Annexe 4 : Matlab

Le but de ce document et de se familiariser ou se rappeler des principales commandes MATLAB (MATrix LABoratory). Cette aide mémoire est bien évidemment non exhaustif, et on consultera l'aide MATLAB ainsi que les guides disponibles sur le site du cours (et/ou Google).

1 Généralités

Au lancement de Matlab, il faut tout d'abord s'assurer d'être dans le bon répertoire de travail. La commande `pwd` vous indiquera ou vous êtes actuellement. La fenêtre principal contient le prompt `>>` indiquant que MATLAB attend les instructions. Si celles-ci se terminent par `;` alors le résultat ne sera pas affiché. Plusieurs instructions peuvent figurer sur une même ligne si elles sont séparées par une virgule ou un point virgule.

1.1 Aide

Le lancement de l'aide peut s'effectuer de plusieurs façons. Tout d'abord en tapant `help` suivi du nom de la commande (aide en ligne), exemple

```
>> help det
```

ou en tapant directement le nom de la fonction dont on désire de l'aide dans la barre de recherche en haut à droite de l'interface (aide html).

La commande `who` affiche toutes les variables utilisées dans l'espace de travail (workspace), la commande `clear y` permet d'effacer la variable y. La commande `clear all` permet de tout effacer.

1.2 Scripts et fonctions

Le script est une séquence d'instructions ou de commandes, directement exécutable. Pour être exécuté par MATLAB, celui-ci doit être enregistré avec l'extension '.m' : un m-file. Par exemple, si le script est enregistré avec le nom `test.m` alors on peut l'exécuter en tapant simplement `test` dans MATLAB.

Les fonctions sont généralement dans des fichiers séparés. On définit une fonction de la manière suivante

```
function [s1,s2,...,sn] = nomfonction (e1,e2,...,en)
  séquence d'instructions
```

où

- `s1,s2,...,sn` sont les sorties de la fonction
- `e1,e2,...,en` sont les entrées de la fonction
- `séquence d'instructions` est le corps de la fonction

Les fonctions sont enregistrés dans un fichier de nom `nomfonction.m` (le nom de la fonction doit correspondre au nom du fichier pour pouvoir être appelé en dehors). En pratique on écrit un script contenant les appels aux différentes fonctions utiles pour les calculs effectués. Ce que l'on doit mettre ou non dans une fonction ou dans le script est choix de l'utilisateur. On fera en sorte d'avoir un script le plus épuré et le plus lisible possible. Pour le devoir, il est conseillé d'avoir une seule fonction par fichier (dans ce cas le `end` à la fin de la fonction est inutile).

Remarques/conseils :

— On commentera les fonctions en début de fichiers, par exemple

```
%fonction permettant de renvoyer le max de (a,b)
function [m] = maxDeAB (a,b)
m=max(a,b);
end
```

De manière générale, on abusera des commentaires (des bons commentaires !). Si le fichier ne contient qu'une seule fonction le `end` est facultatif.

— Il est bon de noter que **le type des sorties dépend du type des entrées ! ! !** Dans l'exemple précédent, si a et b sont des réels, la sortie ne sera pas la même qui si a et b sont des vecteurs (ou des matrices) !

— Il est bon de commencer un script avec un `clear variables; close all`, ce qui permet de nettoyer le workspace et ainsi d'éviter d'utiliser par inadvertance des variables qui ne devraient plus être utilisées.

1.3 Afficher des informations et variables

L'affichage de variables et d'informations peut se faire de plusieurs façons. Pour afficher la valeur d'une variable, on peut simplement retirer le ; à la fin. Cela est souvent utile pour debugger. Pour un affichage plus maîtrisé, voici deux manières d'afficher du texte et des variables sur une même ligne

```
fprintf('La valeur de x est : %g et la valeur de y est : %g \n',x,y)
disp(['La valeur de x est : ', num2str(x),' et la valeur de y est : ',num2str(y)])
```

2 Variables et opérations

Il n'est pas utile de déclarer le type de la variable en MATLAB. Le type est automatiquement établi suivant la valeur affectée à la donnée.

2.1 Vecteurs et matrices

On construit un vecteur ligne de la manière suivante

```
>> x=[1,2,3]; %vecteur ligne
```

Un vecteur colonne s'obtient en séparant les entrées par un ;

```
>> y=[1;2;3]; %vecteur colonne
```

voici plusieurs commandes utiles

```
>> x=-2:0.1:10  % vecteur ligne allant de -2 à 10 par pas de 0.1, x=[-2, -1.9,...,10]
>> length(x)    % longueur du vecteur x
>> x'           % transconjugée du vecteur x (transposé si x réel)
>> norm(x,inf)  % norme infinie de x
>> x(1)         % premier élément du  vecteur x
>> x(end)       % dernier élément du  vecteur x (équivalent à x(length(x)))
```

En MATLAB, **le premier indice d'un vecteur est obligatoirement 1**. Il faut faire très attention aux opérations avec les vecteurs et les matrices. Par exemple, pour deux vecteurs x et y, l'opération `x*y` est différente de l'opération `x.*y`. Pour la première opération, si les deux vecteurs x et y sont des vecteurs lignes (ou colonnes) MATLAB va générer un message d'erreur indiquant que les dimensions ne sont pas bonnes, car MATLAB fait l'opération `*` au sens de l'algèbre linéaire. Pour la deuxième opération, MATLAB réalise l'opération «terme à terme», le résultat de `x.*y` sera alors un vecteur, dont les composantes seront $x_1y_1, x_2y_2, \ldots$ Voici un exemple

```
>> x=[1 2 3];
>> y=[4 5 6];
>> x.*y
ans =

     4    10    18
```

Une matrice se déclare de la manière suivante :

```
>> A=[1 2 3; 2 5 6; 7 8 9]        % matrice 3x3, 1ere ligne=1 2 3,...
>> A'           % transconjugée de la matrice (tranposée si matrice réelle)
>> norm(A,inf)  % norme infinie de la matrice
>> cond(A,inf)  % conditionnement pour la norme infinie
>> det(A)       % déterminant de la matrice
```

Voici un type de matrice qui vous sera très utile pour le devoir, **les matrices tridiagonales**. Voici une méthode simple permettant de créer une matrice tridiagonale

```
>> A=toeplitz([2 -1 zeros(1, 4)]) % matrice tridiagonale
ans =

     2    -1     0     0     0     0
    -1     2    -1     0     0     0
     0    -1     2    -1     0     0
     0     0    -1     2    -1     0
     0     0     0    -1     2    -1
     0     0     0     0    -1     2
```

À noter que ces matrices contiennent beaucoup de 0, pour ne stocker en mémoire que les valeurs différentes de 0, on pourra utiliser `A=sparse(A)`. Pour voir la structure de la matrice (les non-zéros) on pourra utiliser la commande `spy(A)`.

Enfin pour résoudre un système $Ax = b$, on pourra utiliser la commande `x=A\b` (MATLAB fait alors une méthode LU avec pivotage).

2.2 Boucles `for` et `while`

La syntaxe des boucles est relativement classique. Attention, pour les boucles `for` les indices sont forcément des entiers (`i` n'a pas besoin d'être «déclaré» avant)

```
for i=1:10
  i
end
```

```
for i=x(1):x(end)
    x(i)
end

while (x>y) && (a<b)
  ...
end
```

3 Graphiques

La syntaxe pour afficher un graphique est la suivante

```
plot(abscisse1, ordonnee1, abscisse2, ordonnee2)
```

ceci affichera deux courbes définies par les vecteurs entre les parenthèses, par exemple

```
>> x1=-2:0.01:5; y1=x1.^2;
>> x2=-3:0.1:2; y2=2+exp(x2).*cos(x2.^3)
>> plot(x1,y1,x2,y2)
```

On pourra ajouter des légendes à l'aide des commandes suivantes

```
>> l=legend('Fonction x^2', 'Fonction 2+exp(x)*cos(x^3)'); % affiche deux légendes
>> set(l,'fontsize',18);    % taille police légende
>> xlabel('x');             % légende axe x
>> ylabel('f(x)');          % légende axe y
>> set(gca, 'fontsize',16); % taille chiffres axe x et y
```

Il peut être pratique d'afficher dynamiquement des graphiques (par exemple quand la variable dépend du temps) dans une boucle. Voici une façon parmi d'autres de faire cela

```
x=0:0.01:10;
h = plot(x,sin(x),'-o','LineWidth',1.5) ; % graphique initial
legend('sin(i*x)');    % légende fixe pour tous les graphiques
axis([0 1 -1 1])       % axes fixes pour tous les graphiques
for i=2:100
    set(h(1),'xdata',x,'ydata',sin(i*x));
    drawnow;
end
```

Il est également possible d'enregistrer le résultat de l'animation dans une vidéo, voici toujours un exemple rapide

```
video_sol = VideoWriter('solution.mp4','MPEG-4');
open(video_sol);
x=0:0.01:10;
h = plot(x,sin(x),'-o','LineWidth',1.5) ; % graphique initial
legend('sin(i*x)');    % légende fixe pour tous les graphiques
axis([0 1 -1 1])       % axes fixes pour tous les graphiques
```

```
for i=2:100
    set(h(1),'xdata',x,'ydata',sin(i*x));
    drawnow;
    f=getframe;
    writeVideo(video_sol,f);
end
close(video_sol);
```

La vidéo sera enregistrée dans le même répertoire que celui ou se trouve le script, sous le nom indiqué dans la première ligne.

4 Variables globales

Afin d'éviter d'avoir des fonctions avec un trop grand nombre de paramètres, il peut être utile de passer par des variables globales. L'idée est de définir ces variables dans le script principal, et de donner connaissance de ces variables aux fonctions qui en ont besoins. Par exemple, on pourrait avoir ceci dans le script `mon_script.m`

```
global longueur vitesse
longueur=1;
vitesse=20;
```

ces variables apparaissent en bleu dans MATLAB pour indiquer clairement que ce sont des variables globales. Supposons que `longueur` intervienne dans la fonction `ma_fonction.m`, mais je ne souhaite pas passer cette variable en paramètre d'entrée de la fonction. Je peux alors faire

```
fonction y=ma_fonction(x1,x2,x3)
global longueur
y=longueur*(x1+x2+x3)
```

`longueur` vaut alors la valeur définie dans le script `mon_script.m`. **On sera extrêmement prudent avec l'utilisation des variables globales**, en effet si vous modifiez une variable globale dans une fonction, vous modifiez sa valeur pour toute la suite du script `mon_script.m`. En réservera l'utilisation des variables qui ne doivent pas changer, comme des constantes physiques.