TP 1

Prise en main de MatLab

1.1 Objectif du TP

`MatLab` *(MATrix LABoratory)* est un logiciel de calcul plus puissant qui permet de réaliser des programmes structurés, travailler sur des matrices de grande dimension, faire des calculs sur des nombres complexes et tracer des graphes en deux et trois dimensions ($2D$ et $3D$). Ce logiciel est considéré comme un langage de programmation similaire aux langages C et Pascal. En revanche, sa particularité est qu'il s'agit d'un langage interprété (c'est-à-dire, les instructions sont exécutées immédiatement après avoir été tapées). `MatLab` présente comme avantage la disponibilité d'un nombreux algorithmes et fonctions de calcul numérique et traitement du signal, ainsi que l'existence d'une aide en ligne (commande `help`). De plus, il constitué d'un noyau de base extensible à l'aide de nombreuses boîtes à outils (toolboxes) et des boites à outils personnelles.

L'objectif de ce TP est de familiariser l'étudiant avec logiciel `MatLab` qui sera utilisé pour tous les TP de traitement de signal. Il s'agit donc de lui présenter :

— Les fonctionnalités de base et les commandes les plus utiles qui interviennent dans la simulation de différents types des signaux.

— L'objet de base manipulé par `MatLab`, à savoir les vecteurs et les matrices.

— L'architecture des scripts, fonctions, ainsi que les structures de contrôles qui peuvent être utiles dans le traitement du signal.

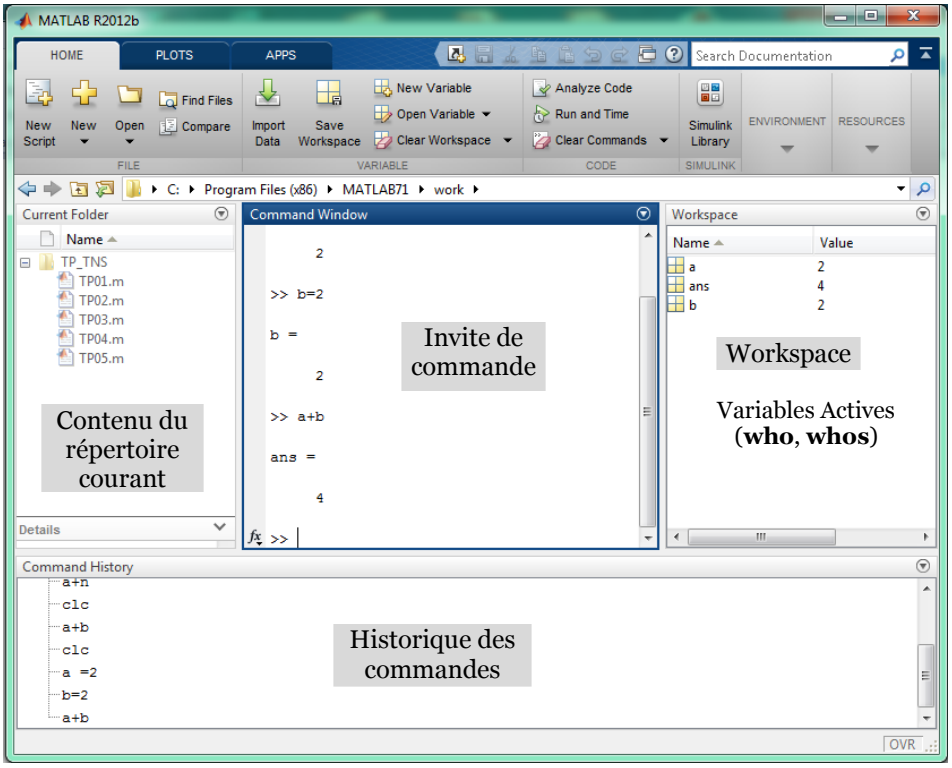
— Les commandes qui permettant de tracer les graphiques en 2D et 3D.

— La modélisation effectuée avec Simulink.

1.2 Bref descriptif de MatLab

Après avoir cliqué sur l'icône *MatLab* dans WINDOWS, vous vous trouvez dans la feuille de calcul de MatLab que l'on appellera **fenêtre** MatLab.

A partir du menu déroulant **View**, quelques fenêtres peuvent être ouvertes : les fenêtres : **Répertoire courant** *(current directory)*, **Commandes** *(command)* et **Historique** *(command history)*.



Fenêtre *Répertoire courant* : Cette fenêtre permet de sélectionner le répertoire de travail et de visualiser son contenu. Le répertoire dans lequel vous souhaitez travailler doit être sélectionné.

Fenêtre *Command Window* : Cette fenêtre permet de saisir des valeurs, de taper des instructions ligne par ligne et d'exécuter directement des programmes (en tapant le nom du programme). L'exécution des instructions et des programmes doit être validée par la touche « entrée » du clavier pour être prise en compte par MatLab.

Fenêtre *Historique* : Cette fenêtre contient toutes les instructions saisies dans la fenêtre de commandes. L'ouverture du fenêtre command history permet de voir cette liste. De plus, nous pouvons effacer le contenu de cette fenêtre en sélectionnant l'option "clear command history" dans le menu déroulant "Edit".

1.3 Les variables sous MatLab

Le logiciel MatLab gère de façons différentes les nombres entiers, réels, complexe, chaines de caractère ainsi que les tableaux de nombre.

Généralités sur les variables

Dans MatLab :

— Les variables sont définies à l'aide d'expressions ;

— Le nom de variable validé par MatLab est : lettre + nombre quelconque de lettres, chiffres ou _. Par exemple : `nom_de_variable` ;

— Aucune déclaration préalable de type de variable ou de dimension de tableau/vecteur est requiert ;

— Majuscules $\neq$ minuscules dans les noms de variables, fonctions ... etc. Par exemple : variable $abc \neq Abc$;

— Les noms de toutes les constantes et fonctions prédéfinies sont en minuscules ;

— Lorsqu'un nouveau nom de variable est déclaré, MatLab crée la variable correspondante et y associe l'espace de stockage approprié dans le *Workspace*. Dans ce cas, si la variable existe déjà, son contenu est changé et un nouvel espace de stockage est alloué en cas de redimensionnement de tableau.

Types de nombres (réels, complexes, entiers)

Réels : 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 opérateurs ou fonctions qui mélangeant des paramètres de types entier et réel, donnent un résultat de type entier. Alors que, dans MatLab certaines opérations qui mixent 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.

Complexes : stockés sous forme de nombres réels double precision

Exemples : `nb_complexe = 4e-13 - 5.6i ou -45+5*j`.

La lettre j est utilisée pour noter le nombre complexe j ($j^2 = -1$).

On peut également extraire la partie réelle, la partie imaginaire, le complexe conjugue, le module et l'argument d'un nombre complexe (voir Annexe).

Format des nombres

Dans MatLab, les calculs sont exécutes avec une grande précision, mais les résultats ne sont affichés, de façon standard, qu'avec 4 chiffres après le point décimal. Pour afficher le résultat de façon plus précise, taper `format long`, et pour revenir au format standard, taper `format short`.

Dans l'exemple qui suit, la saisie de l'un des instructions suivie de pi donne différents valeurs de π.

`format short`	3.1416
`frmat long`	3.141592653589793
`format short e`	$3.1416e+00$
`format long e`	$3.141592653589793e+00$
`format hex`	$400921fb54442d18$

Chaînes de caractères

L'instruction `strfun` permet de voir la liste des fonctions relatives aux chaînes de caractères :

Pour définir une chaıne de caractères, on utilise les quotes `'`. Par exemple, `mot = 'abcd'` définit une chaine de caractère stockée dans une mémoire appelée mot et contenant 4 caractères.

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

Les informations concernant la manipulation des chaines de caractères peuvent être obtenues dans l'aide par l'index string (=chaine 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.

Exemple : `section = 'Département d''électronique'`.

L'instruction `string(i:j)` : partie de string comprise entre le i-ème et le j-ème caractère.

L'instruction `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 « *électronique* ».

Opérateurs arithmétiques

Pour connaitre la liste des opérateurs et caractères spéciaux sous MatLab taper la commande `helpwin ops`.

Ces operateurs arithmétiques S'appliquent à des scalaires, vecteurs ou matrices.

Opérateurs relationnels

Les opérateurs relationnels permettent de faire des tests numériques en construisant des "expressions logiques", c-à-d des expressions retournant les valeurs vrai (1) ou faux (0).

Pour des vecteurs ou matrices, ces operateurs relationnels s'appliquent à tous les éléments et retournent donc également des vecteurs ou des matrices.

Opérateurs logiques

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

Si les expressions sont des matrices, ces operateurs logiques s'appliquent à tous les éléments et retournent donc également des des matrices.

1.4 Vecteurs et Matrices

Définition

Une *matrice* est un tableau de nombres. Sur le plan mathématique d'algèbre linéaire, elle peut représenter une application linéaire.

Un *vecteur* est une matrice particulière. Du point de vue des mathématiques, les vecteurs se présentent sous formes de colonnes, mais en informatique, un vecteur est simplement une suite finie de nombres, mis sous forme de colonnes ou de lignes.

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

Il y a deux façons simples de définir un vecteur en MatLab :

Pour créer un *vecteur ligne*, on utilise : `V = [v1 v2 ... vn]`

Exemple : `V1 = [1 -4 5], V2 = [-3,sqrt(4)], V3 = [V2 V1 -3]`

Pour créer un *vecteur colonne*, on utilise : `V = [v1 ; v2 ;... ; vn]`

ou `V = [v1 ; v2 ;... ; vn]'` (c'est-à-dire la transposée du vecteur V)

Exemple : `V4 = [-3;5;2*pi], V5 = [11 ; v4], V6 = [3 4 5 6]'`

Il est aussi possible de créer un vecteur ligne ou colonne avec l'operateur « : » :

`V = [début:pas:fin]` ;

Si le pas est non spécifié, les vecteurs sont définit avec une incrémentation unité.

Exemple : `V7 = [1 :5], V8 = [1 :2 :10]'`

Adressage de vecteurs

Pour accéder à une valeur particulière d'un vecteur, on écrira par exemple

`V(i)` : qui donne simplement accès au ième élément du vecteur ligne ou colonne `V` ;

`V(i:p:j)` : donne accès aux éléments qui vont d'indices `i` à `j` du vecteur ligne ou colonne `V` avec un pas de 1 ou de p si spécifié ;

`V([i j k:l])` : donne accès aux éléments qui vont d'indices `i`,`j` et `k` à `l` ;

`V(:)` le symbole « : » sans autres précisions signifiant tout le vecteur.

Les opérations et fonction sur les vecteurs sont résumés dans l'annexe.

Affectation de matrices

Pour `MatLab`, une matrice est un tableau à 2 dimensions de `NxM` éléments (`N` lignes et `M` colonnes).

Il est aussi possible de créer une matrice avec l'operateur « [ ] », et accéder aux ses éléments avec l'operateur «( )» (comme pour les vecteurs).

Pour crée une matrice `n` lignes x `m` colonnes, on utilise :

```
M = [v11 v12 ... v1m ;
     v21 v22 ... v2m ;
     ... ... ... ... ;
     vn1 vn2 ... vnm ] ;
```

Exemples : `M1 = [-2:0 ; 4 sqrt(9) 3].`

`M2 = [[1 2 6]' [2 4 7]' [3 5 8]'].`

`V1 = 1:3:7 et V2 = 9:-1:7, M3 = [v2;v1].`

Adressage de matrices

Pour accéder à une valeur particulière d'un matrive, on écrira par exemple :

`M(i,j)` : qui donne accès au élément `(i,j)` du matrice `M` ;

`M(i:j,k:m)` : qui donne accès au sous-matrice (lignes `i` à `j`, colonnes `k` à `m`) ;

`M(i,:)` : qui donne accès aux éléments du ligne `i` ;

`M(i:j,:)` : qui donne accès aux éléments des lignes `i` à `j` ;

`M(:,k)` : qui donne accès aux éléments de colonne `k` ;

`M(:,k:m)` : qui donne accès aux éléments des colonnes `k` à `m`.

Il existe un certain nombre de commandes (voir annexe) qui génèrent automatiquement des matrices particulières comme une matrice de taille quelconque contenant des coefficients identiques.

Il est aussi possible d'utiliser des commandes pour créer des matrices non carrées, par exemple `eye(5; 3)`. Ainsi que, des matrices par blocs. Dans ce cas, il suffit d'écrire :

```
B = [eye(3) eye(3) ;ones(3) zeros(3) ;]
```

ou bien en utilisant les colonnes

```
B = [[eye(3) ones(3)]' [eye(3) zeros(3)]']
```

ou bien encore

```
B = [[eye(3) ; ones(3)] [eye(3) ;zeros(3)]]
```

1.5 Structure de Contrôles

Dans leur forme élémentaire, les structures de contrôle de `MatLab` fonctionnent de la même manière que dans la plupart des autres langages de programmation.

La boucle `for`

Cette boucle permet d'exécuter une séquence d'instructions de manière répétée. Cette boucle consiste à effectuer une boucle pour les valeurs d'un indice, incrémenté à chaque itération, variant entre deux bornes données (borne supérieur et borne inferieur).

Syntaxe :

```
for indice = borne_inf : pas: borne_sup
    <Séquence d'instructions>;
end
```

Où :

`indice` : est une variable appelée l'indice de la boucle ;

`borne_inf` et `borne_sup` : sont deux constantes réelles (paramètres de la boucle) ;

`séquence d'instructions` : le traitement à effectuer pour les valeurs d'indices variant entre `borne_inf` et `borne_sup` avec un pas défini (corps de la boucle).

Par exemple, étant donnée un nombre n, le programme suivant calcul $n!$.

```
n = 4;
nfac = 1;
for k = 1:n
     nfac = nfac*k;
end
nfac
```

La boucle `while`

Cette boucle permet d'exécuter une séquence d'instructions de manière répétée. Dans cette boucle les instructions sont exécutées de façon répétée tant que la condition est satisfaite.

Syntaxe :

```
while expression logique
     <séquence d'instructions>;
end
```

Où :

`expression logique` est une expression dont le résultat peut être vrai ou faux ;

`séquence d'instructions` est le traitement à effectuer tant que expression logique est vraie.

Le traitement de séquence d'instructions est exécuté sous forme d'une boucle tant que l'expression logique est vraie. Dans le cas où l'expression logique devient fausse, le traitement des séquences d'instruction est terminé et le programme passe immédiatement à l'instruction qui suit l'instruction de fin de boucle `end`.

Par exemple, le programme suivant détermine la fraction du nombre n ($n!$) :

```
n = 4;
k = 1;
nfac = 1;
while k <= n
       nfac = nfac*k;
       k = k+1;
end
nfac
```

La boucle `if`

Cette boucle permet d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée au préalable.

La boucle conditionnelle la plus simple a la forme suivante :

Syntaxe :

```
If expression logique
    <séquence d'instructions>;
end
```

Où :

`expression logique` est une expression dont le résultat peut être vrai ou faux ;

`séquence d'instructions` est le traitement à effectuer si expression logique est vraie.

La séquence d'instructions est exécutée uniquement pour un résultat de l'évaluation de l'expression logique vraie (c'est-à-dire vaut 1). Dans le cas contraire, l'exécution des séquences des instructions est arrêtée et le programme passe à l'exécution de l'instruction qui suit la fin de la boucle `end`.

Contrairement à certains langages de programmation, il n'y a pas de mot clé `then` dans cette instruction conditionnée. Notez également que la marque de fin de bloc conditionné est le mot clé `end` et non pas `endif`.

Il existe une séquence conditionnée sous forme d'alternatives :

Syntaxe :

```
If expression logique
    <séquence d'instructions 1>;
else
    <séquence d'instructions 2>;
end
```

Si expression logique est vraie, la séquence d'instructions 1 est exécutée. Dans le cas contraire, (c'est-à-dire dans le cas où l'expression logique est faux.) c'est la séquence d'instructions 2 qui est exécutée. Le déroulement du programme reprend ensuite à la première instruction suivant le mot clé `end`.

Il est possible d'effectuer un choix en cascade :

Syntaxe :

```
if expression logique 1
    <séquence d'instructions 1>;
elseif expression logique 2
    <séquence d'instructions 2>;
    ...
elseif expression logique N
    <séquence d'instructions N>;
else
    <séquence d'instructions par
défaut>;
end
```

Dans ce cas, si l'expression logique 1 est vraie la séquence d'instructions 1 est exécutée et le programme reprend ensuite à la première instruction suivant le mot `end`. Alors que si l'expression logique 2 est vraie la séquence d'instructions 2 qui est exécutée et le programme reprend ensuite à la première instruction suivant le mot `end`,... etc. Si aucune des expressions logiques 1 à N n'est vraie la séquence d'instructions par défaut qui est exécutée.

Généralement, un choix en cascade est utilisé lors d'initialisation de données. Par exemple, pour affecter à une matrice A des valeurs qui dépend de leur indice, on initialise la matrice de la manière suivante :

```
nrows = 10;
ncols = 10;
A = ones(nrows, ncols);
for r = 1:nrows
    for c = 1:ncols
        if r == c
           A(r,c) = 2;
        elseif abs(r - c) == 1
            A(r,c) = -1;
         else
            A(r,c) = 0;
         end
    end
end
```

La boucle `switch`

La boucle `switch` est une alternative de l'utilisation d'une séquence d'instructions conditionnées pour effectuer un choix en cascade existe.

Syntaxe :

```
switch var
case cst1,
   <séquence d'instructions 1>;
case cst2,
    <séquence d'instructions 2>;
    ...
case cstN,
    <séquence d'instructions N>;
otherwise
    <séquence d'instructions par défaut>;
end
```

Où :

`var` est une variable numérique ou chaîne de caractères ;

`cst1, ..., cstN` sont des constantes numérique ou chaîne de caractères ;

`séquence d'instructions i` : est la séquence d'instructions à exécuter si le contenu de la variable `var` est égal à la constante `csti` (`var == csti`).

Dans le cas de l'égalité de la variable `var` à l'une des constantes `cst1, ... cstN`, (par exemple `cst3`), la séquence d'instructions correspondante (ici séquence d'instructions 3) est exécutée. Le programme reprend ensuite à la première instruction qui suit la fin de la boucle `end`. Dans le cas contraire, c'est-à-dire la variable `var` n'est égale à aucune des constantes, la séquence d'instructions par défaut est exécutée.

Par exemple, le programme suivant affiche un texte diffèrent selon la valeur saisie par l'utilisateur :

```
mynumber = input('Enter a number:');
switch mynumber
case -1
  disp('negative one');
case 0
  disp('zero');
case 1
  disp('positive one');
otherwise
  disp('other value');
end
```

Interruption d'une boucle de contrôle

L'instruction `break` permet de sortir des boucles `for` et `while`. L'exécution de cette instruction se poursuit séquentiellement à partir de l'instruction suivant la fin de la boucle `end`.

L'instruction `return` provoque un retour au programme principal, et donc toutes les instructions qui suivant cette instruction ne sont pas exécutées. Généralement, elle est utilisée conjointement avec une structure de contrôle par exemple pour tester dans le corps d'une fonction si les paramètres d'entrée ont les valeurs attendues.

L'instruction `error(' message d''erreur ')` permet d'arrêter un programme et d'afficher un message d'erreur.

L'instruction `warning(' message de mise en garde ')` permet d'afficher un message de mise en garde sans suspendre l'exécution du programme. Pour indiquer au `MatLab` de ne pas

afficher les messages de mise en garde d'un programme en tapant `warning off` dans la fenêtre de commandes.

L'instruction `pause` permet d'interrompre l'exécution du programme. L'exécution normale reprend dès que l'utilisateur enfonce une touche du clavier. L'instruction `pause(n)` suspend l'exécution du programme pendant n secondes.

1.6 Scripts et fonctions

Les fichiers M-files qui ont une extension **.m* contient des instructions MatLab. Ces fichiers peuvent être édités avec tout éditeur de texte. Dans MatLab, on distingue deux types de M-files : les *scripts* (ou programmes MatLab) et les *fonctions*.

Les scripts sont des suites d'instructions enregistrées dans un fichier appelé fichier M d'extension *(*.m)*. Ces fichiers n'ont pas d'arguments d'entrée/sortie et travaillent dans le Workspace. Ils sont créés avec le logiciel MatLab en sélectionnant *Nouveau fichier* dans le menu déroulant *Fichier*.

Toutes les variables créées/modifiées lors de l'exécution d'un script sont des variables globales qui peuvent également être déclarées par global, et donc sont des variables visibles dans le Workspace.

Les fichiers scripts sont composés de trois blocs essentiels :

1. Bloc d'acquisition des données (déclarer les variables).
2. Bloc de traitement :
   - Programmer des opérations répétitives ;
   - Effectuer les opérations algébriques ;
   - Appeler les fonctions ;
3. Bloc d'affichage (tracer les courbes).

Les fonctions, quant à elles, sont des fichiers **.m* qui permet d'enregistrer une succession d'opérations en spécifiant des arguments d'entrée/sortie. Les fonctions n'interagissent avec le workspace qu'à travers leurs variables d'entrée/sortie, les autres variables manipulées restant internes (locales) à ces fonctions. Ces variables ne sont pas disponibles à l'invite MatLab.

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

L'instruction `function` permet donc de construire une nouvelle fonction. La fonction se développe dans un fichier différant. En revanche, à partir *d'arguments d'entrée*, elle détermine des *variables de sortie*.

Par exemple, la fonction $y = \dfrac{x^2}{(2+\sin(x))}$ définie sur R est programée. Dans ce cas, x est la variable d'entrée et y est la variable de sortie.

```
function y = mafonction(x);
y = x.^2./(2+sin(x));
```

Le fichier doit être sauvegardé et nommer de la même manière que la fonction. La fonction `mafonction.m` peut être ensuite appelée par un programme ou directement dans la ligne de commande. Par exemple, `mafonction(2)` donnera 1.3749.

Dans la fonction `mafonction.m`, le point *'.'* placé avant les opérateurs permet d'appliquer directement la fonction à un vecteur ou à une matrice.

Par exemple :

```
x = -50:0.2:50;
y = mafonction(x);
figure(1)
plot(x,y)
```

Toutes les variables utilisées dans la fonction `mafonction.m` sont locales, sauf les variables en argument et les variables de sortie. Par exemple, dans le programme suivant, la variable a est locale :

```
function y = mafonction(x);
a =1;
y = x.^2./(2+sin(x))+a;
```

```
function y = mafonction(x);
global a
y = x.^2./(2+sin(x))+a;
```

1.7 Représentation graphique sous MatLab

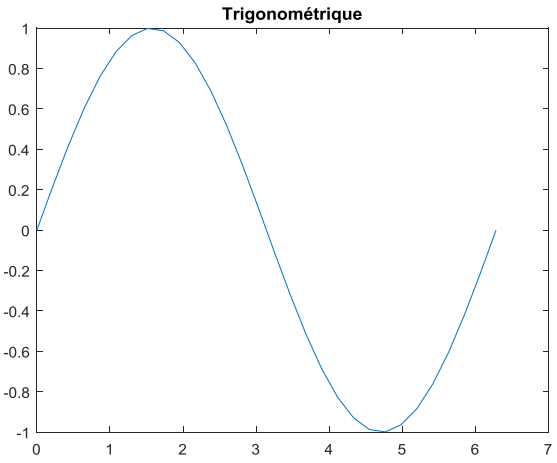
Les fonctionnalités graphiques de `MatLab` se séparent en trois catégories principales : les représentations $2D$ regroupées sous la terminologie `graph2d`, les représentations $3D$ regroupées sous la terminologie `graph3d` et les représentations spécialisées (maillage par exemple) sous la terminologie `specgraph`.

Graphiques à 2D

L'instruction `graph2d` permet de connaitre la liste des graphiques $2D$ et tableaux des fonctions disponibles.

Le programme suivant illustre un exemple :

```
x = linspace(0,2*pi,30);
y = sin(x);
figure(1);
plot(x,y, '-');
title('Trigonométrique');
```

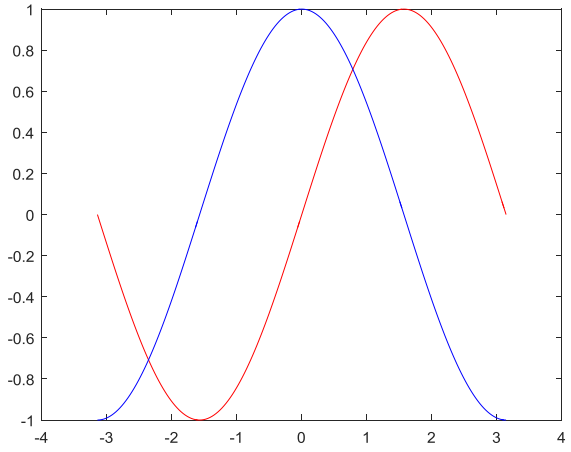


De nombreuses options existent sous `MatLab` pour **contrôler l'affichage** des courbes (Axe, couleur, commentaires, légende, échelle logarithmique,... etc) (voir Annexe).

Pour **ajouter une courbe** à un graphique existant, il suffit d'utiliser l'instruction `hold on`. Dans ce cas, les instructions graphiques qui suivent seront réalisées sur la même figure.

Par exemple, le programme suivant permet de tracer graphiquement en mode superposé les fonctions $\sin(x)$ et $\cos(x)$ sur l'intervalle $[-\pi, \pi]$ avec 201 points.

```
x = -pi:pi/100:pi;
y = sin(x);
z = cos(x);
plot(x,y, 'r');
hold on;
plot(x,z, 'b');
hold off;
```



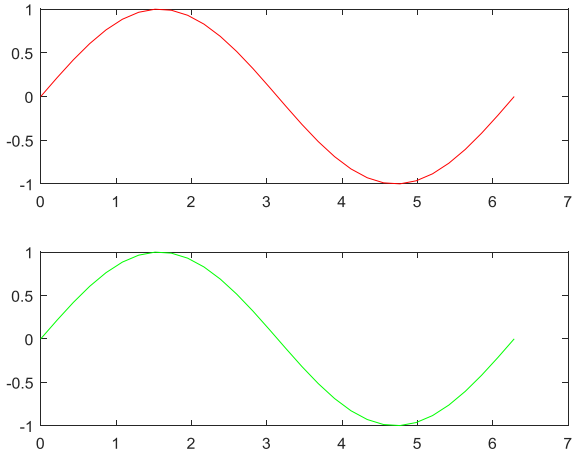
Pour **subdiviser la fenêtre graphique** et donc tracer plusieurs graphiques sur la même figure, nous utilisons l'instruction `subplot` (sous-graphique). Sa syntaxe est :

`suplot(nombre_lignes, nombre_colonnes, numéro_subdivision)`

Les subdivisions sont numéroter de 1 à $nombre\_lignes * nombre\_colonnes$, de la gauche vers la droite puis de haut en bas.

Un exemple est donné ci-dessous pour illustrer le fonctionnement de cette instruction :

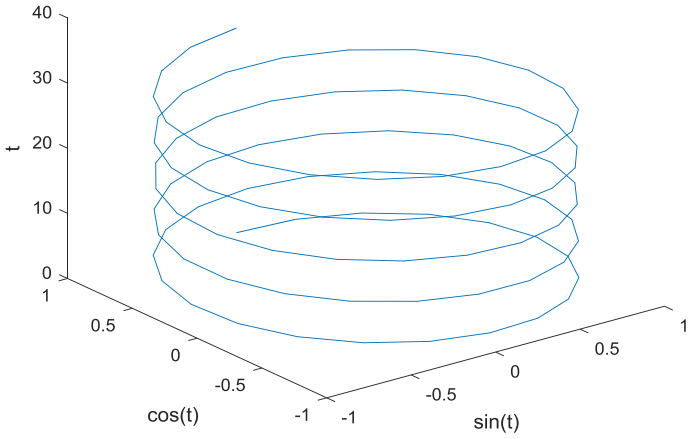
```
x = linspace(0,2*pi,30);
y = sin(x);
z = cos(x);
subplot(2,1,1);
plot(x,y, 'r');
subplot(2,1,2);
plot(x,y, 'g');
```



Graphiques à 3D

Pour tracer une courbe paramétrée avec 3 coordonnées, nous utilisons l'instruction `plot3`.

```
t = linspace(0,10*pi);
x = sin(t);
y = cos(t);
plot3(x,y,t);
xlabel('sin(t)');
ylabel( 'cos(t)');
zlabel('t');
```



Les images : image et imagesc

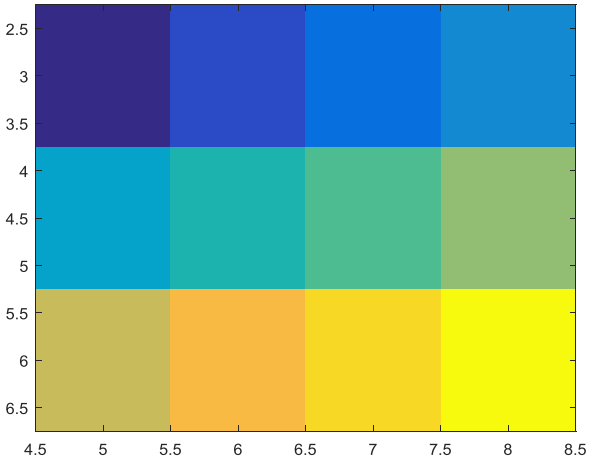
Dans Mtalab, une image pixélisée peut être vue de manière schématique comme une matrice dans laquelle chaque élément à une valeur correspondant à une couleur donnée.

L'instruction `image(x, y, A)` permet de représenter l'image contenue dans une matrice A. Celui-ci permet de spécifier les échelles des axes en prenant les valeurs contenues dans les vecteurs x et y.

L'instruction `imagesc(x, y, A)` permet d'utiliser toutes la palette de couleur disponible sans modifier le contenu de la matrice. Les lettres *sc* ajoutées correspondent à l'abréviation de *scaling*, qui signifie *renormaliser*.

Le script suivant illustre l'utilisation de l'instruction `imagesc` :

```
x = [5 8];
y = [3 6];
A = [0  2  4  6;
     8  10 12 14;
     16 18 20 22];
imagesc(x,y,A);
```

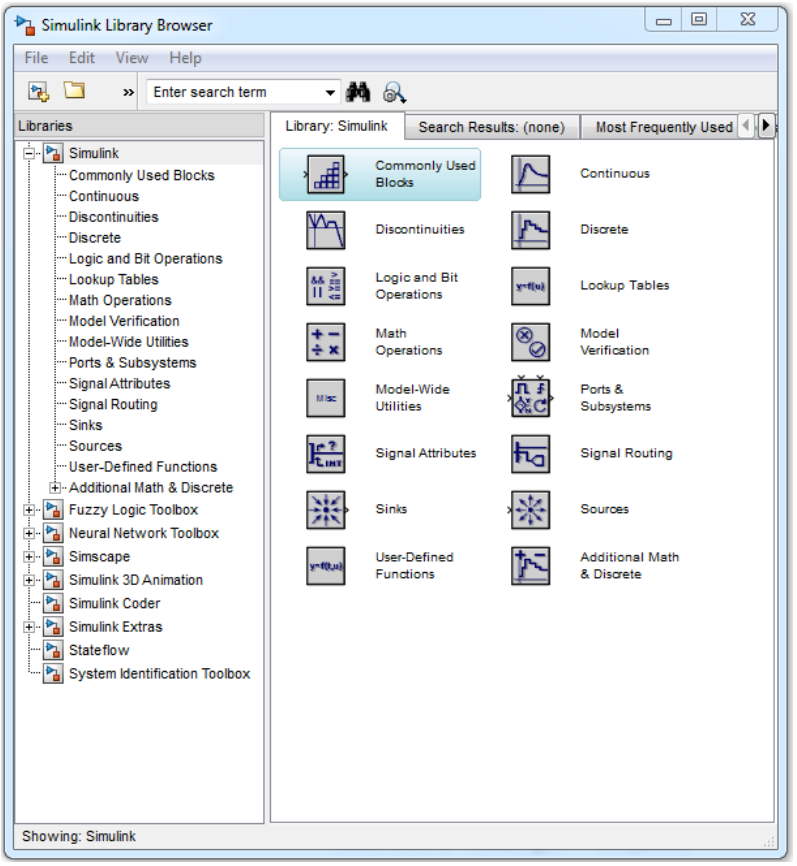


1.8 Introduction à Simulink

Simulink est l'interface graphique de `MatLab` qui permet de représenter les fonctions mathématiques et les systèmes sous forme de diagramme en blocs, et de simuler leurs fonctionnements. Simulink possède des bibliothèques de blocs, regroupés dans des *Blocksets* (Simulink Communications blockset pour les systèmes de communications par exemple). Simulink propose une approche causale de la modélisation. Le comportement dynamique d'un système est caractérisé par un bloc contenant, par exemple, la fonction de transfert du système avec une entrée et une sortie. L'information circulant dans les connexions entre deux blocs est un signal numérique orienté.

Pour démarrer Simulink, cliquer sur l'icône *Simulink bloc Library* (ou taper *Simulink* dans la fenêtre commande de `MatLab`).

Cette fenêtre contient des collections de blocs que l'on peut ouvrir en double-cliquant dessus :



`Sources`	Blocs pour générer ou importer des données de signal tels que le signal sinus
`Continuous`	Blocs des fonctions continues tels que les dérivés et l'intégrateur
`Discontinuities`	Blocs des fonctions discontinues tels que les saturations
`Discrete`	Blocs des fonctions à temps discrets tels que délais d'unité
`Logic and Bit Operations`	Blocs des opérateurs logiques ou de bits tels que l'opérateur logique et l'opérateur relationnel
`Math Operations`	Blocs des fonctions mathématiques comme le gain, le produit et la somme

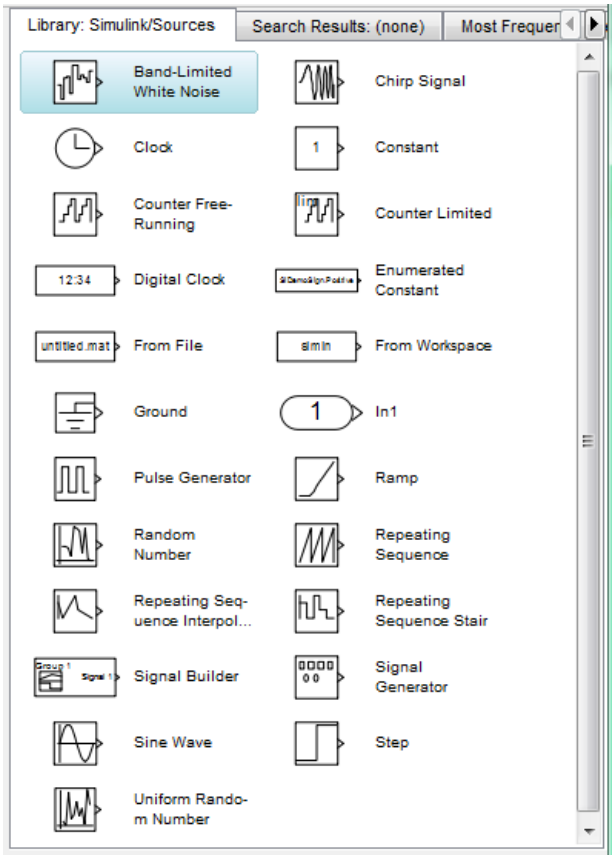
Construction d'un modèle Simulink

La création d'un nouvelle modèle sous Simulink se déroule, en général, selon les étapes suivantes :

— Choisir *New-Model* dans le menu *File*. Une fenêtre de travail nommée Untiled s'ouvrira.

— Ouvrir les collections des de blocs en double-cliquant dessus.

— Faire glisser les blocs nécessaires pour dans la fenêtre de travail.



— Faires des liaisons entres les blocs à l'aide de la souris.

— Changer les paramètres des blocs en double-cliquant sue l'icône de *bloc*.

— Fermer la fenêtre de dialogue.

— Une fois le diagramme est terminé, enregistrer le modèle dans un fichier : dans le menu *File*, choisis *Save As* et donner un nom *(*.mdl)* au fichier.

Avant de lancer la simulation, nous devons choisir les paramètres adaptés au modèle du système. Pour cela, dans le menu *Simulation*, lorsque nous choisissons Parameters, une fenêtre *Simulation Parameters* s'ouvrira. Nous devons alors choisir les paramètres pour *Solver*, *Diagnostics*, ...etc.

Pour lancer la simulation, taper sur le bouton *Start* dans le menu de la simulation.