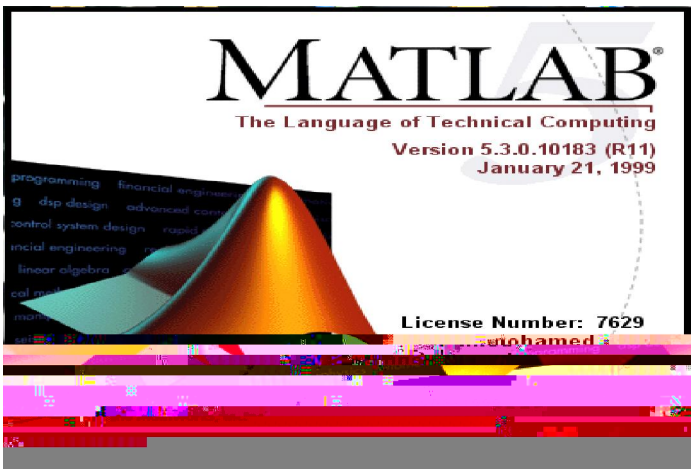
Direction **G**énérale des **E**tudes **T**echnologiques
Institut **S**upérieur des **E**tudes **T**echnologiques de **K**ebilli
Département **G**énie **E**lectrique

Exemples pour Commencer à Utiliser MATLAB

Pour les groupes Génie Electrique et Génie mécanique



BEN BOUBAKER Mohamed
©2008/2009

Notes introductives à Matlab

Le logiciel Matlab consiste en un langage interprété qui s'exécute dans une fenêtre dite d'exécution. L'intérêt de Matlab tient, d'une part, à sa simplicité d'utilisation : pas de compilation, déclaration directe des variables utilisées et, d'autre part, à sa richesse fonctionnelle : arithmétique matriciel et nombreuses fonctions de haut niveau dans de nombreux domaines (analyse numérique, graphique, ...). La programmation sous Matlab consiste à écrire des scripts de commandes Matlab, exécutables dans la fenêtre d'exécution. Et grâce aux diverses Toolboxes spécialisés (ensemble de fonctions Matlab), Matlab s'enrichit au fur et à mesure.

1. Lancement de Matlab (Personnellement j'utilise Matlab 5.3)

Lors de son lancement (avec la commande Matlab par exemple) la fenêtre d'exécution s'ouvre. Il est alors possible d'exécuter différents types de commandes dans cette fenêtre, par exemple :

```
help
HELP topics: (le nombre de function diffère d'une version de Matlab à une autre)
toolbox\local       - Local function library.
matlab\datafun      - Data analysis and Fourier transform functions.
matlab\elfun        - Elementary math functions.
matlab\elmat        - Elementary matrices and matrix manipulation.
matlab\funfun       - Function functions - nonlinear numerical methods.
matlab\general      - General purpose commands.
matlab\color        - Color control and lighting model functions.
matlab\graphics     - General purpose graphics functions.
matlab\iofun        - Low-level file I/O functions.
matlab\lang         - Language constructs and debugging.
matlab\matfun       - Matrix functions - numerical linear algebra.
matlab\ops          - Operators and special characters.
matlab\plotxy       - Two dimensional graphics.
matlab\plotxyz      - Three dimensional graphics.
matlab\polyfun      - Polynomial and interpolation functions.
matlab\sounds       - Sound processing functions.
matlab\sparfun      - Sparse matrix functions.
matlab\specfun      - Specialized math functions.
matlab\specmat      - Specialized matrices.
matlab\strfun       - Character string functions.
matlab\dde          - DDE Toolbox.
matlab\demos        - The MATLAB Expo and other demonstrations.
toolbox\wintools    - GUI tools for MATLAB for MS Windows.
For more help on directory/topic, type "help topic".
```

La commande suivante permet d'affiner l'aide sur les fonctions mathématiques élémentaires :

```
help elfun
 Elementary math functions.
 Trigonometric.
   sin        - Sine.
   sinh       - Hyperbolic sine.
   asin       - Inverse sine.
   asinh      - Inverse hyperbolic sine.
   cos        - Cosine.
   cosh       - Hyperbolic cosine.
   acos       - Inverse cosine.
   acosh      - Inverse hyperbolic cosine.
   tan        - Tangent.
   tanh       - Hyperbolic tangent.
   atan       - Inverse tangent.
   atan2      - Four quadrant inverse tangent.
   atanh      - Inverse hyperbolic tangent.
   sec        - Secant.
   sech       - Hyperbolic secant.
   asec       - Inverse secant.
   asech      - Inverse hyperbolic secant.
   csc        - Cosecant.
   csch       - Hyperbolic cosecant.
   acsc       - Inverse cosecant.
   acsch      - Inverse hyperbolic cosecant.
   cot        - Cotangent.
   coth       - Hyperbolic cotangent.
   acot       - Inverse cotangent.
   acoth      - Inverse hyperbolic cotangent.
 Exponential.
   exp        - Exponential.
   log        - Natural logarithm.
   log10      - Common logarithm.
   sqrt       - Square root.
 Complex.
   abs        - Absolute value.
   angle      - Phase angle.
   conj       - Complex conjugate.
   imag       - Complex imaginary part.
   real       - Complex real part.
 Numeric.
   fix        - Round towards zero.
   floor      - Round towards minus infinity.
   ceil       - Round towards plus infinity.
```

```
   round      - Round towards nearest integer.
   rem        - Remainder after division.
   sign       - Signum function.
```

permettant ainsi de voir toutes les fonctions mathématiques élémentaires dont dispose Matlab. On peut
maintenant préciser la recherche si l'on veut avoir une idée plus précise de la fonction log par exemple :

```
help log
 LOG Natural logarithm.
  LOG(X) is the natural logarithm of the elements of X.
   Complex results are produced if X is not positive.
  See also LOG10, EXP, LOGM.
```

♦ Quelques commandes d'environnement :

path : permet de savoir quels sont les dossiers que Matlab référence et de spécifier de nouveaux dossiers où se trouvent des ressources personnelles, par exemple :

```
path=(path,'~/monapplication/mesfichiersmatlab')
```

cd : positionne Matlab dans un dossier, par exemple :

```
cd ~/monapplication/mesfichiersmatlab
```

dir ou **ls** : permet de faire la liste des objets du dossier courant.

clear all : efface tous les objets en mémoire

clf : détruit les figures

Si monscript.m est un script Matlab que vous avez écrit et qui est accessible (soit via le path ou un commande cd) pour l'exécuter, il suffira de saisir la commande :

```
monscript
```

Par défaut, Matlab inscrit le résultat à la suite de la commande. Si l'on ne désire pas voir le résultat d'une commande il suffit de terminer cette commande par **;**. Pour certaine commande (ls,help,...) celà ne change rien.

2. Les variables sous Matlab

Matlab gère de façon automatique : les nombres entiers, réels, complexes de façon indifférente, les chaines de caractères ainsi que les tableaux de nombre. En aucun cas, il n'est utile de déclarer le type de la variable que l'on manipule, y compris les tableaux. Ainsi les instructions suivantes, déclarent les variables lors de leur affectation :

```
>>a=1
a =
    1
>>b=1.01
b =
   1.0100
>>X=1.0e+05
X =
     100000
>>nom='    mon nom'
nom =
   mon nom   4
>>c=1+2i
c =
```

```
   1.0000 + 2.0000i
```

la constante i est le nombre imaginaire prédéclaré, de même que certaines constantes (e,pi,...)

On déclare un vecteur colonne de la façon suivante :

```
>>u=[1;3;-1]
u =
     1
     3
    -1
```

un vecteur ligne de la façon suivante :

```
>>v=[1,3,-1]
v =
     1     3    -1
```

et une matrice d'ordre 3x2 :

```
>>A=[1,2 ; -1, 3; 4, 0]
A =
     1     2
    -1     3
     4     0
```

la ',' sert à séparer les éléments d'une ligne et ';' les éléments colonnes. En fait, on peut remplacer la ',' par un espace , ce qui améliore la lisibilité :

```
>vb=[1 3 -1]
vb =
     1     3    -1
```

Pour spécifier un élément d'un vecteur, d'une matrice, on utilise la syntaxe suivante :

```
>>u(2)
ans =
     3
>>v(3)
ans =
    -1
>>A(3,2)
ans =
     0
```

L'utilisation d'indice hors limite provoque une erreur, comme le montre cet exemple:

```
>>A(3,3)
???  Index exceeds matrix dimensions.
```

On peut utiliser des raccourcis bien utiles et plus efficaces pour remplir des vecteurs ou des tableaux . Exemples :

```
>u1=1:10        (incrémentation automatique de 1 à 10 avec pas de 1)
u1 =5
     1     2     3     4     5     6     7     8     9    10
>>v1=1:2:10          (incrémentation automatique de 1 à 10 avec pas de 2)
v1 =
     1     3     5     7     9
```

```
>>Id3=eye(3)   (matrice identité  d'ordre3)
Id3 =
    1     0     0
    0     1     0
    0     0     1
>>Un=ones(2)    (matrice constituée de 1 d'ordre 2)
Un =
    1     1
    1     1
>>Z=zeros(2,3)  (matrice nulle  d'ordre 2x3)
Z =
    0     0     0
    0     0     0
```

De même, il existe des syntaxes particulières permettant d'extraire des lignes, des colonnes de matrices :

```
>>A1=[11 12 13;21 22 23;31 32 33]
A1 =
   11    12    13
   21    22    23
   31    32    33

>>A1(:,1)     (colonne 1 de la matrice A1)
ans =
   11
   21
   31
>>A1(2,:)   (ligne 2 de la matrice A1)
ans =
   21    22    23
```

Une des difficultés que rencontre le débutant est le mauvais maniement des matrices du à une mauvaise connaissance des tailles des matrices. Pour vérifier ces tailles, on pourra utiliser la commande size:

```
>>size(A1)
ans =
    3     3
>>size(u)
ans =
    3     1
```

3. Opérations élémentaires sous Matlab

Les opérations sur les scalaires sont standards : addition +, soustraction -, multiplication *, division /, puissance ^.

La racine carrée s'obtient par la function sqrt.On dispose de toutes les fonctions usuelles sur les scalaires, faire help elfun pour plus de détails.

Remarque : les fonctions peuvent renvoyer des résultats complexes, exemples :

```
>>sqrt(-1)
ans =
      0 + 1.0000i
```

acos(2) (function arc cosinus, ici on a affaire au prolongement dans le plan complexe de cette fonction)

```
ans =
      0 + 1.3170i
```

En ce qui concerne les vecteurs et matrices ces opérateurs se prolongent au sens du calcul vectoriel et matriciel.

En particulier, il faut vérifier la compatibilité des tailles des variables.

Exemples :

```
>>u=[1 2 3]
u =
    1     2     3
>>v=[-1 1 1]
v =
   -1     1     1
>>w=u+v     (addition)
w =
    0     3     4
>>ut=u'       (transposition d'un vecteur ligne)
ut =
    1
    2
    3
>>ut2=[ut ut]  (concaténation de deux vecteurs colonnes donne une matrice 3x2)
ut2 =
    1     1
    2     2
    3     3
>>ps=v*ut     (Produit qui conduit au produit scalaire)
ps =
    4
>>M=ut*v      (Produit qui conduit à une matrice)
M =
   -1     1     1
   -2     2     2
   -3     3     3
>>L=M+2*eye(3)
L =
    1     1     1
   -2     4     2
   -3     3     5
```

```
>>Y=L\ut              (Résolution du système linéaire L.Y=ut)
Y =
   0.1667
   0.3333
   0.5000   7
>>E=u/L'          (Résolution du système linéaire E.L' =u)
E =
   0.1667    0.3333    0.5000
```

Si les matrices ne sont pas inversibles, un message d'erreur sera affiché après la division..
On peut effectuer des opérations tensorielles sur les vecteurs et matrices, par exemple effectuer le produit de deux vecteurs colonnes, composante par composante par l'adjonction d'un **.** à l'opérande *. Par exemple :

```
>>ut.*Y
ans =
   0.1667
   0.6667
   1.5000
>>ut.^Y
ans =
   1.0000
   1.2599
   1.7321
```

De même, Matlab autorise l'utilisation de toutes les fonctions scalaires dans un contexte vectoriel. Ainsi, si h est un vecteur de dimension n, sin(h) sera un vecteur de même dimension :

```
>>h=0:pi/4:pi
h =
        0    0.7854    1.5708    2.3562    3.1416
>>sin(h)
ans =
        0    0.7071    1.0000    0.7071    0.0000
```

Pour les multiples opérations sur les matrices (inverse, puissance, trace, déterminant, factorisation, ...) faire help elmat et help matfun.
Pour ce qui est des opérations sur les chaînes de caractères, ces dernières étant considérées comme des vecteurs ligne de caractères ascii, la concaténation de deux chaînes s'effectuera de la façon suivante :

```
>>c1='texte'
c1 =
texte
  >>c2=' et suite de texte'
c2 =
 et suite de texte
>>c3=[c1 c2]
c3 =
texte et suite de texte
```

pour les autres opérations voir help strfun.

4. Contrôle de flux

♦ Opérateurs booléens :

Avant de décrire la syntaxe du test sous Matlab, indiquons les principaux opérateurs de relation ainsi que les opérateurs booléens qu'utilisent Matlab.

< strictement inférieur à
<= inférieur ou égal à
> strictement supérieur à
>= supérieur ou égal à
== égal à
~= différent de
& et logique (and)
| ou logique (or)
~ non logique (not)

Le résultat d'un test est un booléen, qui sous Matlab, prend la valeur 1 pour vrai et 0 pour faux. Par exemple, on
a les résultats suivants :

```
r=1<2
r =
    1
r=~((1>2)|(0~=0))          (traduction Matlab de l'expression logique : non (1>2 ou 0≠0))
r =
    1
```

Il existe d'autres fonctions booléennes, par exemple xor, isfinite, isnan, isinf,..dont on trouvera la description en faisant help ops.

♦ Syntaxe du test (if)

```
if  expression booléenne                    if  expression booléenne if  expression booléenne
        instructions                                instructions instructions
    end                                       else elseif expression booléenne
     instructions instructions
  end else
instructions
end
```

♦ Syntaxe du branchement (switch)

```
switch expression (expression est un scalaire ou une chaîne de caractères)
case value1
            instructions (instructions effectuées si expression=value1)
case value2
            instructions
        ...
        otherwise
            instructions
end
```

♦ Syntaxe de boucle (while et for)

```
while expression  for indice=debut:pas:fin (si le pas n'est pas précisé, par défaut il vaut 1)
instructions instructions
end end
```

5. Utilisation des fonctions

La notion de fonction existe sous Matlab. Sa syntaxe est la suivante :
function [args1,args2,...] = nomfonction(arge1,arge2,...)
instructions
args1,args2,... sont les arguments de sortie de la fonction et peuvent être de n'importe quel type
arge1,arge2,... sont les arguments d'entrée de la fonction et peuvent être de n'importe que type
instructions est un bloc d'instructions quelconque devant affecter les arguments de sortie args1,args2,...
Lorsqu'il n'y a qu'un seul argument de sortie, on peut utiliser la syntaxe plus simple :
function args = nomfonction(arge1,arge2,...)
L'appel à la fonction s'opère de la façon suivante :
[vars1,vars2,...] = nomfonction(vare1,vare2,...)
avec compatibilité des variables d'entrées vare1,vare2,... avec les arguments d'entrée arge1,arge2,... et
compatibilité des variables de sorties vars1,vars2,..., si celles-ci ont déjà été utilisées, avec les arguments de
sortie args1,args2,....
Remarque : il n'est pas obligatoire de fournir tous les arguments d'entrées et de sortie lors de l'appel d'une
fonction, mais ceux que l'on omet doivent être les derniers des listes d'entrée ou de sortie. Ainsi, supposons
nomfonction soit une fonction à 2 arguments d'entrée et 2 arguments de sortie, on peut alors utiliser l'appel
suivant :
[vars1]= nomfonction(vare1) mais pas l'appel : [vars2]= nomfonction(vare2).
La limitation du nombre d'arguments de sortie est gérée de façon automatique dans Matlab. Par contre, les
variations du nombre d'arguments d'entrée doivent être gérées par le concepteur de la fonction à l'aide du
paramètre nargin qui indique le nombre d'arguments en entrée lors de l'appel de la fonction. Voici un exemple
de calcul d'un produit scalaire ou d'une norme au carrée, illustrant son utilisation :

```
function r = psnorm2(a,b) (a,b sont des vecteurs colonne)
    if (nargin==1)
      r=a'*a
    elseif (nargin==2)
    r=a'*b
    end
```

psnorm2(u,v) renvoie le produit scalaire et psnorm2(u) renvoie la norme au carrée de u. Remarquer que
psnorm2(u,u) renvoit également la norme au carrée de u.
Bien qu'il soit parfois dangereux d'utiliser cette possibilité, il est important de la connaître car de nombreuses
fonctions natives de Matlab l'utilisent.

♦ Portée d'une fonction10

On peut déclarer des fonctions dans un script principal à la suite des instructions du script principal. Mais il est souvent plus recommandé, pour des raisons d'organisation, de les placer dans un autre script, voire plusieurs. Dans ce cas, le nom du script externe devra porter le nom de la fonction avec l'extension .m. Si dans ce script externe, on y place plusieurs fonctions, seule la fonction dont le script porte le nom sera accessible depuis le script d'appel. Les autres fonctions ne seront accessibles que dans le script où elles ont été déclarées; on parle, dans ce cas, de sous-fonctions.

Egalement liée aux fonctions, la notion de variable globale permet de rendre visible des variables d'un script à l'autre. Par défaut, aucune variable n'est globale. Supposons que l'on ait un script principal et un script secondaire contenant la déclaration de la fonction nomfonction. Pour déclarer de façon globale, la variable de nom varg, on introduira l'instruction suivante dans le script principal et la fonction nomfonction :

global varg

♦ Fonction comme argument d'une fonction

Pour clore cette partie sur les fonctions, mentionnons l'existence de commandes permettant d'évaluer une chaîne
de caractères comme une commande Matlab. La command eval s'utilise ainsi :

```
comd='s1=sin(1)'
comd =
s1=sin(1)
eval(comd)
s1 =
   0.8415
```

La commande feval permet quant à elle de transférer, en temps que chaîne de caractères, le nom d'une fonction
que l'on veut évaluer :

```
fon='sin'
fon =
sin
s1=feval(fon,1)     (derrière le nom de la fonction à évaluer, apparaissent les arguments)
s1 =
   0.8415
```

6. Lecture et écriture

Comme il l'a déjà été mentionné, par défaut toute commande exécutée produit un résultat qui apparaît dans la fenêtre d'exécution à la suite de la commande. On peut empêcher l'affichage du résultat en terminant la commande par ';'. Ainsi, on a :

```
H=[1 2 3]           (Résultat affiché)
H =
    1     2     3
H=[1 2 3];     (Aucun résultat affiché)11
```

Dans les scripts d'exécution, l'affichage d'un résultat est exceptionnel. Il sert, essentiellement, à détecter les erreurs et à afficher des résultats finaux. On prendra donc garde à ne pas oublier le ';' à la fin de chaque ligne de commande.

♦ Impression à l'écran

La commande standard d'écriture dans la fenêtre d'exécution est fprintf qui a la structure générale suivante :

fprintf(format,var1,var2,...) où format est une chaîne de caractères décrivant le format d'écriture des variables var1,var2,... que l'on souhaite afficher. Les principaux types de formats d'écriture sont :

%d entier %5d : entier de longueur 5, par exemple 34562

%f réel %5.2f : réel de longueur 5 avec 2 chiffres après la virgule, par exemple 32.42

%e exponentiel %10.8e : nombre de la forme -21.01e+05

%g réel double précision mode automatique de détection entre %e et %f

%s chaîne de caractères

Par souci de simplicité, on peut se contenter d'utiliser les formats %d,%f,%e sans spécifier de longueur précise. Par ailleurs on dispose de certains opérateurs de mise en forme , par exemple \n pour passer à la ligne. Ainsi, on écrira, par exemple :

```
fprintf('\n Convergence en %d iterations ',it)
```

où it désigne une variable contenant un entier.

Pour plus de détails, faire help fprintf.

♦ Impression dans un fichier

Il est également possible d'écrire les résultats dans un fichier (et souhaitable lorsqu'il y en a beaucoup). Pour ce faire, on utilise encore la commande fprintf, mais en spécifiant un numéro nfic associé à un nom de fichier de résultats, nommé ici ficres. On effectue, les opérations suivantes :

```
nfic=fopen(ficres,'rw');          (ouvre le fichier ficres en mode lecture et écriture)
fprintf(fid,'\n Convergence en %d iterations ',it); (écrit dans le fichier ficres)
status=fclose(fid); (ferme le fichier ficres)
```

L'opération d'ouverture de fichier par la commande fopen a échoué si nfic vaut -1 et status renvoie 0 si l'opération de fermeture par la commande fclose est réussie et -1 sinon.

♦ Lecture de données

Afin de lire des données utiles à l'exécution, on peut procéder de deux façons : soit en interrompant l'exécution du programme et en demandant à l'utilisateur d'indiquer les données, soit en lisant un fichier de données. Cette deuxième solution étant bien souvent préférable.

Pour interrompre l'exécution et demander une valeur, on utilise la commande input, dont voici un exemple d'utilisation :

```
data=input('Donnez votre valeur (par defaut 0)');12
```

A l'issue de la réponse, la variable data contiendra la réponse envoyée, qui peut prendre n'importe quel type et même prendre la valeur vide [] si on a taper sur enter. C'est d'ailleurs par ce moyen que l'on gère les valeurs par défaut :

```
if (data ==[])
data=0;
   end
```

Pour lire des fichiers de donnée, on utilise la commande fscanf dont le principe de fonctionnement est voisin de la commande fprint. Pour lire la donnée data dans le fichier ficdon, on utilisera la suite de commande suivante :

```
nfic=fopen(ficdon,'r');          (ouvre le fichier ficdon en mode lecture)
```

data=fscanf(fid,'%d'); (lit dans le fichier ficdon)
status=fclose(fid); (ferme le fichier ficdon)
Pour plus d'informations, faire help fscanf, help fopen et help fclose. Il existe d'autres méthodes de lecture et d'écriture sur fichier, faire help iofun pour de plus amples informations.

7. Graphique sous Matlab

Afin de terminer cette brève introduction à Matlab, indiquons quelques fonctionnalités graphiques de Matlab.
Donnons deux exemples : le tracé d'une courbe et le tracé d'un champ d'isovaleurs.
Supposons que l'on veuille représenter graphiquement la courbe y=sin(x) sur l'intervalle [-π,π] avec 200 points.
On exécute alors les commandes suivantes :

```
x=-pi:pi/100:pi;
y=sin(x);
plot(x,y);
```

-4 -3 -2 -1 0 1 2 3 4
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

Il existe de nombreuses options pour contrôler l'affichage des courbes (couleur, axe, commentaire ...), faire help
plot et plus généralement help graph2d.13
Maintenant, supposons que l'on veuille représenter la surface définie par la fonction z=sin(xy) sur le carré
suivant [-π/2,π/2]x[-π/2,π/2] à l'aide d'une grille de points 30x30.
On utilise la séquence de commandes suivantes :

```
[xi,yi]=meshgrid(-pi/2:pi/30:pi/2);
zi=sin(xi.*yi);
surf(xi,yi,zi,zi)
```

Pour les autres possibilités faire help graph3d.

Exemples d'applications :

1. Calcul sur les polynômes.....14-20
2. Graphe et calcul de la dérivée d'une fonction f '(x).....21-23
3. Equation non lineaire à une inconnue et racine d'une fonction...24-25
4. Calcul d'integrale.....26-27

Calcul sur les polynômes

Fonctions :

conv	produit de polynômes
residue	décomposition en éléments simples
roots	trouve les racines d'un polynôme
poly	trouve le polynôme à partir des ses racines
polyval	évalue le polynôme(la valeur du polunôme en un point)

1. Racines d'un polynôme

- 1er exemple :

$3x^2 - 5x + 2 = 0$

On commence par définir un " vecteur " qui contient les coefficients du polynôme : (par ordre décroissant en degré de x)

```
>> p = [ 3 -5 2 ]

p =

3 -5 2

>> roots(p)

ans =

1.0000

0.6667

>> roots( [ 3 -5 2 ])

ans =

1.0000

0.6667
```

- 2ème exemple :

$x^2 - 4x + 4 = 0$

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

p =

1 -4 4

>> roots(p)

ans =

2

2
```

Ø 3ème exemple :

$x^2 + 3x + 8 = 0$

>> p= [1 3 8]

p =

1 3 8

>> roots(p)

ans =

-1.5000 + 2.3979i

-1.5000 - 2.3979i

Ø 4ème exemple :

$x^5 + 2x^4 - 2x^3 + 4x^2 + 3x + 5 = 0$

>> p = [1 2 -2 4 3 5]

p =

1 2 -2 4 3 5

>> roots(p)

ans =

-3.0417

0.9704 + 1.0983i

0.9704 - 1.0983i

-0.4495 + 0.7505i

-0.4495 - 0.7505i

>> format long e

>> roots(p)

ans =

-3.041684725314715e+000

9.703604093970790e-001
+1.098343294996758e+000i

9.703604093970790e-001 -
1.098343294996758e+000i

-4.495180467397220e-001
+7.504870344729816e-001i

-4.495180467397220e-001 -
7.504870344729816e-001i

Ø 5ème exemple :

polynôme à coefficients complexes :

$(1+i)x^2 + (2-5i)x + 3,5 = 0$

>> format short

>> p = [1+i 2-5i 3.5]

p =

1.0000 + 1.0000i 2.0000 - 5.0000i
3.5000

>> roots(p)

ans =

1.7116 + 4.0248i

-0.2116 - 0.5248i

2. Détermination des coefficients d'un polynôme à partir des ses racines

- 1er exemple :

```
>> a = [ 2 1 ]

a =

2 1

>> poly(a)

ans =

1 -3 2
```

(c'est-à-dire : x^2 -3x +2)

- 2ème exemple :

```
>> a = [ 2 2 3 -5 ]

a =

2 2 3 -5

>> poly(a)

ans =

1 -2 -19 68 -60
```

- 3ème exemple :

```
>> a = [ 2+i 2-3i 5]

a =

2.0000 + 1.0000i 2.0000 - 3.0000i
5.0000

>> poly(a)

ans =

1.0000 -9.0000 + 2.0000i 27.0000 -
14.0000i -35.0000 +20.0000i
```

Vérification :

```
>> p = ans

p =

1.0000 -9.0000 + 2.0000i 27.0000 -
14.0000i -35.0000 +20.0000i

>> roots(p)

ans =

2.0000 - 3.0000i

5.0000 - 0.0000i

2.0000 + 1.0000i
```

3. Produit de polynômes

- Ø 1er exemple

(x –2)(x – 1) = ?

```
>> p1=[ 1 -2 ]

p1 =

1 -2

>> p2=[ 1 -1 ]

p2 =

1 -1

>> conv( p1 , p2 )

ans =

1 -3 2
```

Autrement dit :

(x –2)(x – 1) = x^2 -3x +2

Ø 2ème exemple

($3x^2$ - 5x + 2)(x^2 + 3x + 8) = ?

```
>> p1=[ 3 -5 2 ]

p1 =

3 -5 2

>> p2=[ 1 3 8 ]

p2 =

1 3 8

>> conv( p1 , p2 )

ans =

3 4 11 -34 16
```

Autrement écrit :

($3x^2$ - 5x + 2)(x^2 + 3x + 8) = $3x^4 + 4x^3 + 11x^2 - 34x + 16$

4. Décomposition en éléments simples

$$\frac{n(x)}{d(x)} = \frac{r_1}{x - p_1} + \frac{r_2}{x - p_2} + \ldots + k(x)$$

p_1 , p_2 … désignent les " pôles ".

Ø Exemple :

$$\frac{6}{x^4 + 6x^3 + 11x^2 + 6x} = ?$$

polynôme du numérateur :

```
>> n =[ 6 ]

n =

6
```

polynôme du dénominateur :

```
>> d =[ 1 6 11 6 0 ]

d =

1 6 11 6 0

>> [ r , p , k ] = residue ( n , d)

r =

-1.0000
```

3.0000

-3.0000

1.0000

p =

-3.0000

-2.0000

-1.0000

0

k =

[ ]

$$\frac{6}{x^4+6x^3+11x^2+6x}=\frac{-1}{x+3}+\frac{3}{x+2}+\frac{-3}{x+1}+\frac{1}{x}$$

5. Représentation graphique

- Exemple :

y = f(x) = 3x² - 5x + 2

5.1. Utilisation de la fonction plot

>> p = [3 -5 2]

p =

3 -5 2

Calcul de f(x = 1) :

>> polyval(p , 1)

ans =

0

Calcul de f(x = 2) :

>> polyval(p , 2)

ans =

4

Création du vecteur x :

>> x = 0 : 0.01 : 2

x =

Columns 1 through 7

0 0.0100 0.0200 0.0300 0.0400 0.0500 0.0600

…

Columns 197 through 201

1.9600 1.9700 1.9800 1.9900 2.0000

Création du vecteur y :

>> y = polyval(p , x)

y =

Columns 1 through 7

2.0000 1.9503 1.9012 1.8527 1.8048
1.7575 1.7108

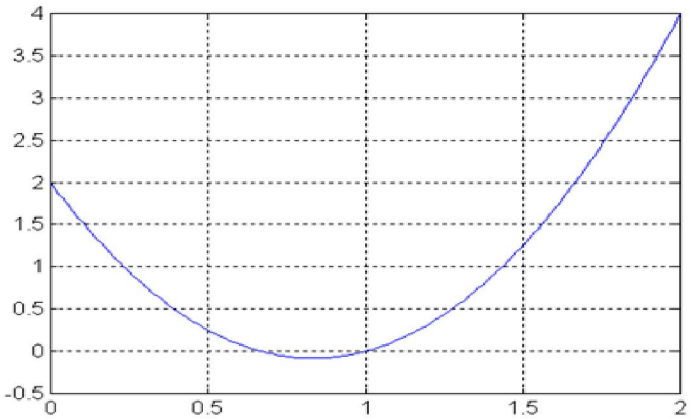
…

Columns 197 through 201

3.7248 3.7927 3.8612 3.9303 4.0000

>> plot (x , y)

>> grid on

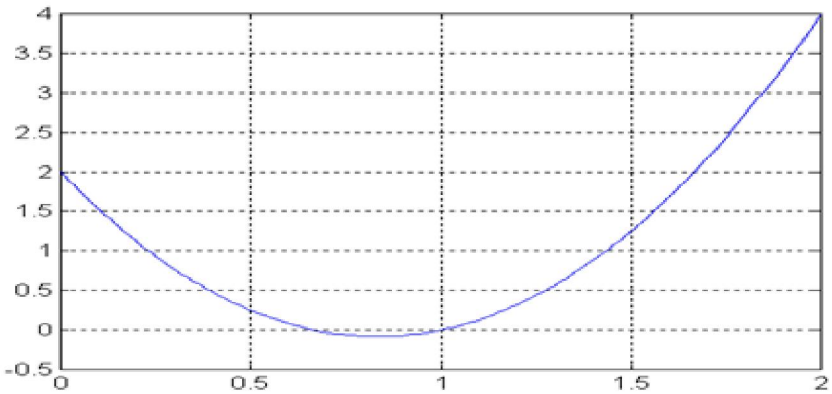


5.2. Utilisation de la fonction fplot

5.2.1. Première méthode

>> fplot ('3*x^2 - 5*x + 2' , [0 2])

>> grid on



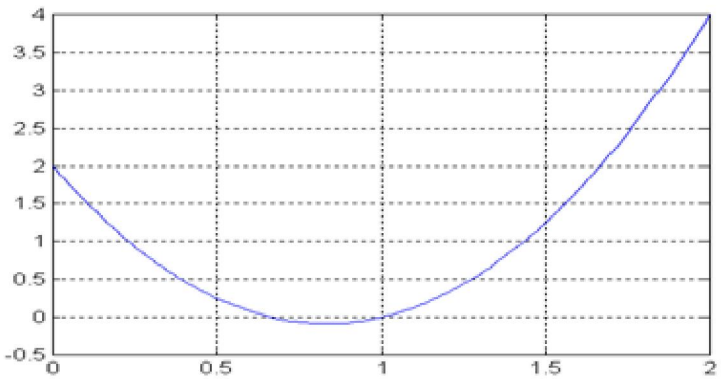
5.2.2. Deuxième méthode

Créer le fichier f1.m de la fonction (Files-new-M-files) :

Vous pouvez clicker directement sur « New M-files »

```
function y=f1(x);
p=[3 -5 2];
y=polyval(p,x);
```

```
>> f1(2)

ans =

4

>> f1(0)

ans =

2

>> fplot ('f1', [ 0 2] )

>> grid on
```



On peut également définir le polynôme de la manière suivante :

```
function y=f1b(x);
y=3.*x.^2 -5.*x + 2;
```

Graphe de la dérivée d'une fonction f '(x)

- Prenons l'exemple suivant :

$$y = f_2(x) = \sin(x) + \frac{\cos(3x)}{2} - \frac{\sin(5x)}{5}$$

Commençons par créer le fichier f2.m de la fonction :

```
function y=f2(x);
y=sin(x)+0.5.*cos(3.*x)-0.2.*sin(5.*x);
```

>> [x y] = fplot ('f2' , [0 2*pi])

x =		
0	6.2643	0.4986
0.0126	6.2832	…
0.0251	y =	0.4966
…	0.5000	0.4992
6.2455	0.4997	0.5000

Une approximation de la dérivée est donnée par :

>> dy = diff(y)./diff(x)

dy =

-0.0276, -0.0804 , -0.1292 , … , 0.1370 , 0.0438

>> plot (x , dy)

??? Error using ==> plot

Vectors must be the same lengths.(c.a.d les vecteurs doivent avoir la meme dimention).

>> length(x)

ans =

307

>> length(dy)

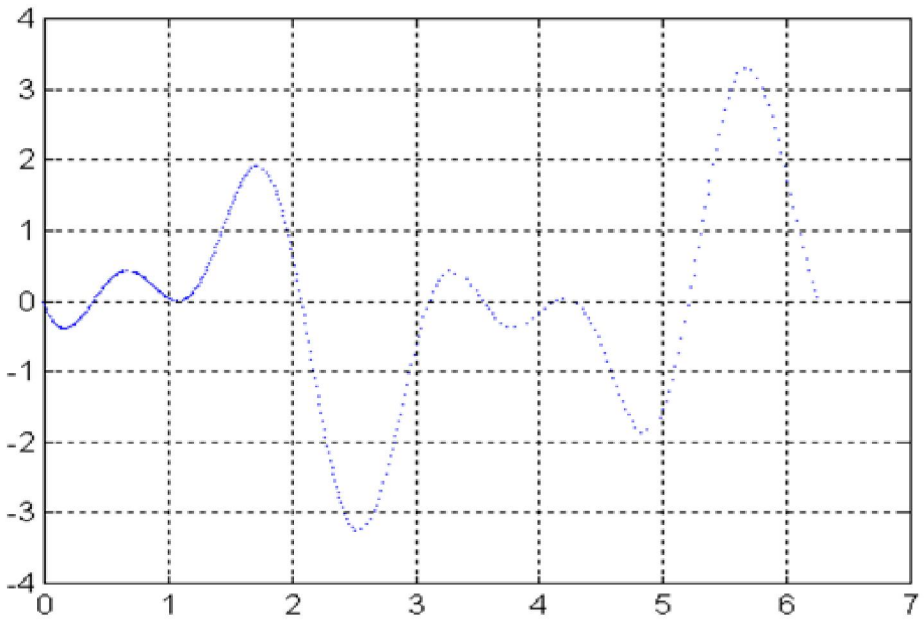
ans =

306

Le vecteur dy a un élément de moins que le vecteur x, ce que n'apprécie pas la fonction plot. Nous allons ignorer le dernier élément du vecteur x

>> plot (x (1 : length(x)-1 , :) , dy , '. ')

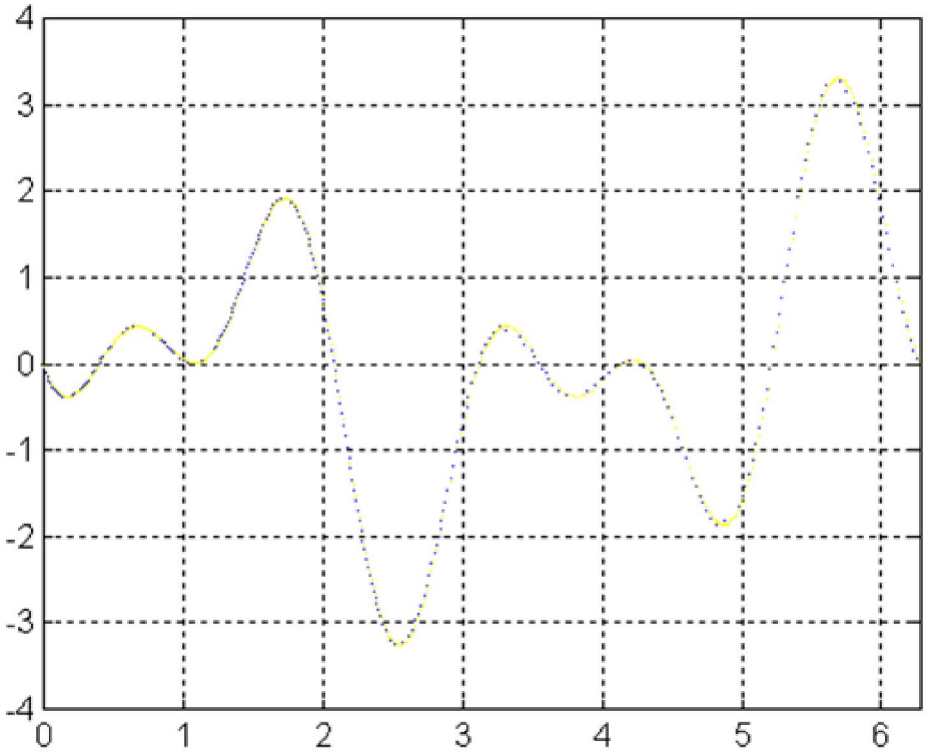
>> grid on



Comparons avec la dérivée exacte (couleur jaune) :

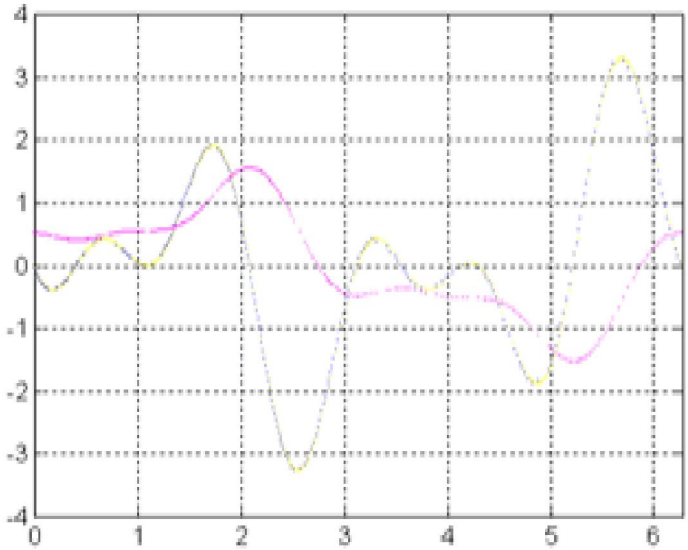
>> hold on

>> fplot ('cos(x) - 1.5*sin(3*x) - cos(5*x)' , [0 2*pi] , 'b.')



En violet, la fonction f2(x) :

```
>> fplot ( 'f2' , [ 0 2*pi ] , 'g.' )
```



Equation non linéaire à une inconnue & Racines d'une fonction

1. Fonction : fzero

Ø Exemple :

Soit à résoudre l'équation :

$-6+2x+sin(x^2)=0$

Cela revient à trouver les racines de la fonction :

$y(x) = -6+2x+sin(x^2)=0$

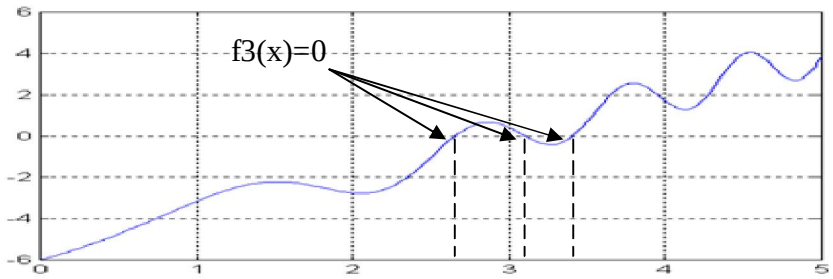
Il faut commencer par créer le fichier f3.m de cette fonction :

```
function y=f3(x);
y=-6+2.*x+sin(x.^2);
```

Représentation graphique de la fonction :

>> fplot('f3', [0 5])

>> grid on % celà pour localizer les zones où s'annule la function.



Cela fait apparaître 3 racines, aux environs de 2,7 3,1 et 3,4.

```
>> fzero('f3', 2.7)

ans =

2.6545

>> format long e

>> fzero('f3', 2.7) % avec plus de précision

ans =

2.654461193307640e+000

>> format short

>> fzero('f3', 3.1)

ans =

3.1039

>> fzero('f3', 3.4)

ans =

3.4078
```

Vérification :

```
>> f3(ans)

ans =

5.5511e-016
```

Calcul d'intégrale

- Fonctions :

quad : algorithme de Simpson

quad8 : algorithme de Newton-Cote (version 4.2.)

quadl (version 6.5.)

Exemple :

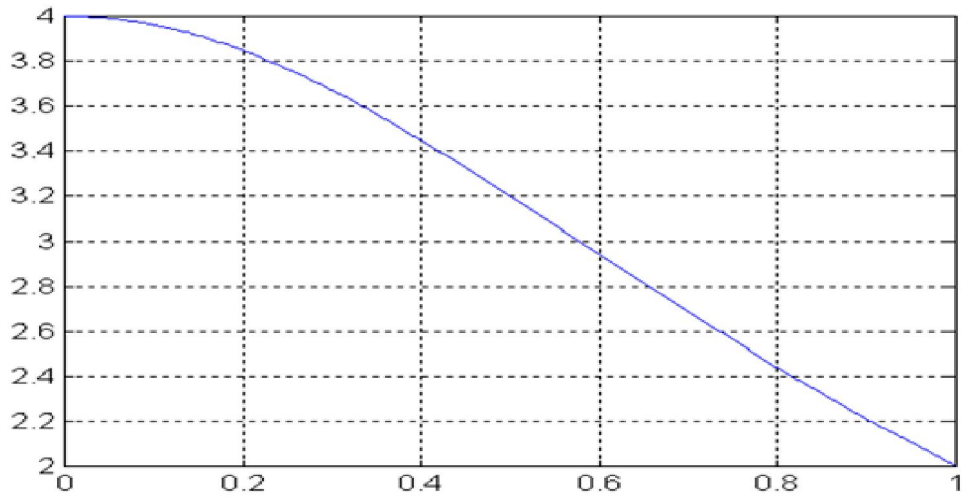
$$\int_{x=0}^{1} \frac{4}{1+x^2}\,dx$$

Il faut commencer par créer le fichier f4.m de la fonction à intégrer :

```
function y=f4(x);
y=4./(1+x.*x);
```

>> fplot('f4', [0 1])

>> grid on



```
>> quad( 'f4' , 0 , 1 )

ans =

3.1416

>> format long e

>> quad( 'f4' , 0 , 1 )

ans =

3.141592502458707e+000
```

La valeur exacte de cette intégrale est pi.

```
>> pi

ans =

3.141592653589793e+000
```

Améliorons la précision avec une tolérance de calcul de 1e-10 (la tolérance par défaut est 1e-3) :

```
>> quad( 'f4' , 0 , 1 , 1e-10 )

ans =

3.141592653589792e+000

>> quad8( 'f4' , 0 , 1 , 1e-10)

ans =

3.141592653589793e+000
```