# TRAVAUX PRATIQUES

# INITIATION MATLAB

Matlab est un logiciel à licence commerciale qui fournit un environnement de calcul pour des applications scientifiques. C'est à la fois un langage de script (en quelques lignes, il peut servir à illustrer un algorithme) et un langage de programmation permettant d'accéder facilement à de nombreuses librairies numériques : calcul matriciel, calcul intégral, équations différentielles, …

Matlab peut exécuter des instructions :

- en ligne de commande : prompt « >> », l'instruction est tapée puis validée à l'aide de la touche « Entrée », le résultat est affiché à la suite (sauf si la ligne se termine par un « ; »).
- dans un fichier de commandes (fichier de scripts d'extension .m) contenant des instructions au format texte.

Domaines d'application de Matlab :

- calculs mathématiques : calcul matriciel, traitement du signal, traitement d'images, …
- visualisation des données : 2D, 3D, cartographie, animation, …
- traitement de fichiers : lecture/écriture dans un fichier, import/export (images, animations), …

En Matlab tout est matrice : ce langage est basé sur le principe que tout calcul (programmation ou tracé graphique) peut se faire à partir de matrices rectangulaires de nombres. Les scalaires (réels ou complexes) sont des matrices 1x1. Les vecteurs sont des matrices nx1 (matrices colonnes) ou 1xn (matrices lignes).

**Exercice 1**

1) Construction

Les objets ne sont jamais déclarés ou alloués explicitement, ils ont un type dynamique et leur taille peut changer dynamiquement. Voici quelques exemples à taper en ligne de commande :

```
>>A=[1,2,3;2,5,-3]            % définit une matrice 2x3 (le ";" sépare les lignes)

>>A'                          % matrice transposée de A

>>B=zeros(2,2)                % définit la matrice nulle de dimension 2

>>A=[A,B]                     % la taille de A augmente (concaténation de A et B)

>>clear('A'),clear('B');      % destruction explicite de A et de B (attention :
                              % chaîne de caractère en paramètre)

>>A=rand(2,3)                 % définit une matrice 2x3 dont les coefficients sont
                              % des nombres aléatoires entre 0 et 1

>>B=ones(10,3)                % définit une matrice 10x3 dont les 30 coefficients
                              % sont initialisés à 1

>>B=eye(4,4)                  % définit la matrice unité en dimension 4

>>who                         % affiche toutes les variables

>>clear, who                  % toutes les variables définies par l'utilisateur ont
                              % disparu

>>A=1.5:0.2:3.5               % crée un vecteur dont les coefficients vont de 1.5
                              % à 3.5 de façon croissante et avec un pas de 0.2
```

```
>>A=3.5:-0.2:1.5              % crée un vecteur dont les coefficients vont de 3.5
                              % à 1.5 de façon décroissante et avec un pas de -0.2

>>A=linspace(0,1,11)          % définit un vecteur ligne à 11 composantes
                              % régulièrement réparties entre 0 et 1

>>b=rand(3,1)                 % définit une matrice colonne à 3 composantes dont
                              % les coefficients sont générés de façon aléatoire

>>C=diag(b)                   % définit une matrice diagonale dont les éléments
                              % diagonaux sont les composantes de b

>>size(C)                     % renvoie les dimensions de la matrice B

>>[m,n]=size(C)               % idem mais les dimensions sont récupérées dans les
                              % variables m (lignes) et n (colonnes)

% ne pas confondre
>>v=[0;10];            % matrice colonne dont les 2 coefficients sont 0 et 10
>>v=[0,10];            % matrice ligne dont les 2 coefficients sont 0 et 10
```

2) Opérations

*a.* Modification

On peut modifier un élément, une ligne, une colonne d'une matrice, pourvu que le résultat reste une matrice.

Exemple (ligne de commande) :

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

>>A(1,3)=27                   % affecte la valeur 27 au coefficient situé à
                              % la 1ère ligne et 3ème colonne

>>A(:,[1,3])=[10,30;40,60]    % modification des colonnes 1 et 3 avec
                              % affectation de nouvelles valeurs

>>A(1,:)=[1,2,3]              % modification de la ligne 1

>>A(1,:)=0                    % mise à zéro des coefficients de la ligne 1

>>A(:,2)=[]                   % suppression de la 2ème colonne (pour supprimer
                              % des éléments)
```

*b.* Concaténation

La concaténation consiste à juxtaposer des matrices lorsque les dimensions sont compatibles.

Exemple (ligne de commande) :

```
>>A=[1,2,3 ;4,5,6]
>>B=[-1,-2 ;-3,-4]
>>C=rand(3,3)
>>M=[A,B]           % concaténation horizontale
>>M=[A;C]           % concaténation verticale (attention au « ; »)
>>M=[A,A']          % erreur (incompatibilité des dimensions)
>>M=[A;A']          % idem
```

```
>>M=[C,C']          % OK car C est une matrice carrée
>>M=[C;C]           % idem
>>M=[A,[10;20];[7,8,9],30]
```

*c.* Opérations algébriques

On peut additionner, soustraire, multiplier, diviser terme à terme les coefficients d'une matrice. On peut également multiplier et inverser des matrices, appliquer une fonction à une matrice …

Exemple (ligne de commande) :

```
>>A=[1,2,3;4,5,6]; B=[7,8,9;10,11,12];          % deux matrices 2x3

>>C=[2,4,3;5,-7,1;-1,2,4];                      % une matrice 3x3

>>A+B        % addition des matrices A et B

>>A.*B       % produit terme à terme (attention « .* » et non
             % « * »)à ne pas confondre avec le produit matriciel

>>A./B       % division terme à terme

>>A*C              % produit matriciel de A et C (résultat : matrice 2x3)

>>C*A              % erreur (dimensions incompatibles)

>>A+10             % on ajoute 10 à chaque coefficient de A

>>(1)./A           % matrice dont les coefficients sont les inverses des
                   % coefficients de A

>>10*A             % multiplication par un scalaire
```

3) Programmer avec Matlab

Matlab est également un langage de programmation qui permet d'écrire des scripts (ensemble d'instructions Matlab) et des fonctions dans des fichiers d'extension « `.m` ».

*a.* Création de programmes (scripts)

- taper le programme suivant dans un fichier de script (Onglet « HOME » puis « New ») :

```
clear variables;  % efface les variables
close all;        % ferme toutes les fenêtres

x=0:0.01:2*pi;    % vecteur ligne dont les composantes vont de
                  % 0 à 2π avec un pas de 0.01

y=exp(-x/2).*cos(2*x).^2;   % vecteur ligne de même dimension que x
                            % Les composantes de y sont les images
                            % des composantes de x par la fonction
                            % x ⟼ e^(-x/2) cos²(2x)

plot(x,y,'color','b','linewidth',2);  % affiche la courbe de la
                                      % fonction x ⟼ e^(-x/2) cos²(2x) sur
                                      % l'intervalle [0,2π]
grid on;          % affiche un quadrillage
```

- sauvegarder le fichier au format « .m ».
- pour exécuter le programme cliquer sur l'icône Run (triangle vert) dans l'onglet « EDITOR », ou en tapant directement sur la touche F5.

*b.* Création d'une fonction

Une fonction est un script qui commence par le mot « function ». Il reçoit en entrée 0, 1 ou plusieurs paramètres et il donne en sortie 0, 1 ou plusieurs paramètres. Les paramètres d'entrée et/ou de sortie peuvent être des matrices.

Syntaxe

```
function [y1,y2,…,yn]= MyFunction(x1,x2,…,xp)
```

où x1,x2,…,xp sont les paramètres d'entrée, et y1,y2,…,yn les paramètres de sortie.

Exemple

- Taper le script suivant et enregistrer le dans un fichier portant obligatoirement le même nom que celui de la fonction (c'est-à-dire ici MyFunction.m) :

```
% Fonction MyFunction
% Calcule le produit scalaire (r1) et les produits
% matriciels (r2 et r3) de deux vecteurs (a et b)
function [r1,r2,r3]=MyFunction(a,b)

    if length(a)~=length(b)
        error('Les vecteurs doivent être de même longueur !');
    end
    r1=sum(a.*b);  % produit scalaire (remarquer le '.' devant '*')
    r2=a'*b;       % produit matriciel 1 (ici pas de '.')
    r3=a*b';       % produit matriciel 2 (idem)
```

- La fonction est appelée dans le programme principal dont le script est donné ci-dessous :

```
clear variables;  % efface les variables
close all;        % ferme toutes les fenêtres
clc;              % efface le contenu de la zone de commande

u=[2,-1,3,7,1];   % vecteur ligne
v=[1,4,-3,8,2];   % vecteur ligne (de même dimension que u)
[ps,pm1,pm2]= MyFunction(u,v); % appel de la fonction MyFunction

disp('Produit scalaire :');disp(ps);
disp('Produit matriciel 1 :');disp(pm1);
disp('Produit matriciel 2 :');disp(pm2);
```

- Enregistrer ce script dans un fichier nommé MyProg.m (par exemple) et le placer dans le même répertoire que le script MyFunction.m.
- Exécuter le programme principal MyProg.m.

*c.* Fonction anonyme

Lorsque que le code d'une fonction n'est pas trop long, on peut définir la fonction directement dans le script du programme principal ; on parle alors de *fonction anonyme*. Reprenons le programme vu en 3) *a*.

```
clear variables;  % efface les variables
close all;        % ferme toutes les fenêtres
```

```
% implémentation de la fonction x ⟼ e^(-x/2) cos²(2x) sous forme de fonction
% anonyme
f=@(x)(exp(-x/2).*cos(2*x).^2);  % f est le nom de la fonction,
                                 % x est le paramètre d'entrée

x=0:0.01:2*pi;     % vecteur ligne dont les composantes vont de
                   % 0 à 2π avec un pas de 0.01

y=f(x);            % appel de la fonction anonyme f, on obtient un
                   % vecteur ligne de même dimension que x
                   % Les composantes de y sont les images
                   % des composantes de x par la fonction
                   % x ⟼ e^(-x/2) cos²(2x)

plot(x,y,'color','b','linewidth',2);  % affiche la courbe de la
                                      % fonction x ⟼ e^(-x/2) cos²(2x) sur
                                      % l'intervalle [0,2π]
grid on;           % affiche un quadrillage
```

Taper et exécuter ce code.

*d.* Fonction graphique `plot`

Taper le code suivant dans un nouveau script Matlab. Ce script donne des exemples d'utilisation de la commande `plot` qui permet de tracer un graphique dans une fenêtre (noter l'absence de boucle `for`). On essaiera d'en comprendre chaque ligne.

```
clear variables;  % efface les variables
close all;        % ferme toutes les fenêtres
clc;              % efface le contenu de la zone de commande

x=0:0.1:2*pi;
y1=sin(x);
y2=sin(2*x);


figure(1);

subplot(2,2,1);
plot(x,y1,'*r',x,y2,'ob');

subplot(2,2,2);
plot(x,y1,'r',x,y2,'b');
axis equal;
xlabel('x');
ylabel('f(x)');

subplot(2,2,3);
plot(x,y1,'*r-',x,y2,'ob-');
axis([0,2*pi,-1,1]);
xlabel('x');
ylabel('f(x)');
title('Fonction graphique plot');

subplot(2,2,4);
hold on;
plot(x,y1,'color',[1,1,0],'linewidth',3);
plot(x,y2,'color',[1,0,1],'linewidth',4);
axis([0,2*pi,-1.2,1.2]);
xlabel('x');
ylabel('f(x)');
title('Fonction graphique plot');
legend('sin(x)','sin(2x)');
```

```
figure(2);
hold on;
x=0:0.01:2*pi;        % on a changé la valeur du pas
z=sin(x).*sin(2*x);   % on remarquera le .*
plot(x,z,'color',[0,0,0],'linewidth',3);
```

**Exercice 2**

1) Consulter l'aide en ligne pour connaître la syntaxe des boucles sous Matlab (commandes `for` et `while`). Pour cela taper, en ligne de commande :

```
>> doc « nom de la commande »
```

Par exemple, pour avoir des informations sur la commande `for` :

```
>> doc for
```

2) On considère la série suivante :

$$\sum_{n=1}^{+\infty} \frac{1}{n^2}$$

*a.* Calculer la somme des 15 premiers termes de cette série.

*b.* Cette série converge vers $\pi^2/6$. Combien de termes faut-il ajouter pour obtenir ce résultat avec une précision de $10^{-3}$ ?

3) On considère la série suivante :

$$\sum_{n=1}^{+\infty} \frac{(-1)^{n+1}}{n}$$

Calculer une valeur approchée à $10^{-4}$ près de la somme de cette série et donner le nombre de termes à ajouter pour y parvenir.

Indication : la valeur exacte de la somme de cette série est $\ln 2$ (commande `log`).

**Exercice 3**

*Les 2 questions sont indépendantes.*

1) En utilisant la commande `trapz` de Matlab (taper `doc trapz` pour plus d'informations) vérifier que :

$$\int_0^1 \frac{x+3}{\sqrt{2x+1}}dx = 3\sqrt{3} - \frac{8}{3} \quad \text{et} \quad \int_0^{+\infty} \frac{\sin^4 x}{x^4}dx = \frac{\pi}{3}$$

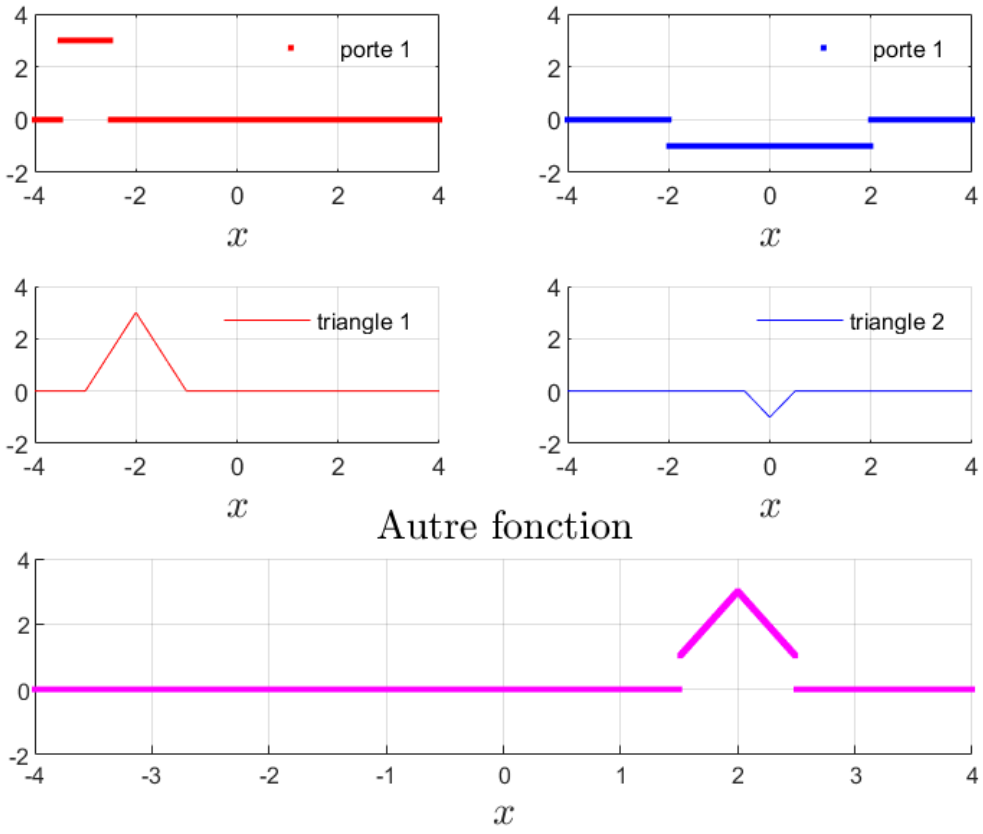
Votre programme ne doit contenir aucune boucle.

2) On propose les deux fonctions suivantes pour tracer les courbes représentatives d'une fonction porte et d'une fonction triangle à l'aide de Matlab :

```
function y=porte(x)
    y=abs(x)<0.5;
```

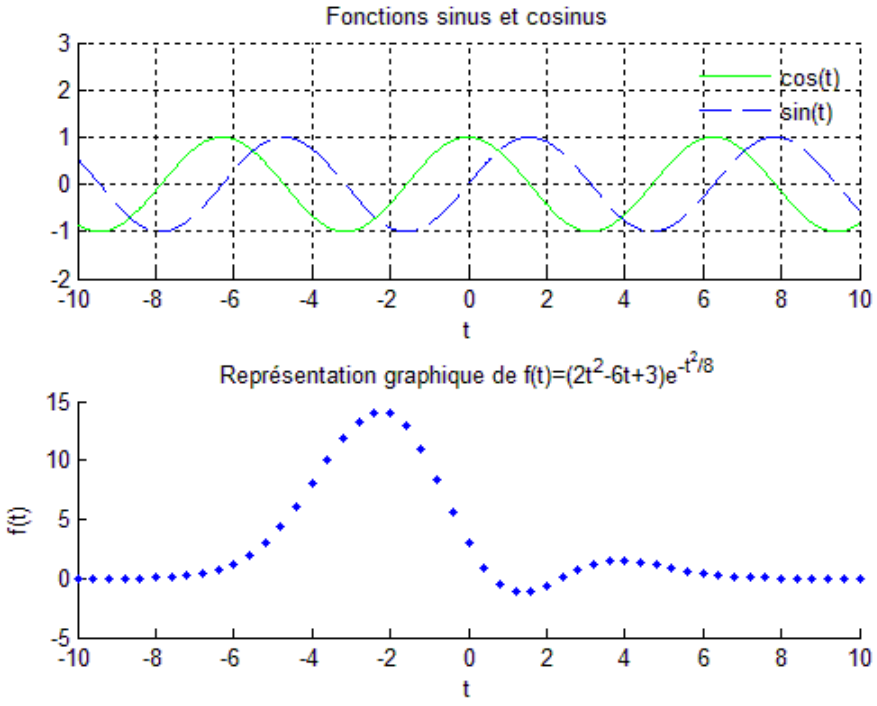
```
function y=triangle(x)
    y=(1-abs(x)).*(abs(x)<1);
```

- Copier ces fonctions chacune dans un fichier, l'un portant le nom `porte.m` et l'autre `triangle.m`
- Ecrire un programme Matlab (ne contenant aucune boucle) permettant d'obtenir la figure ci-dessous (utiliser les commandes `subplot, plot, legend, title, xlabel, axis, grid, …`):




**Exercice 4**

1) Ecrire un programme Matlab (sans aucune boucle `for` ou `while`) permettant d'afficher la figure suivante :

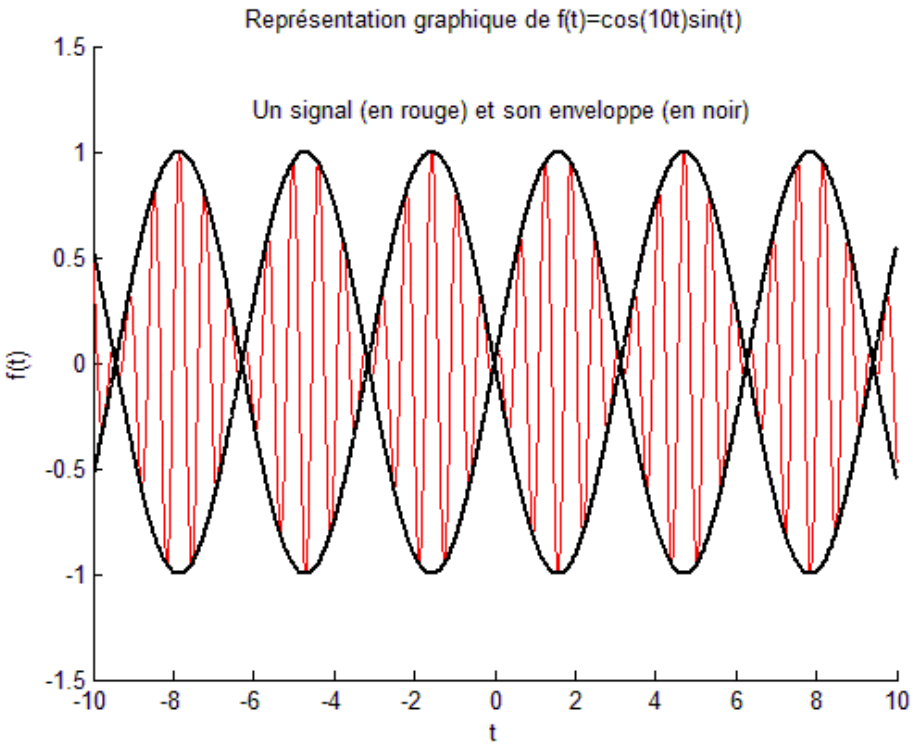



(voir indications page suivante)

Indications :

- Pour l'axe des abscisses, créer un vecteur ligne nommé `t` dont les valeurs sont comprises entre -10 et +10 avec un pas de 0.01
- Graphique du haut : utiliser les commandes `plot, legend, xlabel, title, grid, ylim`
- Graphique du bas :
  - afficher les points bleus avec un pas de 40 (par exemple pour les abscisses `t(1:40:end)`)
  - utiliser les commandes `plot, xlabel, ylabel, title`

2) Compléter votre programme afin d'afficher la figure suivante (toujours sans boucle `for` ni `while`) :




Indication : utiliser les commandes `plot, xlabel, ylabel, title, text, ylim`