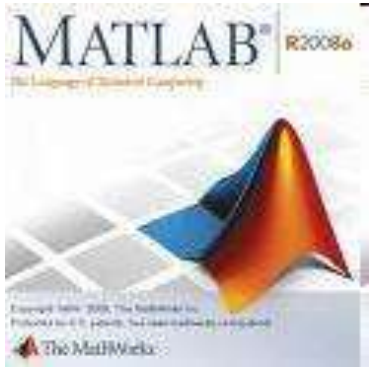
UE optionnelle

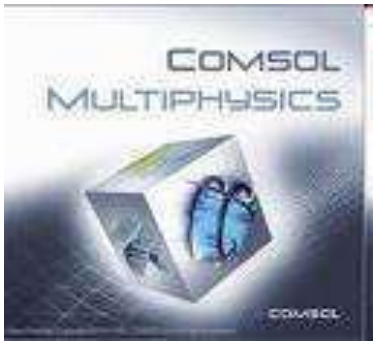
Atelier Logiciel

MSM24 – S1

Master SdI 1ère année



Cours du 14 / 09 / 2010



Fatiha Bouchelaghem (MCF) fatiha.bouchelaghem@upmc.fr

Objectifs :

- Initiation au langage Matlab (calcul et éléments de programmation);
- Modélisation par Eléments Finis – interface Comsol Multiphysics/Matlab;
- Application à la mécanique des structures.

8 h Cours + 24 h TP

Au programme :

Matlab

– ***« Matlab = hyper-calculatrice »***

Affectation de variables (scalaires/vecteurs/matrices), calcul matriciel, résolution de systèmes linéaires ou non, EDO, analyse et tracé de fonctions etc…

– ***« Matlab : environnement de développement »***

Programmation – fichiers *script* et *function* (« M-files ») définis par l'utilisateur: variables, boucles, tests, etc…

– ***« Matlab : modélisation, discrétisation par différence finies ou éléments finis »***

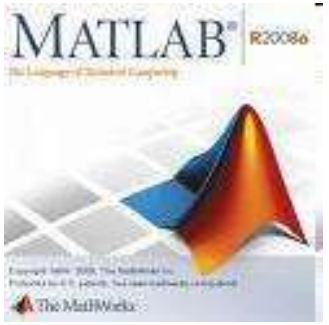
Géométrie, maillage, conditions aux limites etc.

Comsol

Modélisation éléments finis 1D, 2D et 3D, interface Matlab

MATLAB

1. Introduction :



MATLAB = Matrix **Lab**oratory

console d'exécution (shell)

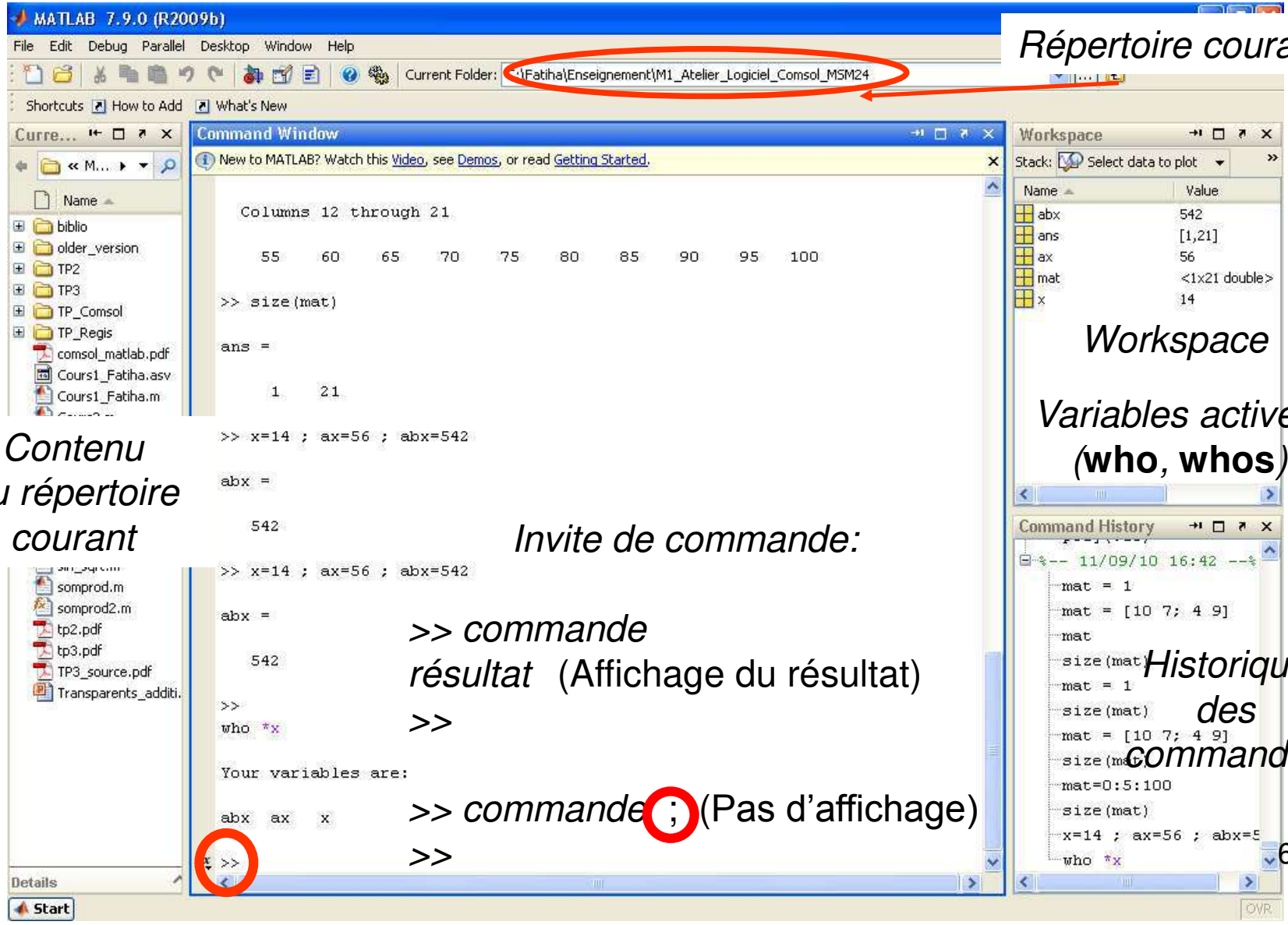
Logiciel commercial de calcul numérique matriciel, de visualisation et de programmation (société *The Mathworks*)

Eléments de données de base : matrices

Le langage MATLAB est interprété, i.e. chaque expression MATLAB est traduite en code machine au moment de son exécution. Un programme MATLAB (script, M-file) n'a donc pas besoin d'être compilé avant d'être exécuté.

2. Lancer MATLAB…

Interface Graphique Utilisateur (Graphical User Interface) :



Répertoire courant

Workspace

*Variables actives (**who**, **whos**)*

Contenu du répertoire courant

Invite de commande:

>> commande
résultat (Affichage du résultat)
>>

>> commande ; (Pas d'affichage)
>>

Historique des commandes

2. Lancer MATLAB…

Répertoires de travail :

Matlab permet d'ouvrir, de créer, de modifier, etc … des fichiers. Travail dans un répertoire par défaut qu'il est possible de modifier par la commande

>> **cd** *{chemin}*

Ou en lançant le « path browser » : **File**/ **Set Path** (ajout, déplacement, suppression de répertoires de travail)

Aide en ligne :

Menu déroulant **Help** ou icône  ou F1

>> **help** { *nom_commande* } : description + syntaxe de la commande

>> **helpwin** *{ nom_commande }*

>> **helpdesk**

>> **lookfor** *{ -all } mot-clé*

3. Notions de base

Généralités sur les variables :

- Le langage Matlab ne requiert aucune déclaration préalable de type de variable ou de dimension de tableau/vecteur ;

- Lorsque Matlab rencontre un nouveau nom de variable, il crée la variable correspondante et y associe l'espace de stockage approprié dans le « workspace » ;

- Si la variable existe déjà, Matlab change son contenu, et lui alloue un nouvel espace de stockage en cas de redimensionnement de tableau ;

- Les variables sont définies à l'aide d'expressions.

3. Notions de base

Généralités sur les variables (suite) :

- Nom de variable valide : lettre + nombre quelconque de lettres, chiffres ou _

Exemple : x_min, COEFF55a, tres_long_nom_de_variable

noms non valides : 86ab, coeff-555

>> **namelengthmax** % Taille maximale du nom de variable

- Attention ! Matlab est "case-sensitive", majuscules ≠ minuscules dans les noms de variables, fonctions ... : variable abc ≠ Abc

- Les noms de toutes les constantes et fonctions prédéfinies Matlab sont en minuscules

- Pour désigner un ensemble de variables, on peut utiliser * (remplace 0, 1 ou plusieurs caractères quelconques) ou ? (remplace 1 caractère quelconque)

Exemple: clear mat* % Détruit toutes les variables dont le nom commence par mat

3. Notions de base

Types de nombres (réels, complexes, entiers) :

Réels : de façon interne, Matlab calcule et stocke par défaut tous les nombres en virgule flottante « **double precision** » (précision finie de 16 chiffres décimaux)

Exemple : 3, 123, -99, 0.000145, -1.6341e20, 4.521e-5

Entiers : les fonctions **int8**, **int16**, **int32** et **int64** génèrent des variables entières signées stockées respectivement sur 8 bits, 16 bits, 32 bits ou 64 bits ; les valeurs réelles (double ou simple précision) sont arrondies au nombre le plus proche (équivalent de **round**)

- opérateurs ou fonctions mélangeant des opérandes/paramètres de types entier et réel, retournent un résultat de type entier !

- Certaines opérations mixant des données de type réel avec des données de type entier 64 bits ne sont pas autorisées.

Exemple : l'expression 13.3 * int64(12) génère une erreur.

3. Notions de base

Types de nombres (réels, complexes, entiers) :

Complexes : stockés sous forme de nombres réels double precision

Exemples : 4e-13 - 5.6i, -45+5*j

Partie réelle : **real**(*nb_complexe*)

Partie imaginaire : **imag**(*nb_complexe*)

Conjugué : **conj**(*nb_complexe*)

Module : **abs**(*nb_complexe*)

Argument **:** **angle**(*nb_complexe*)

3. Notions de base

Chaînes de caractères :

Dans Matlab, une chaîne est un vecteur-ligne contenant autant d'éléments que de caractères.

>> string = '*chaîne de caractères*'

Enregistre la *chaîne de caractères* sur la variable *string* (= vecteur-ligne). Si la chaîne contient une apostrophe, il faut la dédoubler (≠ fin de chaîne)

Exemple : section = 'Sciences de l''Ingenieur'

>> string(*i*:*j*) : partie de *string* comprise entre le *i*-ème et le *j*-ème caractère

>> string(i : **end** *)* ou *string(i :* **length***(string))* : fin de la chaîne à partir du caractère i

Exemple : section(15:23) ou section(15:end) retourne la chaîne « Ingénieur »

>> helpwin strfun : liste des fonctions relatives aux chaînes de caractères

4. Scalaires et constantes

Scalaire : matrice de 1 x 1 élément

Exemples : a = 12.34e-12, w = 2^3, r = sqrt(a)*5, s = pi*r^2 , w = -5+4i, z = w

Quelques constantes prédéfinies :

Constante	Description
pi	3.141592653589793 =
i ou j	Racine de -1 (sqrt(-1)) nombre imaginaire
Inf ou inf	Infini (exemple : 10/0)
NaN ou nan	Indéterminé (exemple : 0/0)
realmin	Plus petit réel positif (double precision)
realmax	Plus grand réel positif (double precision)
eps	Précision relative en virgule flottante
true ou 1	Vrai
false ou 0	Faux

5. Opérateurs de base

>> **helpwin ops** : liste des opérateurs et caractères spéciaux sous Matlab

Opérateurs arithmétiques :

Opérateur ou fonction	Description	Préséance
+ ou fonction **plus**	Addition	4
- ou fonction **minus**	Soustraction	4
* ou fonction **mtimes**	Multiplication	3
/ ou fonction **mrdivide**	Division	3
^ ou fonction **mpower**	Puissance	2
( )	Modifie l'ordre de préséance	1

S'appliquent à des scalaires, vecteurs ou matrices

5. Opérateurs de base

Opérateurs relationnels:

Les **opérateurs relationnels** permettent de faire des **tests numériques** en construisant des "*expressions logiques*", i.e des expressions retournant les valeurs vrai (1) ou faux (0)

Valable pour des vecteurs et des matrices

Opérateur ou fonction	Description
a == b ou a **eq** b ou **isequal**(a,b)	Test d'égalité – retourne 1 si a = b et 0 sinon
a ~= b ou a **ne** b	Test de différence – retourne 0 si a=b et 1 sinon
a < b ou a **lt** b	Test d'infériorité - 1 si inégalité vérifiée, 0 sinon
a > b ou a **gt** b	Test de supériorité - 1 si inégalité vérifiée, 0 sinon
a <= b	1 si inégalité vérifiée, 0 sinon
a >= b	1 si inégalité vérifiée, 0 sinon

Pour des vecteurs ou matrices : s'appliquent à tous les éléments et retournent donc également des vecteurs ou des matrices.

5. Opérateurs de base

Opérateurs logiques :

Les **opérateurs logiques** ont pour arguments des *expressions logiques* et retournent les valeurs logiques vrai (1) ou faux (0)

>> **helpwin relop** : liste des opérateurs relationnels et logiques

Opérateur	Description
~a **not** (a)	Négation logique – retourne 1 si a=0 et 0 si a=1
a & b, **and**(a,b)	ET logique (0 ET 0 => 0; 0 ET 1 => 0; 1 ET 1 => 1)
a | b, **or**(a,b)	OU logique (0 OU 0 => 0 ; 0 OU 1 => 1 ; 1 OU 1 => 1)
xor(a,b)	OU EXCLUSIF 0 OU EXCL 0 => 0 ; 0 OU EXCL 1 => 1 ; 1 OU EXCL 1 => 0)

ET et OU logiques : Si les expressions sont des matrices, retournent une matrice

6. Calcul matriciel

Affectation de vecteurs = matrice 1xn (ligne) ou nx1 (colonne)

- Vecteur ligne : *vec* = [*val1 val2 val3* ...] vec = [*val var expr* ...]

Exemple: v1=[1 -4 5], v2=[-3,sqrt(4)], v3=[v2 v1 -3]

- Vecteur colonne : *vec* = [*val* **;** *var* **;** *expr* ...**]**, vec **=** [*var val var val* ...] **'**

Transposition

vec=[*val1*
val2
...]

Exemple: v4=[-3;5;2*pi], v5=[11 ; v4], v6=[3 4 5 6]'

- L'opérateur **:** permet de construire des séries linéaires (vecteurs ligne) :

vec=[*début*{:*pas*}:*fin*] % si pas non spécifié = 1

6. Calcul matriciel

Addressage de vecteurs :

- *vec*(*i*) : ième élément du vecteur ligne ou colonne vec
- *vec*(*i*{:*p*}:*j*) : adressage des éléments d'indices *i* à *j* du vecteur ligne ou colonne *vec* avec un pas de "1" ou de "*p*" si spécifié
- *vec*([*i j k*:*l*]) : adressage des éléments i,j et k à l

>> **length**(*vec*) % dimension (= nombre d'éléments) du vecteur *vec*

Autres :

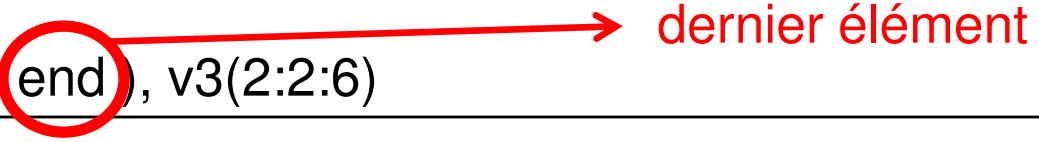
>> **norm**(*vec*) % norme du vecteur *vec*

>> **mean**(vec) % moyenne arithmétique

>> **dot**(*vec1,vec2*) % produit scalaire

>> **cross**(*vec1,vec2*) % produit vectoriel

Exemples : v4(6 : end), v3(2:2:6)



6. Calcul matriciel

Affectation de matrices :

Pour Matlab, une matrice est un tableau à 2 dimensions de NxM éléments (N lignes et M colonnes) → affectation [ ] et adressage ( ) généralisent à 2 dimensions de ce qui a été vu pour les vecteurs

mat=[*v11 v12 ... v1m* ;
v21 v22 ... v2m ;
... ... ;
vn1 vn2 ... vnm]

% crée une matrice n lignes x m colonnes

Exemples :

m1=[-2:0 ; 4 sqrt(9) 3]

v1=1:3:7 et v2=9:-1:7,m2=[v2;v1]

6. Calcul matriciel

Adressage de matrices :

- *mat*(*i*,*j*) % élément (i,j) de *mat*
- *mat*(*i*:*j*,*k*:*m*) % sous-matrice de *mat* (lignes i à j, colonnes k à m)
- *mat*(*i*,:) % ligne *i*
- *mat*(*i*:*j*,:) % lignes *i* à *j*
- *mat*(:,*k*) % colonne *k*
- *mat*(:,*k*:*m*) % colonnes *k* à *m*

>> **numel**(*mat*) % nombre d'éléments de *mat*

>> **size**(mat) % nombre de lignes x nombre de colonnes

>> **rank**(*mat*) % rang (nb de lignes ou colonnes linéairement indépendants)

6. Calcul matriciel

Opérateurs matriciels :

Concaténation	[ ]
Séparateur de ligne	;
Séparateur de colonne	
Transposée	**transpose**(M)=M'
Addition, soustraction	A **+** B, A **–** B (! dim(A) = dim(B); ou bien A ou B est un scalaire)
Multiplication par un scalaire α	α * M
Multiplication	A * B (nb colonnes A = nb lignes B) ou **mtimes**
Multiplication élément par élément	A **.*** B (! dim(A) = dim(B)) ou **times**
Inversion	**inv**(M) ou M^-1 (M = matrice carrée)
Déterminant	**det**(M)
Valeurs propres	**eig**(M)

7. Fonctions prédéfinies

Les fonctions Matlab sont implémentées soit au niveau du noyau Matlab (fonctions "built-ins") soit au niveau de M-files et packages (dont on peut voir et même changer le code).

Exemple:

>> **which** sin

>> **who -builtins** sin % indique que sin est une fonction built-in, alors que

>> **which** axis % montre dans quel M-file est implémentée la fonction axis.

Attention : les noms de fonction Matlab ne sont pas réservés et il est donc possible de les écraser. Pour restaurer la fonction originale : **clear** *nom_fonction*

>> **helpwin elfun** % liste des fonctions mathématiques *élé*mentaires

>> **helpwin specfun** % liste des fonctions mathématiques avancées (*spéc*ialisées)

7. Fonctions prédéfinies

Principales fonctions mathématiques disponibles dans Matlab :

Fonctions courantes	
cosinus, sinus, tangente , cotangente de x ; x en radians	cos(x), sin(x), tan(x), cot(x)
$\cos^{-1}$, $\sin^{-1}$, $\tan^{-1}$ de x ; x en radians	acos(x), asin(x), atan(x), atan2(x)
Cosinus, sinus, tangente hyperboliques de x	cosh(x), sinh(x), tanh(x)
Logarithme en base *e* et exponentielle	log(x), exp(x)
Logarithme en base *10*	log10(x)
Puissance x^a, racine carrée et racine nème	power(x,a), sqrt(x), *x*^(1/*n*)
Valeur absolue	abs(x)
Arrondis : entier le plus proche, supérieur, inférieur	round(x), ceil(x), floor(x)
Factorielle de n	factorial(*n*)
Fonction *var1* "modulo" *var2*	mod(*var1*,*var2*)

- Utilisées sur des vecteurs ou matrices, les fonctions sont appliquées à tous les éléments et retournent donc des vecteurs ou matrices
- Fonctions trigonométriques : angles en [radians].

Analyse de fonctions

Extrema : maxima et minima d'une fonction :

max(V) et min(V) + position de l'élément dans le vecteur V

>>[*fmin*, *posmin*]=**min(***V***)**;

>>[*fmax*, *posmax*]=**max(***V***)**;

Moyenne d'une fonction=moyenne de l'ensemble des valeurs du vecteur :

>>**mean(***V***)**;

Dérivée : % Approximation

>>**diff(***X***)** *X* : vecteur de valeurs discrètes **dim**(X)-1=dim(diff(X))

Intégration numérique : % Approximation

En TP…

>> **quad(***X,x1,x2***)** *X :* vecteur de valeurs discrètes
x1, x2 : bornes d'intégration

8. Graphiques

help graph2d % liste des fonctions graphiques 2D

Les tracés s'effectuent dans une fenêtre graphique (**figure**, **figure(*n*)** pour la n^{ème} figure)

Graphiques 1D :

>> **plot**(*x1*, *y1* {,*linespec*} {, *x2*, *y2* {,*linespec*} ...})

>> **plot**(*vec*)

>> **plot**(*mat*)

>> **plot**(*var1*,*var2* ...)

>> **fplot** % function plot

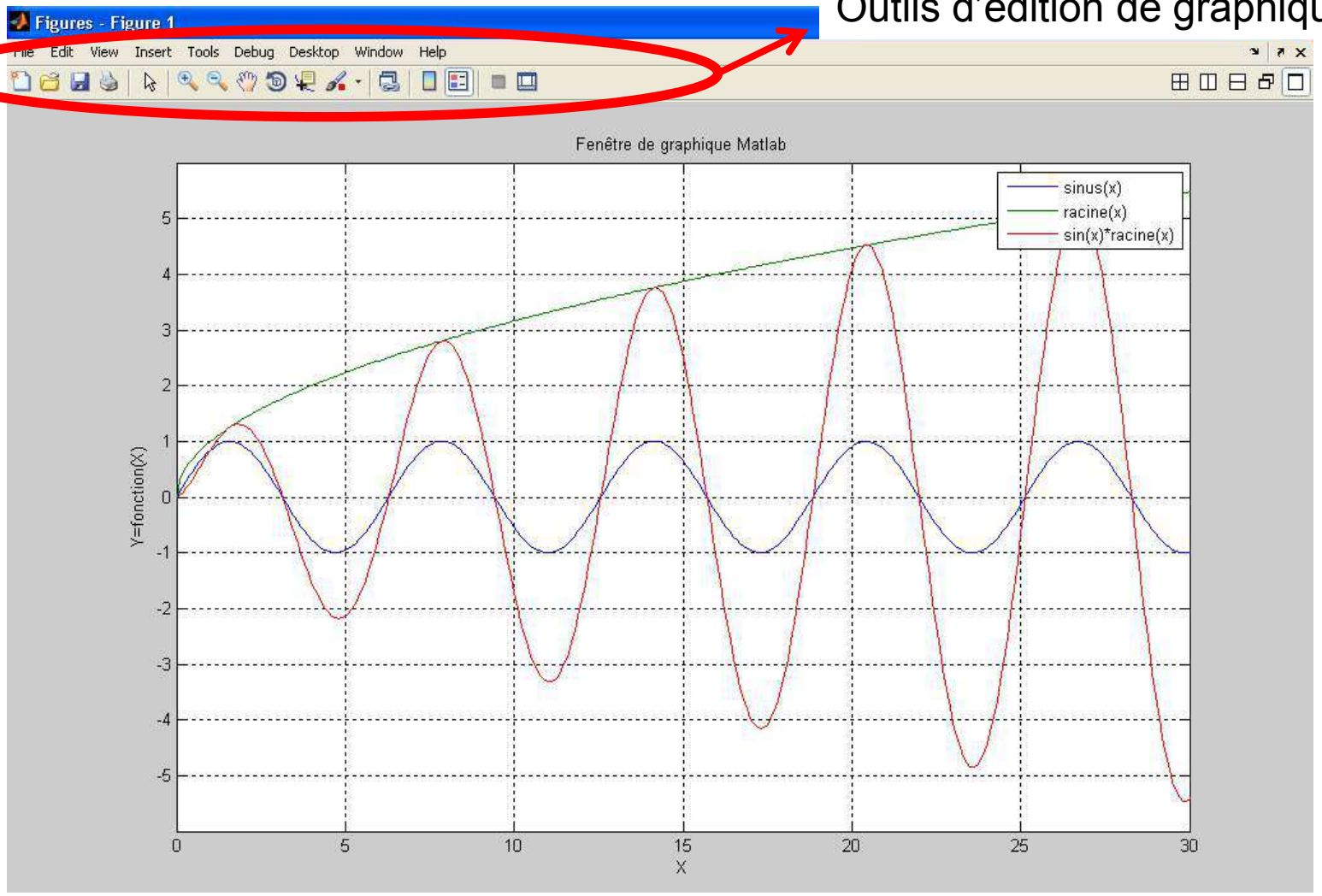
les arguments de la fonction **plot** sont des vecteurs. Il s'agit donc du tracé d'une fonction discrétisée (échantillonnée)

Sauvegarder une figure en tant qu'objet : **commande saveas**

8. Graphiques

Exemple 1: *Traité dans le workspace*

Outils d'édition de graphique

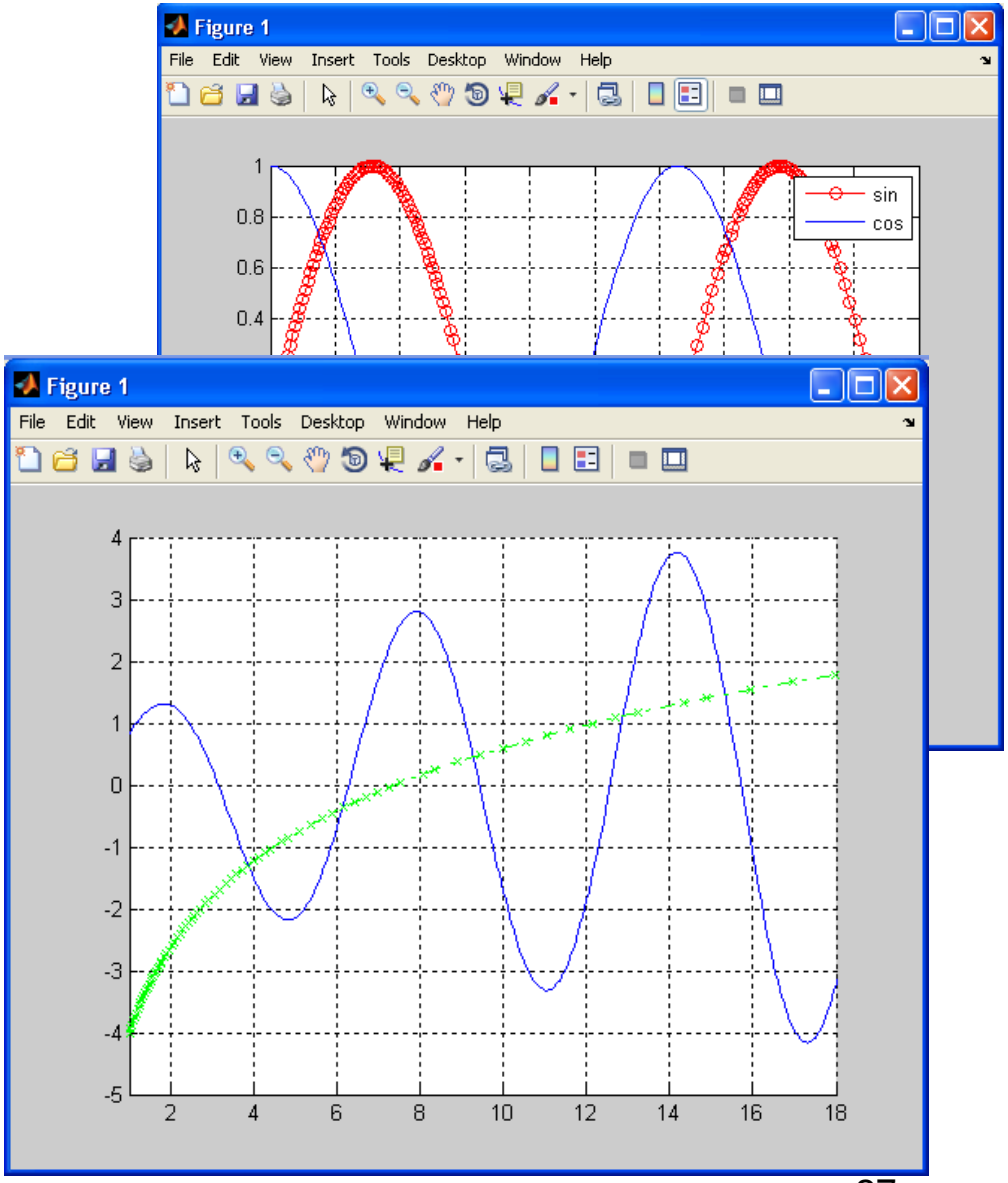


8. Graphiques

Tracé de fonctions : fplot

- Fonction prédéfinie (built-in)
- Fonction quelconque
- Définition puis tracé d'une fonction utilisateur

Exemple 2: *Traité dans le workspace*



8. Graphiques

Autres :

>> **refresh** ou **refresh**(n) % redessine les graphiques,

>> **clf** ou **clf**(n) % clear figure

>> **close** ou **close**(n) % referme la fenêtre graphique

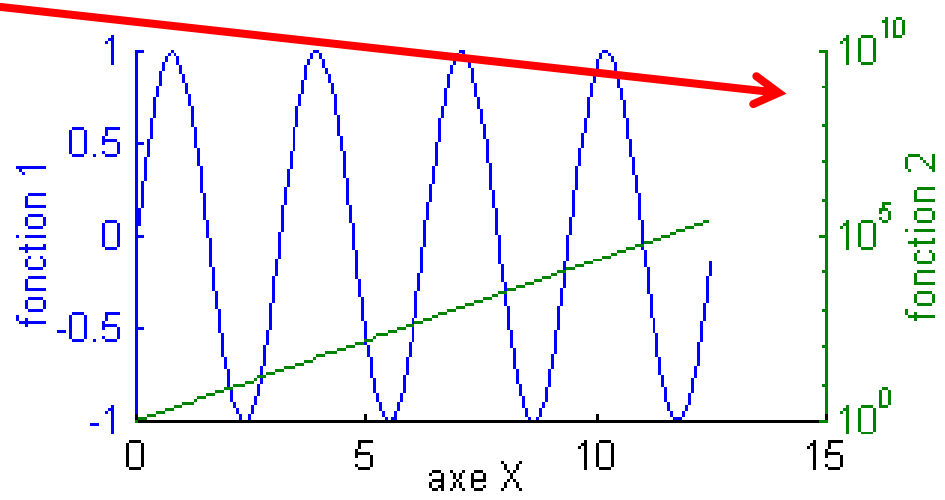
>> **close all** % referme toutes les fenêtres graphiques

>> **plotyy**(*x1*, *y1*, *x2*, *y2* {,'*type1*' {,'*type2*'}}) % fonction pour des graphiques à 2 axes Y

>> **semilogx**, **semilogy** et **loglog** % pour des graphiques avec axes logarithmiques

Exemple 3:

Traité dans le workspace



8. Graphiques

Graphiques 2D : **imagesc, contour, contour3, contourf, surface …**

Exemple 4:

```
Z = rand(10,20);          % matrice de nombre aléatoires pris sur [0;1]

contour(Z);               % isolignes de Z, Z interprétée comme des élévations /
                            plan (x,y)
contourf(Z);              % isolignes avec remplissage
contour(Z,n);             % isolignes de Z, avec n contours


X=[1:0.5:10.5];
Y=[0.1:0.1:1];
contour(X,Y,Z);           % isolignes de Z, avec les limites des axes X et Y spécifiées
contour(X,Y,Z,n);         % length(X) = m, length(Y) = p et [p,m] = size(Z)

imagesc(Z);               % représentation de Z sous forme d'image, chaque élément
                          % de Z représente une surface rectangulaire
```

8. Graphiques

Graphiques 3D :

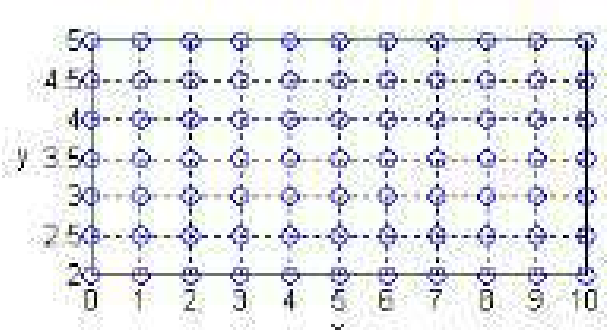
help graph3d % fonctions relatives aux graphiques 3D

Exemple 5 : Détermination et visualisation, par un graphique 3D, de la surface
z = fct(x,y) = sin(x/3)*cos(y/3) en "échantillonnant" cette fonction selon une grille x-y

de dimension de maille 1 en x et 0.5 en y; $0 \leq x \leq 10$ et $2 \leq y \leq 5$.

Pour représenter graphiquement cette surface, il s'agit au préalable de calculer une matrice z dont les éléments sont les "altitudes" z correspondant aux points de la grille.

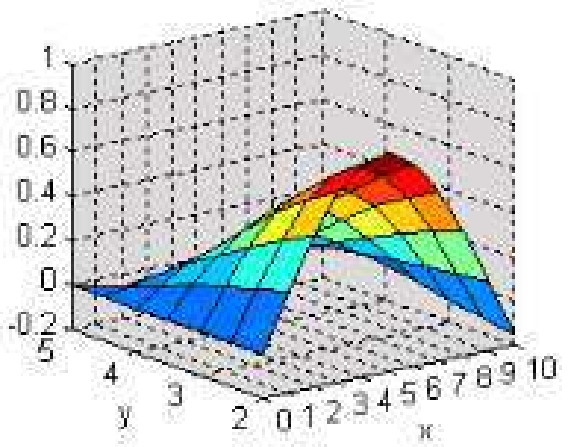
```
    x  0 1 2 3 4 5 6 7 8 9 10
y     ------------------------
2     |                      |
2.5   |                      |
3     |       matrice  z     |
3.5   |        (7 x 11)      |
4     |                      |
4.5   |                      |
5     |                      |
      ------------------------
```



8. Graphiques

- Solution 1 : Méthode classique ne faisant pas intervenir les capacités de vectorisation de MATLAB; 2 boucles *for* imbriquées qui parcourent tous les points de la grille pour calculer les éléments de z

```
x=0:1:10; y=2:0.5:5;                % domaine des valeurs de la grille en X et Y
for k=1:length(x)                   % parcours de la grille, colonne après colonne
for l=1:length(y)                   % parcours de la grille, ligne après ligne
z1(l,k)= sin(x(k)/3)*cos(y(l)/3);   % calcul de z, élément par élément
end
end
surf(x,y,z1);
```



- Solution 2 : **meshgrid**

```
x=0:1:10; y=2:0.5:5;
[Xm,Ym]=meshgrid(x,y);
z2=sin(Xm/3).*cos(Ym/3);  % calcul de z en une seule instruction vectorisée
                          % notez bien que l'on fait produit .* élément par élément et
                          % non pas * (vectoriel)
surf(x,y,z2);             % visualisation de la surface
```

8. Graphiques

- **meshgrid**

sur la base des 2 vecteurs x et y (en ligne ou en colonne) décrivant le domaine des valeurs de la grille en x et y, la fonction meshgrid génère 2 matrices Xm et Ym

Xm est constituée par recopie, en length(y) lignes, du vecteur x
Ym est constituée par recopie, en length(x) colonnes, du vecteur y

dim(Xm) = dim(Ym) = length(y) lignes * length(x) colonnes

```
x = 0  1  2  3  4  5  6  7  8  9  10 (11 él.)
```

```
    -------------(7x11)---------------
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
Xm=| 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
   | 0  1  2  3  4  5  6  7  8  9  10 |
    ----------------------------------
```

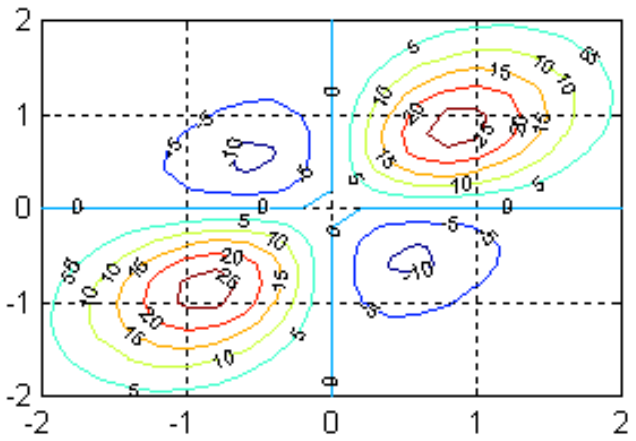
```
y = 2  2.5  3  3.5  4  4.5  5  (7 élém.)
```

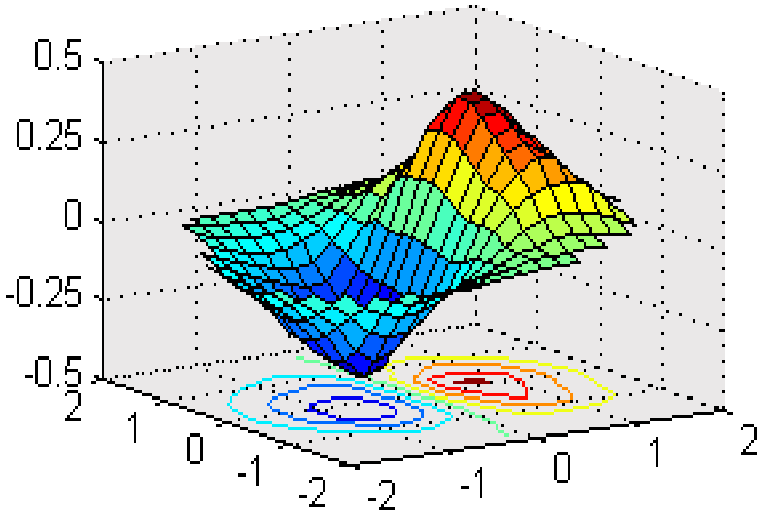
```
    ------------------------(7x11)------------------------
   | 2    2    2    2    2    2    2    2    2    2    2   |
   | 2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5 |
   | 3    3    3    3    3    3    3    3    3    3    3   |
Ym=| 3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5  3.5 |
   | 4    4    4    4    4    4    4    4    4    4    4   |
   | 4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5 |
   | 5    5    5    5    5    5    5    5    5    5    5   |
    ---------------------------------------------------------
```

z=fct(x,y) peut être calculée par une seule instruction Matlab vectorisée (donc sans boucle *for*) en faisant usage des opérateurs "terme à terme"

8. Graphiques

Graphiques 3D : **mesh**, **griddata, plot3, contour3 …**





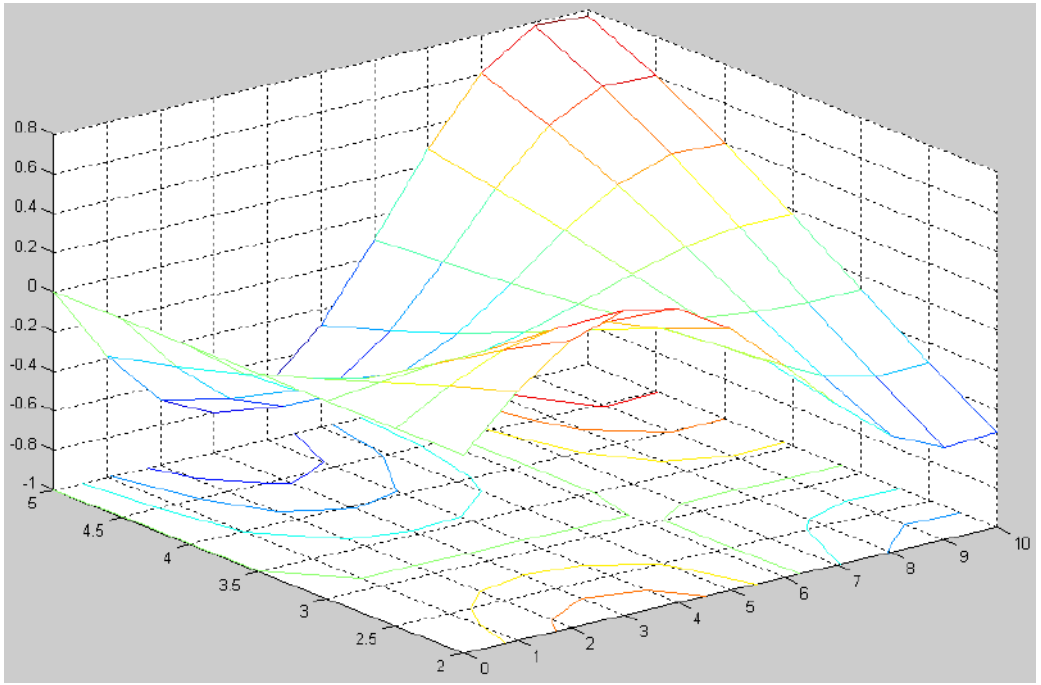
Exemple 6:

```
x= 4*rand(1,50) -2;                          % vecteur de 50 val. aléatoires entre -2 et +2
y= 4*rand(1,50) -2;
z = sin(x/3).*cos(y/3);
xi= -2:0.2:2; yi= xi';                       % grille régulière en x et y
[XI,YI,ZI]= griddata(x,y,z,xi,yi,'cubic');   % interpolation sur grille régulière
surfc(XI,YI,ZI);                             % affich. surf. interpolée et contours
% pour superposer sur ce graphique le semis de points irréguliers :
hold('on');
plot3(x,y,z,'.');
```

8. Graphiques

Graphiques 3D: Exemple 7:

```
x=0:1:10; y=2:0.5:5;
vx = sin(x./2);          % vx : 1 ligne x 11 colonnes
vy = cos(y./2);          % vy : 1 ligne x 7 colonnes
z = vy'*vx;              % x correspond aux colonnes et y correspond aux lignes
                         % z a autant de lignes que vy et autant de colonnes que vx
mesh(x,y,z);                    length(x) = n, length(y) = m et [m,n] = size(Z)
meshc(x,y,z);            % avec isocontours sous le tracé
meshz(x,y,z);
```



9. Racines d'un polynôme

On recherche les racine du polynôme du 5ème degré suivant:

$$P(x) = x^5 + 3x^4 - 8x^3 + 12x^2 - x + 4 = 0$$

Dans Matlab, un polynôme se définit par le vecteur de ses coefficients pris par ordre décroissant :

>> polynom = [*1 3 -8 12 -1 4*]; % vecteur qui définit le polynôme P(x)

Détermination des racines :

>> *Vec*=**roots**(polynom)

>> **poly**(Vec) % convertit les racines données dans le vecteur Vec, en polynôme

10. Equations non linéaires

fonction **fzero**('*fonction*',*x0*) (fstandard): calcule, par approximations successives, en partant d'une valeur donnée x = x0, les racines d'une fonction non linéaire y = f(x), c'est-à-dire les valeurs x1, x2, x3 … pour lesquelles f(x) = 0

fsolve (toolbox Optimisation):

Exemple : Soit la fonction de 3e degré : y = - 0.5*x^3 - x^2 + 15*x – 8

- Définition de l'équation sous forme de fonction Matlab; M-file 'fct_deg3.m' :

```
function [Y]=fct_deg3(X)
Y = - 0.5 * X.^3 - X.^2 + 15*X - 8
return
```

- Tracé de la fonction sur -20 ≤ x ≤ 20 : fplot('fct_deg3(x)',[-20 20])

- Zoom pour rechercher les zéros : fplot('fct_deg3(x)',[0 5]) ; grid('on')

- x0 = 5, recherche de la première solution : x1=fzero('fct_deg3',5) donne x1 = 4.2158

- x0 = 0, recherche de la seconde solution : x2=fzero('fct_deg3',0) donne x2 = 0.5601

11. Programmation

M-file: fichier au format texte (donc lisible et éditable) qui contient des instructions Matlab et qui porte l'extension ***.m**.

- Deux types de M-files : les scripts (ou programmes Matlab) et les fonctions.

• Les scripts n'ont pas d'arguments d'entrée/sortie, travaillent dans le workspace, et toutes les variables créées/modifiées lors de l'exécution d'un script sont ensuite visibles dans le workspace (= variables globales, qui peuvent également être déclarées par **global** *var*).

• Les fonctions, quant à elles, n'interagissent avec le workspace qu'à travers leurs variables d'entrée/sortie, les autres variables manipulées restant internes (locales) à ces fonctions.

Un fichier *.m peut être édité avec tout éditeur de texte

Editeur Matlab : menu File / New / … M-file ou File / Open, edit M-file, open *M-file* ou icône 

Intérêt : débugger intégré, coloriage syntaxique, indentation

11. Programmation

helpwin lang % liste des fonctions orientées programmation

Fichier script

Suite de commandes Matlab sauvegardées dans un M-file (*.m).

Les variables utilisées dans le script sont disponibles à l'invite MATLAB.
Habituellement, on utilise les fichiers script pour :

- programmer des opérations répétitives
- initialiser le système ;
- déclarer les variables ;
- effectuer les opérations algébriques ;
- appeler les fonctions ;
- tracer les figures.

Fichier function

Fichier *.m qui permet d'enregistrer une succession d'opérations en spécifiant des arguments d'entrée et des arguments de sortie.

function [a,b] = ma_fonction (x,y)

Les variables « internes » ou locales ne sont pas disponibles à l'invite MATLAB.
Habituellement, on utilise les fichiers function pour :

- programmer des opérations répétitives en limitant le nombre de variables dans l'invite ;
- structurer le programme.
- étendre des fonctions Matlab

Rq : il faut « *appeler* » une function

Le fichier *.m doit porter le même nom que la fonction

11. Programmation

Structures de contrôle: boucles, exécutions conditionnelles (tests)

Travail en "format libre" (<espace> et <tab> non significatifs) : penser à bien "indenter" le code lorsqu'on utilise des structures de contrôle, afin de faciliter la lisibilité et la maintenance du programme.

Boucles

if-elseif-else

```
if CONDITION 1
        Action 1;
elseif CONDITION 2
        Action 2;
else
        Action3;
end;
```

for

```
for
i=val_init:incr:val_fin
        Action 1;
        Action 2;
              .
              .
        ActionN;
end;
```

while

```
while CONDITION
        Action 1;
        Action 2;
              .
              .
        ActionN;
end;
```

switch

```
switch CAS
        case {CAS1}
              Action 1;
        case {CAS2}
              Action 2;
        otherwise
              Action3;
end;
```

blocs else et else if facultatifs

bloc otherwise facultatif

>> **continue** : sauter les instructions dans une boucle et passer à la prochaine itération

>> **break** : sortie prématurée de boucle

11. Programmation

Structures de contrôle:

Dans Matlab, il faut "penser instructions matricielles" (on dit aussi parfois "vectoriser" son algorithme) avant d'utiliser systématiquement ces structures de contrôle qui, parce que Matlab est un langage interprété, sont moins rapides que les opérateurs et fonctions matriciels de base.

Exemple : l'instruction y=sqrt(1:100000); est beaucoup plus efficace que la boucle

```
for n=1:100000,
y(n)=sqrt(n);
end
```

(bien que, dans les 2 cas, ce soit un vecteur de 100 000 éléments qui soit créé, contenant les valeurs de la racine de 1 jusqu'à la racine de 100 000).

Temps écoulé = 0.0051 s (allocation vectorielle) vs 36.5744 s (boucle for)

11. Programmation

Interactions avec l'utilisateur :

>> **disp**(*variable*) % affiche le contenu de la variable spécifiée (et pas son nom)
>> **disp**('*chaîne*') % affiche le contenu de la chaîne de caractères spécifiée

Exemple :

```
M=[1 2;3 5] ;
disp('La matrice M vaut :')        % affiche "La matrice M vaut :" sur une ligne
disp(M)                            % puis valeurs de la matrice M sur les lignes suivantes
```

>> **fprintf**('*format*',*variable(s)*) % affiche de façon formatée les variables spécifiées

```
v=444; t='chaîne de car.';
fprintf('variable v= %6.1f et variable t= %s \n',v,t) % affiche sur une seule ligne :
% " variable v= 444.0 et variable t= chaîne de car.   "
```

>> **warning**({'*id*',} '*message*' {,*variable(s)*...}) % avertissement (sans interruption du programme)

>> **error**('*message*' {,*variable(s)*...}) % interruption de l'exécution de la fonction ou du script, ne pas sortir avec **exit** ou **quit**, qui termine aussi Matlab

11. Programmation

Interactions avec l'utilisateur :

variable=**input**('*prompt*') ; % Affiche le prompt et attend la réponse puis ↵
% nombre, un vecteur, une matrice, voire toute expression valide
% qui est alors interprétée et affectée à *variable*

variable_chaîne=**input**('*prompt*', 's') ; % 's' : string, saisie interactive de texte

choix=**menu**('*Titre*','*bouton1*','*bouton2*',...) % menu de choix entre plusieurs options

echo on | off | on all | off all % Active (on) ou désative (off) l'affichage de toutes les
% commandes exécutées par les scripts, et par les fonctions appelées
% depuis les scripts (all)

pause(*secondes*) % se met en attente durant le nombre de *secondes* spécifié

pause % attend que l'utilisateur frappe n'importe quelle touche au clavier.

11. Programmation

Interactions avec l'utilisateur :

return ou **end** % Termine l'exécution de la fonction ou du script

nargin % A l'intérieur d'une fonction, retourne le nombre d'arguments d'entrée passés
% lors de l'appel à cette fonction. Permet par exemple de donner des valeurs
% par défaut aux paramètres d'entrée manquants.

nargout % A l'intérieur d'une fonction, retourne le nombre de variables de sortie
% auxquelles la fonction est affectée lors de l'appel.

Exemple : A l'intérieur d'une fonction-utilisateur mafonction :
- lorsqu'on l'appelle avec mafonction(...) : nargout vaudra 0
- lorsqu'on l'appelle avec *out1*=mafonction(...) : nargout vaudra 1
- lorsqu'on l'appelle avec [*out1 out2*]=mafonction(...) : nargout vaudra 2, etc...

inputname(*k*) % A l'intérieur d'une fonction, retourne le nom de variable du *k*-ème
% argument passé à la fonction

11. Programmation

Performances d'un programme (temps calcul) :

tic % fonction qui démarre un compteur de temps

instructions MATLAB...

elapsed_time = **toc** % arrête le compteur de temps en retournant le temps écoulé en s

Exemple : tic; A=rand(1000,1000); B=inv(A); dt = toc

Sur le net …

Site de la société The MathWorks (éditrice de Matlab) :

http://www.mathworks.com

Page officielle de Matlab avec des vidéos et des cours :

http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

Page mathtools qui offre des programmes dans différents domaines:

http://www.mathtools.net/Fortran/index.html

Fonctions/scripts libres développés pour Matlab :

http://www.mathworks.com/matlabcentral/fileexchange/