> c2=3*(2-sqrt(-1)*3)

c2 =

    6.0000 - 9.0000i

>> angle(c1)

ans =

   -1.1071

>> real(c2)

ans =

     6

>> imag(c2)

ans =

    -9

>> abs(c2)

ans =

   10.8167
```

`real(c1)` `imag(c1)` `abs(c1)` `angle(c1)`
autres fonctions: « `conj` » « `isreal` »

Génération
automatique
de nombre
complexe

```
Command Window
>> (-2.5)^0.5

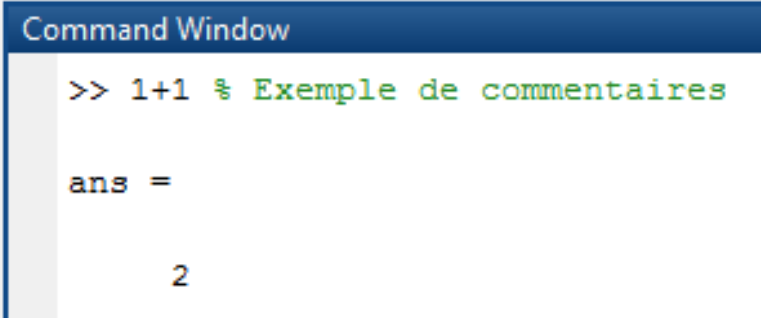
ans =

    0.0000 + 1.5811i

fx >> |
```

Variables : noms

- nom : (Identificateur) Les noms sont de plus 31 caractères (« **underscore** » peut être utilisé)
- Différence entre les **minuscules** et les **majuscules**
- *8var* par exemple est interdit alors que *var8* est licite
- % commentaires :



```
Command Window
>> 1+1 % Exemple de commentaires

ans =

     2
```

Affichages

Calcul en Matlab en double precision

format short: (default) Scaled fixed point format with 5 digits.

format long: Scaled fixed point format with 15 digits.

format short e : format court à 5 chiffres avec notation en virgule flottante.

format long e : format long à 15 chiffres avec notation en virgule flottante.

```
>> format short  
>> pi
```

```
ans =  
  
    3.1416
```

```
>> format short e  
>> pi
```

```
ans =  
  
    3.1416e+00
```

```
>> format long  
>> pi
```

```
ans =  
  
    3.141592653589793
```

```
>> format long e  
>> pi
```

```
ans =  
  
    3.141592653589793e+00
```

Formats de données et entrée/sortie

disp

Output sans formatage

>> disp(pi)
3.1416

et

Command Window

```
>> disp(pi)
3.141592653589793

>> fprintf('%15.11f',pi)
fx 3.14159265359>> |
```

fprintf

Output avec formatage

>> fprintf('%15.11f',pi)
3.14159265359

Commandes principales

Création de séquences

$n = \text{Début} : \text{Fin}$

$p = \text{Début} : \text{incrément} : \text{Fin}$

```
Command Window
>> n=-5:5

n =

    -5    -4    -3    -2    -1     0     1     2     3     4     5

>> p=2:-0.1:1.5

p =

Columns 1 through 3

    2.000000000000000    1.900000000000000    1.800000000000000

Columns 4 through 6

    1.700000000000000    1.600000000000000    1.500000000000000
```

Commandes principales

Pour définir un vecteur, il faut mettre les éléments entre crochets : []
Séparés soit par espace ou virgule (vecteur ligne), soit par ; (vec. colonne)

```
>> A=[0 1;2 3]

A =

     0     1
     2     3

>> B=[4 5;6 7]

B =

     4     5
     6     7
```

```
>> A*B

ans =

     6     7
    26    31

>> A.*B

ans =

     0     5
    12    21
```

« * » Produit matriciel

« . » Préfixe signifiant opération élément par élément

Commandes principales

% Commentaire (dans un programme M-File)... sera imprimé par la commande « help ».

Command Window

```
>> A=[0 1; 2 3]
```

```
A =
```

```
    0    1  
    2    3
```

« ; » Termine une rangée dans une matrice (rappel: sert aussi à éviter l'impression du résultat à l'écran)

```
>> B=[4 5; 6 7]; % lignes de commentaires
```

```
fx  
>> |
```


Variables: tableaux, matrices

Un vecteur est défini:

```
>> v = [1, 2, 4, 5] % vecteur ligne
```

```
>> w = [1; 2; 4; 5] % vecteur colonne
```

$$v = \begin{bmatrix} 1 & 2 & 4 & 5 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}$$

Une matrice (tableau 2D)

```
>> A = [1, 2, 3; 4, -5, 6; 5, -6, 7]
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

Repérage des indices des matrices

Parentheses

```
>> A(2,3)
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

Sous-matrices définies par les vecteurs
par rangée et colonne

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

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

L'ordre est important!!!!

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

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

$$B = \begin{bmatrix} 6 & 5 \\ -5 & 4 \end{bmatrix}$$

Repérage des indices des matrices

Les deux points pour balayer

```
>> A(1,:)
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

On peut limiter

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

```
>> A([1 2],:)
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

Forme générale pour :

```
>> v=1:5
```

```
>> w=1:2:5
```

premier:incrément:dernier

$$v = [1 \ 2 \ 3 \ 4 \ 5]$$

$$w = [1 \ 3 \ 5]$$

Utilisation des indices des matrices

```
» matrice=[1,2,3;4,5,6;7,8,9]
```

matrice =

1	2	3
4	5	6
7	8	9

```
» col2=matrice( : , 2)
```

col2 =

2
5
8

Opérations mathématiques de base

Les opérateurs mathématiques usuels s'appliquent à tous les niveaux:

scalaires

```
Command Window
>> a=2; b=0.56; c=2*a^b

c =

    2.9485

fx >> |
```

vecteurs

```
Command Window
>> vec1=[1 2 3 4]; vec2=vec1*pi

vec2 =

    3.1416    6.2832    9.4248   12.5664

fx >>
```

Opérations mathématiques de base

Les opérateurs mathématiques matriciels usuels sont définis

>> B = A'

1	4	5
2	-5	6
3	6	7

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

>> A*B

14	12	38
12	77	32
38	32	110

>> A+B

2	6	8
6	-10	12
8	12	14

Opérations mathématiques particulières aux tableaux Matlab

>> B.*A

>> A./B

>> A.\B

>> A.^b

2 possibilités

- Opérations élément par élément de 2 tableaux de dimensions identiques
- Opérations entre un tableau et un scalaire. Dans ce cas le « . » est inutile

Opérations mathématiques particulières aux tableaux Matlab

Exemple:

$$B = \begin{bmatrix} 2 & 5 \\ 10 & 20 \end{bmatrix}$$

$$A = \begin{bmatrix} 3 & 10 \\ 7 & 4 \end{bmatrix}$$

Command Window

```
>> B=[2 5; 10 20]; A=[3 10; 7 4]; C=A*B
```

C =

```
106    215  
  54    115
```

Opération matricielle
de multiplication

```
>> D=A.*B
```

D =

```
  6    50  
 70    80
```

Opération sur les éléments du
tableau, 1 à 1

Opérateurs rationnels

MATLAB supporte six opérateurs de relations logiques.

Less Than	<
Less Than or Equal	<=
Greater Than	>
Greater Than or Equal	>=
Equal To	==
Not Equal To	~=

Opérateurs logiques

MATLAB supporte trois opérateurs logiques.

not	~
and	&
or	

Noms réservés en Matlab

Interdit de les utiliser
comme noms de
variables

```
for  
end  
if  
while  
function  
return  
elseif  
case  
otherwise
```

```
switch  
continue  
else  
try  
catch  
global  
persistent  
break
```

Pour connaître la liste exhaustive, taper : `iskeyword`

Structures de branchement : if

If expression
 % exécute ces commandes
end

If expression1
 % exécute ces commandes
elseif expression2
 % exécute ces commandes
else
 % exécute ces commandes
end

Structures de branchement : if

Notez que le décalage vers la droite est une convention de style qui facilite énormément la compréhension des programmes. L'éditeur MATLAB reconnaît de plus les mots-clef et les colore.

```
% Calculate the function f(x,y) based upon
% the signs of x and y
if x >=0 && y >=0
    fun = x + y;
elseif x >= 0 && y<0
    fun = x + y^2;
elseif x <0 && y >=0
    fun = x^2 +y;
else
    fun = x^2 + y^2;
end
```

Structures de branchement : if exemples

```
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```

```
if ((attendance >= 0.90) & (grade_average >= 60))
    pass = 1;
end;
```

Structures de branchement : Switch

Un switch– case--otherwise dans MATLAB

```
switch (expression)
    case expression1,                % est vraie
        ↓
        % exécute ces commandes
    case expression2,                % est vraie
        ↓
        % exécute ces commandes
    otherwise,                        % par défaut
        ↓
        % exécute ces commandes
end
```

Structures de branchement : Switch

```
1 - sp=input('Quel est votre spécialité: \n')
2 - switch sp
3 -     case 'génie chimique',
4 -         disp(' spécialité fondée en 1888')
5 -     case 'génie mécanique',
6 -         disp(' spécialité fondée en 1810')
7 -     case 'génie électrique',
8 -         disp(' spécialité fondée en 1800')
9 -     case 'génie biotechnologique',
10 -         disp(' spécialité fondée en 2003')
11 -     otherwise,
12 -         disp(' inconnu au tableau')
13 - end
14
```


Structures de branchement : Switch

Un switch– case--otherwise dans MATLAB avec plusieurs conditions

```
switch (expression)
    case (expression1, expression2), % au moins une de ces
                                     % expressions est vraie
        % exécute ces commandes
    case expression4,
        % exécute ces commandes
    otherwise, % par défaut
        % exécute ces commandes
end
```

MATLAB Les boucles

Début: incrément: fin



for x = 1: 0.5 : 10

% exécute ces commandes

end

while x <= 10

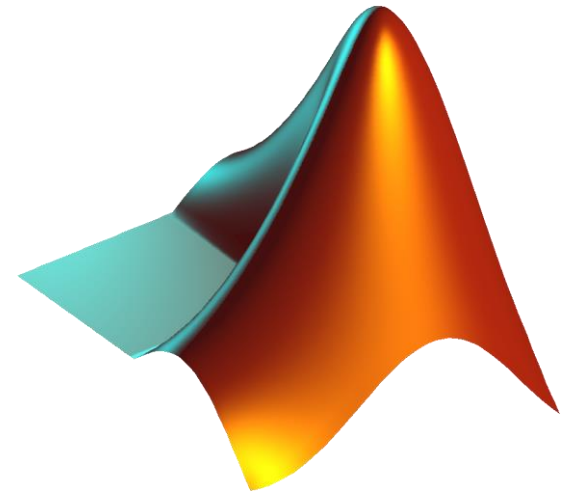
Condition logique



% exécute ces commandes

end

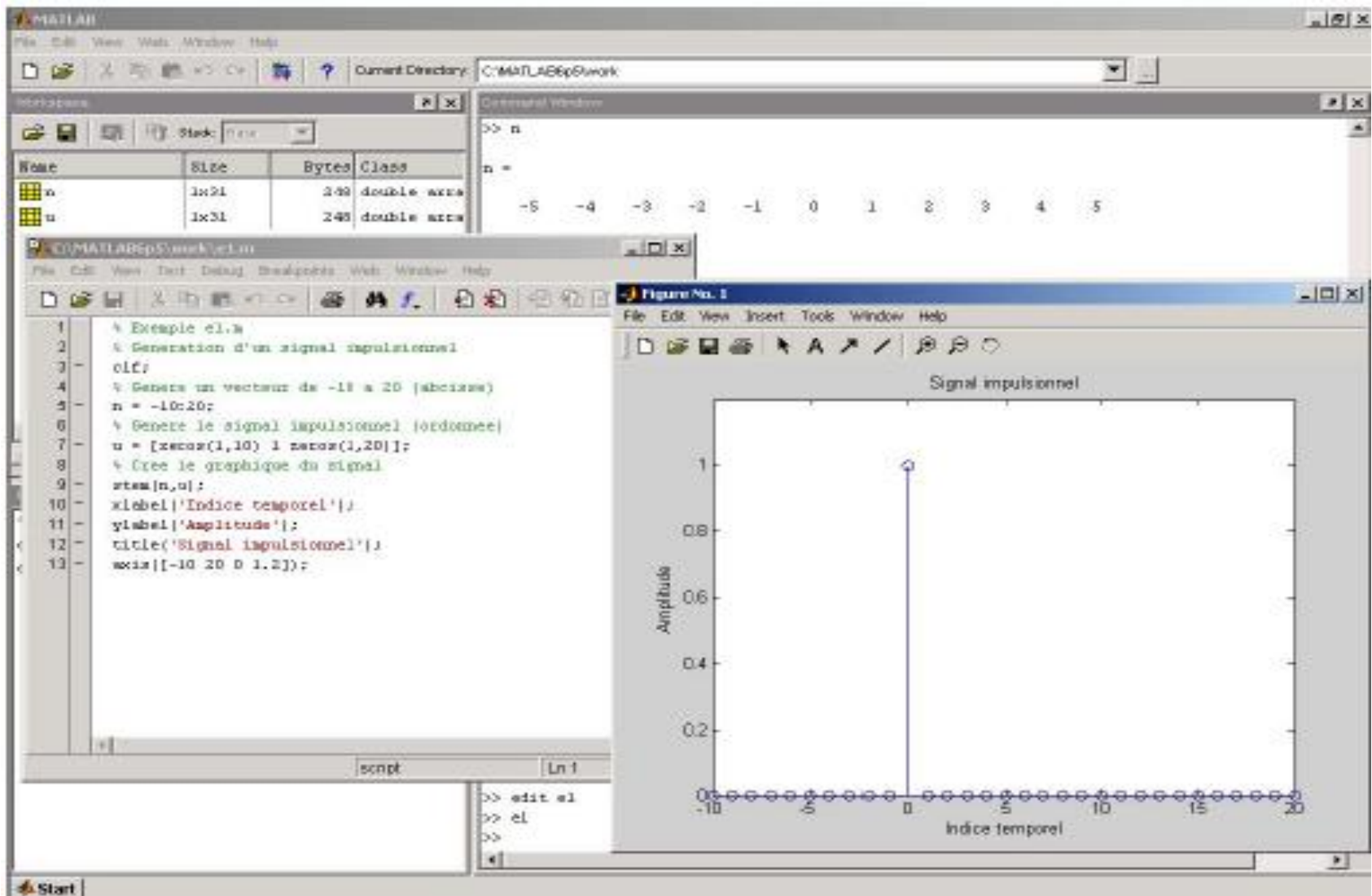
M-files: Scripts, Fonctions



M-Files

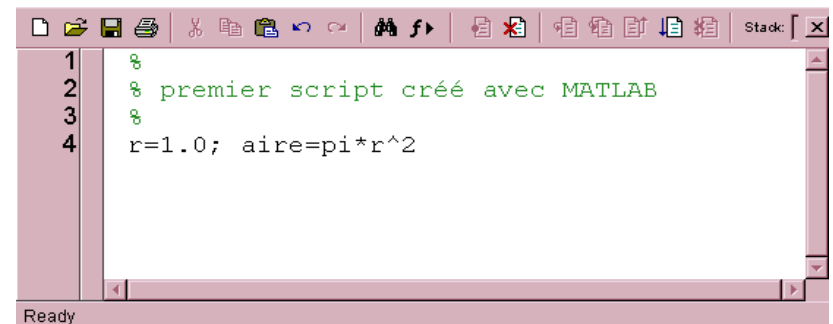
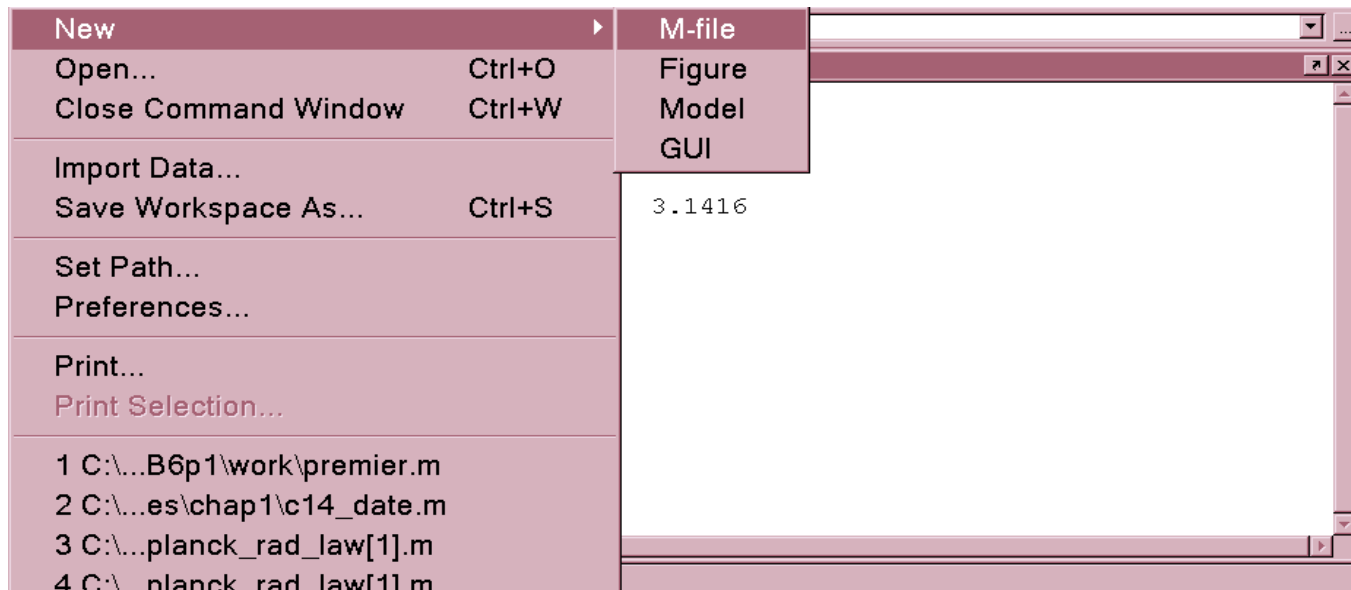
- M-Files \neq X-Files ☺
- Les programmes MATLAB se présentent sous la forme de fichiers possédant l'extension *.m et se trouvant dans le répertoire de travail.
- On les crée à l'aide de la commande `edit`. Sans argument, cette commande crée un nouveau fichier. Avec argument, cette commande nous permet d'éditer le programme désiré.
- On les exécute en invoquant leur nom comme une commande.

M-Files



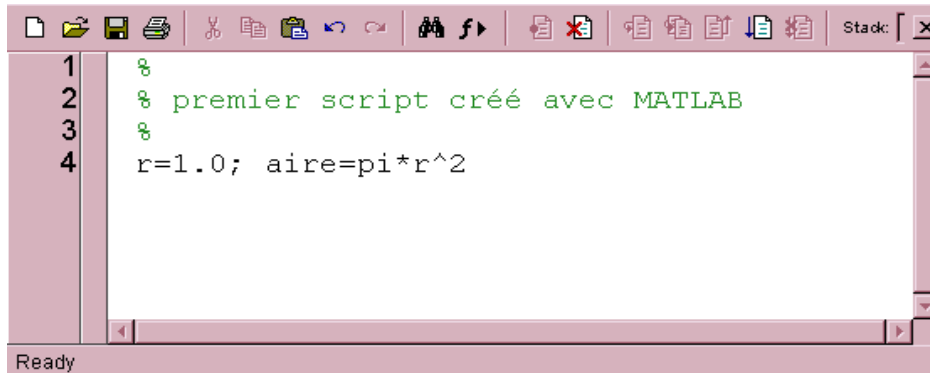
Calcul avec des scripts (m-files)

Matlab permet d'effectuer le travail à partir d'un fichier externe qui porte l'extension.



Calcul avec des scripts (m-files)

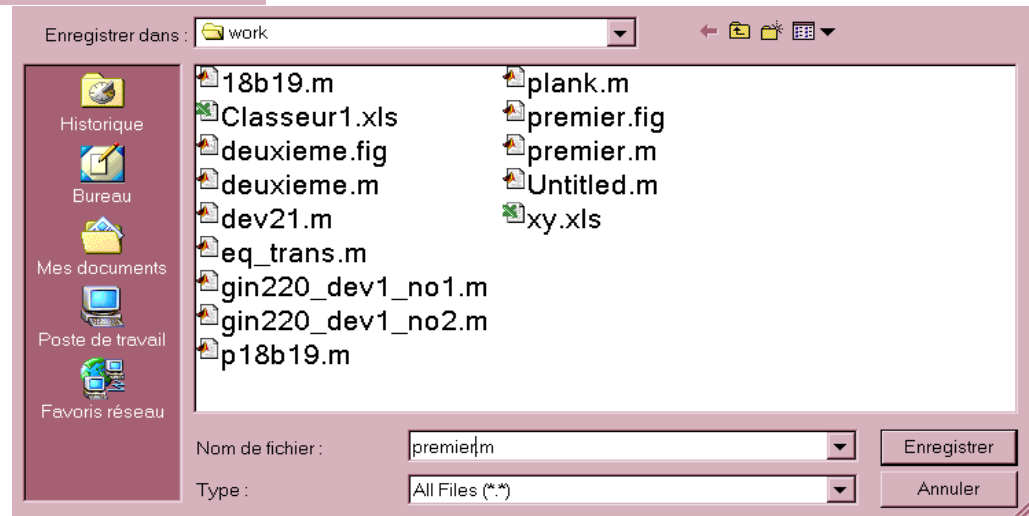
On enregistre ensuite le fichier (par défaut dans le répertoire de travail de MATLAB)



The image shows the MATLAB script editor window. The toolbar at the top includes icons for file operations (new, open, save, print, copy, paste, undo, redo), editing (find, replace), and execution (run, stop). The script content is as follows:

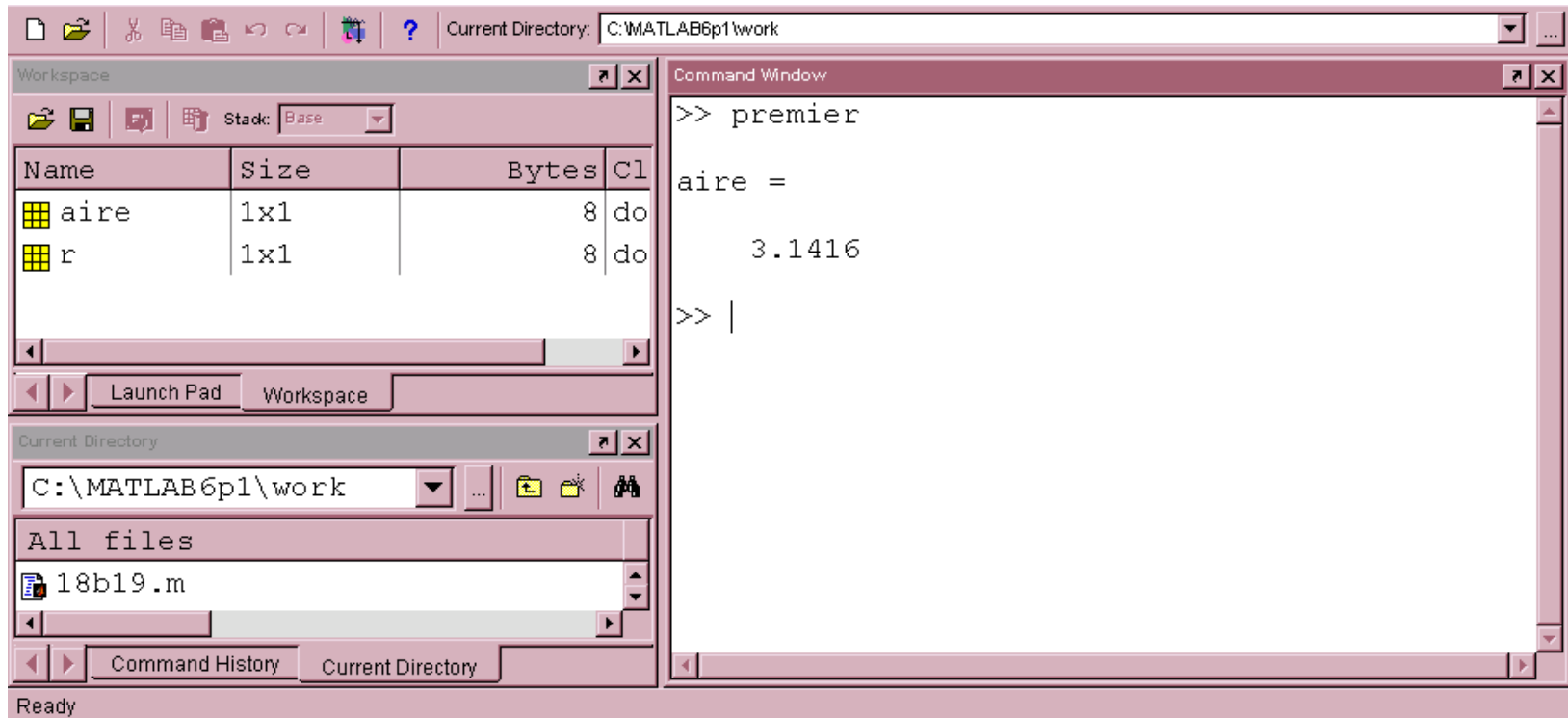
```
1 %  
2 % premier script créé avec MATLAB  
3 %  
4 r=1.0; aire=pi*r^2
```

The status bar at the bottom left indicates 'Ready'.



Calcul avec des scripts (m-files)

On n'a ensuite qu'à appeler le nom (sans le .m) du fichier qui sera ainsi exécuté



Fonctions

Les fonctions sont des M-Files dont la première ligne contient la commande « **fonction** »

```
1 function y=au_carre(x)
2 % Fonction retournant le carré de l'argument x.
3 % Dans le cas d'une matrice, met chacun des éléments au carré un à un.
4 y=x.*x;
```

Output

```
>> help au_carre
Fonction retournant le carré de l'argument x.
Dans le cas d'une matrice, met chacun des éléments au carré un à un.

>> au_carre(10)

ans =

    100

>> au_carre([1 2 3 4])

ans =

     1     4     9    16

fx >> |
```

Sous-programmes (functions)

Une déclaration de fonction

```
function [sortie1, sortie2, sortie3...] = exemple(entrée1, entrée2, ...);  
% fonction exemple  
% Nom Prenom 14/10/2005.
```

Lorsque la fonction est créée avec l'éditeur MATLAB, lorsqu'on sauvera celle-ci, l'éditeur nous proposera le nom exemple.m

Fonction

```
function [sortie1, sortie2, sortie3...] = exemple(entrée1, entrée2, ...);
```

```
% fonction exemple
```

```
% Nom prenom 14/10/2005.
```

```
A=1;
```

```
B=2;
```

```
C=3;
```

```
sortie1=A;
```

```
sortie2=B;
```

```
sortie3=C;
```

Les variables qui se situent à l'intérieur du « corps » de la fonction sont locales, elle ne seront pas accessibles à l'extérieur de la fonction. Ceci assure une certaine indépendance.

Une fonction est différente d'un script car l'intérieur de la fonction est « privé », il contient des variables qui ne sont pas dans le « workspace » général. Elle fait du travail sur les variables d'entrée et nous retourne les variables de sortie.

```
function [mensuel,total] = pret(montant,taux,annees);
```

```
% fonction calculant le taux mensuel d'un pret
```

```
% Nom prenom 2005.
```

```
% Utilisation: pret(montant,taux,annees)
```

```
%         retourne le montant de la mensualité
```

```
%         [m t] = pret(montant,taux,annees)
```

```
%         retourne la mensualité dans m et le montant total dans t
```

La fonction peut avoir
**plusieurs ou aucune
variable de sortie**

```
%         m: montant du prêt en dollars
```

```
%         taux:   taux d'intérêt en pourcent
```

```
%         années: durée de l'emprunt
```

```
%         variables de sortie:
```

```
%         mensuel: mensualités a payer
```

```
%         total:  montant total a rembourser sur la durée du prêt
```

Arguments d'entrée

Ce nom sera celui du fichier (pret.m). **Ce nom sera automatiquement donné par l'éditeur de MATLAB**

Les premières lignes de commentaires constituent le « help » de la fonction. Il est très conseillé de prendre soin de donner à l'utilisateur les conseils sur l'utilisation de la fonction et tous les détails nécessaires a son bon fonctionnement

function [mensuel,total] = pret(montant,taux,annees);

% fonction calculant le taux mensuel d'un pret

% Nom prenom 14/10/2005.

% Utilisation: pret(montant,taux,années)

% retournera le montant de la mensualité

% [m t] = pret(montant,taux,années)

% retournera la mensualité dans m et le montant total dans t

%

% variables d'entree:

% montant: montant du pret en dollars

% taux: taux d'intérêt en pourcent

% annees: duree de l'emprunt

%

% variables de sortie:

% mensuel: mensualités a payer

% total: montant total a rembourser sur la durée du prêt

La première ligne de commentaires est particulièrement importante car c'est dans celle-ci que le lookfor ira chercher le mot clef

```
function [mensuel,total] = pret(montant,taux,annees);
```

```
% fonction calculant le taux mensuel d'un pret
```

```
% Nom prenom 14/10/2005.
```

```
% Utilisation: pret(montant,taux,années)
```

```
%          retournera le montant de la mensualité
```

```
%          [m t] = pret(montant,taux,années)
```

```
%          retournera la mensualité dans m et le montant total dans t
```

```
%
```

```
% variables d'entree:
```

```
%          montant: montant du pret en dollars
```

```
%          taux:   taux d'intérêt en pourcent
```

```
%          annees: duree de l'emprunt
```

```
%
```

```
% variables de sortie:
```

```
%          mensuel: mensualités a payer
```

```
%          total:  montant total a rembourser sur la durée du prêt
```

Fonction

function [mensuel,total] = pret(montant,taux,annees);

% fonction calc

% Nom prenc

% Utilisation:

%

%

%

%

%

% variables

% m

%

%

%

% variables de sortie:

%

%

%

Corps de la fonction. L'utilisateur en appelant la fonction a donné les variables d'entrée, la fonction doit maintenant calculer les variables de sortie et les retourner à l'utilisateur. La fonction est le « sous-traitant » de la programmation.

format bank

tauxmensuel=taux*0.01/12;

a=1+tauxmensuel;

b=(a^(annees*12)-1)/tauxmensuel;

mensuel=montant*a^(annees*12)/(a*b);

total=mensuel*annees*12;

Dans ce cas, mensuel et total doivent être calculées. Aucune instruction spéciale de fin de fonction n'est nécessaire

Fonction

```
+2  funxy.m  switch_ex_cours.m  try_catch_ex1.m  try_catch_ex2.m  try_catch_ex0.m  pret.m
1  function [mensuel,total] = pret(montant,taux,annees);
2  % fonction calculant le taux mensuel d'un pret
3  % Nom prenom 2005.
4  % Utilisation:   pret(montant,taux,annees)
5  %               retournera le montant de la mensualité
6  %               [m t] = pret(montant,taux,annees)
7  %               retournera la mensualité dans m et le montant total dans t
8  %
9  %   variables d'entree:
10 %               montant: montant du prêt en dollars
11 %               taux:    taux d'intérêt en pourcent
12 %               années:  durée de l'emprunt
13 %
14 %   variables de sortie:
15 %               mensuel: mensualités a payer
16 %               total:   montant total a rembourser sur la durée du prêt
17 - format bank
18 - tauxmensuel=taux*0.01/12;
19 - a=1+tauxmensuel;
20 - b=(a^(annees*12)-1)/tauxmensuel;
21 - mensuel=montant*a^(annees*12)/(a*b);
22 - total=mensuel*annees*12;
23
```


Fonction

Command Window

```
>> help pret
fonction calculant le taux mensuel d'un pret
Nom prenom 2005.
Utilisation:  pret(montant,taux,annees)
               retournera le montant de la mensualité
               [m t] = pret(montant,taux,annees)
               retournera la mensualité dans m et le montant total dans t

variables d'entree:
montant: montant du prêt en dollars
taux:    taux d'intérêt en pourcent
années:  durée de l'emprunt

variables de sortie:
mensuel: mensualités a payer
total:   montant total a rembourser sur la durée du prêt
```

Fonction

Command Window

```
>> lookfor mensuel
pret                - fonction calculant le taux mensuel d'un pret
>>
>> [montant_mensuel total_a_rembourser]=pret(150000,5,20)

montant_mensuel =

    985.83

|
total_a_rembourser =

    236598.24

fx >> |
```

Fonction

Types de ligne de définition de fonctions dans MATLAB

`function [a,b,c]=triplet(x,y,z)`

Appel de fonction avec trois paramètres d'entrée et 3 de sortie.

`function [A] = calcule_aire(a,b)`

Appel de fonction avec un paramètre de sortie et deux d'entrée.

`function A = calcule_aire(a,b)`

Avec un seul paramètre d'entrée, pas nécessaire de mettre les [].

`function trace_trajectoire(v0,h0,g)`

Sans paramètre de sortie, ce qui est aussi légal. (par exemple, la fonction plot de MATLAB est de ce type)

Fonctions graphiques

- Graphiques en 2D et en 3D
 - 2D : commandes *plot*, *fplot*, *loglog*, ...
 - 3D : commandes *mesh*, *surf*, *image*, *contourslice*, ...

Gestion des fenêtres graphiques

- Une commande graphique ouvre une fenêtre dans laquelle sera affichée le graphique voulu.
- Si une fenêtre est déjà ouverte, une nouvelle instruction graphique écrasera le graphique précédent.
- Si plusieurs fenêtres sont ouvertes, Matlab utilise la dernière fenêtre active.
- Chaque fenêtre graphique ouverte se voit affectée un numéro (visible dans le bandeau de la fenêtre).

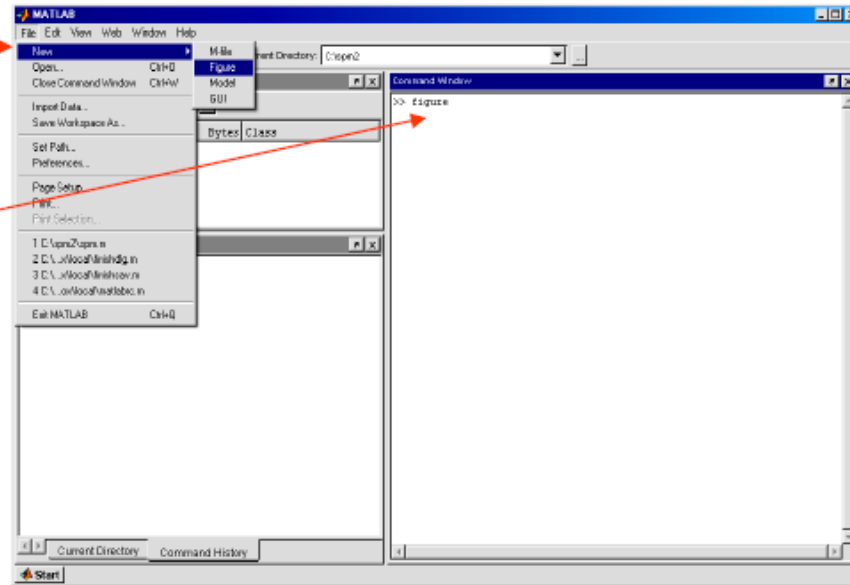
Ouverture/fermeture d'une fenêtre graphique

- Ouverture d'une fenêtre :
 - Ouvrir une nouvelle fenêtre :
 - Commande *figure*
 - Commande *figure(n)* où *n* est le numéro de la fenêtre
- Activation :
 - Cliquer sur la fenêtre graphique
 - Commandes *figure* et *figure(n)* : activent une fenêtre existante
- Fermeture d'une fenêtre :
 - Fermer la fenêtre active :
 - Commande *close*
 - Fermer une fenêtre ouverte :
 - Commande *close(n)* où *n* désigne le numéro de la fenêtre
 - Fermer toutes les fenêtres graphiques :
 - Commande *close all*

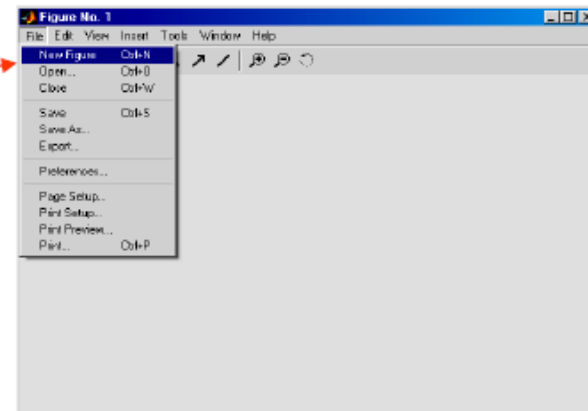
Ouvrir une fenêtre graphique

- Menu File>New>Figure

- Commande *figure*



- Fenêtre graphique :
Menu File>New Figure



Graphiques 2D : commande fplot

- Graphe d'une fonction : commande *fplot*
trace le graphe d'une fonction sur un intervalle donné

Syntaxe : *fplot('nomf', [xmin, xmax, ymin, ymax])*

nomf = le nom d'une fonction matlab incorporée
ou une expression définissant une fonction

xmin, *xmax* déterminent l'intervalle des abscisses pour lequel
est tracé le graphe

ymin, *ymax* déterminent l'intervalle des ordonnées
par défaut : les valeurs minimum et maximum prises par la
fonction sur l'intervalle [*xmin*, *xmax*]

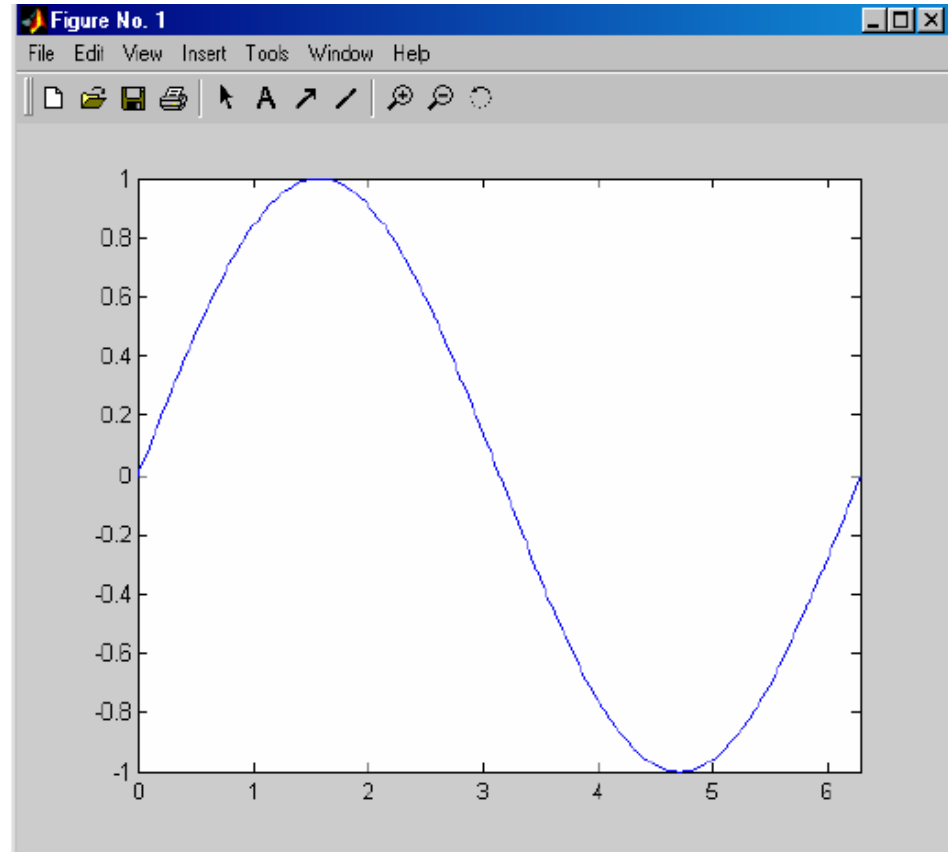
Exemples : commande fplot

- Fonction sinus :

```
int = [0,2*pi];  
fplot('sin',int)
```

ou

```
fplot('sin',[0,2*pi])
```



Exemples : commande fplot

- Fonction $x \cdot \sin(x)$:

```
fplot('x*sin(x)',[-2*pi,2*pi])
```

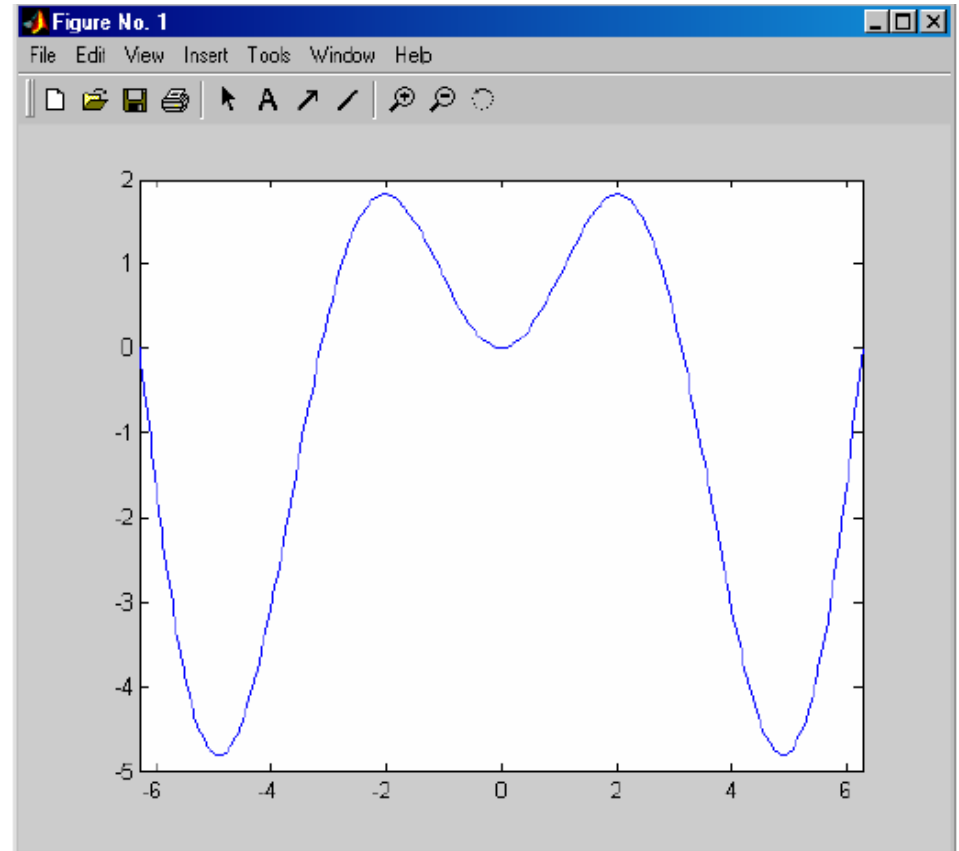
- Définir une fonction :

- Ecrire une fonction h.m :

```
function y=h(x)
```

```
y=x.*sin(x);
```

- `fplot('h',[-2*pi,2*pi])`



Graphiques 2D : commande plot

- Graphe d'une fonction : commande *plot*
 - trace le graphe d'un vecteur y pour un ensemble de valeurs x spécifiées
 - par défaut, relie les points par des segments de droite

Syntaxe : *plot(x,y)*

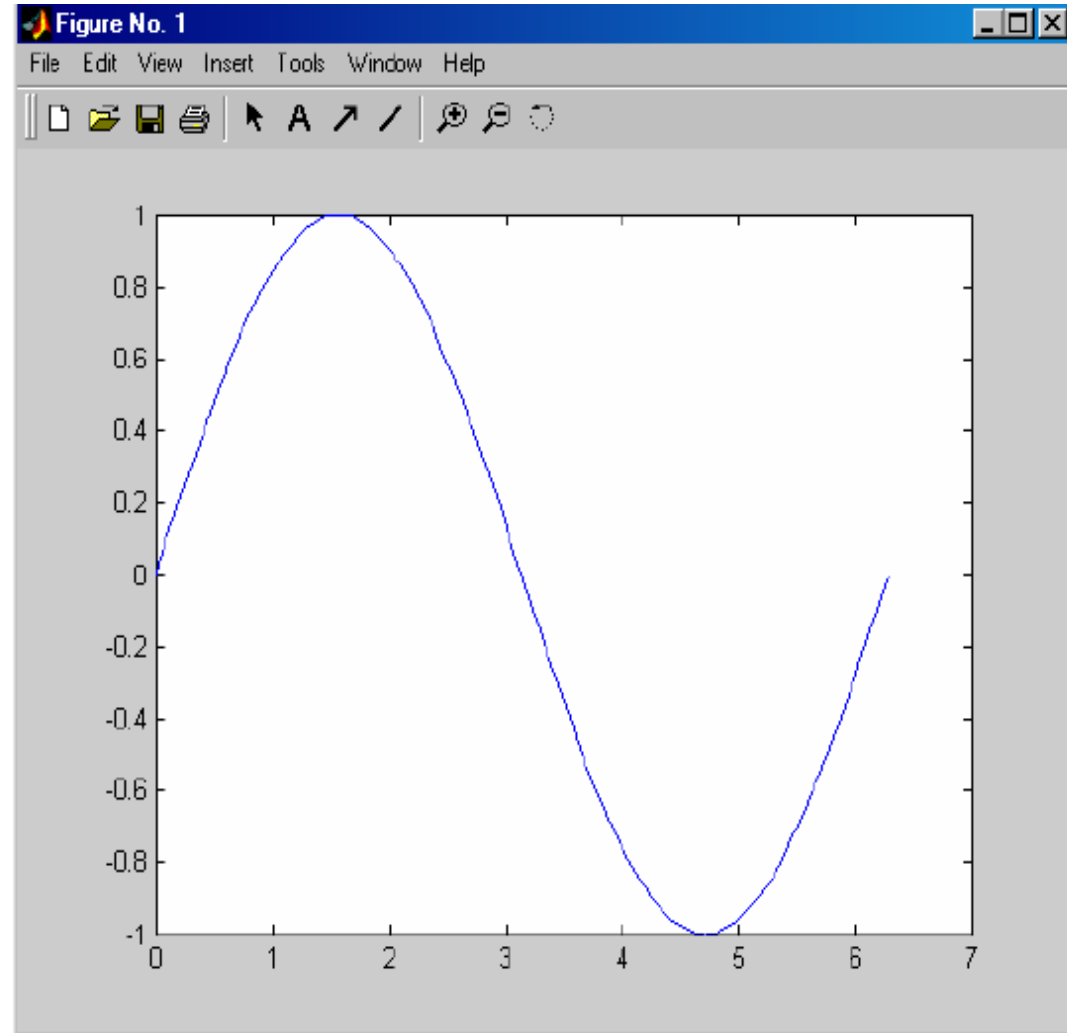
y = vecteur contenant les valeurs en ordonnées

x = vecteur contenant les valeurs en abscisses

x et y sont des vecteurs (ligne ou colonne) de même longueur !

Exemples : commande plot

- Fonction sinus :
Intervalle $[0, 2\pi]$, 201 valeurs
 $x = 0:\pi/100:2*\pi;$
 $y = \sin(x);$
 $\text{plot}(x,y);$
- Sélection automatique de l'étendue des axes



Exemples : commande plot

- Fonction sinus :

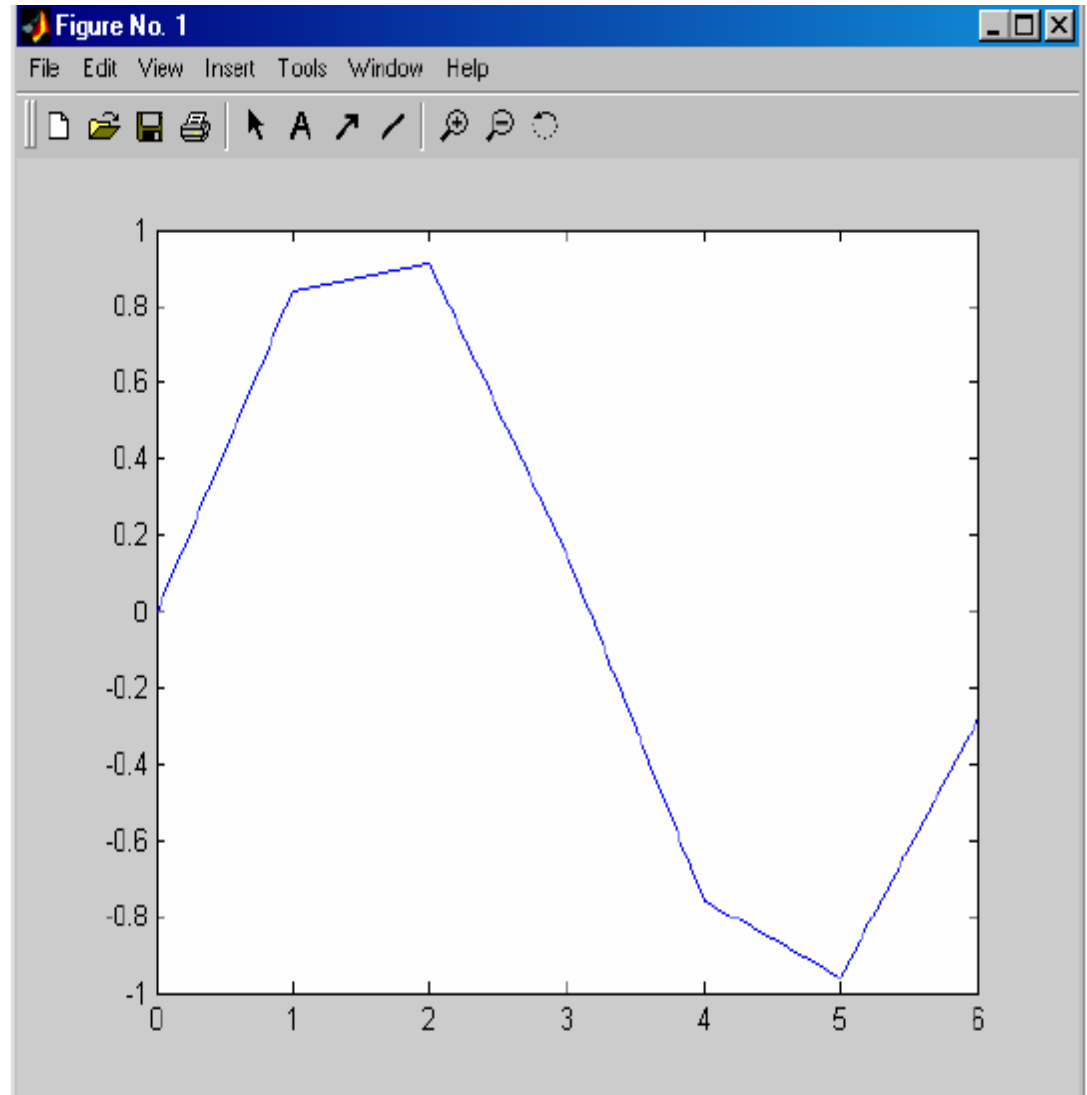
Intervalle $[0, 2\pi]$, 7 valeurs

```
x= 0:1:2*pi;
```

```
y = sin(x);
```

```
plot(x,y);
```

- Points reliés linéairement

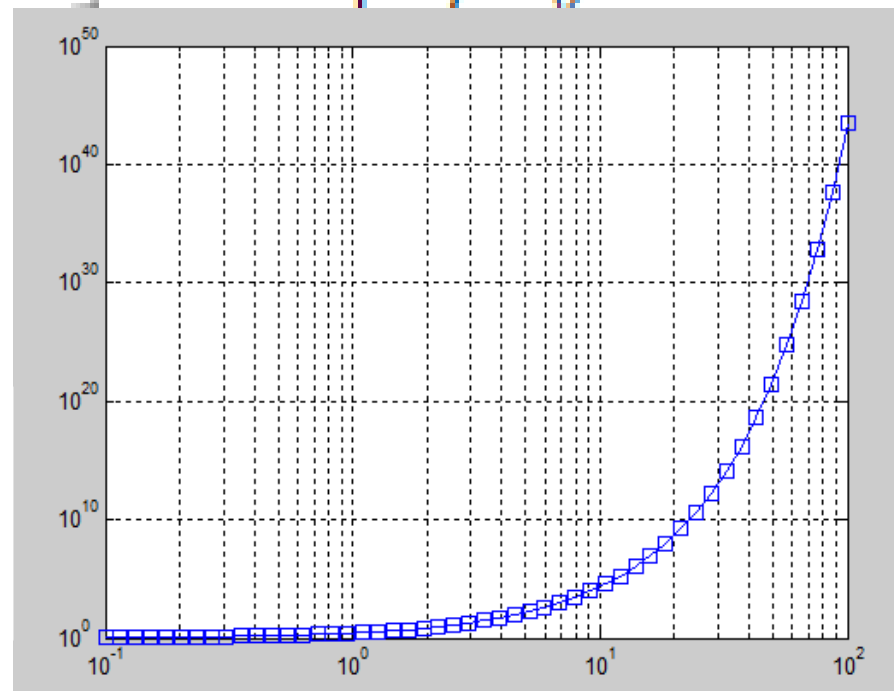


Graphiques logarithmique : commande loglog

- **loglog(X1,Y1,'format_ligne1',,...)**
 - Cette fonction trace un graphique à échelle logarithmique.
 - Les paramètres sont les mêmes que *plot()*.

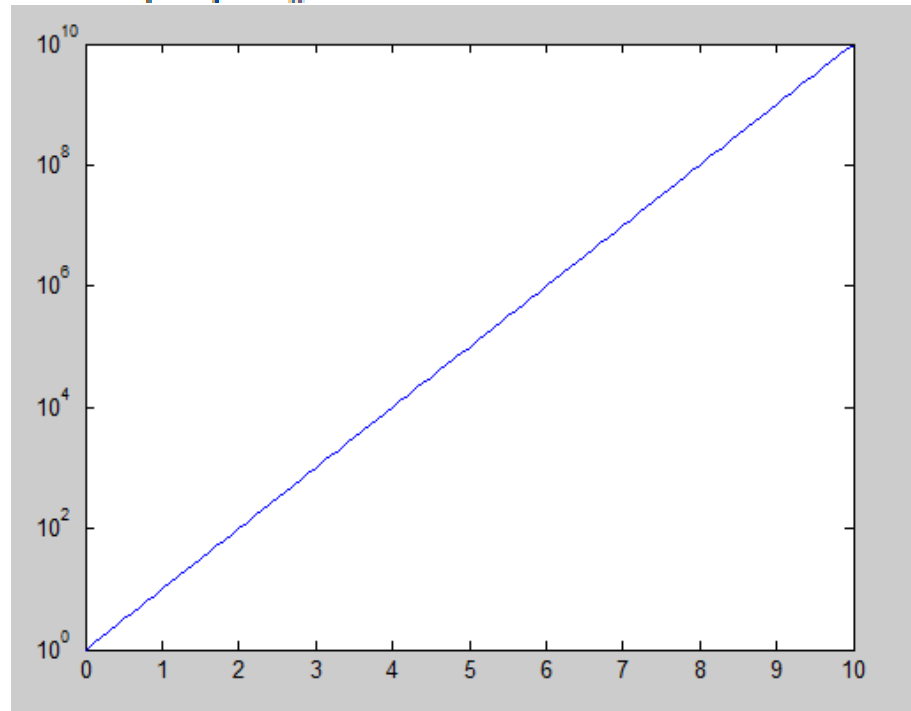
- **Exemple:**

```
x = logspace(-1,2);  
loglog(x,exp(x),'-s');  
grid on;
```



Graphiques semi-logarithmique

- `semilogx(X1,Y1,'format_ligne1',...)`
- `semilogy(X1,Y1,'format_ligne1',...)`
 - Ces fonctions tracent des graphiques à échelle semi-logarithmique.
 - Les paramètres sont les mêmes que `plot()`.
- Exemple:
`x = 0:0.1:10;`
`semilogy(x,10.^x);`



Plusieurs courbes sur un seul graphique

- Plusieurs courbes pour différents vecteurs d'abscisse x :
 - commande *plot* :

plot(x_1, y_1, x_2, y_2)

x_1 = vecteur d'abscisses pour la courbe y_1

x_2 = vecteur d'abscisses pour la courbe y_2

- commandes *hold on* et *hold off* :

hold on

.....
instructions graphiques

.....

hold off



graphes superposés dans la
même fenêtre active

- » utile dans une boucle (cf. programmation)
- » utile pour la commande *fplot*

Plusieurs courbes sur un seul graphique

- Plusieurs courbes pour le même vecteur d'abscisse x :
 - commande *fplot* :

fplot('[nomf1, nomf2]',[xmin, xmax, ymin, ymax])

xmin, *xmax* déterminent l'intervalle des abscisses pour lequel les deux graphes sont tracés

- commande *plot* :

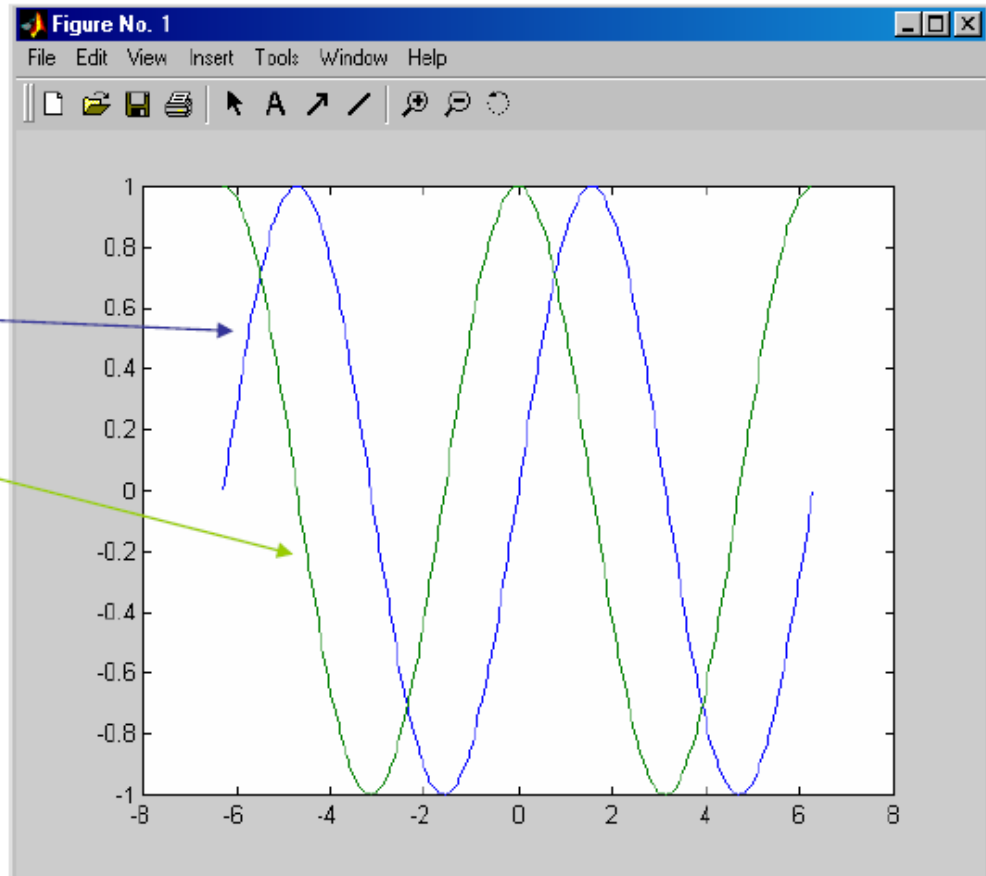
plot(x, y_1, x, y_2)

x = vecteur d'abscisses pour les courbes y_1 et y_2

Plusieurs courbes sur un seul graphique : exemples

- Fonctions sin et cos :
intervalle $[-2\pi, 2\pi]$, 201 valeurs

```
x = -2*pi:pi/100:2*pi;  
y1 = sin(x);  
y2 = cos(x);  
plot(x,y1,x,y2)
```



Plusieurs courbes sur un seul graphique : exemples

- Fonctions sin et cos :
intervalle $[-2\pi, 2\pi]$, 201 valeurs
intervalle $[0, 2\pi]$, 101 valeurs

```
x1 = -2*pi:pi/100:2*pi;  
y1 = cos(x1);  
x2 = 0:pi/100:2*pi;  
y2 = sin(x2);
```

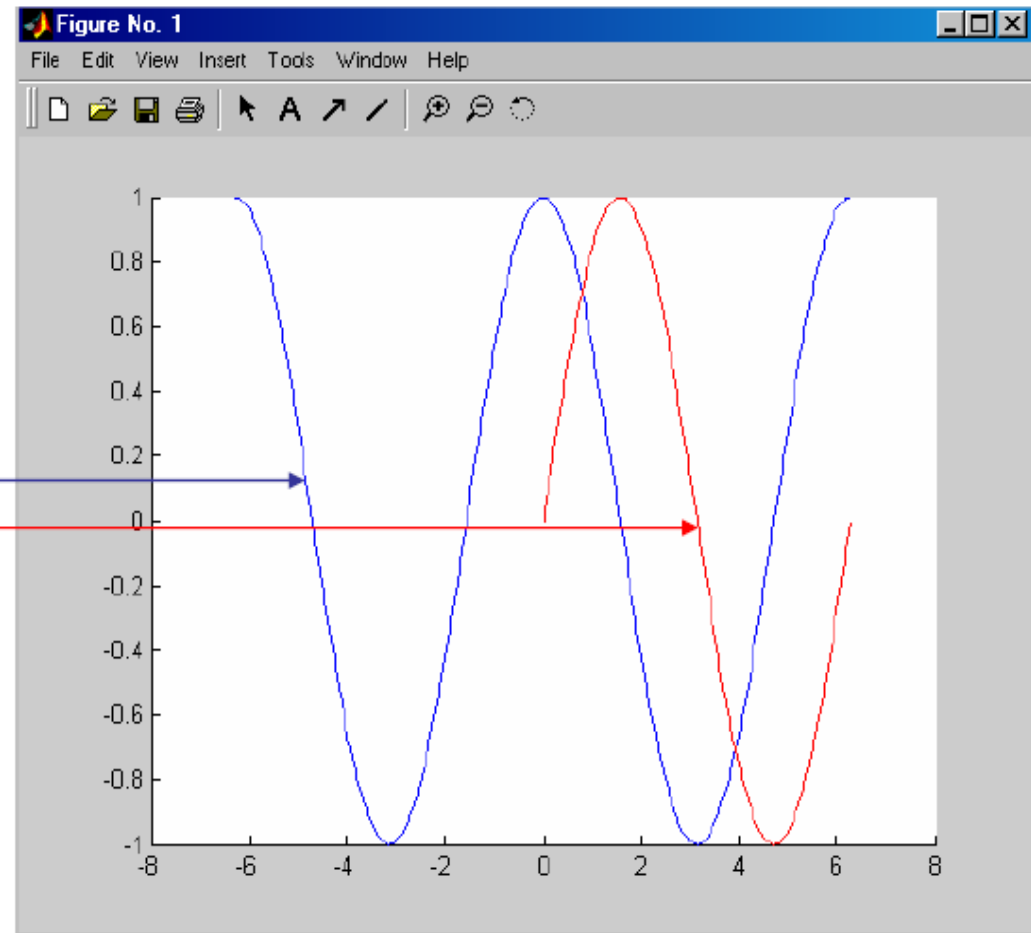
```
hold on;  
plot(x1,y1);  
plot(x2,y2,'r');  
hold off;
```

ou

```
hold on;  
fplot('cos',[-2*pi,2*pi]);  
fplot('sin',[0,2*pi],'r');  
hold off;
```

ou

```
plot(x1,y1,x2,y2,'r');
```



Plusieurs graphiques dans une fenêtre

- Commande `subplot` : décomposer une fenêtre en sous-fenêtres

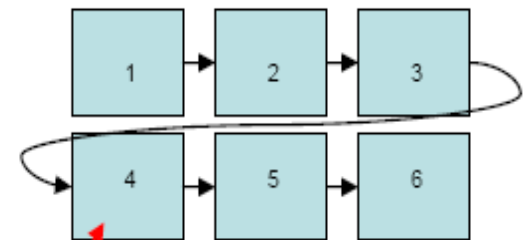
`subplot(m,n,i);`
instruction graphique

m = nombre de sous-fenêtres verticalement

n = nombre de sous-fenêtres horizontalement

i = numéro de la sous-fenêtre dans laquelle le graphique s'affiche

Numérotation de gauche à droite, de haut en bas

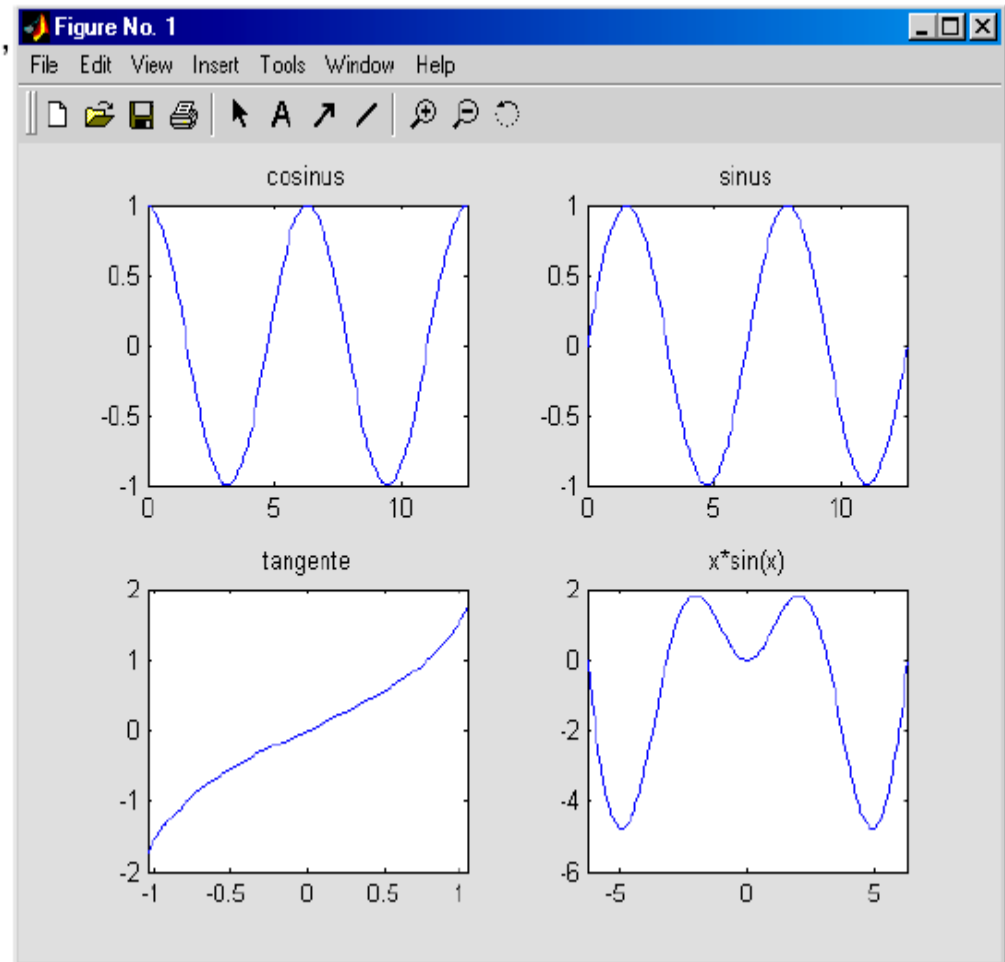


Ex : `subplot(2,3,4);`

Plusieurs graphiques dans une fenêtre: exemple

- Fonctions $\cos(x)$, $\sin(x)$, $\tan(x)$, $x \sin(x)$:

```
figure;  
subplot(2,2,1);  
fplot('cos',[0,4*pi]);  
title('cosinus');  
subplot(2,2,2);  
fplot('sin',[0,4*pi]);  
title('sinus');  
subplot(2,2,3);  
fplot('tan',[-pi/3,pi/3]);  
title('tangente');  
subplot(2,2,4);  
fplot('x*sin(x)',[-2*pi,2*pi]);  
title('x*sin(x)');
```



Améliorer la lisibilité

- Créer des légendes et annotations
- Contrôler les axes
- Options du tracé des courbes

Textes et légendres

- Légendes des axes
 - Commande *xlabel* :
xlabel('légende de l'axe x')
 - Commande *ylabel* :
ylabel('légende de l'axe y')
- Titre du graphique
 - Commande *title* :
title('titre du graphique')

Textes et légendes

- Texte dans la figure :
 - Commande *text* : écrit un texte à une position précise sur la figure

text(posx, posy, 'un texte')

posx et *posy* = coordonnées du point de début de texte

- Commande *gtext* : écrit un texte à une position choisie à l'aide de la souris

gtext('un texte')

Textes et légendres

- Identification des courbes dans un même graphique :
 - Commande *legend* : légende permettant d'identifier les courbes

legend('légende 1','légende 2','légende 3')

Contrôle des axes

commande *axis* = orientation et échelle des plots

- Limiter les axes : par défaut, l'étendue varie du minimum au maximum

axis([xmin xmax ymin ymax])

axis auto = revenir à la sélection par défaut

- Aspect des axes : changement d'échelle

axis square = axes x et y de même longueur

axis equal = pas de même longueur sur les axes x et y

axis normal = revenir à la sélection par défaut

- Autres options : *help axis*

Contrôle des axes

- Visibilité :
 - axis on* = rend les axes visibles (par défaut)
 - axis off* = rend les axes invisibles
- Grille de lignes :
 - grid on* = affiche une grille de lignes
 - grid off* = efface la grille de lignes (par défaut)

Options du tracé des courbes

- Spécifier les couleurs
- Spécifier le style de trait
- Spécifier le symbole à chaque point

`plot(x,y,'color_style_marker','LineWidth',n)`

color_style_marker = chaîne de 3 à 4 caractères définissant la couleur, le style du trait et le symbole

n = épaisseur du trait (par défaut 1)

- Couleur du fond définis par les axes : commande `whitebg('couleur')`
 - Modifie la couleur les propriétés du graphique (couleur des axes, du tracé, ...) pour maintenir un contraste adéquat

Options du tracé des courbes

Couleurs	Style	Symbole
y = jaune	-	point
m = magenta	:	cercle
c = cyan	--	croix
r = rouge	-.	plus
g = vert	none	étoile
b = bleu		carré
w = blanc		losange
k = noir		triangle (bas)
		triangle (haut)
		triangle (gauche)
		triangle (droite)
		pentagone
		hexagone
		aucun

Valeur par défaut : 'b-.' = bleu, trait plein, point

Modifier les plots après création

- Mode édition interactive

- Clic droit sur l'objet à modifier
 - Épaisseur de ligne
 - Style de ligne
 - Couleur
 - Propriétés de l'objet

- Commande `set`

```
h = plot(x,y);  
set(h,'nomprop',valprop)
```

nomprop = désigne la propriété du graphique à modifier

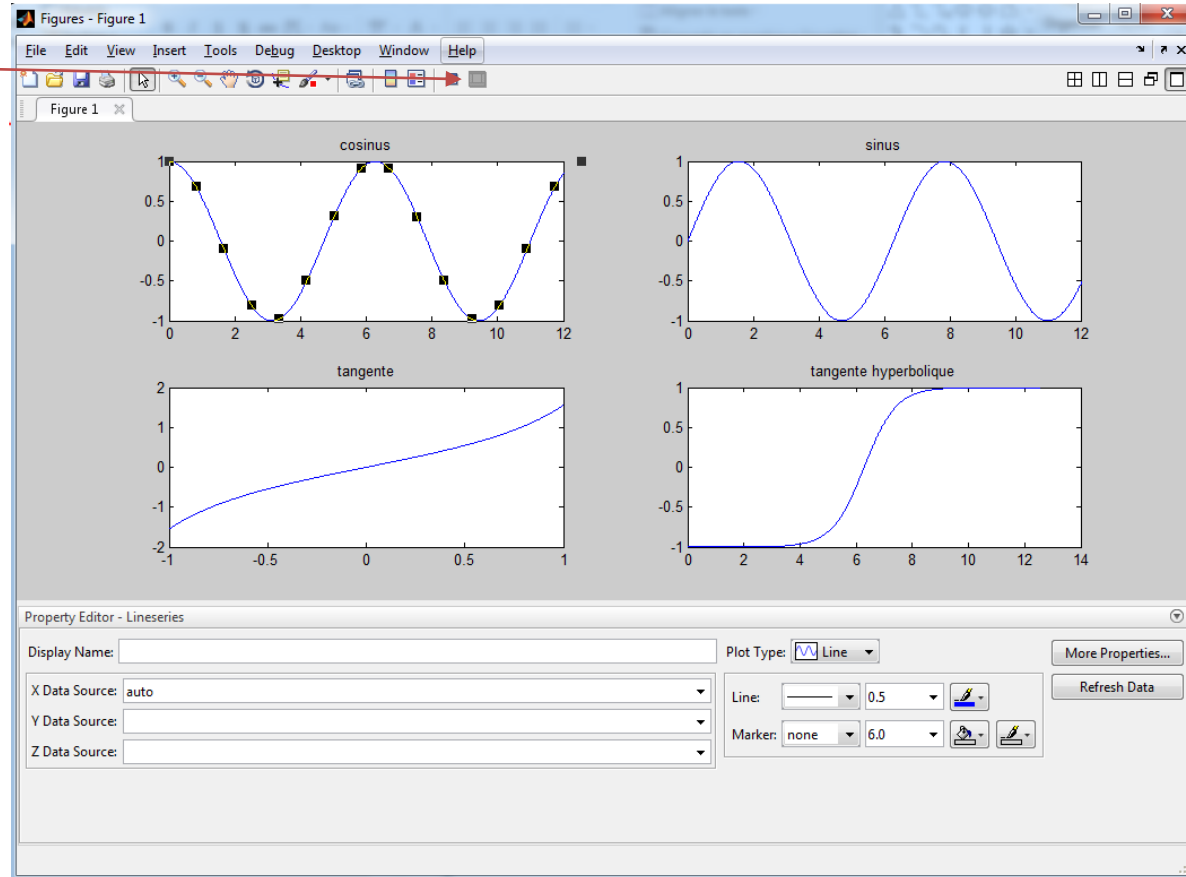
ex : 'Color' modifie la couleur du fond d'écran

valprop = valeur de la propriété

ex : 'r'

n = numéro de la figure à modifier

- Commande `get(h)` : retrouver les propriétés graphiques



Graphiques 2D

imagesc

`imagesc(x,y,I,[min max])`

x: axis x

Y: axis y

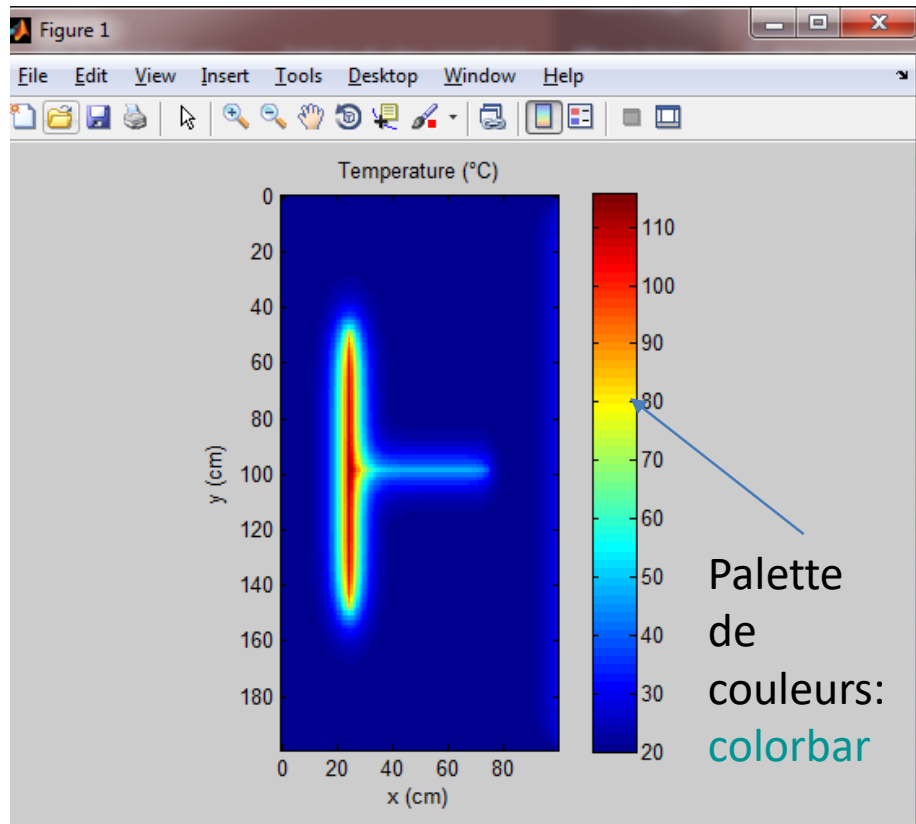
I: tableau 2d (Image)

[min max]: borne de la palette de couleur

```
8 % Load the data in Temp
9 Temp=load('TM_num');
10 I=Temp.TM;
11 N=size(I); NX=N(1);NY=N(2);
12
13 x=[0:NX-1]*100/NX;
14 y=[0:NY-1]*200/NY;
15
16 %%%%%%%%% Figures %%%%%%%%%
17
18 imagesc(x,y,I,[min(min(I)) max(max(I))]);
19 %contourf(x,y,I);
20 %contour3(I);
21 colormap('jet'); colorbar; axis image;
22 title('Temperature (°C)');
23 xlabel('x (cm)');
24 ylabel('y (cm)');
```

Choix du
type de
palette

Afficher
la palette
sur la
figure



Sauvegarder une figure

- Format figure Matlab « *.fig » :
 - Dans la fenêtre graphique : Menu File>Save nomfig.fig

- Exporter
 - Menu File>Export (tiff, jpeg, eps, ...)
 - Commande *saveas* :

```
h=plot(...); (*)  
saveas(h,'nomfich.ext')  
saveas(h,'nomfich','format')
```

format ou *ext* =

- ai = Adobe Illustrator
- eps = EPS level1
- tif = TIFF
- jpg = JPEG
- fig = fichier binaire « nomfich.fig »
- m = fichier « nomfich.fig » et crée une fonction matlab de même nom qui permettra d'appeler la figure via l'instruction *nomfich*

(*) Remarque : on peut utiliser alternativement l'instruction *h=figure(n)* pour sauvegarder l'ensemble des graphiques d'une figure (par exemple, dans le cas de « subplots »)

Sauvegarder une figure

- Exporter
 - Commande *print* :

print -f<num> -d<format> <nomfich>

num = numéro de la fenêtre graphique (par défaut la fenêtre active)

format = format de sauvegarde (tiff, jpeg, ps, eps, epsc, ...)

nomfich = nom du fichier du graphique sauvegardé

Exporte le graphique dans le répertoire de travail actif

Imprimer les graphiques

- Imprimer directement ou après exportation du graphique
- Par le menu File :
 - Page Setup = mise en page
 - Print setup = configuration de l'impression
 - Print preview = visualiser l'impression
 - Print = imprimer le graphique
- Commande *print*
 - Commande *print* : imprime directement sur l'imprimante connectée

Graphiques 3D

- Représenter les lignes de niveaux d'une fonction $z = g(x,y)$ sur le domaine plan $[a,b] \times [c,d]$
- Représenter une surface d'équations $z = g(x,y)$ sur le domaine plan $[a,b] \times [c,d]$
- Représenter une surface paramétrée d'équations
- Représenter des volumes d'équations $v = g(x,y,z)$

Graphiques 3D

- Commande *meshgrid* : création d'un maillage du domaine $[a,b] \times [c,d]$, de maille de pas h

$$[X,Y] = \text{meshgrid}(a:h:b, c:h:d)$$

- Evaluer la fonction g aux nœuds de maillage

$$Z = g(X,Y)$$

Ex : fonction $z=x \exp(-x^2-y^2)$ sur le domaine $[-2,2] \times [-2,2]$
avec un maillage de pas $h=0.2$

$$[X,Y] = \text{meshgrid}(-2:0.2:2, -2:0.2:2);$$

$$Z = X.*\exp(-X.^2-Y.^2);$$

Lignes de niveaux

- Syntaxe

commande `contour(X,Y,Z,n)`

» n = nombre de lignes de niveaux à afficher

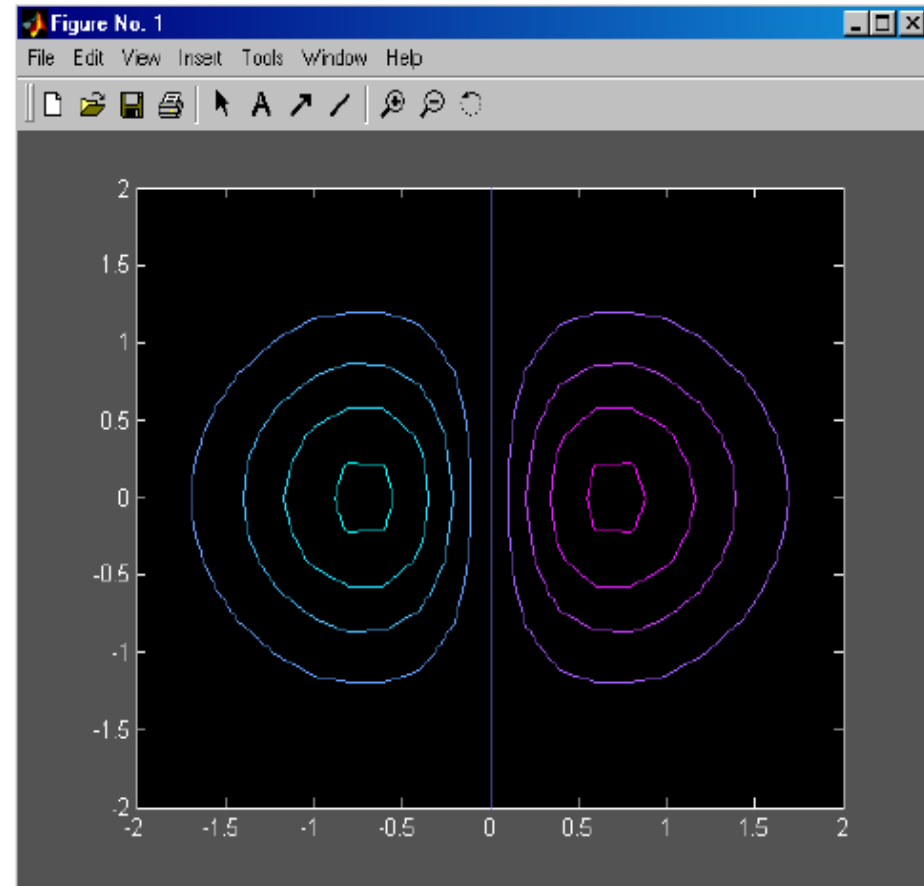
commande `contour(X,Y,Z)`

» Sélection automatique du nombre de lignes de niveaux

- Afficher les valeurs des lignes de niveaux : commande `clabel`

- Toutes les lignes de niveaux :
`[C,h] = contour(X,Y,Z,n);`
`clabel(C,h)`
- Quelques lignes de niveaux :
`[C,h] = contour(X,Y,Z,n);`
`clabel(C,h,'manual')`

Sélection des lignes de niveaux avec la souris



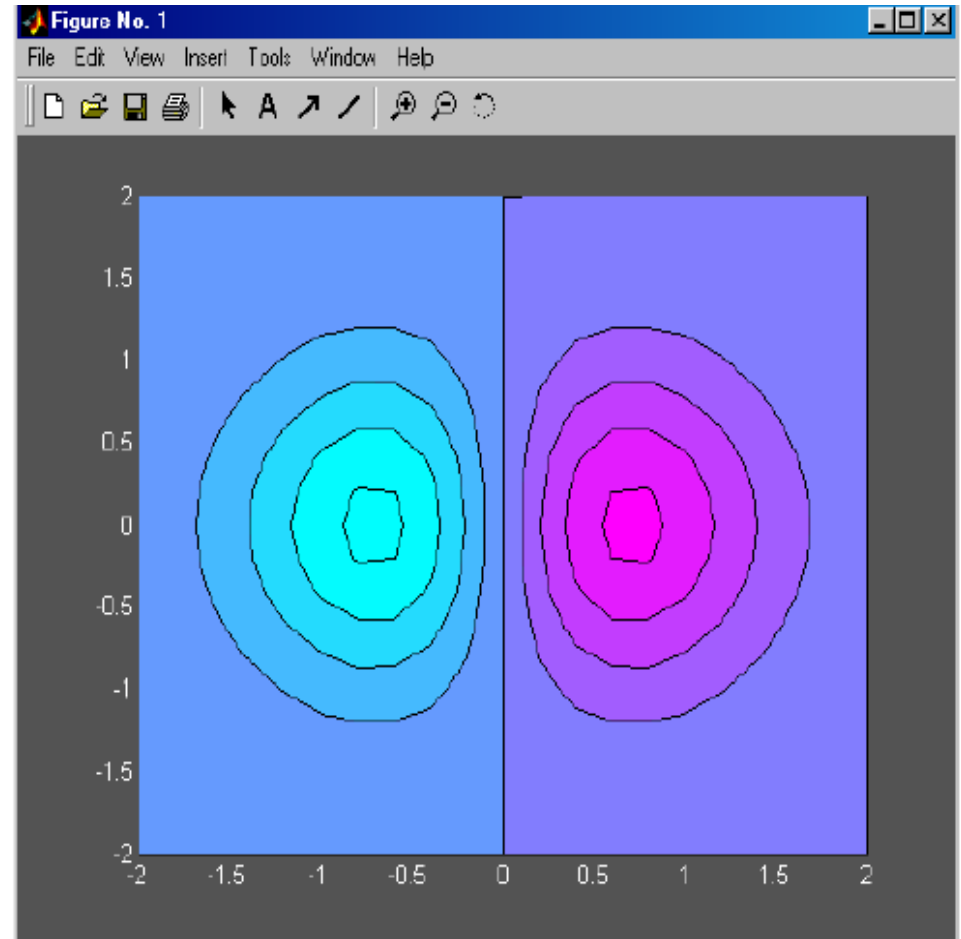
Lignes de niveaux exemples

- Syntaxe

commande `contourf(X,Y,Z,n)`

n = nombre de lignes de
niveaux à afficher

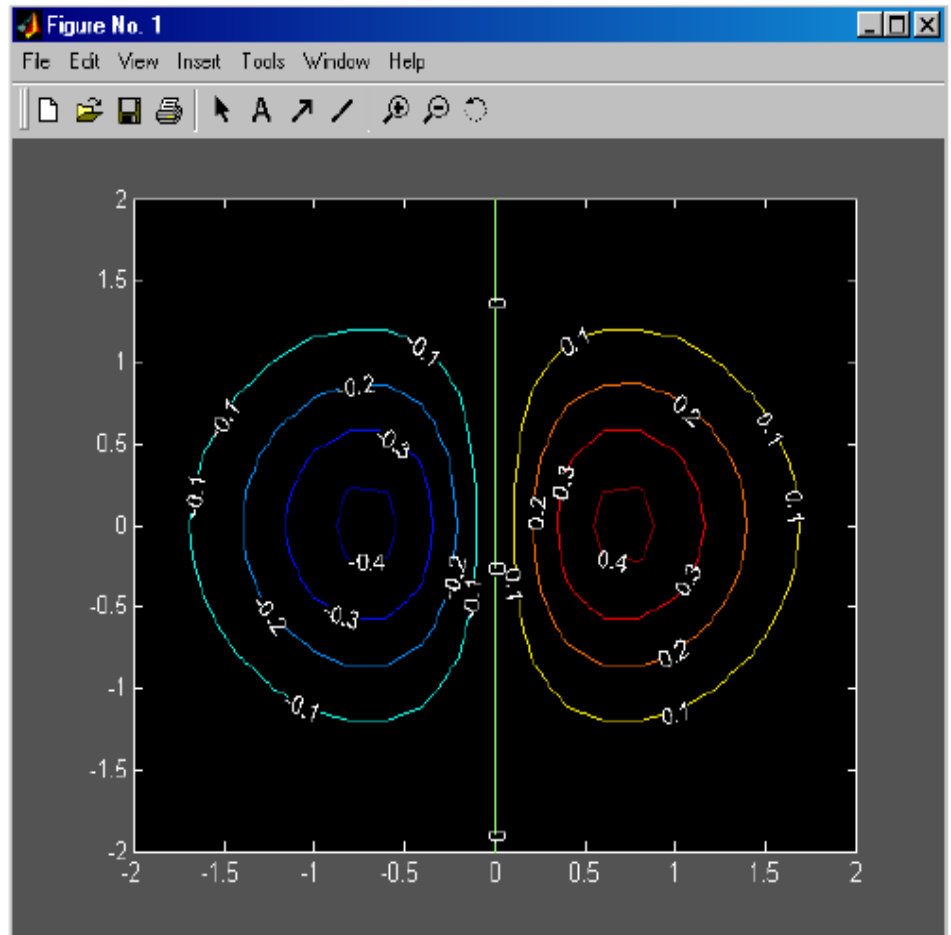
- Affiche en plus de lignes de
niveaux, un dégradé continu de
couleurs qui varient en fonction
des valeurs de la fonction



Lignes de niveaux exemples

- fonction $z = x \exp(-x^2 - y^2)$ sur le domaine $[-2,2] \times [-2,2]$
avec un maillage de pas $h=0.2$

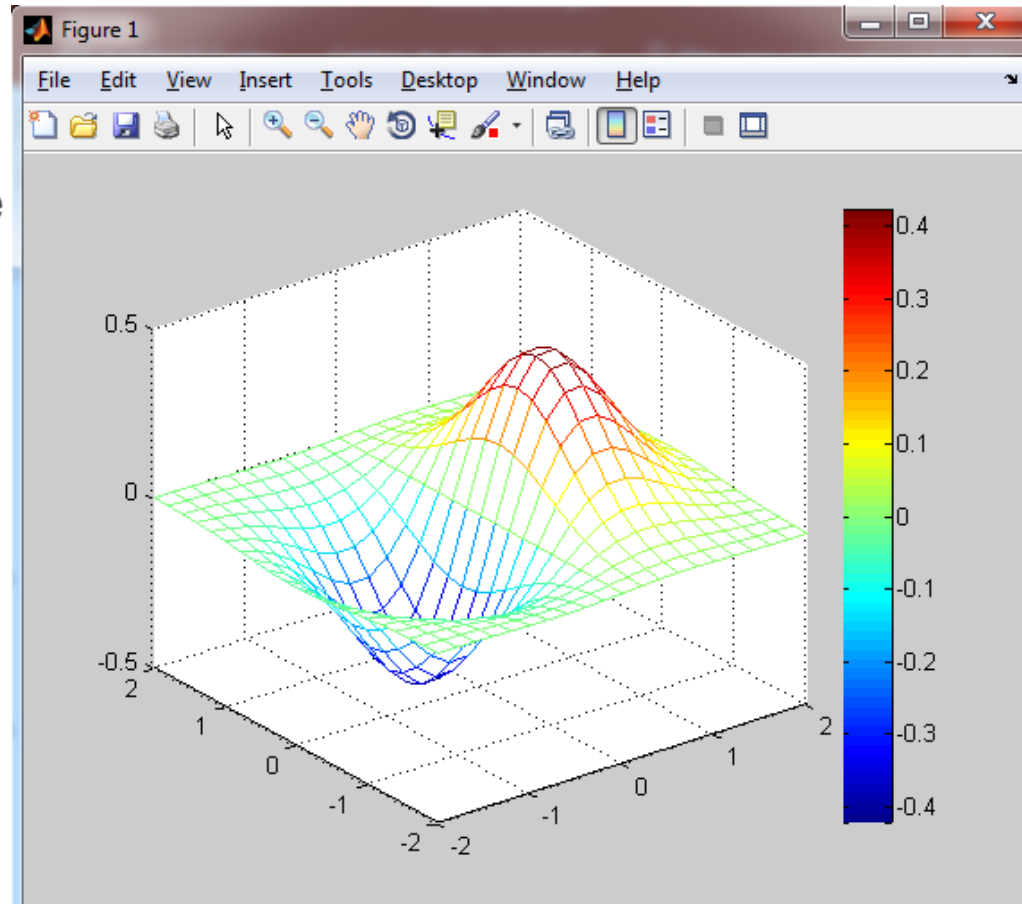
```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);  
Z = X.*exp(-X.^2-Y.^2);  
[C,h] = contour(X,Y,Z);  
clabel(C,h);
```



Surface d'équations

- commande `mesh(X,Y,Z)`
surface de la fonction $Z=g(X,Y)$
- fonction $z=x \exp(-x^2-y^2)$ sur le domaine $[-2,2] \times [-2,2]$
avec un maillage de pas $h=0.2$

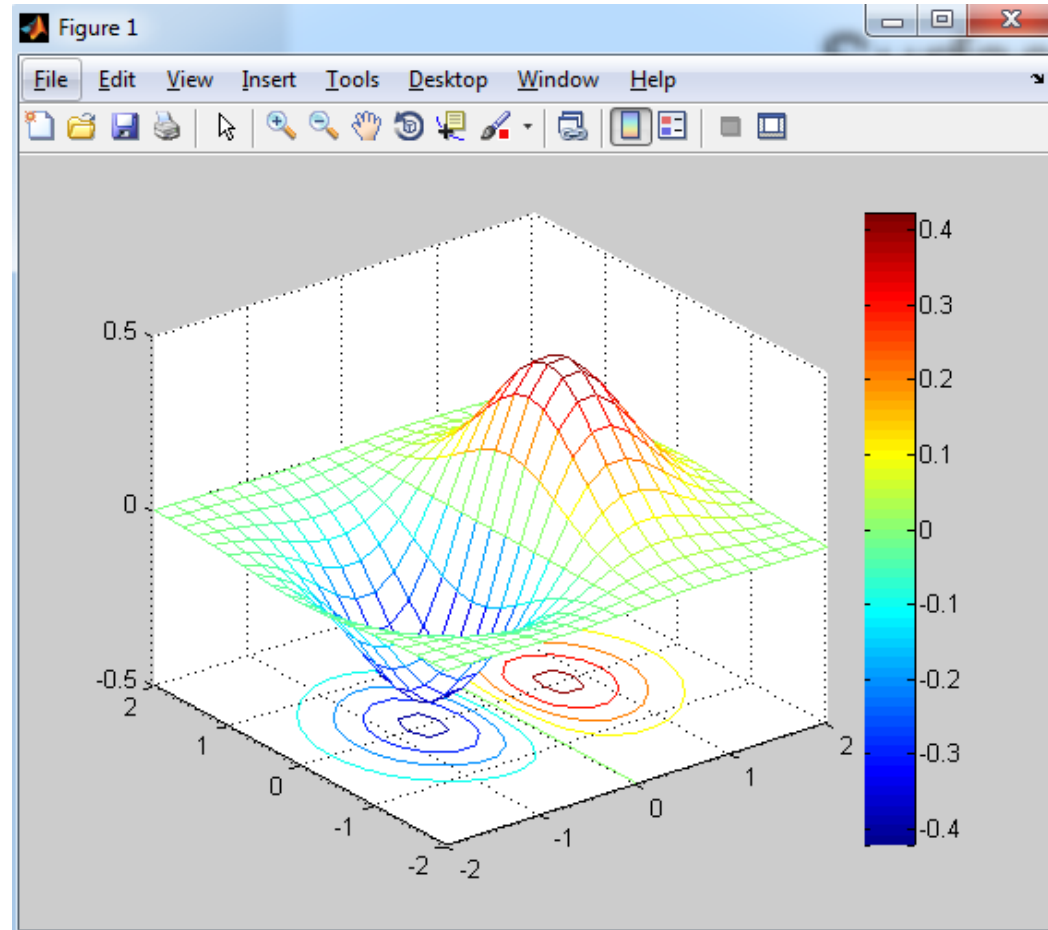
```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);  
Z = X.*exp(-X.^2-Y.^2);  
mesh(X,Y,Z);
```



Surface d'équations

- commande `meshc(X,Y,Z)`
surface de la fonction $Z=g(X,Y)$
et projection des contours sur le
plan défini par $[a,b] \times [c,d]$
- fonction $z=x \exp(-x^2-y^2)$ sur le domaine
 $[-2,2] \times [-2,2]$
avec un maillage de pas $h=0.2$

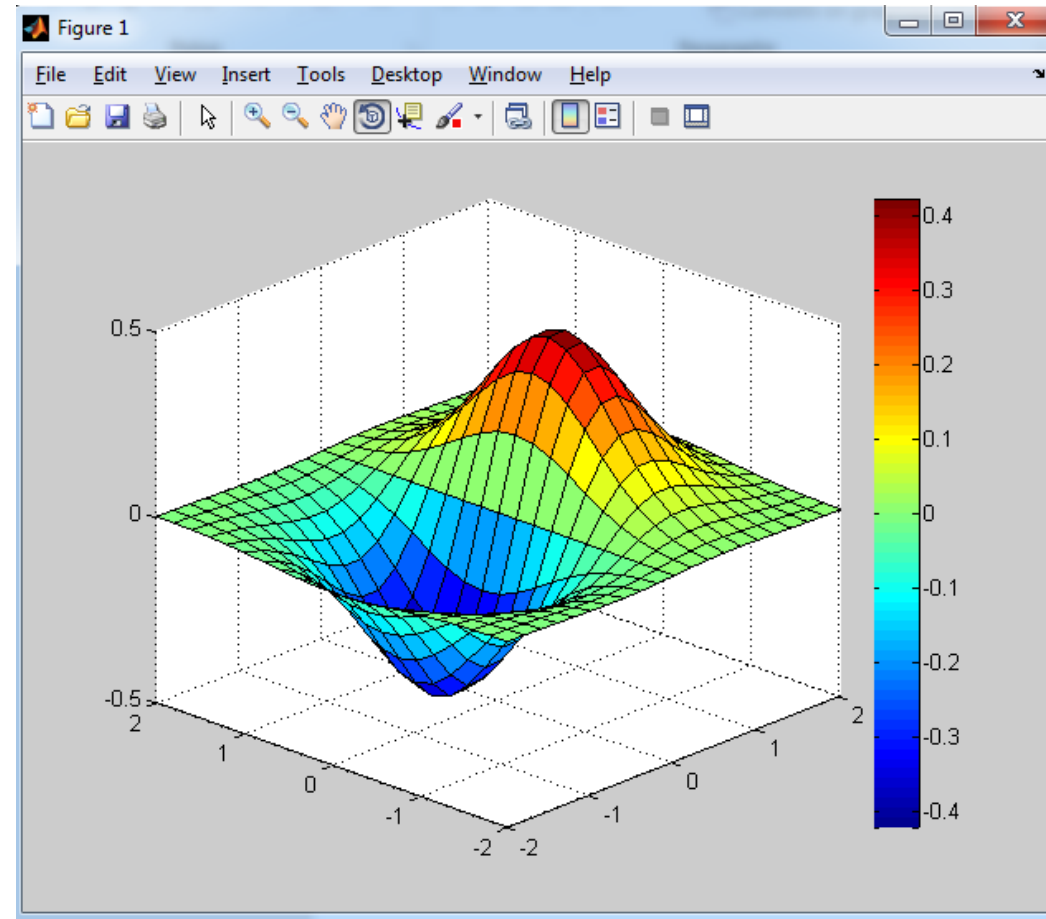
```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);  
Z = X.*exp(-X.^2-Y.^2);  
meshc(X,Y,Z);
```



Surface d'équations

- commande `surf(X,Y,Z)`
surface de la fonction $Z=g(X,Y)$
et colore la surface
- fonction $z=x \exp(-x^2-y^2)$ sur le domaine $[-2,2] \times [-2,2]$
avec un maillage de pas $h=0.2$

```
[X,Y] = meshgrid(-2:0.2:2, -2:0.2:2);  
Z = X.*exp(-X.^2-Y.^2);  
surf(X,Y,Z);
```



Options d'apparence

voir *help graph3d*

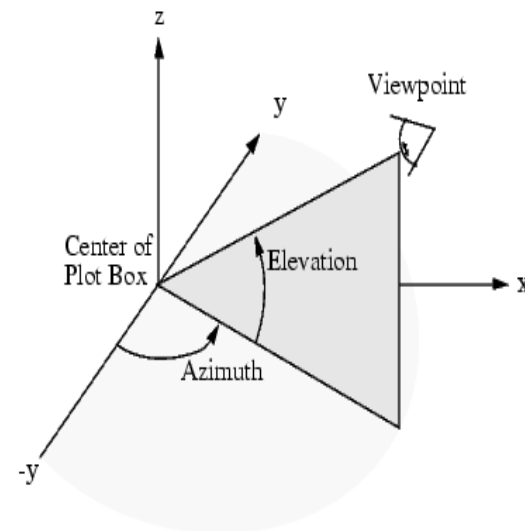
- Palette de couleurs : commande `colormap(palette)`
Ex : `colormap(gray)` = palette de dégradés de gris
`colormap(winter)` = palette de dégradés de bleu et vert
- Couleur unique : `surf(X,Y,Z,'FaceColor','couleur')`
Ex : `surf(X,Y,Z,'FaceColor','r')` colorie la surface en rouge
- Couleur du maillage de la surface : `surf(X,Y,Z,'EdgeColor','couleur')`
Ex : `surf(X,Y,Z,'EdgeColor','none')` supprime le maillage

Options d'apparence

voir *help graph3d*

- Contrôle des axes :
 - commande *axis [xmin xmax ymin ymax zmin zmax]*
- Point de vision :
 - commande *view(visionhor,visionvert)*

Ex : *view(30,65)* = angle (30°,65°)
» Orientation 3D par défaut : commande *view(3)*
équivalent à *view(-37.5,30)*
 - commande *rotate3d* = rotation interactive
- Barre de couleurs : commande *colorbar*
- Eclairage d'une surface :
 - Position de l'éclairage : commande *camlight <position>*
Ex : *camlight left*
 - Mode d'éclairage : commande *lighting <mode>*
Ex : *lighting phong*



Utilisation de Matlab pour le calcul scientifique: Quelques applications

Algèbre linéaire: Système linéaire

Soit le système linéaire suivant:

$$\begin{cases} x + 2y - 3z = 5 \\ -3x - y + z = -8 \\ x - y + z = 0 \end{cases}$$

Ecrire le système matriciel associé

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

```
>> b=[5;-8;0];
```

Résoudre le système en une seule ligne:

```
>> x=A\b;
```

Le symbole `\` s'applique à des systèmes carrés et rectangulaires.

Pour les systèmes rectangulaires, donne la solution des moindres carrés

Algèbre linéaire matricielle

- Introduire la matrice suivante

```
>> mat=[1 2 -3; -3 -1 1; 1 -1 1];
```

- Calculer le rang de cette matrice

```
>> r=rank(mat);
```

- Calculer le déterminant (matrice carrée)

```
>> d=det(mat);
```

- Calculer l'inverse

```
>> E=inv(mat);
```

- $x = A \setminus b$ est la même que $x = \text{inv}(A) * b$

Algèbre linéaire: Décomposition de matrices

MATLAB a intégré les méthodes de décomposition des matrices

Les plus courants sont :

- Décomposition en valeurs propres:
`>> [V,D]=eig(X);`
- Décomposition en valeurs singulières:
`>> [U,S,V]=svd(X);`
- Décomposition QR:
`>> [Q,R]=qr(X);`
- Décomposition LU:
`>> [L,U]=lu(X);`

Algèbre linéaire: exercices

$$\begin{cases} x + 4y = 34 \\ -3x + y = 2 \end{cases}$$

**Résoudre les systèmes
linéaires suivants :**

$$\begin{cases} 2x - 2y = 4 \\ x + y = 3 \\ 3x + 4y = 2 \end{cases}$$

Algèbre linéaire: exercices

$$\begin{cases} x + 4y = 34 \\ -3x + y = 2 \end{cases}$$

```
>> A=[1 4; -3 1];  
>> b=[34;2]  
>> rank(A)  
>> x=inv(A)*b;
```

$$\begin{cases} 2x - 2y = 4 \\ x + y = 3 \\ 3x + 4y = 2 \end{cases}$$

```
>> A=[2 -2; -1 1; 3 4];  
>> b=[4; 3; 2]  
>> rank(A)  
→ rectangular matrix  
>> x1=1\b;  
→ Donne la solutions de moindres carré  
>> error=abs(A*x1-b)
```

Polynômes

- Plusieurs fonctions peuvent être décrits par un polynôme d'ordre élevé
- MATLAB représente un polynôme par un vecteur des coefficients.
Si P est un polynôme, alors

$$ax^3 + bx^2 + cx + d$$

$$P(1) = a$$

$$P(2) = b$$

$$P(3) = c$$

$$P(4) = d$$

- $P = [1 \ 0 \ -2]$ représente le polynôme $x^2 - 2$
- $P = [2 \ 0 \ 0 \ 0]$ représente le polynôme $2x^3$

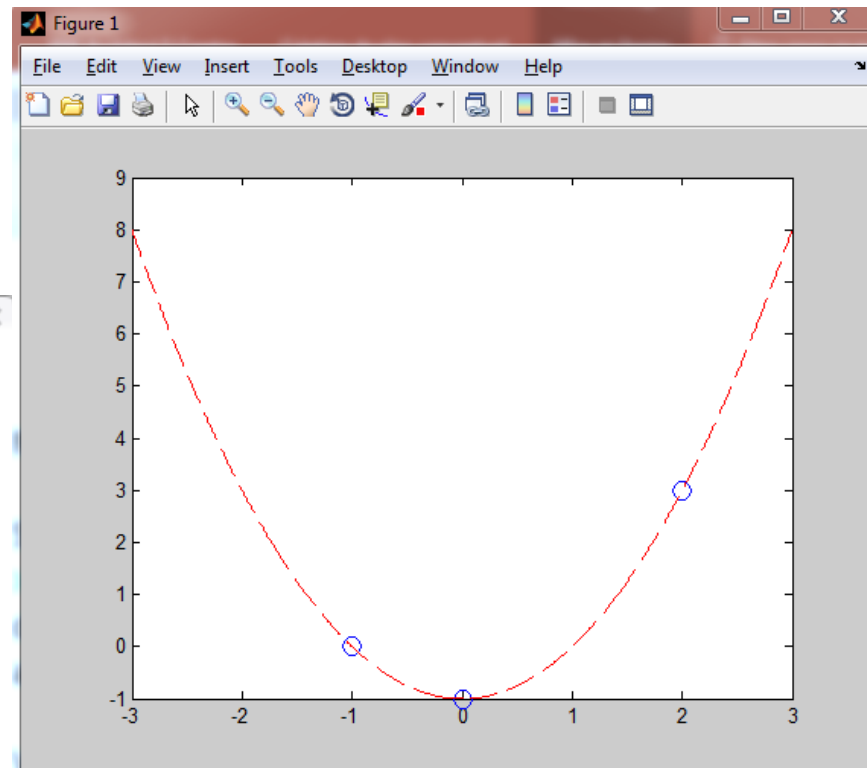
Opérations sur les polynômes

- Un polynôme de longueur $N+1$ est de degré N
- Racines d'un polynôme
`>> r = roots(P)`
 r est un vecteur de longueur N
- On peut reconstruire un polynôme à partir de ses racines
`>> P = poly(r)`
 r est un vecteur de longueur N
- Evaluer un polynôme en un point
`>> y0 = polyval(P,x0)`
 $x0$ et $y0$ sont des scalaires
- Evaluer un polynôme sur un ensemble de points
`>> y = polyval(P,x)`
 x est un vecteur ; y est un vecteur de même taille

Interpolation Polynomiale

- MATLAB interpole des données par un polynôme
- Soit les vecteurs $X = [-1 \ 0 \ 2]$ et $Y = [0 \ -1 \ 3]$
`>> p2 = polyfit(X,Y,2);`
détermine la meilleure interpolation polynomiale de degré 2
aux points $(-1,0)$, $(0,-1)$ et $(2,3)$
- Voir `help polyfit` pour plus d'informations

```
+11 images2d_cours.m x oscillateur.m x equadiff.m x interp_pol_cours.m x
1 - clear; close all; clc;
2
3 - X = [-1 0 2];
4 - Y = [0 -1 3];
5
6 - p2 = polyfit(X,Y,2);
7
8 - plot(X,Y,'o','MarkerSize',10)
9 - hold on
10 - x=-3:0.01:3;
11 - plot(x,polyval(p2,x),'r--')
```



Ajuster une courbe: « fitter une courbe »

L'ajustement de courbe est une technique d'analyse d'une courbe, consistant à construire une courbe à partir de fonctions mathématiques et d'ajuster les paramètres de ces fonctions pour se rapprocher de la courbe mesurée — on parle donc aussi d'ajustement de paramètres. On utilise souvent le terme anglais **curve fitting**.

Voir [help fit](#). Il existe aussi [lsqcurvefit](#)

`fitobject = fit(x,y,fitType)` crée le fit pour les données en `x` et `y` avec le modèle spécifiée par `fitType`. Pour la liste complète, voir [Model Names and Equations](#)

Exemples de Library Model Name	Description
'poly1'	Linear polynomial curve
'poly11'	Linear polynomial surface
'poly2'	Quadratic polynomial curve
'linearinterp'	Piecewise linear interpolation
'cubicinterp'	Piecewise cubic interpolation
'smoothingspline'	Smoothing spline (curve)
'lowess'	Local linear regression (surface)

Ajuster une courbe: « fitter une courbe »

Fitter une courbe quadratique

```
>> load census;  
>> f=fit(cdate,pop,'poly2')  
>> plot(f,cdate,pop)
```

f =

Linear model Poly2:

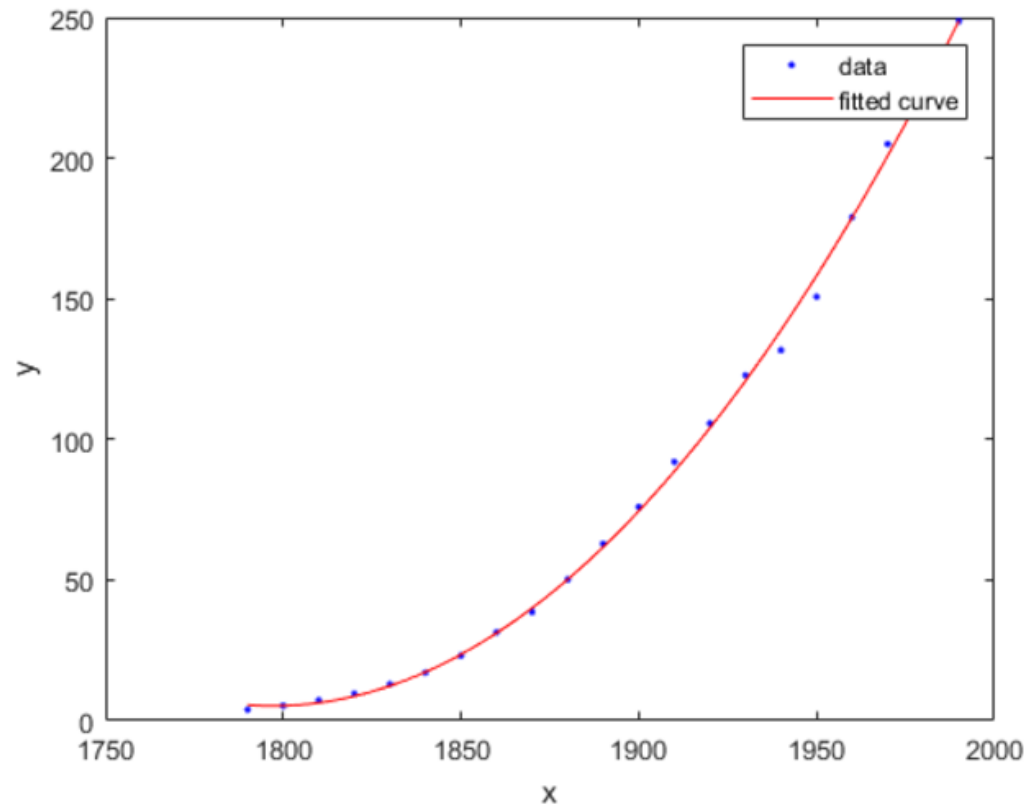
$$f(x) = p1*x^2 + p2*x + p3$$

Coefficients (with 95% confidence bounds):

p1 = 0.006541 (0.006124, 0.006958)

p2 = -23.51 (-25.09, -21.93)

p3 = 2.113e+04 (1.964e+04, 2.262e+04)



Polynômes: Exercices

- Evaluer $y = x^2$ pour $x=-4 :0.1 :4$
- Ajouter un bruit aux données. Utiliser `randn`. Tracer les données bruitées avec le marqueur.
- Interpoler les données bruitées par un polynôme de degré 2
- Tracer le résultat et la donnée initiale dans une même figure avec des couleurs différentes

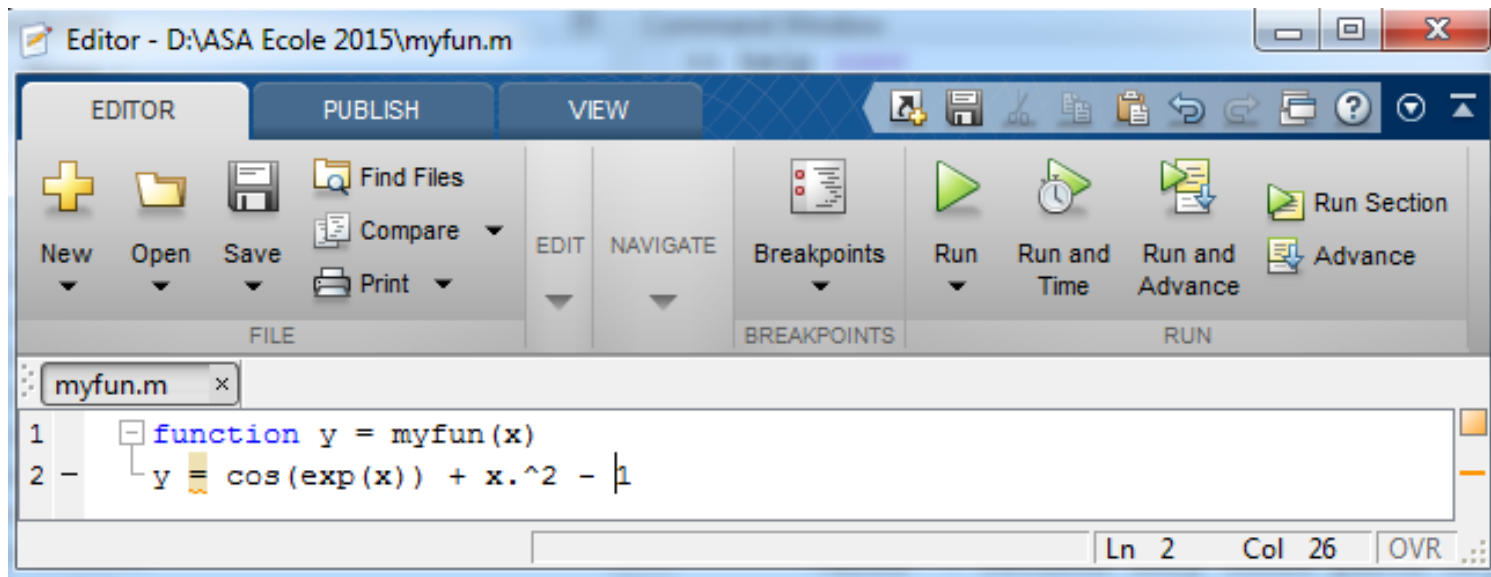
Polynômes: Exercices

- Evaluer $y = x^2$ pour $x=-4 : 0.1 : 4$
`>> x=-4:0,1:4;`
`>> y=x.^2;`
- Ajouter un bruit aux données. Utiliser `randn`. Tracer les données bruitées avec le marqueur.
`>> y=y+randn(size(y));`
`>> plot(x,y,'.');`
- Interpoler les données bruitées par un polynôme de degré 2
`>> p=polyfit(x,y,2);`
- Tracer le résultat et la donnée initiale dans une même figure avec des couleurs différentes
`>> hold on;`
`>> plot(x,polyval(p,x),'r')`


Optimisation: Equation non linéaire

- Beaucoup de problèmes nécessitent de résoudre $f(x) = 0$
- **fzero** calculer les zéros de n'importe quelle fonction
 - **fzero** a besoin d'une fonction comme input
 - 1 indique la recherche du zéro au voisinage de 1

```
» x=fzero('myfun',1)  
» x=fzero(@myfun,1)
```



Optimisation: Minimisation d'une fonction

- `fminbnd` : minimise une fonction sur un intervalle borne
`>> x=fminbnd('myfun',-1,2);`
- `myfun` prend une entrée scalaire et renvoie une sortie scalaire
- `myfun(x)` sera le minimum de `myfun` pour $-1 \leq x \leq 2$
- `fminsearch` : sans contrainte d'intervalle
`>> x=fminsearch('myfun',.5)`
- cherche un minimum local de `myfun` avec un point initiale $x = 0.5$
- On a pas besoin de crée un chier pour définir une fonction
`>> x=fzero(@myfun,1)`
- Au lieu de cela, vous pouvez faire une fonction anonyme
`>> x=fzero(@(x) (cos(exp(x)+x^2-1),1);`


input fonction à évaluer

Optimisation: Exercices

- Déterminer le minimum de la fonction
 $f(x) = \cos(4x) \sin(10x) e^{-|x|}$ dans l'intervalle $[-\pi, \pi]$
Utiliser `fminbnd`
- Tracer cette fonction et vérifier l'existence d'un minimum
- Introduire la fonction suivante :

```
>> function y=myFun(x)  
>> y=cos(4*x).*sin(10*x).*exp(-abs(x));
```

 - Trouvez le minimum dans `command window` :
 - Tracer la fonction et vérifier le résultat

Optimisation: Exercices

- Déterminer le minimum de la fonction
 $f(x) = \cos(4x) \sin(10x) e^{-|x|}$ dans l'intervalle $[-\pi, \pi]$
Utiliser `fminbnd`
- Tracer cette fonction et vérifier l'existence d'un minimum
- Introduire la fonction suivante :

```
>> function y=myFun(x)  
>> y=cos(4*x).*sin(10*x).*exp(-abs(x));
```

 - Trouvez le minimum dans `command window` :

```
>> x0=fminbnd('myFun',-pi,pi);
```
 - Tracer la fonction et vérifier le résultat

```
>> figure; x=-pi:0.01:pi; plot(x,myFun(x);
```

Intégration numérique

- MATLAB contient des méthodes d'intégration
- Adaptive Simpson's quadrature (input est une fonction)

```
>> q=quad('myfun',0,10);
```

➤ q est l'intégrale de la fonction **myfun** de 0 à 10

```
>> q2=quad(@(x) sin(x)*x,0,pi)
```

➤ q2 est l'intégrale de $\sin(x)$ de 0 à pi

- Méthode des trapèzes (input est un vecteur)

```
>> x=0:0.01:pi;
```

```
>> z=trapz(x,sin(x))
```

➤ z est l'intégrale de $\sin(x)$ de 0 à pi

```
>> z2=trapz(x,sqrt(exp(x))./x)
```

➤ z2 est l'intégrale de $\sqrt{e^x}/x$ de 0 à pi

Résolution d'équations différentielles ordinaires (EDO): fonction ODE

- EDO explicites type :

$$\frac{dy}{dt} = f(t, y)$$

- EDO linéairement implicites type :

$$M(t, y) \frac{dy}{dt} = f(t, y)$$

- EDO totalement implicites type :

$$f(t, y, y') = 0$$

Fonctions: ODE

Les principales fonctions de Matlab qui permettent la résolution des équations différentielles sont :

- `ode23`, `ode45`... : méthode de Runge-Kutta
- `ode15s`, `ode23s`, ... : EDO linéairement implicites
- `ode23i` : EDO totalement implicites

Syntaxe : ode23, ode45, ...

$$\frac{dy}{dt} = f(t, y)$$

`[T, Y] = ode45 (odefun, tspan, y0)`

`odefun` : fonction f de l'EDO

`tspan` : intervalle de t

`y0` : conditions initiales

`T` : vecteur contenant les instants auxquels la solution calculée

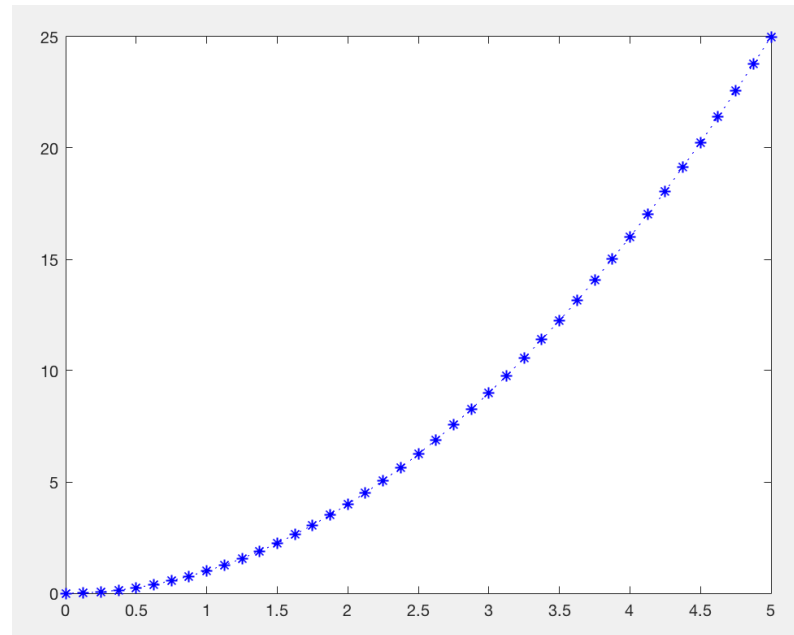
`Y` : la solution

Exemple : EDO d'ordre 1

Résoudre $y' = 2t$, avec la condition initiale $y(0) = 0$ sur $[0; 5]$

`[T, Y] = ode45 (@(t, y) 2 *t, [0 5], 0)`

script.m	f.m
<pre>1 - clear ALL; 2 - [T,Y]=ode45(@(t,y) f(t,y), [0 3 - plot(T,Y,'b*:')</pre>	<pre>1 - function w = f(t,y) 2 - w=2*t; 3 - end</pre>



Exemple : équation de l'oscillateur harmonique (d'ordre 2)

Soit l'équation différentielle du second ordre connue sous le nom de l'équation de l'oscillateur harmonique :

$$\frac{d^2y}{dt^2} + \omega_0^2 y = 0$$

Pour résoudre cette équation à l'aide des solveurs ODE, il faut l'exprimer sous une forme vectorielle, pour cela :

Création de 2 vecteurs de dimension 2 correspondant à l'ordre de l'équation différentielle :

Vecteur \mathbf{v} : solution

- $v(1) = y$
- $v(2) = \frac{dy}{dt}$

Vecteur \mathbf{w} : dérivées

- $w(1) = \frac{dy}{dt}$
- $w(2) = \frac{d^2y}{dt^2}$

Exemple : Equation de l'oscillateur harmonique

Expression sous forme vectorielle

Vecteur \mathbf{v} : solution

- $v(1) = y$
- $v(2) = \frac{dy}{dt}$

Vecteur \mathbf{w} : dérivées

- $w(1) = \frac{dy}{dt}$
- $w(2) = \frac{d^2y}{dt^2}$

Transformation de l'équation différentielle d'ordre 2 en un système de deux équations différentielles du premier ordre :

Equation différentielle:

- $\frac{d^2y}{dt^2} + \omega_0^2 y = 0$
- $\frac{d^2y}{dt^2} = -\omega_0^2 y$

Forme vectorielle:

- $w(1) = \frac{dv(1)}{dt} = v(2)$
- $w(2) = \frac{dv(2)}{dt} = -\omega_0^2 v(1)$

Exemple : Equation de l'oscillateur harmonique

Expression sous forme vectorielle

Pour résumer, on transforme l'équation différentielle d'ordre 2 en un système de deux équations différentielles du premier ordre :

Equation différentielle:

$$\frac{d^2 y}{dt^2} + \omega_0^2 y = 0$$

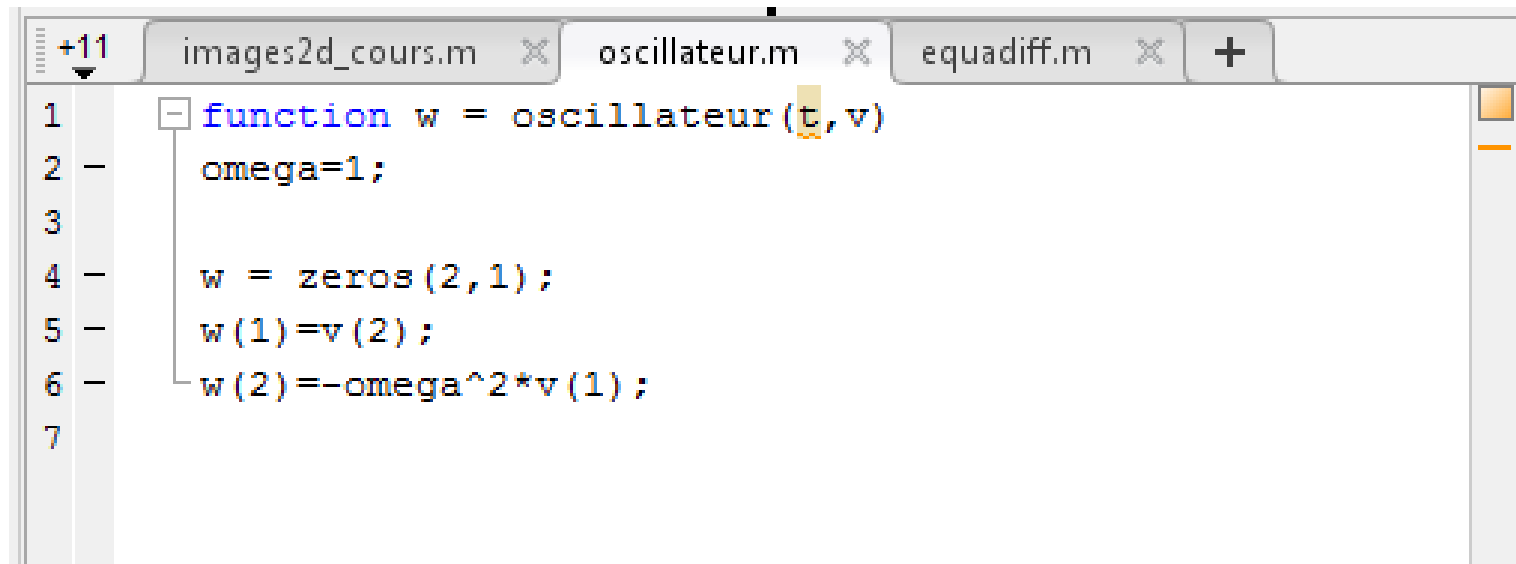
Système d'équations différentielles du premier ordre :

- $y(t) = v_1 = v(1)$
- $\frac{dy(t)}{dt} = v_2 = v(2) = w(1)$
- $\frac{d^2 y(t)}{dt^2} = v_3 = -\omega_0^2 * y(t) = -\omega_0^2 * v(1) = w(2)$

Exemple : Equation de l'oscillateur harmonique

On écrit un fichier "function" correspondant à ces 2 lignes

- $w(1) = v(2)$
- $w(2) = -\omega_0^2 v(1)$



```
+11 images2d_cours.m x oscillateur.m x equadiff.m x +
1 function w = oscillateur(t,v)
2     omega=1;
3
4     w = zeros(2,1);
5     w(1)=v(2);
6     w(2)=-omega^2*v(1);
7
```

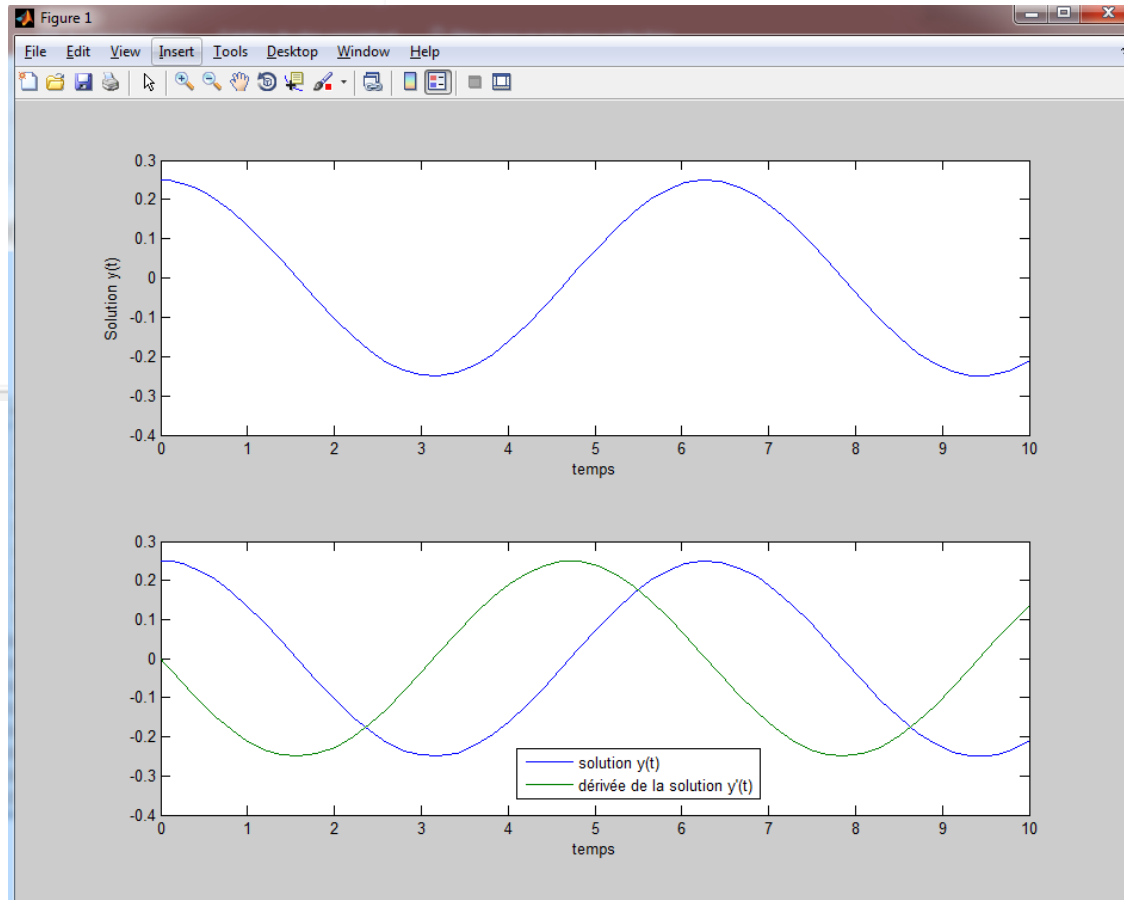
Exemple : Equation de l'oscillateur harmonique

On écrit un script .m appelé par exemple "equadiff" dans lequel :

- 1) on définit le domaine d'étude : $t_{initial} = 0$, $t_{final} = 10$: `deltat = [0, 10]`
- 2) on définit les conditions initiales : $y'(0) = 0$ et $y(0) = 0.25$: `yinit=[0.25,0]`
- 3) on appelle ode45 : `[t, Y]=ode45(@oscillateur, deltat, yinit)`
 - t : vecteur contenant les instants auxquels la solution a été calculée.
 - Y : matrice de deux colonnes :
 - la première colonne $Y(:,1)$ contient $y(t)$
 - la seconde colonne $Y(:,2)$ contient la dérivée y'
- 4) on fait la représentation graphique
 - de la solution en fonction de t : `plot(t,Y(:,1))`
 - ou la solution et sa dérivée en fonction de t : `plot(t,Y)`

Exemple : Equation de l'oscillateur harmonique

```
+10 temp_filter_svd_wavelet.m x images2d_cours.m x oscillateur.m x equadiff.m x
1 - clear; close all; clc
2
3 - deltat = [0, 10];
4 - yinit=[0.25,0];
5
6 - [t, Y]=ode45(@oscillateur, deltat, yinit);
7
8 - subplot(2,1,1)
9 - plot(t,Y(:,1))
10 - xlabel('temps');
11 - ylabel('Solution y(t)')
12
13 - subplot(2,1,2)
14 - plot(t,Y(:,2))
15 - xlabel('temps');
16 - legend('solution y(t)', ...
17 - 'dérivée de la solution y'(t)')
```



Résolution d'équations différentielles avec conditions aux limites : solveurs **BVP**

$$\begin{cases} \frac{dy}{dt} = f(t, y) \text{ pour } t \in]a, b] \\ \text{condition initiale en } a \end{cases}$$

```
[T,y]=ode23('fonc',[a,b],ya);  
plot(T,y)
```

```
fonc.m  ×  +  
function dy=fonc(t,y)  
    dy=f(t,y);  
end
```

$$\begin{cases} \frac{dy}{dt} = f(t, y) \text{ pour } t \in]a, b] \\ \text{condition aux limites} \end{cases} \quad ?$$

Les équations issues des problème physiques sont d'ordre supérieur avec conditions aux limites. Ces problèmes sont connus comme « Boundary Value Problem » (BVP)

Résolution d'équations différentielles avec conditions aux limites : solveurs **BVP**

$$\begin{cases} \frac{d^2 y}{dt^2} + y = 0 \\ \frac{dy}{dt}(0) = 1 \text{ et } y(\pi) = 0 \end{cases} \quad \text{Solution exacte: } y = \sin(t)$$

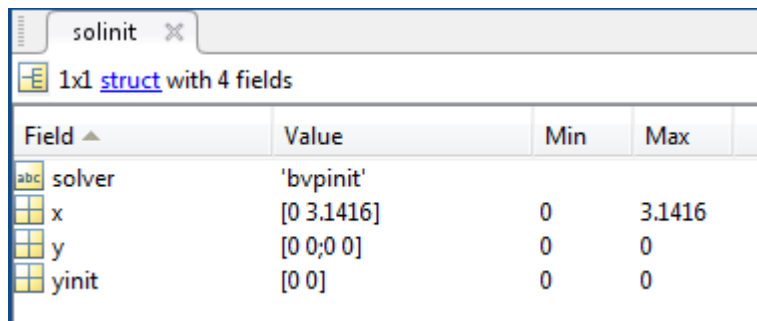
Pour résoudre numériquement, nous devons d'abord réduire l'équation de second ordre à un système d'équations du premier ordre

$$\begin{cases} \frac{dy}{dt} = z \\ \frac{dz}{dt} = -y \end{cases} \quad \text{avec } z(0) = 1 \text{ et } y(\pi) = 0$$

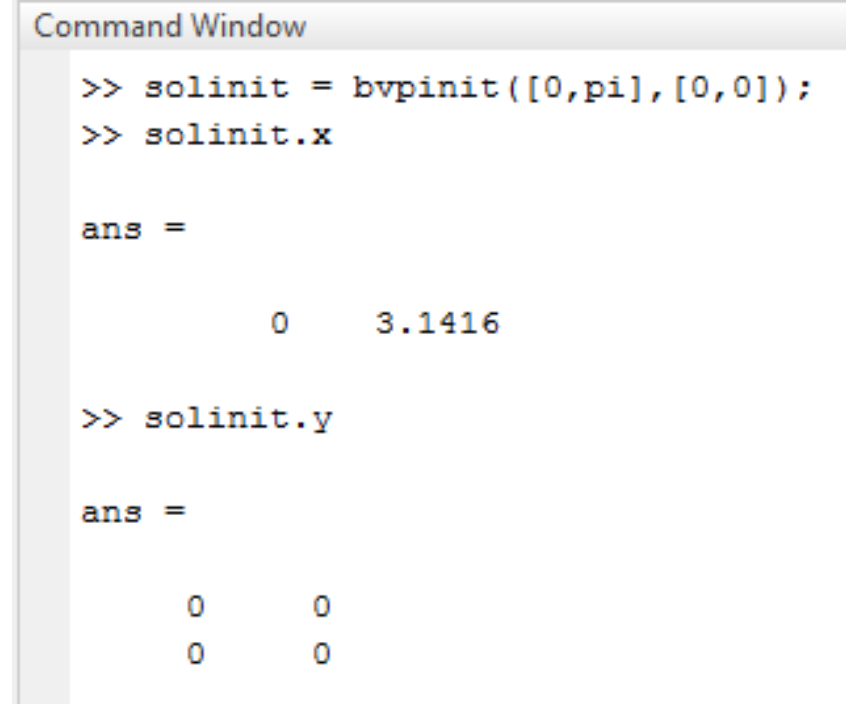
Résolution d'équations différentielles avec conditions aux limites : solveurs BVP

Solveur BVP de Matlab basé sur les étapes suivantes :

- fonction `bvpinit` définit le domaine et une solution initiale
- `solinit` est une structure :



Field ▲	Value	Min	Max
solver	'bvpinit'		
x	[0 3.1416]	0	3.1416
y	[0 0; 0 0]	0	0
yinit	[0 0]	0	0



```
Command Window

>> solinit = bvpinit([0,pi],[0,0]);
>> solinit.x

ans =

           0    3.1416

>> solinit.y

ans =

           0           0
           0           0
```

Résolution d'équations différentielles avec conditions aux limites : solveurs BVP

- `sol=bvp4c(@deriv,@bcs,solinit);`

Utilisation du solveur Matlab bvp4c avec les arguments suivants :

- `@deriv` fonction qui représente les seconds membres du système

```
function dYdx = deriv(x,Y)
    dYdx(1) = Y(2);
    dYdx(2) = -Y(1);
```

- `@bcs` fonction qui définit les conditions aux limites

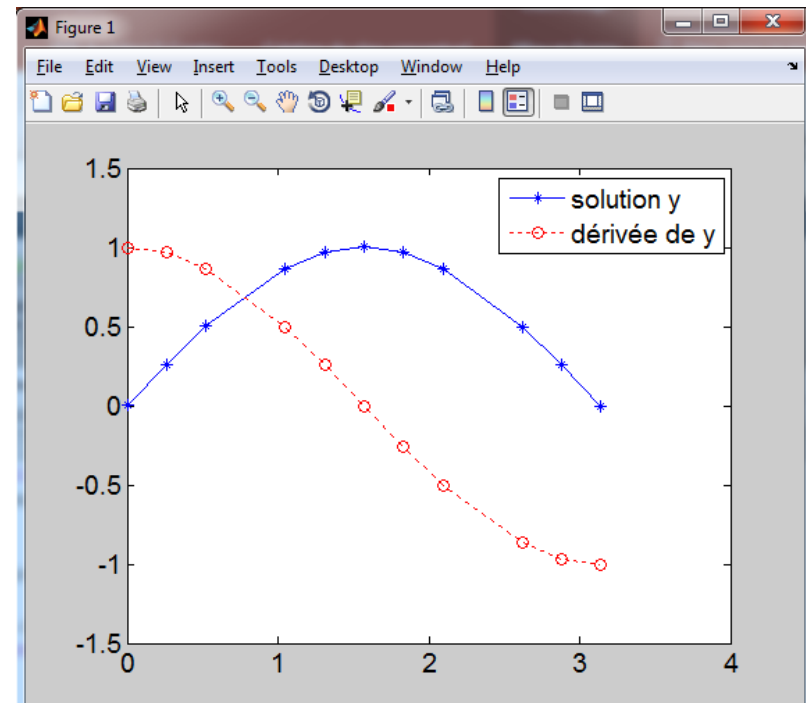
```
% boudary conditions y'(a)=1, y(b)=0.
function res = bcs(ya,yb)
    res = [ya(2)-1
           yb(1)];
```

- `solinit`

- `plot(sol.x,sol.y(1,:),'b-*')`

Résolution d'équations différentielles avec conditions aux limites : solveurs **BVP**

```
1 - clear; close all; clc;
2 - solinit = bvpinit([0,pi],[0,0]);
3 - solinit.x
4 - solinit.y
5 - sol=bvp4c(@deriv,@bcs,solinit);
6
7 - plot(sol.x,sol.y(1,:), 'b-*', sol.x,sol.y(2,:), 'r:o');
8 - legend('solution y', 'dérivée de y')
```



Solveur IBVP: (Initial-Boundary Value problems)

- ❑ Le solveur IBVP résout une PDE sous la forme suivant :

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right) \right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right)$$

$$x \in]a, b] \text{ et } t_0 \leq t \leq t_f, \quad m = \begin{cases} 0 & \text{problème cartésien } (a > 0) \\ 1 & \text{problème cylindrique} \\ 2 & \text{problème sphérique} \end{cases}$$

- ❑ Condition initiale : $u(x, t_0) = u_0(x)$ en $t_0 = t$ et $\forall x$

- ❑ Conditions aux limites : $\forall t$ et $x = a$ ou $x = b$

$$p(x, t, u) + q(x, t) f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0$$

Solveur IBVP: pde

- ❑ Fonction **pde** de Matlab :

$$sol = pdepe(m, pdefun, icfun, bcfun, xmesh, tspan)$$

- ❑ **pdefun** est la fonction qui définit c , f et s

$$[c, f, s] = pdefun(x, t, u, dudx)$$

- ❑ **icfun** est la fonction de la condition initiale :

$$u = icfun(x)$$

- ❑ **bcfun** est la fonction des conditions aux limites :

$$[pl, ql, pr, qr] = bcfun(xl, ul, xr, ur, t)$$

- ❑ **xmesh** est le maillage de l'intervalle $[a, b]$:

$$xmesh = linspace(a, b, n)$$

- ❑ **tspan** est le vecteur des instants où on cherche approximer la solution

- ❑ **sol** : est le vecteur solution où $sol(i, j, 1)$ est la solution à l'instant **tspan**(i) et au point **xmesh**(j)

Solveur IBVP: Exemple

- Equation de la chaleur 1D instationnaire:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}; \quad T(x, 0) = 0; \quad T(1, t) = 0; \quad T(1.5, t) = 200$$

- PDE: $\left(u_x = \frac{\partial u}{\partial x}\right)$

$$m = 0, a = 1, b = 1.5, t_0 = 0, t_f = 3000s$$

$$u = T, \quad c(x, t, u, u_x) = \frac{1}{\alpha}, \quad f(x, t, u, u_x) = \frac{\partial T}{\partial x}, \quad s(x, t, u, u_x) = 0$$

$$p(a, t, u(a, t)) = T(a, t), \quad q(a, t) = 0$$

$$p(b, t, u(b, t)) = T(b, t) - 200, \quad q(b, t) = 0$$

Solveur IBVP: Exemple

- Equation de la chaleur 1D instationnaire:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}; T(x, 0) = 0; T(1, t) = 0; T(1.5, t) = 200$$

```
% Equation de la chaleur 1D instationnaire
```

```
clear; close all; clc;
```

```
m = 0;
```

```
x = linspace(0,1.5,40);
```

```
t = linspace(0,2,10);
```

```
sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
```

```
% Extract the first solution component as u.
```

```
u = sol(:,:,1);
```

```
% A surface plot is often a good way to study a solution.
```

```
surf(x,t,u)
```

```
title('Numerical solution computed with 40 mesh points.')
```

```
xlabel('Distance x')
```

```
ylabel('Time t')
```

```
% A solution profile can also be illuminating.
```

```
figure
```

```
plot(x,u(end,:))
```

```
title('Solution at tf')
```

```
xlabel('Distance x')
```

```
ylabel('u(x,tf)')
```

```
% -----  
function [c,f,s] = pdex1pde(x,t,u,DuDx)
```

```
alpha=0.1;
```

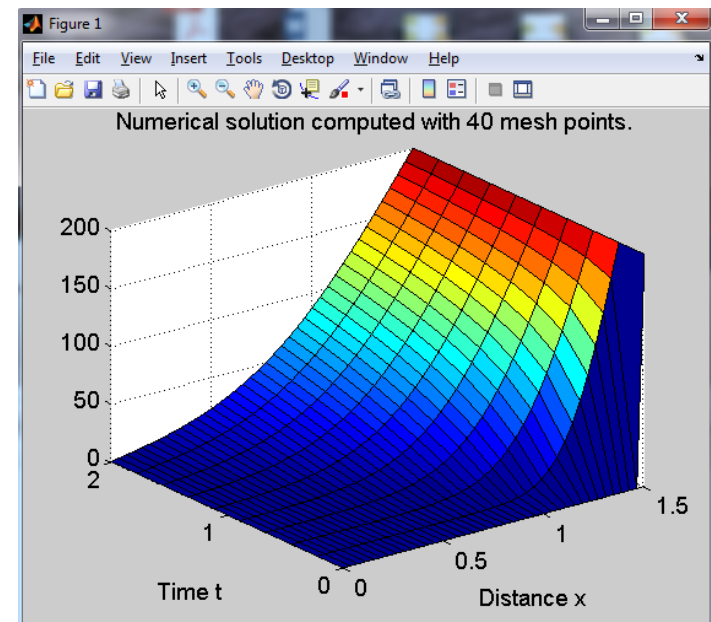
```
c = 1/alpha;
```

```
f = DuDx;
```

```
s = 0;
```

```
% -----  
function u0 = pdex1ic(x)  
u0 = 0;
```

```
% -----  
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)  
pl = ul;  
ql = 0;  
pr = ur-200;  
qr = 0;
```

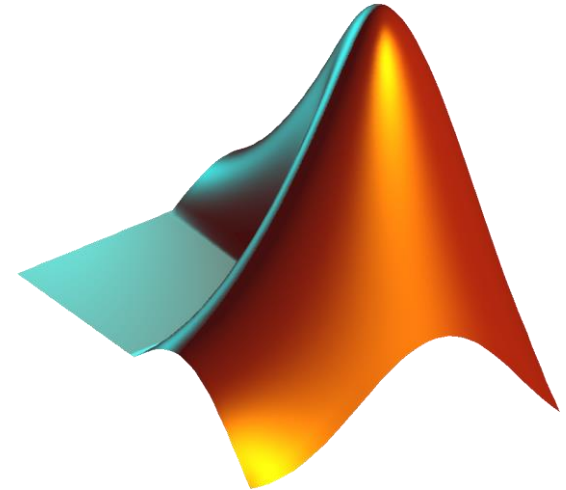


Lire et écrire des données dans différents formats de fichiers

*utiles pour importer des données expérimentales ou
numériques issues de codes/logiciels (fortran,
starccm+, ...)*



*Analyse de données
avec matlab*



Exporter des fichiers .mat

- Format des fichiers « .mat » très pratique pour enregistrer et charger des données
 - Avantages de ces fichiers: rapidité de chargement et petite taille qu'ils occupent.
- Il est possible de sauver tout le workspace : `save mesdonnees`
 - Toutes les variables du workspace seront enregistrées dans le fichier «mesdonnees.mat » dans le dossier courant.
 - Il est également possible de spécifier un chemin
`save mondossier\mesdonnees`
`save D:\chemin\vers\mondossier\mesdonnees`
- Pour sauver uniquement certaines variables, il faut les lister à la suite du nom du fichier :
`save mesvars var1 var2 var3`
`save('mesvars','var1','var2','var3')`

Importer des fichiers .mat

- L'importation des données est effectuée par la commande

`load :`

`load mesvars;`

- Toutes les variables préalablement sauveées sont restaurées avec leur nom d'origine.
- Afin de ne pas écraser d'autres variables présentes dans le workspace, les variables chargées peuvent être attribuées à une nouvelle variable, sous forme de structure (permet de structurer les données dans une même variable).

```
>> mesnouvvars = load('mesvars')
```

```
mesnouvvars =
```

```
var1: 100
```

```
var2: 435
```

```
var3: 54
```

Exporter des fichiers ascii

Fonction **save**

même commande que pour les fichiers .mat mais en précisant le type de fichiers -ascii après les variables. Il est également possible d'ajouter l'option -tabs pour que les valeurs soient séparées par des tabulations.

```
>> save('mesvars.txt','var1','var2','var3','-ascii')
```

```
>> save('mesvars.txt','var1','var2','var3','-ascii','-tabs')
```

➤ Limitations:

- ❖ mise en page n'est pas contrôlable par l'utilisateur
- ❖ Enregistrement restreint aux valeurs numériques: aucun caractère,
- ❖ structure, cell, etc, n'est autorisé.
- ❖ Pertes des noms des variables
- ❖ même nombre de colonne pour chaque ligne

Exporter des fichiers ascii

Autres méthodes `fprintf` (voir `help`)

1) la création du fichier : `fopen`. Les paramètres de la fonction sont le nom du fichier (év. avec le chemin d'accès) et la permission.

La fonction retourne un identifiant pour ce fichier qu'il nous faut conserver.

```
fid = fopen(filename, permission);
```

```
Exemple: >> fid = fopen('d:\chemin\vers\monfichier.txt', 'w');
```

2) L'écriture dans les fichiers: `fprintf`. Les paramètres sont l'identifiant du fichier, le format, puis les données :

```
count = fprintf(fid, format, A, ...)
```

3) La fermeture du fichier: `fclose`.

Pour le fichier log.txt:

```
>> x = 1:1:100;  
>> y = [x; log(x)];  
>> fid = fopen('log.txt', 'wt');  
>> fprintf(fid, '%6.2f %12.8f\n', y);  
>> fclose(fid)
```

Le fichier log.txt contient alors :

```
1.00 0.00000000  
1.10 0.09531018  
1.20 0.18232156  
1.30 0.26236426  
...  
99.90 4.60416969  
100.00 4.60517019
```

Importer des fichiers ascii

Autres méthodes **fscanf** (voir **help**)

La fonction **fscanf** est l'équivalent de la fonction **fprintf** pour la lecture des fichiers

A = fscanf(fid, format)

La fonction lit tout le fichier et converti les données au format spécifié, puis les retourne à la matrice A.

[A,count] = fscanf(fid, format, size)

En reprenant le fichier log.txt:

```
>> fid = fopen('log.txt', 'r');  
>> a = fscanf(fid, '%g %g', [2 inf]); % sur 2 lignes  
>> a = a'; % transposition en colonnes  
>> fclose(fid)
```

Le fichier log.txt contient alors :

```
1.00 0.00000000  
1.10 0.09531018  
1.20 0.18232156  
1.30 0.26236426  
...  
99.90 4.60416969  
100.00 4.60517019
```

Importer des fichiers ascii

Autres méthodes **textread** (voir **help**)

- La différence de **textread** par rapport à **fscanf** est la capacité de celui-ci à extraire de
- multiples variables en une commande.
- La fonction est effectuée jusqu'à ce que le fichier soit entièrement lu.
- **textread** est utile pour lire des fichiers textes avec un format connu

`[A,B,C,...] = textread('filename','format')`

Pour contrôler le nombre d'éléments lus: `[A,B,C,...] = textread('filename','format',N)`

Exemples:

```
>> [names, types, x, y, answer] = textread('mydata.dat', '%s %s %f %d %s', 1)
names =
'Sally'
types =
'Level1'
x =
12.340000000000000
y =
45
answer =
'Yes'
```

Fichier contact.dat Sally Level1 12.34 45 Yes
--

Autres fonction de traitement de fichiers

Pour lire et écrire dans des fichiers ascii:

- `fgets, fgetl`
- `load`
- `dlmread, dlmwrite`
- `csvread, csvwrite`
- `importdata`
- `xlswrite, xlsread, readtable`: écrire et lire des fichiers Excel

Fichier climate.xlsx:

Time	Temp	Visibility
12	98	clear
13	99	clear
14	97	partly cloudy

```
>> T = readtable('climate.xlsx','Sheet','Temperatures')
```

```
T =
```

Time	Temp	Visibility
------	------	------------

12	98	'clear'
----	----	---------

13	99	'clear'
----	----	---------

14	97	'partly cloudy'
----	----	-----------------

```
>> [num,headertext] = xlsread('climate2.xlsx','Temperatures')
```

```
num =
```

12	98
----	----

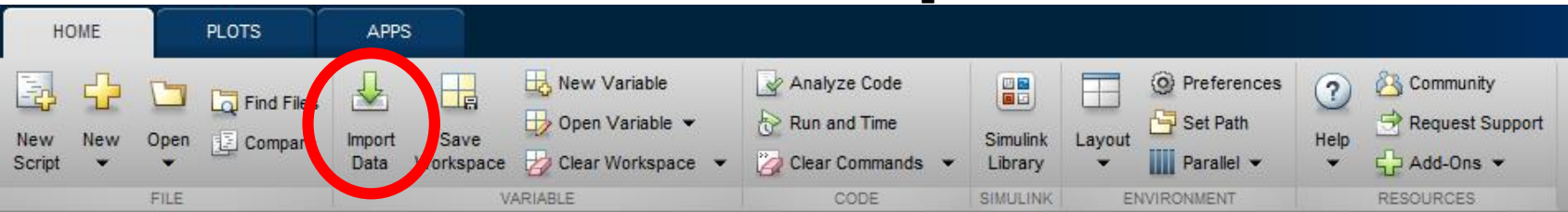
13	99
----	----

14	97
----	----

```
headertext =
```

'Time'	'Temp'
--------	--------

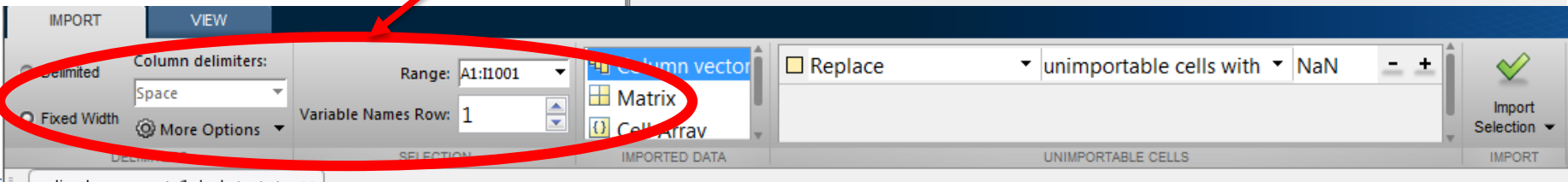
Bouton Import Data



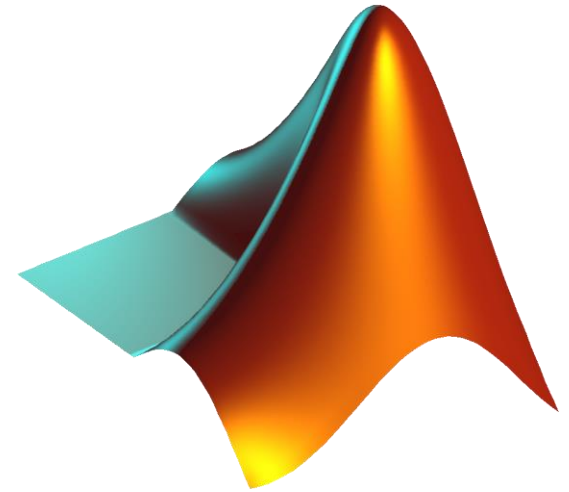
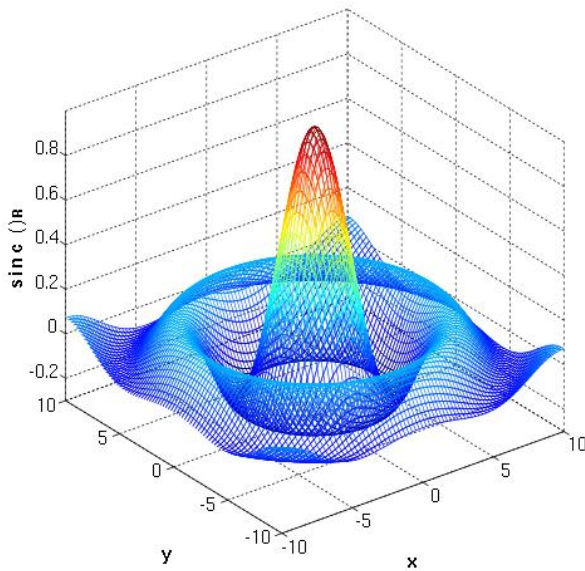
Options

The screenshot shows the MATLAB Import Data dialog box with the 'Import' tab selected. The 'Column delimiters' are set to 'Space', 'Range' is 'A1:I1001', 'Variable Names Row' is '1', and 'Import Selection' is checked. The resulting data table is shown below.

	VarName1	VarName2	VarName3	VarName4	VarName5	VarName6	Var...	VarName8	VarName9
	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NU...	NUMBER	NUMBER
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000		0.00000	0.00000
2	0.100000E-02	0.152836E-02	0.113778E-01	0.352604E-04	0.784721E-03	0.771709E-03		0.103811E-02	0.100000E-02
3	0.200000E-02	0.204151E-02	0.641704E-02	0.293531E-02	0.373678E-02	0.839399E-02		0.203386E-02	0.200000E-02
4	0.300000E-02	0.430725E-02	0.908014E-03	0.100479E-01	0.141942E-01	0.629081E-02		0.224581E-02	0.300000E-02
5	0.400000E-02	0.341191E-02	2.83582E-5	0.324120E-02	0.163404E-01	0.831616E-02		0.317750E-02	0.400000E-02
6	0.500000E-02	0.274556E-02	0.105335E-02	0.801042E-03	0.308452E-01	0.105086E-01		0.381766E-02	0.500000E-02
7	0.600000E-02	0.229966E-02	0.101064E-02	0.214674E-04	0.390711E-01	0.624010E-02		0.422592E-02	0.600000E-02
8	0.700000E-02	0.206351E-02	0.257161E-02	0.208602E-04	0.141018E-01	0.104581E-02		0.460027E-02	0.700000E-02
9	0.800000E-02	0.269145E-04	0.459312E-02	0.111255E-02	0.252653E-01	0.267109E-01		0.790426E-02	0.800000E-02
10	0.900000E-02	0.189583E-04	0.784499E-02	0.584221E-04	0.206489E-01	0.296130E-01		0.874900E-02	0.900000E-02
11	0.100000E-01	0.489569E-03	0.769586E-02	0.172184E-04	0.150123E-01	0.465230E-01		0.957287E-02	0.100000E-01
12	0.110000E-01	0.260165E-02	0.107843E-01	0.425643E-02	0.164326E-01	0.484089E-01		0.115167E-01	0.110000E-01
13	0.120000E-01	0.235016E-02	0.312093E-02	0.418270E-02	0.228042E-01	0.460694E-01		0.110663E-01	0.120000E-01
14	0.130000E-01	0.308422E-02	0.751410E-02	0.162382E-01	0.367007E-01	0.437547E-01		0.124435E-01	0.130000E-01
15	0.140000E-01	0.376022E-02	0.156626E-01	0.126666E-01	0.339872E-01	0.481465E-01		0.147109E-01	0.140000E-01
16	0.150000E-01	0.139323E-01	0.166828E-01	0.134088E-01	0.207504E-01	0.449594E-01		0.143851E-01	0.150000E-01
17	0.160000E-01	0.110843E-01	0.191162E-01	0.325988E-01	0.200622E-01	0.344916E-01		0.163487E-01	0.160000E-01
18	0.170000E-01	0.849628E-02	0.703779E-02	0.347311E-01	0.265648E-01	0.185341E-01		0.148737E-01	0.170000E-01
19	0.180000E-01	0.623905E-02	0.226361E-02	0.475054E-01	0.209945E-01	0.207068E-01		0.155680E-01	0.180000E-01
20	0.190000E-01	0.282120E-02	0.356886E-03	0.271037E-01	0.274235E-01	0.220826E-01		0.160547E-01	0.190000E-01
21	0.200000E-01	0.845229E-04	0.293247E-03	0.197624E-01	0.259688E-01	0.971281E-02		0.157980E-01	0.200000E-01
22	0.210000E-01	0.227595E-03	0.868475E-03	0.192388E-01	0.260630E-01	0.725921E-02		0.171849E-01	0.210000E-01



OUTILS NUMERIQUES POUR L'INGENIEUR I MATLAB



FIN

Rappel:

help, doc et google sont vos amis