MATLAB : LOGICIEL DE CALCUL SCIENTIFIQUE ET LANGAGE DE PROGRAMMATION

Première partie : Le noyau de Matlab

I- Généralités sur Matlab
- I-1- Environnement
- I-2- Principe du Help/Demo
- I-3- Richesse de Matlab : ses fonctions préprogrammées
- I-4- Exemple de programme
- I-5- Notion d'algorithme
- I-6- Où écrire ses programmes/instructions et comment les exécuter :

II- Type et structure des variables
- II-1- Types de données
- II-2- Structures des données

III- Instructions et structures de contrôle
- III-1- Affectations :
- III-2- Instructions de contrôle :
- III-3- Utilisations des fonctions préprogrammées :
- III-4- Écrire ses propres fonctions :

IV- Graphisme sous Matlab
- IV-1- Graphisme 2D
- IV-2- Graphisme 3D
- IV-3- Affichage des images

V- Entrée/Sortie des données
- V-1- lecture/écriture des fichiers .mat
- V-2- lecture/écriture des fichiers binaires
- V-3- lecture/écriture des fichiers texte

VI - Exemples d'application
- VI-1- Calcul des matrices de cooccurrences
- VI-2- TP2 - Géo6333 : reconstitution d'une image satellitaire

Deuxième partie : Les Toolboxes

1- Statistics Toolbox
2- Signal Processing Toolbox
3- Image Processing Toolbox
3- Mapping toolbox
4- Fuzzy Logic Toolbox
5- Neural Networks Toolbox
6- Wavelet Toolbox
7- GUIs

CONCLUSION

NB : Ce tutorial est fait surtout pour la version 5.3 (release 11) mais peut très bien convenir aux autres versions (6.*).

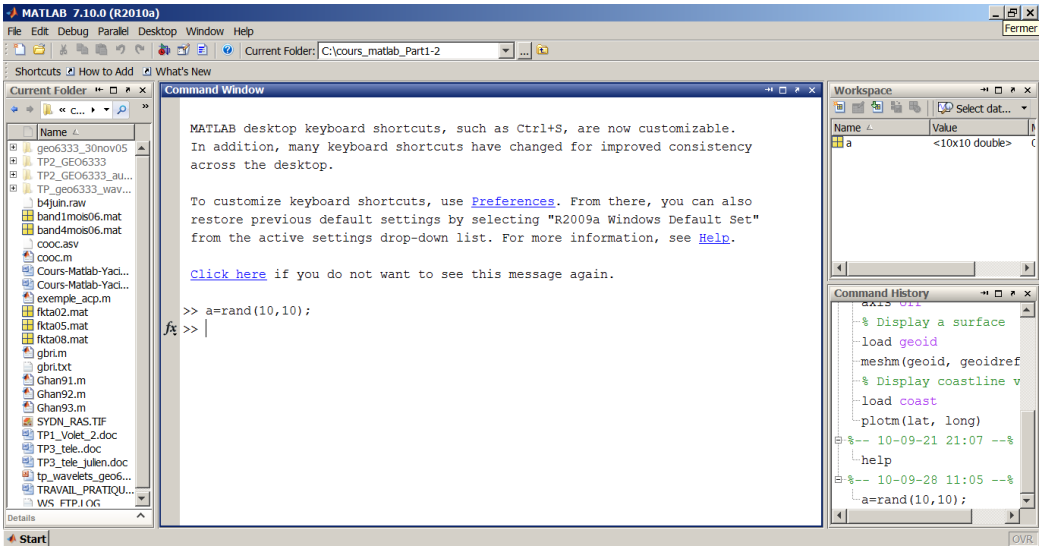
Première partie : Le noyau de Matlab

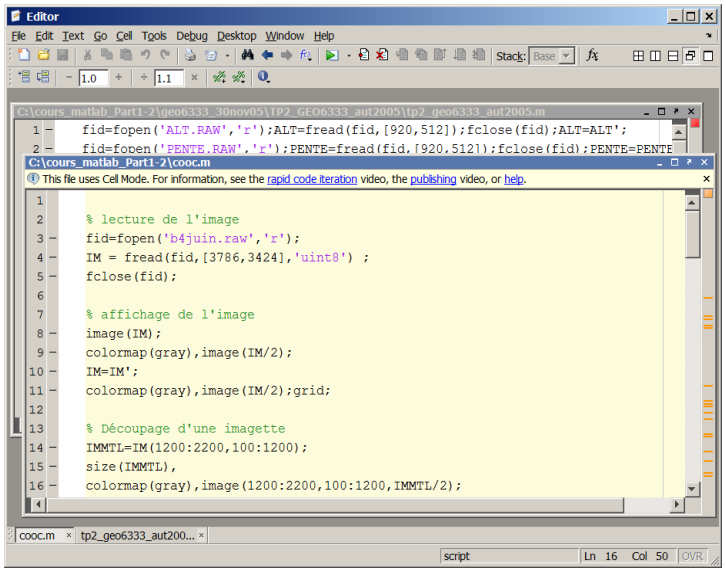
I- Généralités sur Matlab

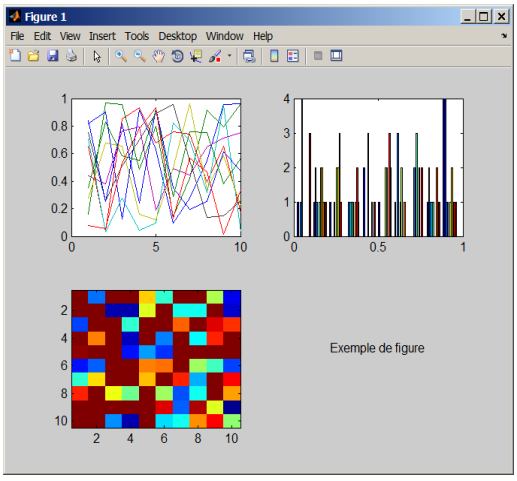
Matlab (Matrix laboratory) est un langage de calcul scientifique très performant intégrant le calcul, la programmation et la visualisation dans un environnement simple à utiliser. C'est un système interactif qui permet de manipuler directement des données structurées (matrices et vecteurs), ce qui n'est pas le cas de C, Java, Fortran ou Pascal par exemple.

I-1- Environnement

Matlab est à la fois un langage et un logiciel. Il possède une fenêtre principale ou de commande et un éditeur de programmes (M-files). Dans l'éditeur de programme, plusieurs fenêtres (programmes) peuvent être ouvertes à la fois. Un éditeur de figure utilisé pour les graphiques.



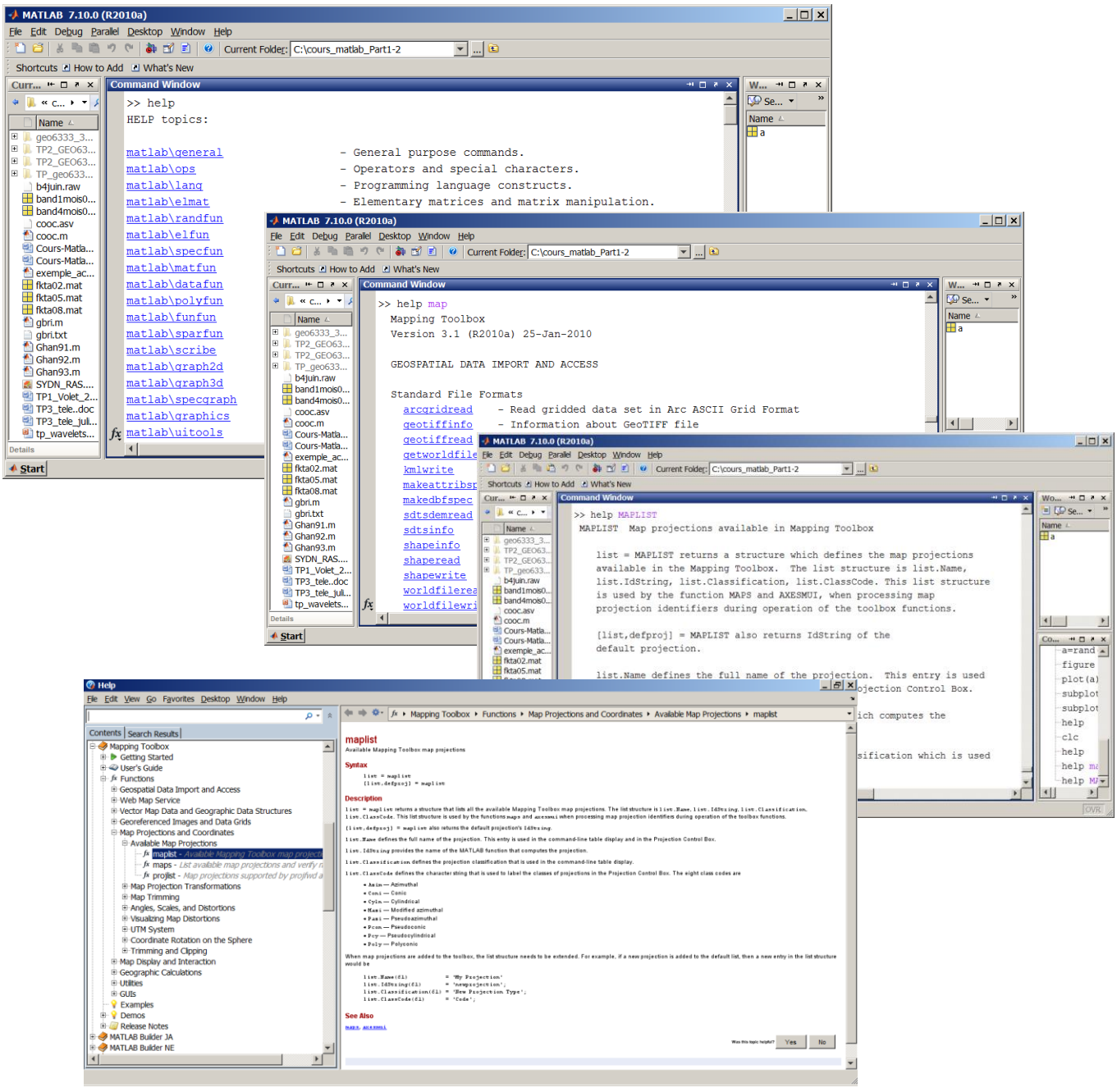




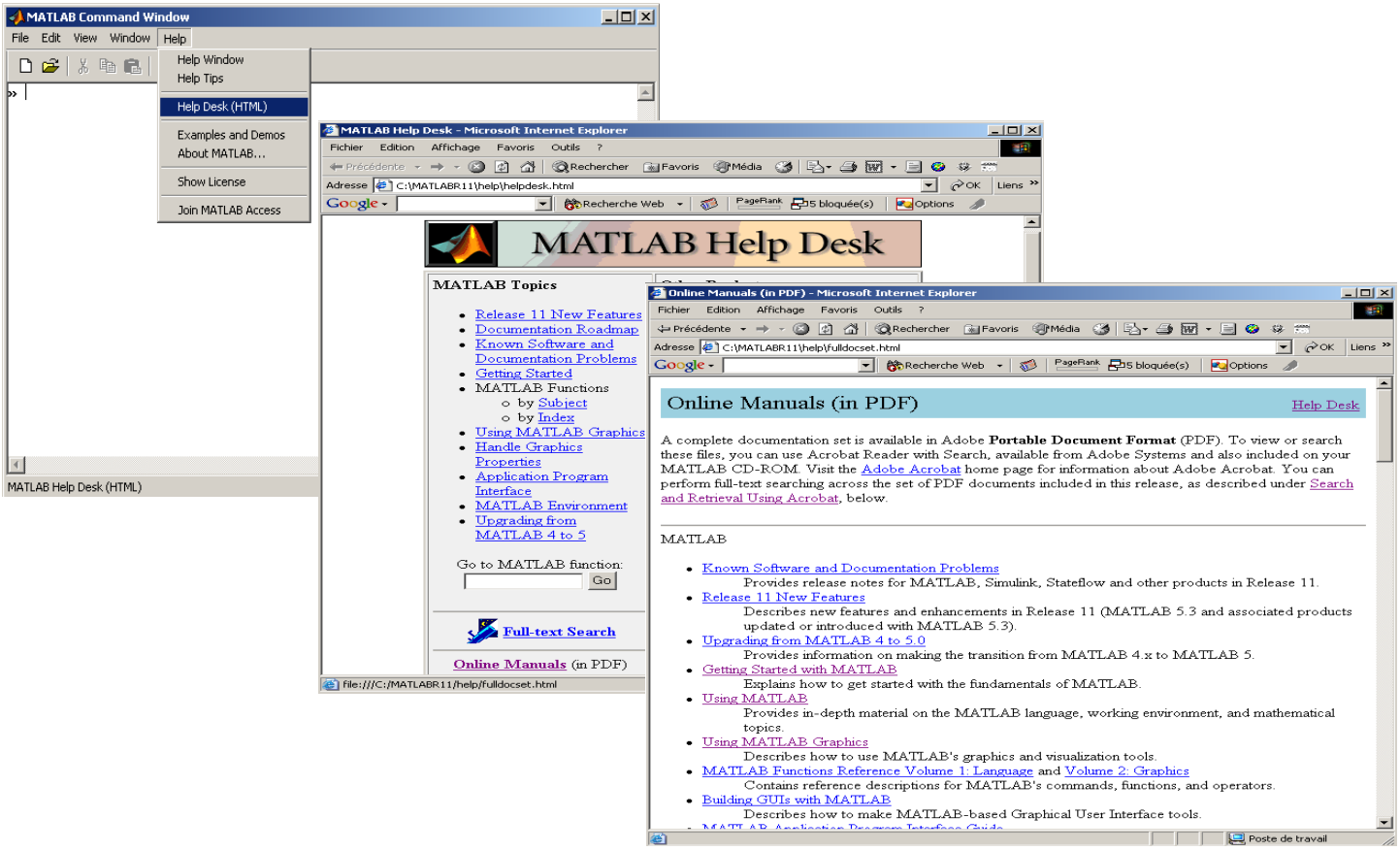
I-2- Principe du Help/Demo

Le help de Matlab est très bien adapté à l'auto-apprentissage autant théorique (documents pdf) que pratique (**`help`** topic) sur le langage, les fonctions disponibles et les techniques qu'il comporte (notamment grâce à ses toolboxes).

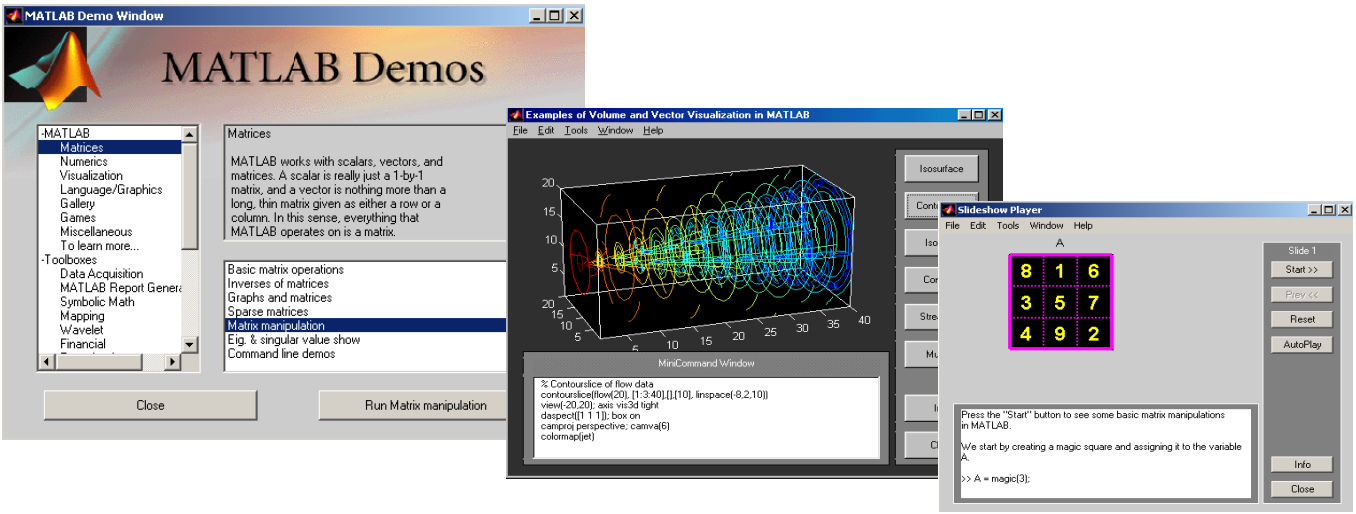
Taper la commande **`help`** permet d'afficher tout le contenu de matlab, ie les familles de fonctions qu'il comporte. Taper **`help`** *famille* affiche toutes les fonctions qui appartiennent à la famille en question et **`help`** *fonction* donne la définition de la fonction, ses options et sa syntaxe.



Une documentation pdf plus élaborée est aussi disponible et accessible par le menu *Help Desk (HTML) – online manuals (in pdf)*. Cette documentation comporte des supports théoriques pour la plupart des techniques offertes par Matlab notamment via les toolboxes. On pourrait y trouver par exemple des cours théoriques et pratiques sur le traitement du signal, la logique floue, les réseaux de neurones, les ondelettes, ect…



Le **`demo`**, apprendre tout en "s'amusant" : taper **`demo`** (démonstration) affiche un environnement interactif qui permet de naviguer dans les différentes ressources de matlab et de lancer des simulations de programmes où les codes (instructions) et les actions résultantes sont affichés en même temps.



I-3- Richesse de Matlab : ses fonctions préprogrammées

Matlab est doté d'une collection de fonctions (m-files) préprogrammées (notamment dans ses Toulboxes) spécifiques à des domaines aussi variés que les statistiques, le traitement du signale et d'image, la logique floue, les réseaux de neurones, les ondelettes, … et qui permettent de résoudre un bon nombre de problèmes relatifs à ces domaines. Pour visualiser ces fonctions, il suffit de taper **`help`** suivi du non de la famille à laquelle appartient la fonction. Pour connaître le nom de ces familles, il suffit juste de taper **`help`**.

Plus de 1500??? fonctions préprogrammées :

- Générale (**`help general`**) : help, demo, dir, cd, !, clear, whos, clear, clc,…
- Opérateurs (**`help ops`**) : +, -, *, /, ^, =, ~=, <, >, &, | , ~, .*, ./, .^, …
- Langage (**`help lang`**) : if, else, for, while, case, …
- fonctions élémentaires sur les matrices (**`help elmat`**) : rand, ones, size, diag, ', …
- fonctions mathématiques élémentaires (**`help elfun`**) : cos, tan, sin, sinh, asin, asinh, exp, log, log10, round, …
- fonctions mathématiques spécialisées (**`help specfun`**) : airy, gcd, lcm, factorial, cart2sph, cart2pol,…
- fonctions sur les matrices (**`help matfun`**) : norm, trace, det, inv, eig, …
- analyse de données (**`help datafun`**) : max, min, hist, diff, corrcoef, conv, conv2, …
- polynômes et interpolations (**`help polyfun`**) : interp, interp2, spline, voronoi, polyarea, roots, polyfit, …

Et beaucoup plus dans les toolboxes :

- traitement du signal (**`help signal`**) : cov, xcorr, xcorr2, modulate, fft, ifft, hilbert, impz, filter, fir1, cheby1, ar, …
- traitement d'images (**`help images`**) : imresize, imcontour, edge, histeq, filter2, fft2, bwlabel, colormap, image, …
- statistiques (**`help stats`**) : betafit, weibfit, betapdf, chi2pdf, poisspdf, betacdf, poisscdf, betarnd, poissrnd, mean, std, kurtosis, skewness, harmmean, geomean, ztest, ttest, polyfit, regress, cluster, dendrogram, princomp,
- calcul symbolique (**`help symbolic`**) : diff, int, limit, solve, dsolve, finverse, fourier, laplace, ztrans, …
- réseaux de neurones (**`help nnet`**) : network, newelm, initlay, trainbfg, …
- logique floue (**`help fuzzy`**) : mfedit, ruleedit, surview, anfisedit, …
- ondelettes (**`help wavelet`**): wavemenu, appcoef, dwt, …
- mapping toolbox (**`help map`**) : antipode, distance, areaint, intrplat, intrplon, distortcalc, mfwdtran, deg2km, …

I-4- Exemple de programme

```
% Premier Programme Matlab
% scalaires
```

```
a=3;
b=6;
c=input('donner la valeur de c : ');
d=a+b/c;
disp(d),

% matrices
A=[ 2 5 3
    4 7 1 ];

B=[ 6 1 7 ; 2 0 9 ];

C=A+B;
C,
disp(C)
```

Donc : **ni entête ni déclarations** de types, de constantes ou de variables. Toutes les ressources de Matlab sont directement disponible sur l'espace de travail et n'on pas besoin d'être chargée via des instructions telles que ***include*** de C ou ***uses*** de Pascal.

I-5- Notion d'algorithme

Un algorithme est un ensemble (séquence) d'opération (ou d'ordres) simples, que l'ordinateur peut exécuter (sous un langage de programmation) et conçu afin de résoudre un problème plus complexe. Notons qu'il y a une "infinité" de façons de concevoir un algorithme mais seule une (ou quelques) façon est optimale.
L'algorithme dépend des actions que l'ordinateur (ou plutôt le langage) peut exécuter. Plus l'ordinateur peut exécuter des opérations complexes plus l'algorithme est simple et inversement.
Les aMatlab ayant des

I-6- Où écrire ses programmes/instructions et comment les exécuter :

Toutes les instructions permises par Matlab peuvent être lancées à partir de la fenêtre de commande, il suffit d'écrire ces instructions en respectant leur syntaxe et de taper enter pour valider. Si nous avons un nombre très réduis d'instructions a exécuter, il est en effet possible de les écrire directement dans cette fenêtre de commande. Toutefois, le plus souvent nous avons besoin d'écrire des programmes assez longs et surtout de sauvegarder ces programmes. L'éditeur de programmes ou de fichiers d'extension ***.m*** est fait pour cela. Pour l'activer, aller dans le menu ***file*** et cliquer sur ***open*** pour ouvrir un fichier existant ou bien sur ***new*** puis ***M-file*** si on veut créer un nouveau fichier ***.m***.
Pour lancer (exécuter) un programme, on peut soit à partir de la fenêtre de programme faire ***tools*** suivi de ***run*** ou bien écrire carrément le nom du programme dans la fenêtre de commande suivi de enter.
Notons que lorsqu'on ouvre Matlab (i.e. la fenêtre de commande), le répertoire de travail par défaut est *c:\matlab\work*. Toutefois le(s) programme(s) qu'on veut exécuter peut se trouver dans un autre répertoire qu'il va falloir atteindre par l'instruction **`cd`** (similaire à celle du DOS) ou encore mieux par la commande ***set path*** qui se trouve dans le menu ***file***. en choisissant le

répertoire de travail par ***Browse***. Une fois ce répertoire sélectionné, sauvegarder ce path par la commande ***save path*** du ***Path Browser***.

II- Type et structure des variables

Types :

'char' : character, 8 bits (signed or unsigned).
'short' : integer, 16 bits.
'int' : integer, 32 bits.
'long' : integer, 32 or 64 bits.
'ushort' : unsigned integer, 16 bits.
'uint' : unsigned integer, 32 bits.
'ulong' : unsigned integer, 32 bits or 64 bits.
'float' : floating point, 32 bits.

'uchar' : unsigned character, 8 bits.
'schar' : signed character, 8 bits.
'int8' : integer, 8 bits.
'int16' : integer, 16 bits.
'int32' : integer, 32 bits.
'int64' : integer, 64 bits.
'uint8' : unsigned integer, 8 bits.
'uint16' : unsigned integer, 16 bits.
'uint32' : unsigned integer, 32 bits.
'uint64' : unsigned integer, 64 bits.
'single' : floating point, 32 bits.
'float32' : floating point, 32 bits.
'double' : floating point, 64 bits.
'float64' : floating point, 64 bits.

'bitN' : N bits (1<=N<=64).
'ubitN' : N bits (1<=N<=64).

Structures des données :

données scalaires : i=2; pi = 3.1416 ; x =1.52e+04; reponse = 'ceci est un champ de caractères',

Les vecteurs :

```
x = [  41    74    27    44    93    68    21    84    63    13 ] ;
x = [   13.60
        76.46
        74.83
        59.55
        25.71  ];
```

Les matrices :

```
x  = [
       72.41       90.28       71.81       30.25       81.62
       28.16       45.11       56.92       85.18       97.71
       26.18       80.45       46.08       75.95       22.19
       70.85       82.89       44.53       94.98       70.37
       78.39       16.63        8.77       55.79       52.21
       98.62       39.39       44.35        1.42       93.29
       47.33       52.08       36.63       59.62       71.34
] ;
```

Les tenseurs :

```
y(:,:,1) = [
   36   24   68   26   18
   29   98   67   12    3
   87   64   13    7   73
   63   23    2   85   54 ];

y(:,:,2) = [
   28   87   59   57   83
   37   25   33   98   19
    1   57   66   79   64
   89   16   86   15   67 ];

y(:,:,3) = [
   77   61   51   41   15
   38   18   21   41   38
   44    0   10    5   31
   48   79   16   94   17 ];
```

Les tenseurs peuvent également avoir une dimension > 3.

III- Instructions et structures de contrôle

La plupart des instructions en matlab sont écrites comme dans un langage de programmation (tel que C par exemple) mais sont exécuter comme dans un logiciel, ie que matlab fait des interprétations et non pas des compilations. Toutefois nous verrons a la fin que certaines instructions peuvent être exécuter a partir de d'interfaces graphiques (GUI).

Les opérations en Matlab sont orientées-matrices, i.e. que les instructions qui se font dans les autres langages (Fortran, Pascal, C, Java) sur des scalaires peuvent se faire en Matlab directement sur les matrices.

III-1- Affectations :

exemple : X=[1 5 3 0; 5 4 1 8; 3 2 1 4]; Y=[4 2 3 7; 2 9 6 0; 7 5 3 2];

Z = 10*X +Y.^2 ;

Z =

26	54	39	49
54	121	46	80
79	45	19	44

T= X*Y ;
dimension de la matrice résultat : (3x4) x (4x3) = (3x3)

T =

23	65	41
87	52	74
47	30	42

III-2- Instructions de contrôle :

instruction alternative **if** :
synthaxe :

```
if condition, instruction(s);
elseif condition, instruction(s);
else instruction(s);
end
```

exemple :

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

instruction alternative **switch** :
synthaxe :

```
switch variable,
  case valeur1, instruction1;
  ...
  case valeurn, instructionn;
end
```

exemple :

```
switch n
   case 1,  disp('Method is linear');
   case 2,  disp('Method is quadratic');
```

```
        case 3,  disp('Method is cubic');
        otherwise disp('Unknown method.')
        end
```

instruction répétitive **for** :

synthaxe : `for expression , instruction(s); end`

```
for i=1:10,
  S = S + x(i) ;
end

for i=1:N, for j=-10:M, for k=5:15, M(i,j,k) = i+j*k; end;end;end

for i=1:4:20,
    for j=20:-1:0,
        T(i,j) = 2*A(i)+j^2 ; if i==j, break; end
    end;
end
```

instruction répétitive **while** :

synthaxe : `while condition , instruction(s); end`

exemple :

```
A=[1 5 3 ; 5 4 1 ; 3 2 1 ];
E = 0*A; F = E + eye(size(E)); N = 1;
while norm(E+F-E,1) > 0,
     E = E + F;
     F = A*F/N;
     N = N + 1;
end
```

III-3- Utilisations des fonctions préprogrammées :

L'utilisation des fonctions de matlab se fait directement par appel (sans aucune déclaration préalable) en respectant leur syntaxe qu'on peut afficher par "`help` *fonction en question* ".

exemple :
supposons qu'on veuille utiliser la fonction **`cart2sph`**. Taper **`help cart2sph`** donne :

```
CART2SPH Transform Cartesian to spherical coordinates.
   [TH,PHI,R] = CART2SPH(X,Y,Z) transforms corresponding elements of
   data stored in Cartesian coordinates X,Y,Z to spherical
   coordinates (azimuth TH, elevation PHI, and radius R).  The arrays
   X,Y, and Z must be the same size (or any of them can be scalar).
   TH and PHI are returned in radians.

   TH is the counterclockwise angle in the xy plane measured from the
```

positive x axis. PHI is the elevation angle from the xy plane.

See also CART2POL, SPH2CART, POL2CART.

Une description détaillée de la fonction (définition et syntaxe) est affichée. Le **help** suggère même des fonctions similaires à la fonction affichée.

Si on tape : [TH,PHI,R] = CART2SPH(10,10,10)
Le résultat affiché serait :

```
TH =
   0.78539816339745

PHI =
   0.61547970867039

R =
  17.32050807568878
```

III-4- Écrire ses propres fonctions :

Une fonction est un programme comme un autre avec en entête le mot réservé **function** suivi de la syntaxe de la fonction sous la forme y=f(x). Taper **help function** pour plus de détails.

Exemples :

```
function nd=numday(j,m)
njpm=[31 28 31 30 31 30 31 31 30 31 30 31];
nm=sum(njpm(1:m-1))+j;
```

si on écrit dans la fenêtre de commande **numday(18,07)** le résultat sera :

```
ans =  199
```

Une fonction Matlab peut avoir plusieurs entrées et plusieurs sorties et qui peuvent être aussi hétérogènes que possible. Chacune des variables (d'entrée ou de sortie) peut avoir une structure ou un type différent des autres.

Exemple :

On peut faire une fonction ou les entrées peuvent être par exemple une chaîne de caractère (nom de fichier par ex), suivie d'une matrice puis d'un scalaire. La sortie peut être par exemple une matrice, un vecteur et une chaîne de caractère.

On lit dans un fichier essai.txt (à rentrer) une matrice A qu'on ajoute a une matrice B (à renter) et qu'on multiplie par un scalaire c(à rentrer). On calcule donc par exemple : D=A+cB, la somme des lignes de D et on affiche un texte qui décrit la matrice D.

IV- Graphisme sous Matlab

Matlab possède des possibilités d'édition de graphes (2D et 3D) et d'images très développées. Voir **`demo`** → visualization.

IV-1- Le graphisme 2D

Parmi les instructions de graphisme 2D, on peut citer à titre d'exemple :

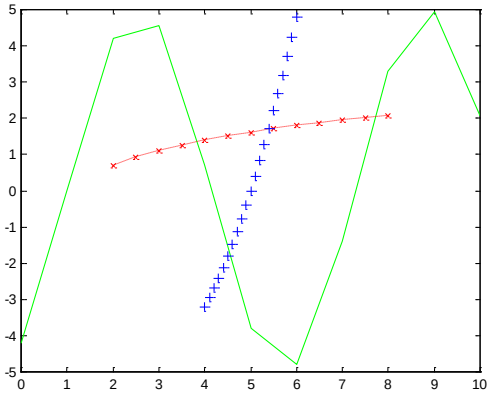
L'instruction `plot` :

Permet de faire un graphe d'une ou de plusieurs séries de données selon différentes options.
syntaxes : (utiliser **`help plot`** plus de détails)
`plot(y)` : affiche le graphe du vecteur (ou des colonnes de matrice) y en fonction de son indice.
`plot(x,y)` : affiche y en fonction de x.
`plot(x,y,'+k')` : ajoute l'option de couleur et de ligne et/ou de marque. ici `'k'` corresponde a la couleur `'black'` et `'+'` à la marque +.
`plot(x1,y1,'-g',x2,y2,':rx',x3,y3,'b+')` : affiche y1 en fonction de x1, y2 en fonction de x2, y3 en fonction de x3, etc… la première est affichées en couleur verte (green) avec une ligne continue, la seconde en rouge (red) avec la marque x et une ligne discontinue superposée et enfin la troisième en bleu avec une marque +.

Exemple :

```
x1=0:10; x2=2:0.5:8; x3=4:0.1:6;
y1=5*sin(x1-1); y2=log(x2); y3=0.8*x3.^2-4*x3;
plot(x1,y1,'-g',x2,y2,':rx',x3,y3,'b+')
```

La figure peut être copiés dans word par la commande *copy figure* du menu *edit*. Le résultat est la figure ci-dessous. Le contrôle des axes, la grille quadrillée, le titre, la légende peut être effectué par les instructions axis, grid, title, et legend respectivement. Mais une manière plus intéressante pour choisir les différentes options du plot s'effectue en cliquant sur le bouton ↖ de la fenêtre figure et de double cliquer sur les axes ou sur les plots pour changer les différentes options du plot.



L'instruction `hist` :

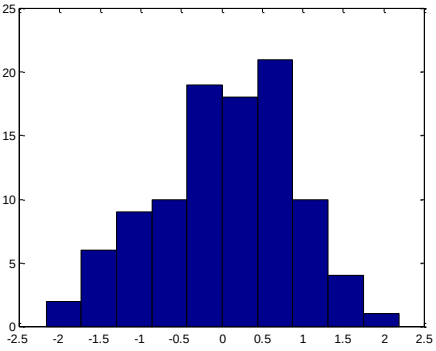
Affiche l'histogramme d'une série de données.
Exemple :

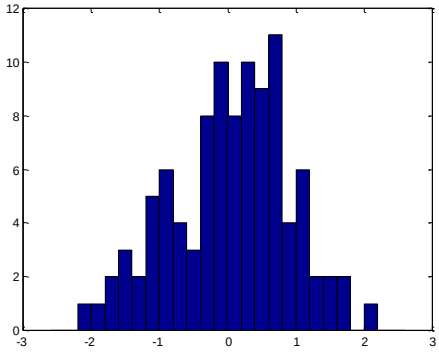
```
x=randn(1,100);
hist(x);
```

donne le premier graphe avec 10 intervalles par défaut

```
hist(x,[-2.5:0.2:2.5]);
```

donne le second graphe avec une définition précise des intervalles





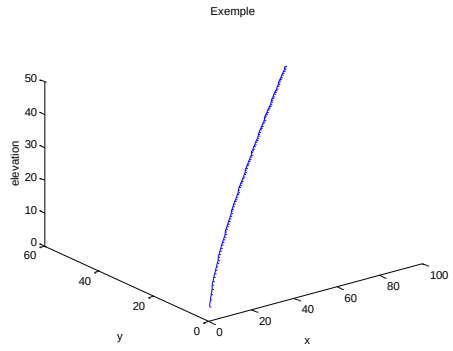
IV-2- Le graphisme 3D

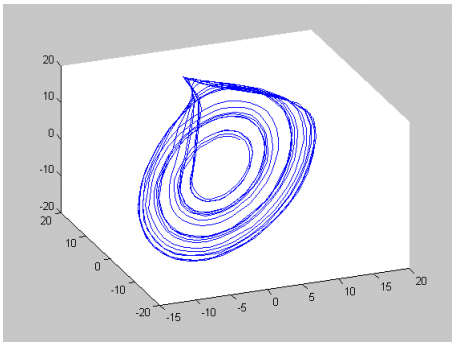
L'instruction `plot3` :
Permet de tracer une variables z en fonction de deux variables x et y (**`help plot3`** pour la syntaxe).
Exemple :

```
x=1:100; y=0.5:0.5:50; z=2*x.^0.5+3*y.^0.5;
plot3(x,y,z),xlabel('x'),ylabel('y'),zlabel('elevation'),title('Exemple')
```

xlabel, ylabel, zlabel et title permettent peuvent être ajoutés à partir de la fenêtre figure en double cliquant sur les axes.

Autre exemple : attracteur de série temporelle



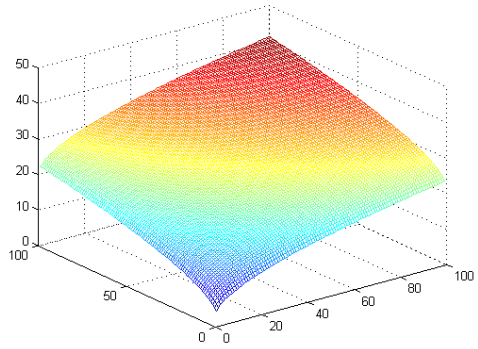


On peut effectuer des zoom, des rotation ou des changement de couleur à partir de la barre d'outils de la fenêtre plot.

L'instruction `mesh` :

Permet de tracer une surface en perspective.
Exemple :

```
x=1:100; y=0.5:0.5:50;
for i=1:100, for j=1:100, z(i,j)=2*x(i)^0.5+3*y(j)^0.5;end;end
mesh(z)
```



Cette instruction servirait par exemple à afficher un Modèle Numérique d'Élévation.

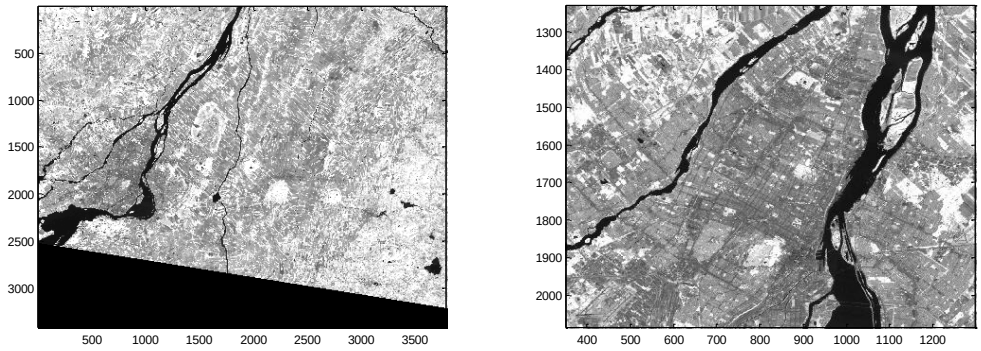
IV-3- L'affichage des images

L'instruction **`image`** :

```
Pour lire l'image :
fid=fopen('b4juin.raw','r');
IM = fread(fid,[3786,3424],'uint8') ;
fclose(fid);

Pour l'afficher :
```

Image(IM) utilise sa palette de couleurs par défaut
Ou
colormap(gray);image(IM) pour affichage en niveaux de gris



Si on veut afficher uniquement la région de Montréal (figure de droite) , soit on limite l'affichage de IM aux lignes et colonnes délimitant la région (`image(IM(1000:2500,1:1000))`), soit on fait un zoom à partir de l'image entière. Les zoom est plus pratique mais moins précis à controler. Plusieurs autres instruction sont disponibles pour l'affichage images et le choix du colormap. (**`help images`**) pour plus de détails

V- Entrée/Sortie des données

V-1- lecture/écriture des fichiers .mat

Les fichiers MAT : Matlab possède un format propre (*.mat) pour sauvegarder (et charger) les résultats. Ce format lui est spécifique et ne peut être lu par un autre logiciel. L'écriture et la lecture dans ce type de fichiers présente l'avantage d'être simple et puissante. Ces instruction sont respectivement **`save`** et **`load`**.

synthaxe :
pour sauvegarder : **`save nomfich var1, var2,`** …
Les variables var1, var2, … peuvent avoir un structure quelconque. L'extension .mat est ajoutée automatiquement à nomfich.
Pour charger : simplement **`load nomfich`**.

Toutefois les instructions **`load`** et **`save`** peuvent manipuler des données ascii et donc lisibles par un éditeur de texte. il suffit d'ajouter le mot `-ascii` à la fin de l'instruction et l'extension `.txt` (ou autre) au nom du fichier pour choisir cette option.
Taper **`help load`** et **`help save`** pour davantage de détails.

V-2- lecture/écriture des fichiers binaires

L'instruction de lecture **`fread`** :
C'est une instruction qui permet la lecture en bloc d'un fichier binaire tout en précisant la taille et le type des données.
syntaxe : elle varie en fonction de la façon dont on veut lire les organiser données : A = fread (fid,size,precision)
ex1 : lecture d'un fichier de nombre réels dans un vecteur :

```
fid=fopen('nomfich.ext','r');
V = fread(fid,1000,'float') ;
fclose(fid);
```

ex2 : lecture d'un fichier image binaire 8 bits dans une matrice :

```
fid=fopen(fichier_raw,'r');
[IMRAW, COUNT] = fread(fid,[3786,3424],'uint8');
fclose(fid);
```

voir **`help fread`** pour plus de détails.

L'instruction d'écriture **`fwrite`** :
Permet d'écrire des données binaires dans un fichier selon un format précisé par l'utilisateur.
Taper **`help fwrite`** pour la syntaxe.

V-3- lecture/écriture des fichiers texte

L'instruction **`textread`** :
Cette instruction permet de lire un fichier ascii organise en lignes/colonnes de données de différents types.
syntaxe : `[A,B,C,...] = TEXTREAD(FILENAME,FORMAT)`

Par exemple, pour un fichier avec deux colonne de string, une colonne de réels, deux colonne d'entiers et une derniére colonne de string. (NB: string = chaine de caractères).
[names,types,x,y,z,answer] = textread('mydata.dat','%s %s %f %2d %s');

L'écriture d'un ficher ascii se fait en utilisant l'instruction **`save`** avec l'option -ascii.

VI - Exemples d'application

Ex1 : TP2 TÉLÉ : en annexe

(Pour les étudiants du cours Geo6333.)

Exemple 2 : Calcul des matrices de cooccurrences

```
% lecture de l'image
fid=fopen('b1juin.raw','r');
IM = fread(fid,[3786,3424],'uint8') ;
fclose(fid);

% affichage de l'image
image(IM);
colormap(gray),image(IM/2);
IM=IM';
colormap(gray),image(IM/2);grid;

% Découpage d'une imagette
```

```
IMMTL=IM(1200:2200,100:1200);
size(IMMTL),
colormap(gray),image(1200:2200,100:1200,IMMTL/2);

% Calcul de la matrice de cooccurrence
s=size(IMMTL);M=s(1);N=s(2);
K=255;%max(max(I))-min(min(I))+1;

alpd=input('  Angle en degrés   :    ') ;
alpha=alpd*pi/180;
r=input(' décalage  :  ');
rsi=round(r*sin(alpha));
rco=round(r*cos(alpha));

for i=1:K,
   for j=1:K,
      Cooc(i,j)=0;
   end
end

for x=1:M,
   for y=1:N,
     if (x+rsi)>=1, if (y+rco)>=1, if (x+rsi)<M+1, if (y+rco)<N+1,
    Cooc(IMMTL(x,y),IMMTL(x+rsi,y+rco))=Cooc(IMMTL(x,y),IMMTL(x+rsi,y+rco))+1;
     end;end;end;end
     if (x-rsi)>=1, if (y-rco)>=1, if (x-rsi)<M+1, if (y-rco)<N+1,
    Cooc(IMMTL(x,y),IMMTL(x-rsi,y-rco))=Cooc(IMMTL(x,y),IMMTL(x-rsi,y-rco))+1;
     end;end;end;end
   end
end

image(Cooc);title('Coocurence');pause,
mesh(Cooc),pause,

% calcul des parametres de texture lies a la matrice de cooccurence

% Homogénéité
Nc=(max(max(IMMTL))-min(min(IMMTL))+1)^2;
Homog=(1/Nc)^2*sum(sum((Cooc.^2)));
disp(Homog),

% Contraste
L=max(max(IMMTL))-min(min(IMMTL))+1;
for l=1:L,
  for a=1:K,
    cont1(l,a)=0;cont2(l,a)=0;cont(l,a)=0;
  end
end
for l=1:L,
  for a=1:K,
    if a <= K-(l-1), cont1(l,a)=Cooc(a,a+(l-1));end
    if a >= 1+(l-1), cont2(l,a)=Cooc(a,a-(l-1));end
    cont(l,a)=cont1(l,a)+cont2(l,a);
  end
    contv(l)=((l-1)^2)*sum(cont(l,:));
end
contrast=(1/(Nc*(L-1)^2))*sum(contv),
```

```
% Entropie
for i=1:K,
  for j=1:K,
     if Cooc(i,j)==0, ent(i,j)=0;
     else, ent(i,j)=Cooc(i,j)*log(Cooc(i,j));end
 end
end
 sent= sum(sum(ent)) / (Nc*log(Nc));
 entrop=1-sent;
 entrop,

%%  Directivité
for i=1:K, vdir(i)=Cooc(i,i); end
direc=sum(vdir)/Nc;

%%  Uniformité
for i=1:K, vunif(i)=Cooc(i,i)^2; end
unifor=sum(vunif)/(Nc^2); unifor,
```