École des HAUTES ÉTUDES INDUSTRIELLES
Département : AUTOMATIQUE

Laboratoire d'Automatique

Salle : H 501

TRAVAUX PRATIQUES DE RÉGULATION

MATLAB - SIMULINK
APPLICATIONS A L'AUTOMATIQUE

Abdel AITOUCHE	**Bureau H503**
Patrick DEBAY	**Bureau H502**
Fabrice VIENNE	**Bureau H502**

SOMMAIRE

0

1 . INTRODUCTION

MATLAB est un système interactif et convivial de calcul numérique et de visualisation graphique destiné aux ingénieurs et scientifiques. Il possède un langage de programmation à la fois puissant et simple d'utilisation. Il permet d'exprimer les problèmes et solutions d'une façon aisée, contrairement aux langages de programmation.

MATLAB est un progiciel qui permet de façon interactive et graphique de modéliser et de simuler des systèmes. ***MATLAB*** s'impose dans les modes universitaire et industriel comme un outil puissant de simulation et de visualisation de problèmes numériques. Dans le monde universitaire ***MATLAB*** est utilisé pour l'enseignement de l'algèbre linéaire, le traitement du signal, l'automatique ainsi que dans la recherche scientifique. Dans le domaine industriel, il est utilisé pour la résolution et la simulation de problèmes pratiques d'ingénierie et de prototypage.

MATLAB, moteur de calcul numérique, intègre des fonctions d'analyse numérique, de calcul matriciel, de traitement du signal, des centaines de fonctions mathématiques avec des visualisations graphiques à 2 Dimensions et à 3 Dimensions. ***MATLAB*** offre un environnement de programmation qui permet de créer, débugger et exécuter ses propres fonctions.

Des sous bibliothèques ***MATLAB*** augmentent sa puissance en fournissant des algorithmes et des fonctions développées. Parmi les domaines couverts, nous trouvons le traitement du signal, l'analyse et la synthèse temporelle et fréquentielle des systèmes linéaires, continus et discrets, la conception des lois de commande, l'identification, l'optimisation, la simulation dynamique, les réseaux neuronaux, etc...

MATLAB permet précisément de faire des calculs en algèbre linéaire : calcul matriciel, calcul des racines des vecteurs propres, racines de polynômes, etc..

MATLAB est conforté par une multitude de boîtes à outils (Toolboxes) spécifiques à des domaines variés. Nous donnons une liste non exhaustive de ces quelques modules :

- module de contrôle : **CONTROL SYSTEM**
- module d'identification : **SYSTEM IDENTIFICATION**
- module de traitement de signal : **SIGNAL PROCESSING**
- module de logique floue : **FUZZY**
- module de réseaux neuronaux : **NEURAL NETWORK**
- module d'optimisation : **OPTIMIZATION**
- module de contrôle robuste : **ROBUST CONTROL**
- module de calcul symbolique : **EXTENDED SYMBOLIC**
- module de simulation dynamique de systèmes : **SIMULINK**
- module d'intégration, de dérivation et d'évaluation : **SPLINE TOOLBOX**
- module de régression linéaire multiple et d'analyse de méthodes de développement quantitatives :

Le module de **"Control System"**, utilisant les fonctions de ***MATLAB***, est une bibliothèque d'algorithmes, écrites en fichiers texte. Ces fichiers ont une extension de type ***.m**. Les Systèmes de Contrôle/Commande peuvent être modélisés sous forme de fonctions de transfert ou sous forme de variables d'état. Cette modélisation peut combiner les techniques traditionnelles et modernes. Nous pouvons étudier les systèmes continus et les systèmes discrets.

Des conversions entres plusieurs modes de représentations sont proposées. Les réponses temporelles, les réponses fréquentielles et le lieu des racines des systèmes peuvent être calculés et représentés par des graphiques.

D'autres fonctions, telles que les méthodes de placement de pôles, de commande optimale et d'estimation sont proposées.

En complément de ***MATLAB***, l'outil additionnel ***SIMULINK*** est proposé pour la modélisation et la simulation de systèmes dynamiques en utilisant une représentation de type schémas blocs.

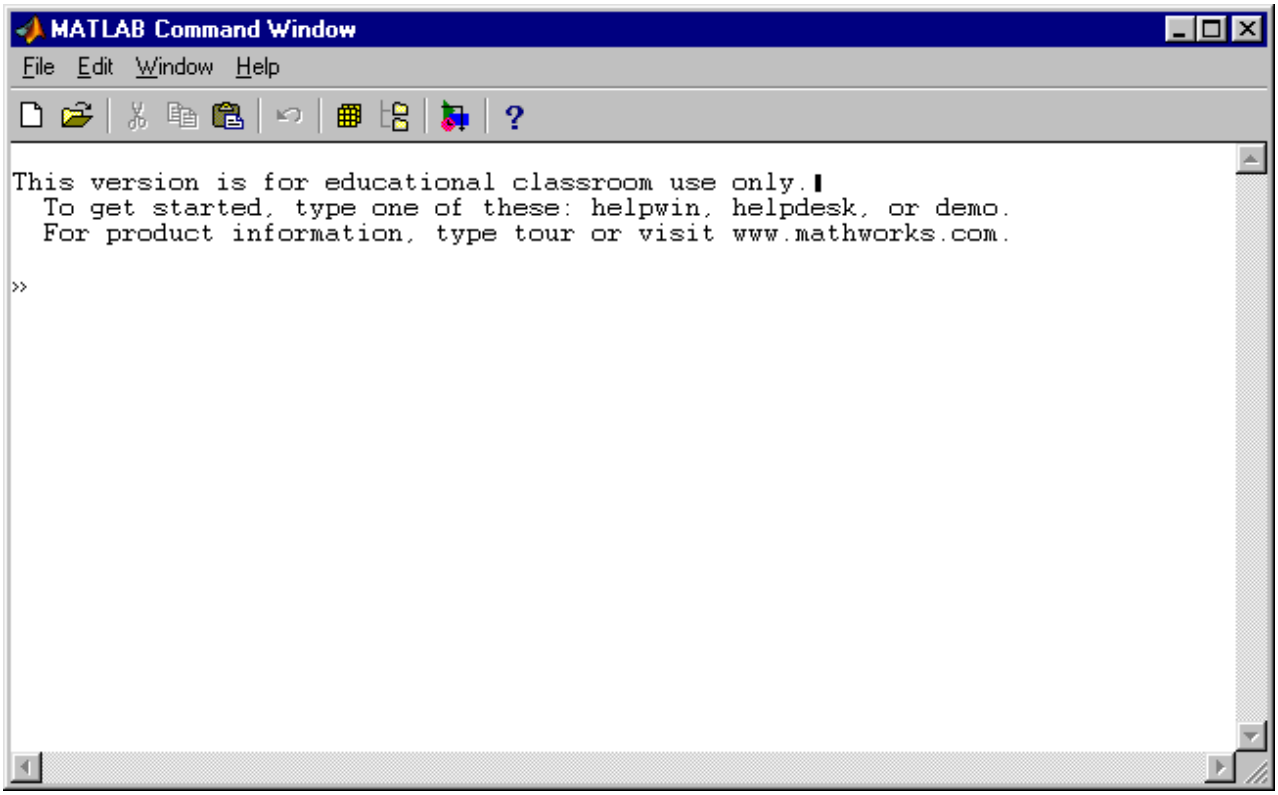
Le langage ***MATLAB*** est interprété. Le listing source n'est ni compilé, ni linké. ***MATLAB*** ne donne donc pas naissance à un code exécutable. Lors de l'exécution d'une séquence, chaque ligne est lue, interprétée, puis exécutée. Ce langage comporte donc l'avantage de ne pas nécessiter de compilation et de linkage, mais possède l'inconvénient d'être lent en exécution.

Pour des applications, où nous avons besoin de rapidité, nous pourront programmer en langage évolué tel que le langage C. Cette programmation doit respecter des règles bien précises pour rendre le programme exécutable compatible avec le fonctionnement de ***MATLAB***. Nous utilisons pour compiler et linker le fichier CMEX.BAT, ce qui permet de créer des MEX-FUNCTIONS en langage C. Les exécutables obtenus sont plus rapides qu'en langage interprété.

12 . UTILISATION DE MATLAB

Pour lancer l'exécution de ***MATLAB*** : sous Windows, il faut effectuer un double clic sur l'icône de ***MATLAB***.

Dans ce cas, une fenêtre ***MATLAB*** doit s'ouvrir. L'invite "»" de ***MATLAB*** doit alors apparaître, à la suite duquel vous entrerez les commandes.



02.1 . L'INSTRUCTION HELP

L'instruction **help** permet de visualiser les différentes fonctions utilisées par ***MATLAB***.

L'instruction **help "nom de la fonction"** donne des renseignements sur l'utilisation de la fonction. Par exemple, si vous souhaitez savoir comment utiliser la commande qui génère des réponses indicielles, tapez :

>> help step

sur l'écran apparaît ce qui suit :

STEP Step response of LTI systems.

STEP(SYS) plots the step response of each input channel of the LTI system SYS (created with either TF, ZPK, or SS). The time range and number of points are chosen automatically.

STEP(SYS,TFINAL) simulates the step response from t = 0 to the final time t = TFINAL. For discrete-time systems with unspecified sampling time, TFINAL is interpreted as the number of samples.

STEP(SYS,T) uses the user-supplied time vector T for simulation. For discrete-time systems, T should be of the form Ti:Ts:Tf where Ts is the sample time of the system. For continuous systems, T should be of the form Ti:dt:Tf where dt will become the sample time of a discrete approximation to the continuous system. The step input is always assumed to start at t = 0 (regardless of Ti).

STEP(SYS1,SYS2,...,T) plots the step response of multiple LTI systems SYS1,SYS2,... on a single plot. The time vector T is optional. You can also specify a color, line style, and marker for each system, as in step(sys1,'r',sys2,'y--',sys3,'gx').

When invoked with left-hand arguments,

[Y,T] = STEP(SYS,...) returns the output response Y and the time vector T used for simulation. No plot is drawn on the screen. If SYS has NU inputs and NY outputs, and LT=length(T), the array Y is LT-by-NY-by-NU and Y(:,:,j) gives the step response of the j-th input channel.

For state-space systems,

[Y,T,X] = STEP(SYS,...) also returns the state trajectory X which is an LT-by-NX-by-NU array if SYS has NX states.

See also INITIAL, IMPULSE, LSIM.

Overloaded methods

Si vous voulez connaître, toutes les fonctions générales, tapez **help general**.

12.2 . GESTION DES COMMANDES ET DES FONCTIONS

Le premier écran de ***MATLAB*** présente quelques commandes :

intro, demo, help help, help, info.

- **intro** : Lance une introduction à ***MATLAB***,
- **help** : Produit une liste des fonctions ***MATLAB*** par catégorie,
- **help help** : Informations sur l'utilisation de l'aide,
- **info** : Informations sur les boîtes à outils (toolbox) disponibles,
- **demo** : Programme de démonstration donnant une présentation des fonctions de base de ***MATLAB***,
- **doc** : Charger la documentation hypertext,
- **what** : Répertoire énumérant des fichiers *.M -, *.MAT - et *.MEX,
- **type** : Liste des fichiers *.M,
- **lookfor** : Mot-clé de recherche par les entrées D'AIDE,
- **which** : Localisation des fonctions et des fichiers,
- **demo** : Exécution de démonstrations,
- **path**

22.3 . GESTION DES VARIABLES ET DE L'ESPACE DE TRAVAIL

- **who** : Énumérer les variables actuelles,
- **whos** : Énumérer les variables actuelles en forme longue,
- **load** : Récupérer des variables du disque,
- **save** : Sauver des variables sur disque,
- **clear** : Effacer des fonctions et des variables,
- **pack** : Consolider la mémoire de l'espace de travail,
- **size** : Taille de matrice,
- **length** : Taille de vecteur,
- **disp** : Exposer sur l'écran une matrice ou du texte.

32.4 . EXECUTION DES INSTRUCTIONS DU SYSTEME D'EXPLOITATION

- **cd** : Changer le répertoire de travail,
- **dir** : Énumérer les répertoires,
- **delete** : Effacer des fichiers,
- **getenv** : Obtenir la valeur d'environnement,
- **!** : Exécuter les ordres du système d'exploitation (ici DOS),
- **diary** : Sauver le texte d'une session ***MATLAB***.

42.5 . CONTROLE DE LA FENETRE D'ECRAN

- **cedit** : Ligne d'ordre d'ensemble,
- **clc** : Effacer les commandes d'écran,
- **home** : Positionne le curseur en haut de l'écran,
- **format** : Permet de choisir un format approprié,
- **echo** : Afficher sur l'écran du texte,

- **more** : Contrôle de la pagination.

52.6 . DEMARRER ET QUITTER *MATLAB*

- **quit** ou **exit** : Quitter ***MATLAB***,
- **startup** : Fichiers *.M exécuté quand ***MATLAB*** est invoqué,
- **matlabrc** : Fichier *.M équivalent à un fichier *.Bat.

62.7 . INFORMATIONS GENERALES

- **info** : Information sur ***MATLAB*** et **MathWork**.
- **subscribe** : Souscription de l'utilisateur de ***MATLAB***.
- **hostid** : Identification du host.
- **whatsnew** : Information sur les nouveautés.
- **ver** : Versions de ***MATLAB***, ***SIMULINK***, et les ***TOOLBOX***.

*2*3 . NOTIONS DE BASE DE *MATLAB*

73.1 . LE MODE INTERACTIF

Dans le mode interactif, ***MATLAB*** peut être utilisé comme une "super-puissante" calculatrice scientifique. Nous disposons des opérations arithmétiques et d'un ensemble important de fonctions de calcul numérique et de visualisation graphique.

83.2 . VARIABLES SPECIALES ET CONSTANTES

Dans MATLAB, nous trouvons des constantes pré-définies :

: 3.14159265358979,
: 2.2204e-016 (distance entre 1.0 et le flottant le plus proche) ;
cette valeur peut être modifiée,
e) : nombre infini,
Number) : n'est pas un nombre, exprime parfois une indétermination.

Et une variable pré-définie :

: variable contenant la dernière réponse.

93.3 . OPERATIONS ARITHMETIQUES

Les opérations arithmétiques de base dans ***MATLAB*** sont : + pour l'addition, - pour la soustraction, * pour la multiplication et / ou \ pour la division.

La division à droite et la division à gauche ne donnent pas les mêmes résultats, ce sont donc deux opérations différentes que ce soit pour les scalaires ou pour les vecteurs et matrices.

a/b : c'est a qui est divisé par b

```
» 4/3
ans =
   1.33333333333333
```

a\b : c'est b qui est divisé par a

```
» 4\3
ans =
   0.75000000000000
```

MATLAB possède l'opérateur "^" pour l'élévation à une puissance entière ou réelle.

```
» 3^4
ans =
   81

» 3^(-1.5)
ans =
   0.19245008972988
```

103.4 . FORMAT DES NOMBRES ET PRECISION DES CALCULS

MATLAB a la possibilité d'afficher les valeurs des variables dans différents formats : flottants courts (**short**), flottants long (**long**), flottant courts en notation scientifique (**short e**), flottants longs en notation scientifique (**long e**), ainsi que les formats monétaire, hexadécimal et rationnel (notation sous forme d'une fraction).

113.5 . ÉDITION DES LIGNES DE COMMANDE

MATLAB conserve l'historique des commandes entrées de façon interactive lors d'une session. Il est donc possible de récupérer des instructions déjà saisies et de les modifier dans le but de les réutiliser.

Si vous avez commis une erreur sur l'expression suivante, il est toujours possible de la rééditer après son exécution.

```
» 4*(4.5-23+2.5)/(8.5-3.2)
```

Au lieu de **23** nous voulons plutôt **2.3**. Pour corriger l'expression et l'exécuter une nouvelle fois : utilisez les touches de déplacement vers le haut (↑) pour revenir à la ligne précédente, ensuite la touche de déplacement vers la gauche(←) pour se positionner entre le **2** et le **3** du **23** et insérez un point. Vous pouvez utiliser la touche "retour Arrière" (***BackSpace***) pour effacer des caractères.

Résumé de l'édition des lignes de commandes

:	passer à l'instruction précédente,
:	passer à l'instruction suivante,
:	aller vers la gauche sur la ligne de commandes,
:	aller vers la droite sur la ligne de commandes,
ckSpace :	effacer le caractère à gauche du curseur,
ppr :	effacer le caractère à droite du curseur,
c :	effacer la ligne de commandes,
but, Fin :	se positionner au début ou à la fin de la ligne de commandes.

En tapant quelques caractères suivis de la touche ↑, la dernière commande commençant par ces caractères est alors rappelée sur la ligne de commandes.

34 . VECTEURS OU TABLEAUX A 1 DIMENSION

Le moyen le plus simple de saisir un vecteur est d'entrer ses éléments en les séparant par des blancs ou par des virgules.

Saisie du tableau x, vecteur ligne

```
» x=[6 4 5]
x =
   6     4     5
```

Afin d'éviter l'affichage du résultat d'une expression quelconque, nous terminerons celle-ci par un point virgule.

```
» x=[6 4 5];
```

Autre façon de saisir un vecteur ligne

```
» x=[6, 4, 5]
x =
   6     4     5
```

Ce vecteur est considéré comme une matrice à une ligne et trois colonnes.

```
» size(x)
ans =
   1     3
```

Les dimensions d'un tableau quelconque peuvent être récupérées sous forme d'un vecteur **[m n]**, m et n étant respectivement le nombre de lignes et de colonnes.

```
» [m n]=size(x)
m =
   1
n =
   3
```

La longueur d'un tableau quelconque est, par définition, sa plus grande dimension.

```
» longueur_x = length(x)
longueur_x =
    3
```

Si nous voulons créer le vecteur v=[6 4 5 1 2 8], nous pouvons utiliser le vecteur x précédent.

```
» v=[x 1 2 8]
v =
    6     4     5     1     2     8
```

De même pour avoir le vecteur w = [1 6 4 5 1 2 8], nous avons deux possibilités :

```
» w=[1 v]
w =
    1     6     4     5     1     2     8
```

ou bien :

```
» w=[1 x 1 2 8]
w =
    1     6     4     5     1     2     8
```

Dans ***MATLAB***, les indices d'un tableau commencent à 1. Pour récupérer une composante d'un vecteur, il faut spécifier son indice entre parenthèses.

```
» w(3)
ans =
    4
```

Si les composantes d'un vecteur sont espacées d'un pas constant et si la première et la dernière valeur sont connues, alors ce vecteur peut être décrit de la manière suivante :

```
» debut=0;
» fin=5;
» pas=1;
» t=[debut:pas:fin]
t =
    0     1     2     3     4     5
```

L'addition et la soustraction de vecteurs de mêmes dimensions se font élément par élément.

```
» x=[0 4 3];
» y=[2 5 7];
» x-y
ans =
   -2    -1    -4
» x+y
ans =
    2     9    10
```

Ajouter ou retrancher un scalaire à un vecteur, revient à ajouter ou retrancher le scalaire à toutes les composantes du vecteur.

```
» 3+x
ans =
   3     7     6
» x-2
ans =
   -2     2     1
```

De même, la multiplication et la division d'un vecteur par un scalaire sont réalisées sur toutes les composantes du vecteur.

```
» 2*x
ans =
   0     8     6
» x/4
ans =
   0     1.0000     0.7500
```

La transformation d'un vecteur ligne en un vecteur colonne et inversement, sera réalisée à l'aide de l'opérateur de transposition " ' ".

```
» tx=x'
tx =
   0
   4
   3
```

Le produit d'un vecteur par sa transposée donne le carré de la norme de celui-ci.

```
» x*tx
ans =
   25
```

Le produit d'un vecteur colonne de taille n par un vecteur ligne de taille m donne une matrice de dimensions (n, m)

```
» tx*y
ans =
   0     0     0
   8     20    28
   6     15    21
```

En précédant d'un point les opérateurs *****, \, / et ^, nous réalisons des opérations élément par élément.

```
» x.*y
ans =
   0    20    21
» x.^2
ans =
   0    16     9
» x./y
ans =
   0    0.8000    0.4286
» x.\y
Warning : Divide by zero.
ans =
   Inf    1.2500    2.3334
```

Plusieurs fonctions opérant directement sur des vecteurs sont disponibles sous ***MATLAB***. Nous pouvons citer :

sum : somme des composantes d'un vecteur,

```
» sum(x)
ans =
   7
```

prod : produit des composantes d'un vecteur

```
» prod(y)
ans =
   70
```

sqrt : racines carrées des composantes d'un vecteur,

```
» sqrt(x)
ans =
   0    2.0000    1.7321
```

mean : moyenne des composantes d'un vecteur

```
» mean(x)
ans =
   2.3333
```

45 . MATRICES OU TABLEAUX A 2 DIMENSIONS

Le tableau à 2 dimensions (matrice) est l'élément de base de ***MATLAB***. Un vecteur n'est autre qu'une matrice à une ligne ou à une colonne.

MATLAB travaille avec des matrices rectangulaires dont les coefficients peuvent être réels ou complexes. Dans certaines situations, un scalaire est une matrice 1*1.

125.1 . SAISIE D'UNE MATRICE

Les matrices peuvent être introduites de différentes manières:

- explicitement,
- générées par des fonctions,
- crées par des fichiers *.m,
- chargées par des fichiers externes.

De façon explicite, pour entrer par exemple une matrice 2x2, tapez :

Les lignes sont séparées par un point virgule

```
» A = [1 2; 3 4]
A =
    1     2
    3     4
```

Les lignes de la matrice sont séparées par un retour à la ligne

```
» A=[1 2
3 4]
A =
    1     2
    3     4
```

produit le même résultat.

Dimensions de la matrice x

```
» a=[1 2 3; 4 5 6]
a =
    1     2     3
    4     5     6
» [m,n]=size(a)
m =
    2
n =
    3
```

Accès à un élément de la matrice A

Attention la matrice **A** est différente de la matrice **a**.

Pour visualiser l'élément A(2,1) :

```
» A(2,1)
ans =
   3
```

Pour visualiser la ligne 1 de la matrice A :

```
» A(1,:)
ans =
   1     2
```

135.2 . MATRICES PARTICULIERES

Dans ***MATLAB***, nous trouvons certaines matrices spéciales ou particulières, nous en donnons ci-après quelques exemples :

Matrice identité (matrice carrée)

```
» ident=eye(3)
ident =
   1     0     0
   0     1     0
   0     0     1
```

Matrice nulle

```
» Matrice_nulle=zeros(2,3)
Matrice_nulle =
   0     0     0
   0     0     0
```

Matrice unité

```
» Matrice_unité=ones(3,2)
Matrice_unité =
   1     1
   1     1
   1     1
```

Matrice aléatoire

```
» Matrice_aléatoire=rand(2,3)
Matrice_aléatoire =
   0.9501    0.6068    0.8912
   0.2311    0.4859    0.7621
```

La fonction **rand** donne un réel de façon aléatoire compris entre 0 et 1. Pour une matrice carrée unité, nulle ou aléatoire, il suffit d'indiquer l'ordre comme seul argument des fonctions **ones**, **zeros** et **rand**.

145.3 . OPERATIONS ELEMENTAIRES SUR LES MATRICES

Plusieurs autres opérations matricielles peuvent être utilisées telles que la somme (**+**), le produit (*****) de matrices, l'exponentielle (**expm**), la norme (**norm**), le logarithme (**logm**), la diagonalisation (**diag**) d'une matrice, la décomposition, la factorisation et toutes les opérations possibles sur les matrices.

Soient les matrices carrées x et y suivantes :

```
» x=[1 2 3; 4 5 6; 7 8 9]
x =
   1   2   3
   4   5   6
   7   8   9
» y=[2 3 4; 5 6 7; 8 9 2]
y =
   2   3   4
   5   6   7
   8   9   2
```

La somme de deux matrices x et y se note **x+y.**

```
» z=x+y
z =
   3   5   7
   9   11  13
   15  17  11
```

Le produit des deux matrices x et y se note **x*y**.

```
» produit=x*y
produit =
   36   42   24
   81   96   63
   126  150  102
```

Le produit de deux matrices rectangulaires (non carrées) ne peut se faire que si les dimensions des deux matrices respectent la contrainte suivante : si x est une matrice (n, m) et y une matrice (p, q), le produit x*y ne peut se faire que si **m=p** ; le résultat est une matrice (n, q).

Élévation à une puissance d'une matrice

```
» x^2
ans =
   30   36   42
   66   81   96
   102  126  150
```

En précédant un opérateur par un point, celui-ci s'applique individuellement à tous les éléments de la matrice.

```
» x.^2
ans =
   1    4    9
```

```
    16    25    36
    49    64    81
```

Nous disposons dans ***MATLAB*** de tout un ensemble de fonctions élémentaires et spécialisées pour la manipulation et la gestion des matrices.

Vous pouvez utiliser les opérations de base du calcul matriciel.

La transposée de A :

```
» B = A'
B =
    1     3
    2     4
```

L'inverse de A :

```
» inv(A)
ans =
   -2.0000    1.0000
    1.5000   -0.5000
```

Le déterminant de A :

```
» det(A)
ans =
   -2
```

Les valeurs propres de A :

```
» eig(A)
ans =
   -0.3723
    5.3723
```

Les vecteurs propres et les valeurs propres de A :

```
» [D,E] = eig(A)
D =
   -0.8246   -0.4160
    0.5658   -0.9094
E =
   -0.3723         0
         0    5.3723
```

Les colonnes de la matrice D représentent les vecteurs propres de A et les éléments diagonaux de E sont les valeurs propres de A.

Le rang de la matrice A :

```
» rank(A)
ans =
    2
```

La dimension de la matrice A :

```
» length(A)
ans =
    2
```

La trace de A :

```
» trace(A)
ans =
    5
```

Nous calculons avec **trace** la somme des éléments de la diagonale principale de la matrice **A.**

Pour résoudre un système d'équations linéaires, nous utilisons l'instruction "****". Par exemple**,** résolvons le système AX=B avec A = [1 2; 3 4], B=[5 6]' et X=[X1 X2]'. Nous écrivons :

```
» A=[1 2; 3 4]
A =
    1    2
    3    4
» B=[5 6]'
B =
    5
    6
» X=A\B
X =
   -4.0000
    4.5000
```

D'autres types de matrices spécifiques sont générées telles que les matrices triangulaire (**tri**), Hankel (**hankel**), Toepliz (**toepliz**), compagnon (**compan**), Hadamard (**hadamard**) et rotation d'une matrice de 90° (**rot90**), etc...

56 . LES NOMBRES COMPLEXES

156.1 . DEFINITION

L'imaginaire pur **i** ($i^2=-1$) est noté **i** ou **j**. Un nombre complexe est donc de la forme **z=a+ib** ou **a+jb**. Mais ***MATLAB***, dans ses réponses donne toujours le symbole **i**.

```
» i^2
ans =
   -1.0000 + 0.0000i

» j^2
ans =
   -1.0000 + 0.0000i
```

166.2 . CONJUGUE

Le conjugué d'un nombre complexe est obtenu par la fonction **conj**. Considérons le nombre complexe **z1.**

```
» z1=3-8i
z1 =
   3.0000 - 8.0000i
```

Son conjugué, noté z1conjugué, est

```
» z1conjugué=conj(z1)
z1conjugué =
   3.0000 + 8.0000i
```

176.3 . OPERATIONS COURANTES

Nous pouvons effectuer les opérations courantes sur les complexes telles que l'addition, la multiplication, l'élévation à une puissance et la division.

Addition de deux complexes

```
» z1=3-8i
z1 =
   3.0000 - 8.0000i
» z2=6+6j
z2 =
   6.0000 + 6.0000i
» total=z1+z2
total =
   9.0000 - 2.0000i
```

Multiplication

```
» z1*z2
ans =
   66.0000 -30.0000i
```

Division

```
» z1/z2
ans =
   -0.4167 - 0.9167i
```

Élévation à une puissance

```
» z1^5
ans =
   4.4403e+004 +1.0072e+004i
```

Remarques

A partir de la version 4 de ***MATLAB***, il n'est pas nécessaire d'écrire le signe de multiplication "*****" avant **i** (ou **j**) ; nous pouvons noter **z = 4+2i** au lieu de **z=4+2*i**.

Il est même conseillé d'utiliser la première notation pour éviter toute ambiguïté, surtout si nous avons défini une variable "**i**" à laquelle nous avons affecté une valeur.

Par exemple, si nous définissons une variable i à laquelle nous affectons la valeur 5,

```
» i=5
i =
    5
```

La définition du complexe z sans le signe "*" avant i, donne bien le nombre complexe désiré.

```
» z=4+2i
z =
   4.0000 + 2.0000i
```

Alors que si nous utilisons le signe "*", i représente la variable précédemment affectée de la valeur -3 au lieu du symbole i des nombres complexes ; dans ce cas, l'instruction précédente revient à une simple opération arithmétique.

```
» z=4+2*i
z =
    14
```

Attention si nous voulons déclarer z comme un complexe z=3+i il faut faire de la façon suivante (si nous avons déclaré une variable i bien sur) :

```
» z=3+1i
z =
   3.0000 + 1.0000i
```

Si nous oublions le 1 devant le i nous obtenons le résultat suivant :

```
» z=3+i
z =
    8
```

Les fonctions **real** et **imag** permettent d'obtenir respectivement les parties réelle et imaginaire d'un nombre complexe.

```
» partie_réelle=real(z1)
partie_réelle =
    3

» partie_imaginaire=imag(z1)
partie_imaginaire =
   -8
```

MATLAB propose les fonctions **abs** et **angle** qui permettent d'obtenir directement le module et l'argument d'un complexe.

```
» module=abs(z1)
module =
   8.5440
» argument=angle(z1)
argument =
   -1.2120
```

La démarche inverse est possible. Si nous connaissons le module et la phase, nous pouvons déduire la forme complexe :

```
» complexe= module*exp(j*argument)
complexe =
   3.0000 - 8.0000i
```

67 . LES POLYNOMES

MATLAB représente un polynôme sous la forme d'un tableau avec ses coefficients classés dans l'ordre des puissances décroissantes.

187.1 . SAISIE D'UN POLYNOME

Le polynôme P d'expression $P(x) = x^2 - 6x + 9$ est représenté par le tableau à 1 dimension suivant :

```
» p=[1 -6 9]
p =
   1    -6    9
```

Le nombre d'élément du tableau est égal au degré du polynôme +1.

197.2 . RACINES D'UN POLYNOME

Nous pouvons déterminer les racines du polynôme p à l'aide de la fonction **roots**.

```
» roots(p)
ans =
   3
   3
```

207.3 . ÉVALUATION DE POLYNOMES

Pour évaluer un polynôme en un point, nous utilisons la fonction **polyval**.

Valeur du polynôme p en 1 et celle du polynôme p en -1.52

```
» polyval(p,1)
ans =
   4
» polyval(p,-1.52)
ans =
   20.4304
```

217.4 . DETERMINATION D'UN POLYNOME A PARTIR DE SES RACINES

Nous pouvons aussi déterminer les coefficients d'un polynôme à partir de ses racines en utilisant la fonction **poly**.

Nous cherchons, par exemple, le polynôme qui a pour racines **1, 2 et 3**. Celles-ci peuvent être définies comme les éléments d'un vecteur **r**.

```
» r=[1 2 3]
r =
    1     2     3
```

le polynôme recherché est alors :

```
» K=poly(r)
K =
    1    -6    11    -6
```

Qui correspond à : $K(x) = x^3 - 6x^2 + 11x - 6$ (en multipliant par un réel non nul tous les coefficients de K, nous trouvons un autre polynôme ayant les mêmes racines que K).

Nous vérifions bien que les racines du polynômes K sont 1, 2 et 3.

```
» racines=roots(K)
racines =
    3.0000
    2.0000
    1.0000
```

227.5 . POLYNOME CARACTERISTIQUE D'UNE MATRICE

Le polynôme caractéristique d'une matrice A s'écrit :

```
» A=[1 2; 3 4]
A =
    1     2
    3     4
» P=poly(A)
P =
    1    -5    -2
```

Les éléments de ce vecteur représentent les coefficients du polynôme par puissance décroissante.

Les racines du polynôme caractéristique représentent les valeurs propres de A.

```
» roots(P)
ans =
   5.3723
   -0.3723
```

Si nous comparons avec les valeurs propres de A

```
» eig(A)
ans =
   -0.3723
   5.3723
```

Soit le polynôme P d'expression $P(x) = x^2 - 6x + 9$ et le polynôme Q d'expression $Q(x) = x^3 + 4x^2 + 2x + 3$

```
» P=[1 -6 9]
P =
   1    -6    9
» Q=[1 4 2 3]
Q =
   1    4    2    3
```

Si nous voulons faire le produit de deux polynômes P par Q, nous pouvons utiliser l'opération de convolution représentée par la commande **conv.**

```
» Résultat=conv(P, Q)
Résultat =
   1    -2    -13   27    0     27
```

La division de deux polynômes se fait par la fonction **deconv**. Le quotient Q et le reste R de la division peuvent être obtenus sous la forme d'éléments d'un tableau.

```
» P1=[1 2]
P1 =
   1    2
» P2=[1 -2 1]
P2 =
   1    -2    1
» [Q, R]= deconv(P2, P1)
Q =
   1    -4
R =
   0    0    9
```

D'autres opérations sur les polynômes sont possibles telles que le filtrage polynomiale utilisé en régression linéaire (**polyfit**), le calcul du résidu du quotient de 2 polynômes (**residue**), etc ...

Pour générer systématiquement des vecteurs nous utilisons l'opérateur ":".

```
» B=1:5
B =
   1     2     3     4     5
```

Dans ce cas nous créons un vecteur B compris entre 1 et 5 avec des pas de 1.

Cela revient à faire la commande suivante :

```
» debut=1;
» fin=5;
» pas=1;
» B=[debut:pas:fin]
B =
   1     2     3     4     5
```

Si vous tapez :

```
» C = 0:0.5:3
C =
   0.0000    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
```

Ici le pas est de 0.5.

78 . REPRESENTATION GRAPHIQUE

Les données peuvent être tracées en utilisant des commandes graphiques. Plusieurs types de tracés peuvent être envisagés : tracé cartésien en coordonnées X-Y normal (**plot**), tracé log-log en X-Y (**loglog**), tracé semi-logarithmique en X (**semilogx**), tracé semi-logartihmique en Y (**semilogy**), tracé en coordonnées polaires (**polar**) et tracé en tridimensionnel (**mesh**).

238.1 . EXEMPLE N°1

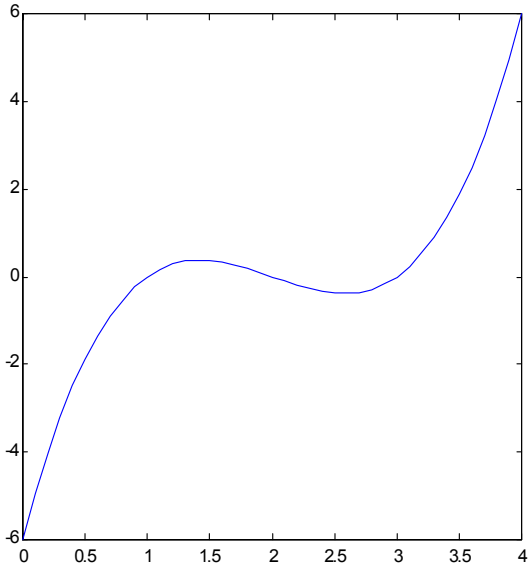
Pour tracer la représentation graphique d'un polynôme $K(x) = x^3 - 6x^2 + 11x - 6$, définissons un domaine pour la variable x qui contient les racines de K. Une fois, le domaine défini calculons (Y) les valeurs du polynôme K(x) au différents points du domaine de x.

```
» K=[1 -6 11 -6]
K =
   1     -6     11     -6
» x=[0:0.1:4];
» y=polyval(K, x);
```

Tracé de la fonction y=K(x)

```
» plot(x,y)
```

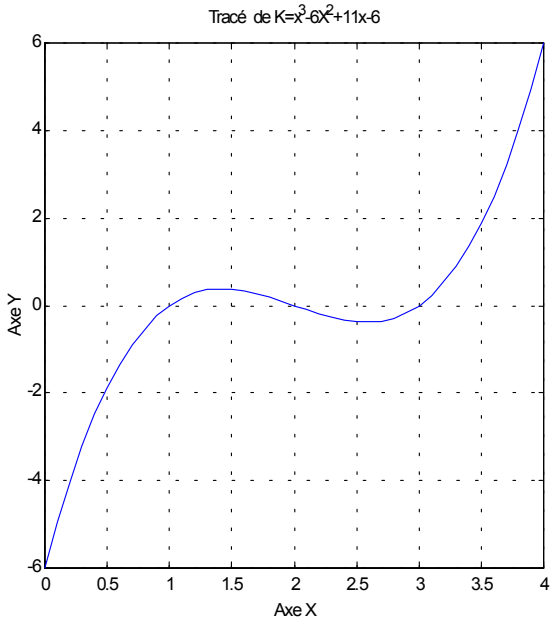
Dans ce cas nous obtenons la courbe suivante :



Les courbes tracées sur l'écran peuvent avoir des titres sur les axes (**xlabel**, **ylabel**), titre du graphique (**title**), texte positionné arbitrairement (**text**), texte positionné par la souris (**gtext**), des lignes graduées (**grid**), des graphiques avec des rectangles utilisés dans les histogrammes (**bar**), des graphiques en escalier utilisés dans les systèmes échantillonnés (**stairs**).

```
» grid
» title('Tracé de K=x^3-6X^2+11x-6')
» xlabel('Axe X')
» ylabel('Axe Y')
```

Dans ce cas nous obtenons la courbe suivante :

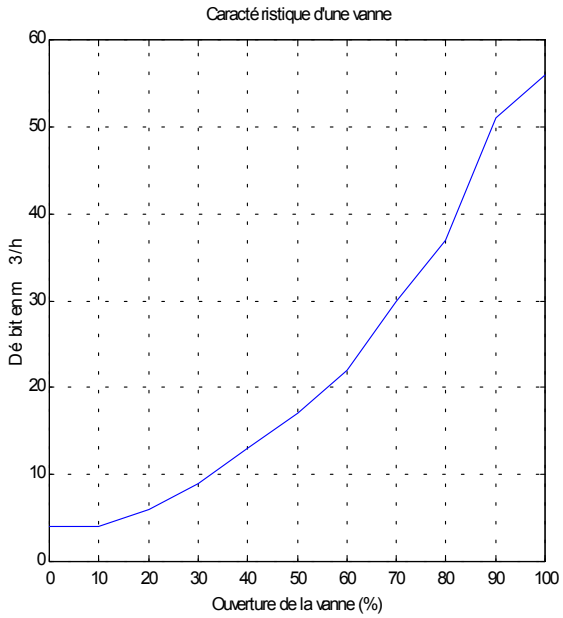


plot : tracé d'une représentation graphique,
grid : affiche une grille,
title : attribue un titre au graphique,
xlabel : attribue un test à l'axe des abscisses,
ylabel : attribue un texte à l'axe des ordonnées.

248.2 . EXEMPLE N°2

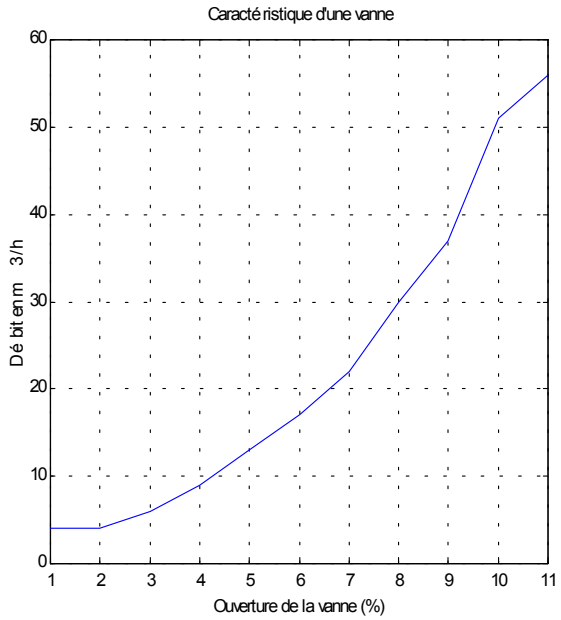
Nous voulons tracer la courbe suivante :

```
» X = 0:10:100;Y = [ 4 4 6 9 13 17 22 30 37 51 56];
» plot(X,Y);xlabel('Ouverture de la vanne (%)');ylabel('Débit en m3/h') ;
» title('Caractéristique d''une vanne') ; grid
```



Dans le cas où nous ne précisons pas l'axe des abscisses, ***MATLAB*** le génère automatiquement. Nous écrira alors **plot(Y)**.

```
» plot(Y)
» xlabel('Ouverture de la vanne (%)');ylabel('Débit en m3/h') ;
» title('Caractéristique d''une vanne') ; grid
```



Dans le cas où nous voulons que des axes de coordonnées fixés manuellement, nous utilisons l'instruction **axis**.

» help axis

AXIS Control axis scaling and appearance.
AXIS([XMIN XMAX YMIN YMAX]) sets scaling for the x- and y-axes on the current plot.
AXIS([XMIN XMAX YMIN YMAX ZMIN ZMAX]) sets the scaling for the x-, y- and z-axes on the current 3-D plot.
V = AXIS returns a row vector containing the scaling for the current plot. If the current view is 2-D, V has four components; if it is 3-D, V has six components.

AXIS AUTO returns the axis scaling to its default, automatic mode where, for each dimension, 'nice' limits are chosen based on the extents of all line, surface, patch, and image children.
AXIS MANUAL freezes the scaling at the current limits, so that if HOLD is turned on, subsequent plots will use the same limits.
AXIS TIGHT sets the axis limits to the range of the data.
AXIS FILL sets the axis limits and PlotBoxAspectRatio so that the axis fills the position rectangle. This option only has an effect if PlotBoxAspectRatioMode or DataAspectRatioMode are manual.

AXIS IJ puts MATLAB into its "matrix" axes mode. The coordinate system origin is at the upper left corner. The i axis is vertical and is numbered from top to bottom. The j axis is horizontal and is numbered from left to right.
AXIS XY puts MATLAB into its default "Cartesian" axes mode. The coordinate system origin is at the lower left corner. The x axis is horizontal and is numbered from left to right. The y axis is vertical and is numbered from bottom to top.

AXIS EQUAL sets the aspect ratio so that equal tick mark increments on the x-,y- and z-axis are equal in size. This makes SPHERE(25) look like a sphere, instead of an ellipsoid.

AXIS IMAGE is the same as AXIS EQUAL except that the plot box fits tightly around the data.

AXIS SQUARE makes the current axis box square in size.

AXIS NORMAL restores the current axis box to full size and removes any restrictions on the scaling of the units. This undoes the effects of AXIS SQUARE and AXIS EQUAL.

AXIS VIS3D freezes aspect ratio properties to enable rotation of 3-D objects and overrides stretch-to-fill.

AXIS OFF turns off all axis labeling, tick marks and background.

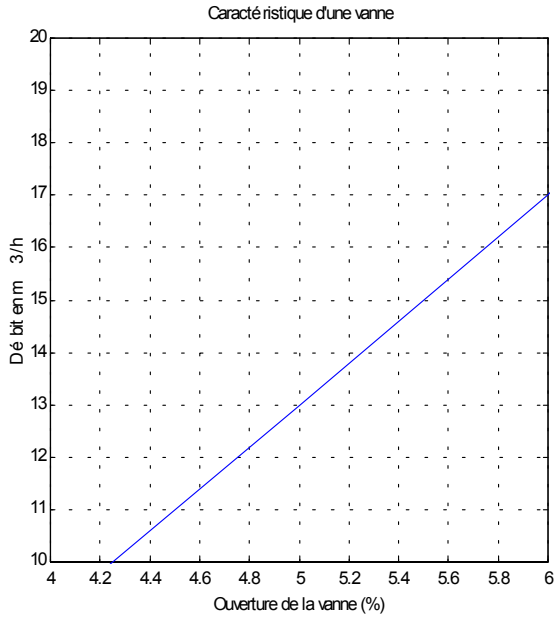
AXIS ON turns axis labeling, tick marks and background back on.

See also AXES.

Dans l'exemple une partie de la courbe nous intéresse. Cette partie a des valeurs de "X" comprises entre 4 et 6 et pour des valeurs de "Y" comprises entre 10 et 20.

```
» axis([4 6 10 20])
```

Nous obtenons la figure suivante :



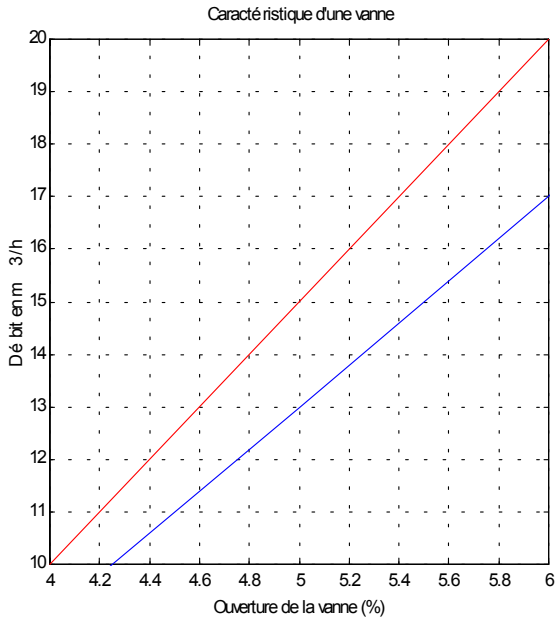
Pour maîtriser les graphiques, Nous pouvons utiliser les commandes graphiques de contrôle :

hold : maintien du graphique courant,
shg : appel du graphique courant,
clg : effacement du graphique.

La fonction **hold** permet par exemple de superposer deux courbes sur le même graphique.

```
» plot(Y)
» xlabel('Ouverture de la vanne (%)');ylabel('Débit en m3/h') ;
» title('Caractéristique d''une vanne') ; grid
» axis([4 6 10 20])
» hold on
» plot([4 6], [10 20], 'r')
» hold off
```

Nous obtenons la courbe suivante :



» hold on

Signifie : mise en marche du maintient de la figure courante. Cette fonction permet l'affichage de plusieurs courbes sur la même figure.

» plot([4 6], [10 20], 'r')

Le "**r**" permet de tracer une courbe avec la couleur rouge, d'autres couleurs sont disponible :

y : yellow,
m : magenta,
c : cyan,
r : red,
g : green,
b : blue,
w : white,
k : black.

» hold off

Signifie : l'arrêt du maintient de la figure courante. Attention si vous oubliez de faire cette fonction toutes les courbes seront affichées sur la même figure.

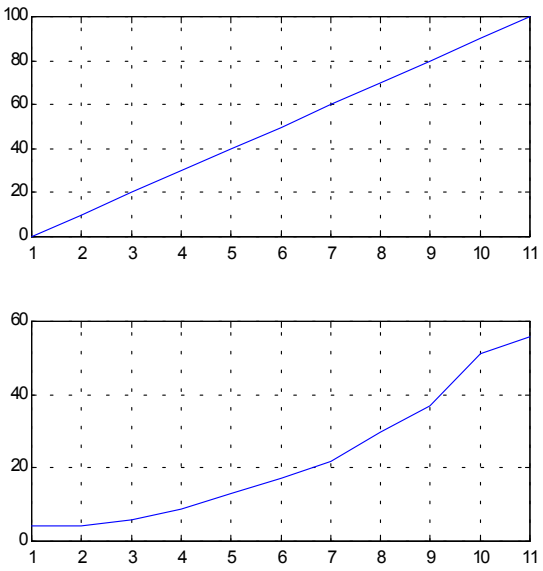
258.3 . SUBDIVISER LA FENETRE GRAPHIQUE

Avec ***MATLAB***, il est possible de tracer plusieurs graphiques dans une même fenêtre à l'aide de la commande **subplot** en divisant cette dernière en plusieurs zones.

subplot(m, n, p) divise la fenêtre graphique courante en **m*n** zones graphiques (**m** lignes et **n** colonnes) et trace le graphique qui suit cette instruction dans la zone de numéro **p** (la numérotation se fait de gauche à droite et ligne par ligne).

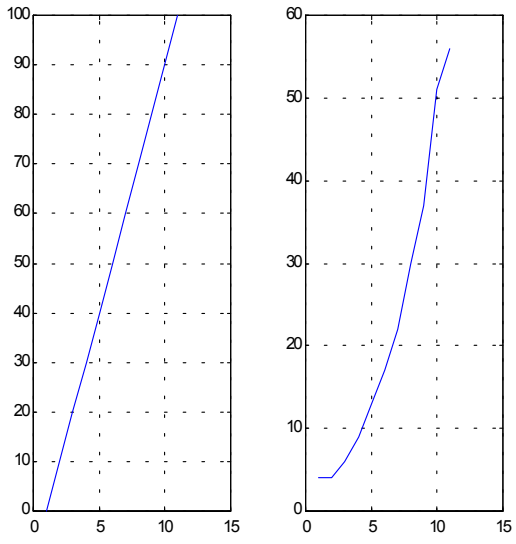
Par exemple si nous voulons tracer X et Y sur 2 fenêtres positionnées en ligne, nous procédons comme suit :

```
» subplot(211)
» plot(X)
» grid
» subplot(212)
» plot(Y)
» grid
```



Par exemple si nous voulons tracer X et Y sur 2 fenêtres positionnées en colonne, nous procédons comme suit :

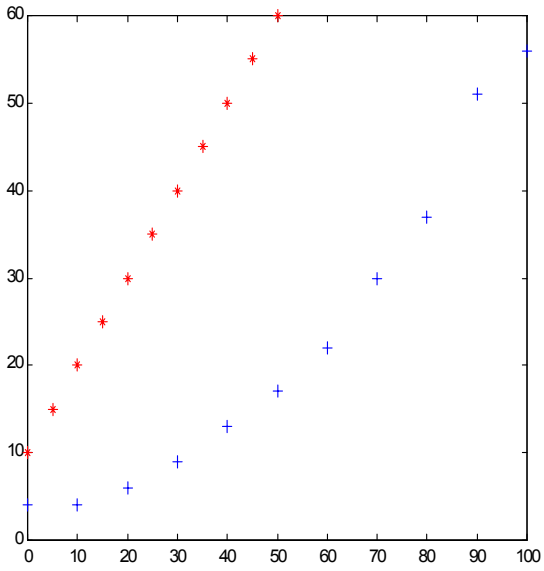
```
» subplot(121)
» plot(X)
» grid
» subplot(122)
» plot(Y)
» grid
```



268.4 . DIFFERENTES FORME DE POINTS

Le tracé d'un graphique se fait par défaut en utilisant des points. Dans le cas où nous voulons distinguer sur un même graphique des courbes entre elles, nous pouvons utiliser d'autres marques telles que **+**, *****, **o**, **x**, **-**, **--**, **:**, **-.** et préciser la couleur. Par exemple, créons 2 nouveaux vecteurs **X1=[0 :5 :50]** et **Y1=[10:5:60]**

```
» X = 0:10:100;Y = [ 4 4 6 9 13 17 22 30 37 51 56];
» X1=[0 :5 :50]
X1 =
    0     5    10    15    20    25    30    35    40    45    50
» Y1=[10:5:60]
Y1 =
   10    15    20    25    30    35    40    45    50    55    60
   » plot(X, Y, '+b', X1, Y1, '*r')
```



278.5 . ZOOM

La commande **zoom**, applicable seulement aux graphique 2D, permet de "zoomer" une partie rectangulaire du graphique courant, sélectionnée à l'aide du bouton gauche de la souris. Le bouton droit annule le "**zoom**" précédent.

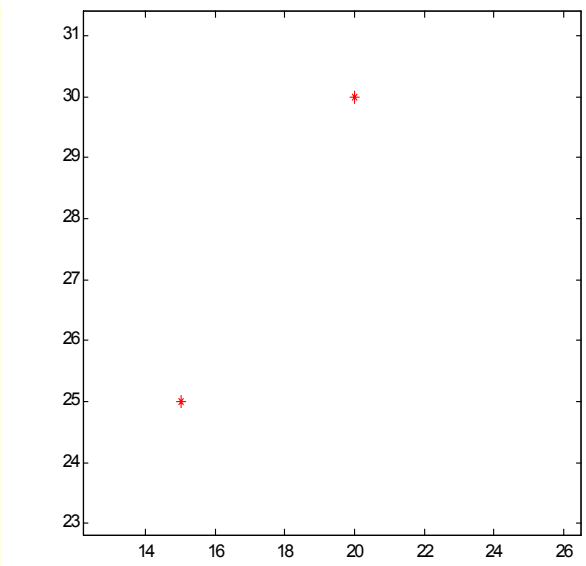
zoom off rend inactif le "**zoom**" et la commande **zoom out** rétablit le graphique avec ses dimensions initiales.

Nous pouvons faire un **zoom** uniquement suivant l'axe x avec la commande **zoom xon** et rend le zoom inactif par la commande **zoom off**.

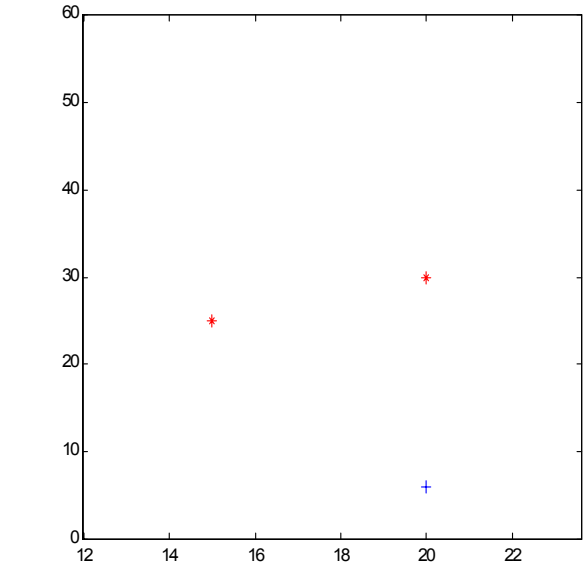
Nous pouvons faire un **zoom** uniquement suivant l'axe y avec la commande **zoom yon** et rend le zoom inactif par la commande **zoom off**.

La commande **zoom** est intéressante pour une lecture précise du point d'intersection de deux courbes.

```
» plot(X, Y, '+b', X1, Y1, '*r')
» zoom
```



» zoom off
» zoom out
» zoom xon



Même déroulement pour la fonction **zoom yon**.

288.6 . RECUPERER LES COORDONNEES D'UN POINT

Le problème précédent peut être résolu par la fonction **ginput** qui récupère les valeurs des abscisses et des ordonnées des points sélectionnés sur la fenêtre graphique courante par l'intermédiaire de la souris.

La syntaxe de la commande **ginput** est la suivante :

```
[x, y] = ginput(n)
```

x, y : vecteurs des coordonnées des points sélectionnés,

n : nombre de points que nous désirons sélectionner. Cet argument est facultatif, s'il n'est pas spécifié, la sélection des points se termine quand nous appuyons sur la touche "Entrée" du clavier.

```
» [x, y]=ginput(1)
x =
   35.0230
y =
   44.7368
```

Nous constatons que ces valeurs sont moins précises que celles obtenues avec la commande **zoom**. La combinaison de la commande **zoom** et de la fonction **ginput** peut donner une meilleure précision.

Cette instruction intéressante **ginput** sera utile en asservissements. En effet, si nous traçons une réponse indicielle d'un système du premier ordre et que nous voulons connaître précisément la valeur de la constante de temps, nous utiliserons cette fonction.

Reprenons l'exemple de la caractéristique d'une vanne et tapons :

```
» X = 0:10:100;Y = [ 4 4 6 9 13 17 22 30 37 51 56];
» plot(X,Y);xlabel('Ouverture de la vanne (%)');ylabel('Débit en m3/h') ;
» title('Caractéristique d''une vanne') ; grid
» [x, y]=ginput(2)
x =
   93.3180
   20.0461
y =
   52.6316
    6.1404
```

Le pointeur de la souris apparaît sur l'écran et choisissez les coordonnées sur lesquelles vous allez cliquer votre souris. Dans cet exemple, vous cliquez 2 fois et les valeurs des coordonnées x et y apparaissent sur l'écran sous forme de vecteurs colonnes de dimension 2.

298.7 . GRAPHIQUES 2D

MATLAB peut introduire des graphiques couleurs 2D et 3D impressionnants. Il fournit aussi les outils et moyens de personnaliser et de modifier pratiquement tous leurs aspects, facilement et de manière parfaitement contrôlée.

Pour avoir un aperçu des possibilités de ***MATLAB***, il est conseillé de consulter le programme de démonstration en entrant la commande **demo** à la suite de l'invite "**»**".

La commande **plot** permet de tracer des graphiques xy. Avec **plot(x,y)** nous traçons y en fonction de x ; x et y sont des vecteurs de données de mêmes dimensions. (voir les exemples N°1 et N°2).

Tracé de fonction

fplot permet le tracé de la courbe d'une fonction y=f(x) dont nous spécifions l'expression sous la forme d'une chaîne de caractère **'f(x)'**.

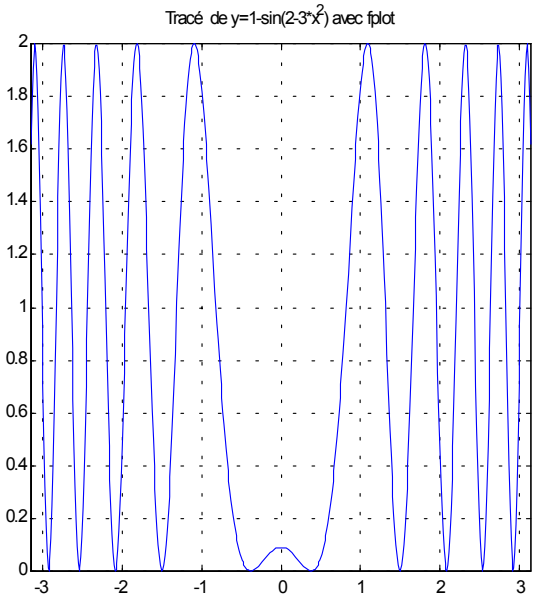
Cette expression peut être remplacée par le nom d'un fichier *.m (fichier de fonction) dans lequel est programmé la fonction **f**.

La syntaxe est la suivante :

fplot('expression', [xMin xMax])

Programme ***MATLAB***

```
» fplot('1-(sin(2-3*x^2))', [-pi pi])
» grid
» title('Tracé de y=1-sin(2-3*x^2) avec fplot')
```

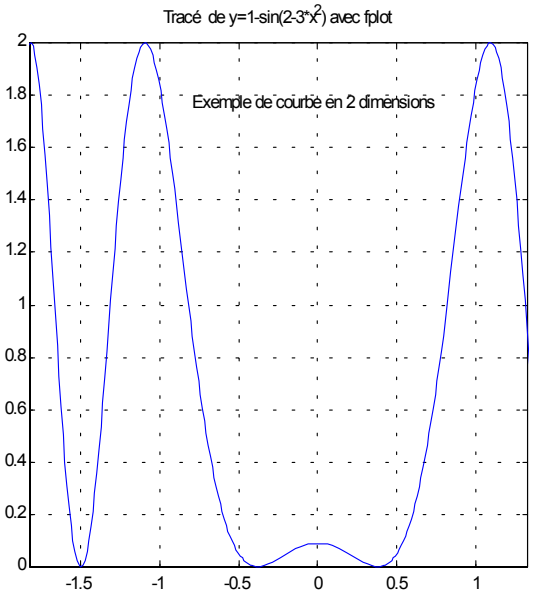


La commande **figure(gcf)** permet de passer du mode intéractif à la fenêtre graphique courante. Vous pouvez aussi utiliser le menu "windows" et sélectionner la fenêtre graphique.

La commande **figure(n)** permet de créer une nouvelle fenêtre avec le numéro **n**.

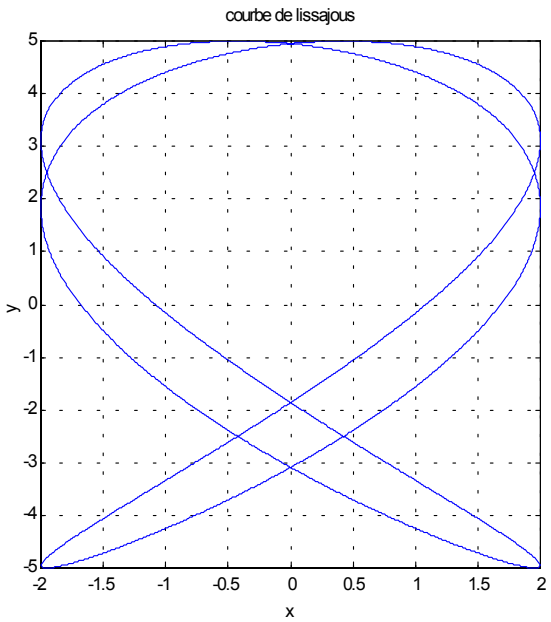
Au graphique courant vous pouvez ajouter un titre, une grille, une légende pour les axes ou du texte sur le graphique grâce aux commandes : **title**, **grid**, **xlabel**, **ylabel**, **text** et **gtext**.

```
» gtext('Exemple de courbe en 2 dimensions')
```



Nous pouvons aussi tracer des courbes paramétriques.

```
» t=[0:0.001:2*pi];
» x=2*sin(4+3*t);
» y=5*sin(2+2*t);
» plot(x,y);
» grid;
» xlabel('x');
» ylabel('y');
» title('courbe de lissajous');
```



308.8 . GRAPHIQUES 3D

Soit l'exemple suivant d'une fonction à 2 variables : $K = \frac{\sin\left(x^2 + y^2\right)}{x^2 + y^2}$ pour x et y variant de $-\pi$ à π avec un pas de $\pi/10$.

```
» x=[-pi:pi/10:pi];
» y=x;
```

Dans la prochaine étape nous générons deux matrices carrées X et Y qui définissent le domaine de calcul de z, nous utiliserons pour ceci la fonction **meshgrid.**

```
» [X, Y]=meshgrid(x, y);
```

Nous évaluons la fonction Z et nous stockons les données dans la variable Z.

```
» Z=sin(X.^2+Y.^2)./(X.^2+Y.^2);
   Warning : Divide by zero.
```

Nous dessinons la surface représentative de la fonction.

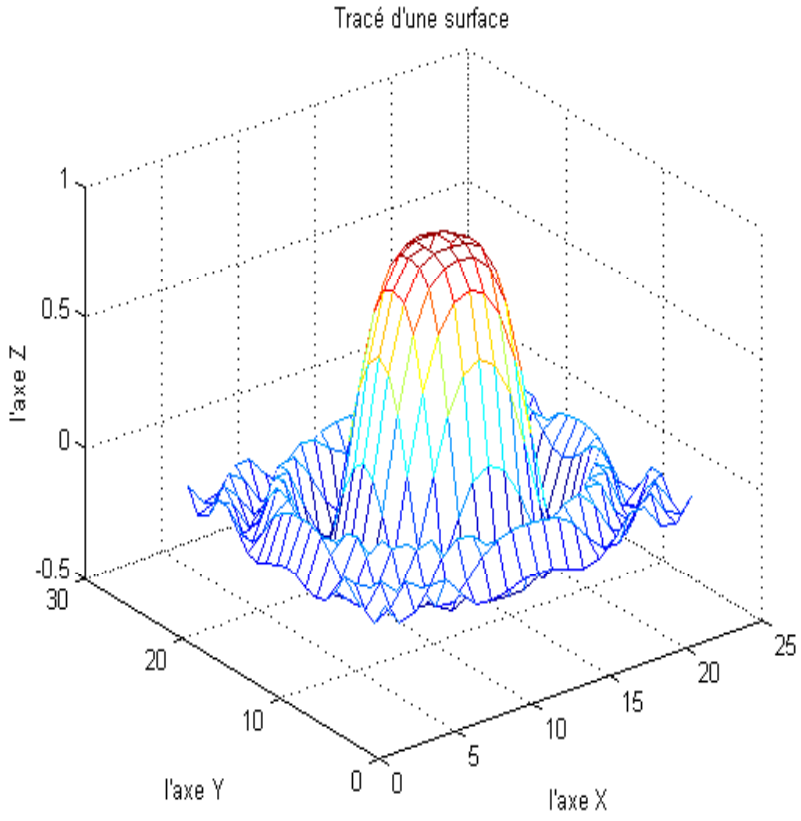
```
» mesh(Z)
```

Nous pouvons rajouter un titre pour le tracé (**title**), des légendes pour les axes (**xlabel**, **ylabel** et **zlabel)** ainsi qu'un quadrillage (**grid**).

```
» xlabel('l''axe X'); ylabel('l''axe Y') ; zlabel('l''axe Z');
» grid; title('Tracé d''une surface');
```

Si une apostrophe est présente dans une chaîne de caractères, il faut la doubler car ***MATLAB*** l'utilise comme délimiteur de chaînes.

D'autres commandes comme **plot3**, **surf**, etc. étendent considérablement les capacités graphiques de ***MATLAB***.



89 . CRÉATION DE FONCTIONS MATLAB

Dans ***MATLAB***, il y a deux types de fichiers : les fichiers de données et les fichiers de programmes (scripts et fonctions).

319.1 . FICHIERS DE DONNEES

En plus des fichiers de données que nous pouvons définir et utiliser par programmation dans ***MATLAB*** nous trouvons les fichier MAT. Ce sont des fichiers binaires (d'extension "mat") qui permettent de stocker et de restituer des variables utilisées dans l'espace de travail. Ces opérations sont réalisées respectivement par les commandes **save** et **load**.

329.2 . FICHIERS DE COMMANDES ET DE FONCTIONS

MATLAB peut exécuter une séquence d'instructions stockées dans un fichier. Ce fichier est appelé fichier M (M-file). Ce nom provient du fait que l'extension est "*.m". La majorité de votre travail avec ***MATLAB*** sera liée à la manipulation de ces fichiers. Il y a deux types de fichiers M : les fichiers de commandes (fichiers scripts) et les fichiers de fonctions.

Manipulation des fichiers M

Au cours de l'utilisation de ***MATLAB***, vous serez amené à éditer des fichiers M (avec votre éditeur de textes préféré) et à retourner à ***MATLAB***. Si vous souhaitez laisser les deux applications actives et basculer de l'une vers l'autre, cela est possible, si vous disposez d'un système d'exploitation multitâches ou si votre environnement graphique vous permet l'ouverture de plusieurs fenêtres applications.

Si ce n'est pas votre cas, vous pourrez exécuter toute commande système ou application externe à ***MATLAB*** à la suite de l'opérateur "!".

Si votre éditeur de texte s'appelle **notepad**,

```
» !notepad monfichier.m
```

Vous pouvez éditer un nouveau fichier auquel vous voulez attribuer le nom "monfichier" ou modifier un fichier existant, de même nom. Une fois les modifications terminées et enregistrées, en quittant votre éditeur, vous retrouverez l'environnement initial de ***MATLAB***.

Les fichiers de commandes (scripts)

Un fichier de commandes ou script est une séquence d'instructions ***MATLAB***. Les variables de ces fichiers sont locales à l'espace de travail. Les valeurs des variables de votre environnement de travail peuvent être modifiées par des instructions des fichiers scripts.

Les fichiers de commandes (scripts) sont aussi utilisés pour la saisie de données. Dans le cas de grandes matrices, l'utilisation de scripts vous permet de corriger facilement et rapidement les erreurs de saisie. Un fichier script peut appeler un autre ou s'appeler lui même de façon récursive.

Nous proposons, dans ce qui suit, un exemple de script, stocké dans un fichier appelé **courbe1.m**, dont le code permet de tracer la courbe de la fonction $y=x^2+5$, sur l'intervalle [-5, 5].

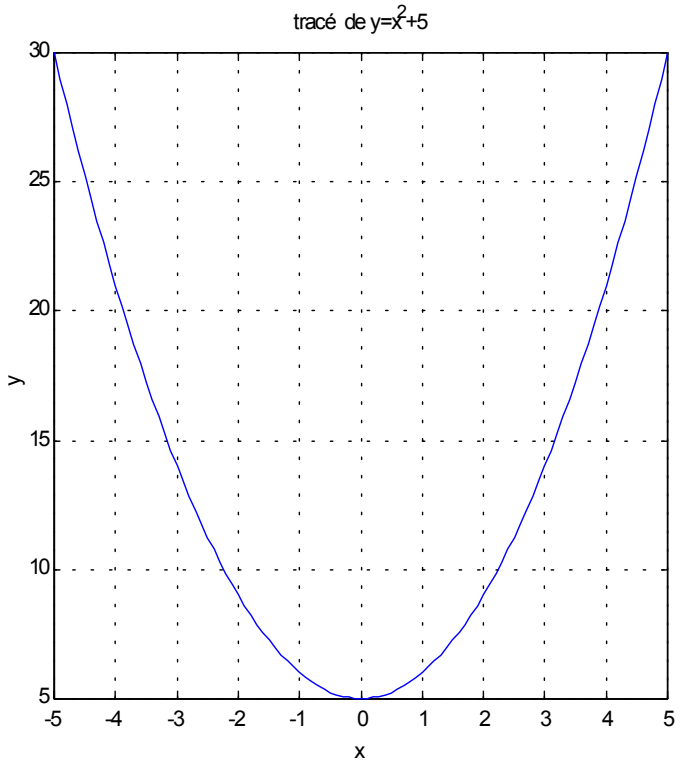
Fichier courbe1.m

```
%domaine du tracé
x=[-5:0.1:5];
%calcul des valeurs de la fonction
y=x.^2+5;
%tracé de la courbe
plot(x,y)
%quadrillage, titre et légendes
grid
title('tracé de y=x^2+5')
xlabel('x')
ylabel('y')
```

Pour invoquer un script, il suffit de l'appeler par son nom.

```
» courbe1
```

L'exécution de ce script donne le résultat suivant :



Remarque

Pour insérer des commentaires dans un programme, il suffit d'utiliser le symbole "%". Tout ce qui suit ce symbole, jusqu'à la fin de la ligne, est considéré comme un commentaire et n'est pas, par conséquent, interprété par ***MATLAB***.

Les fichiers de fonctions

Les fichiers fonctions fournissent une extensibilité à ***MATLAB***. Vous pouvez créer de nouvelles fonctions spécifiques à votre domaine de travail qui auront le même statut que toutes les autres fonctions ***MATLAB***.

Les variables dans les fonctions sont par défaut locales, mais à partir de la version 4.0 de ***MATLAB***, nous pouvons définir des variables globales.

Exemple de fichier fonction

Nous allons écrire une fonction qui nous renvoie le résultat de l'équation $S = (n+1) * n/2$.

Il est intéressant de générer des fonctions qui seront utilisées dans un programme principal. Une fonction ***MATLAB*** a la structure suivante :

function [paramètres de sorties]=nom de la fonction(paramètres d'entrées)

Par exemple, créons une fonction **sn** qui calcule l'opération $S = (n+1) * n/2$. Nous procédons comme suit :

```
function S=sn(n)

% sn.m
% sn : somme de n nombres
% Cette fonction calcule la somme de n nombres
%
   S=(n+1)*n/2;
```

Cette fonction doit être sauvegardée sous le même nom que la fonction. Dans cet exemple, ce sera **sn.m**. Par exemple, nous calculons l'opération avec n=9, le résultat de **S=45**.

```
» S=sn(9)
S =
   45
```

Si vous tapez **help sn**, sur l'écran apparaissent les commentaires suivants :

```
» help sn
sn.m
sn : somme de n nombres
Cette fonction calcule la somme de n nombres
```

10 . UTILISATION DES FONCTIONS DU CONTROL TOOLBOX

Le module de contrôle peut être utilisé avec les systèmes linéaires invariants continus ou discrets. Deux types de modélisation peuvent être envisagés :

- modélisation par fonction de transfert,
- modélisation par espace d'état.

Fonctions ***MATLAB*** utiles en automatique

Nom:	Utilité:
A'	Renvoie la transposée de la matrice A
AXIS	Modifie l'échelle dans un graphique (permet de faire des zooms)
BODE	Trace le lieu de Bode
CLEAR	Détruit toutes les variables
CLG	Efface une fenêtre graphique
FEEDBACK	Fournit la fonction de transfert en BF à partir de la fonction de transfert en BO
CONV	Effectue le produit de convolution entre 2 vecteurs
DISP	Affiche un message ou une variable
FOR	Boucle pour
GINPUT	Récupère les coordonnées de positions pointées à l'écran
HELP	Donne une aide sur l'utilisation d'une fonction
HOLD	Permet de superposer ou non des graphiques de même type
IMPULSE	Trace la réponse impulsionnelle
INPUT	Permet la saisie de données
INV	Inverse les matrices
NGRID	Trace les isophases et les isogains sur un lieu de Black
NICHOLS	Trace le lieu de Black
NYQUIST	Trace le lieu de nyquist
PADE	Approximation des retards ou des fonctions exponentielles
PAUSE	Attend l'appui d'une touche au clavier
PLOT	Trace des courbes dans un repère cartésien
ROOTS	Donne les racines d'un polynôme
STEP	Trace la réponse indicielle
SUBPLOT	Divise un écran graphique en plusieurs sous-écrans
WHILE	Boucle tant que
WHO	Affiche le nom de toutes les variables en cours

Dans les paragraphes suivants, nous envisagerons uniquement les systèmes continus.

3310.1 . MODELISATION PAR FONCTION DE TRANSFERT

010.1.1 . Comment introduire une fonction de transfert sous MATLAB

MATLAB est limité aux systèmes à une entrée, avec une ou plusieurs sorties (SIMO).

Exemple n°1 :

Considérons 2 réservoirs d'eau montés en cascade, l'entrée étant le débit d'eau sur le premier réservoir et la sortie la hauteur d'eau du deuxième réservoir.

La fonction de transfert du système est déterminée et est égale à :

$$H(p) = \frac{0.5}{(50p+1)*(120p+1)}$$

Les constantes de temps sont égales respectivement à $\theta_1 = 50s$ et $\theta_1 = 120s$.

Le gain statique est égal à $k = 0.5$.

Créons un fichier **niveau2.m** sur le logiciel "**notepad**" ou sur le "**debbuger**" de ***MATLAB***.

```
% Ce programme représente la fonction de transfert
% d'un double réservoir
% H(p)=0.5/(50p+1)(120p+1)
num = 0.5;
den1=[50 1];
den2=[120 1];
den = conv(den1,den2);
printsys(num,den,'p')
```

Sur ***MATLAB*** nous obtenons la fonction de transfert suivante :

```
» niveau2
num/den =
            0.5
    ------------------------
    6000 p^2 + 170 p + 1
```

Les variables **num**, **den1**, **den2** et **den** représentent respectivement le numérateur et le dénominateur de la fonction de transfert H(p).

La commande **printsys** permet d'afficher à l'écran la fonction de transfert sous forme de num(s)/den(s), s étant l'opérateur de Laplace par défaut.

Si nous voulons écrire la fonction de transfert en **p**, nous précisons dans **printsys** l'option '**p**'.

Si nous voulons définir une fonction de transfert dans ***MATLAB*** il faut utiliser la fonction **tf**. Cette fonction **tf** permet de définir la fonction de transfert

```
» sys=tf(num,den)
Transfer function:
            0.5
    -------------------------
    6000 s^2 + 170 s + 1
```

Dans ce cas, la variable **sys** représente la fonction de transfert H(p)

Exemple n°2 :

Nous pouvons aussi modéliser une fonction de transfert par factorisation des pôles et des zéros.

Cette fonction est représentée par $H(p) = k * \frac{\prod_{i=1}^{m}(p - z_i)}{\prod_{j=1}^{n}(p - p_j)}$ où k représente le gain et les p_j et les z_i représentent respectivement les pôles et les zéros de la fonction de transfert.

Considérons l'exemple suivant :

$$H(p) = \frac{0.5}{120*50} * \frac{1}{\left(p+\frac{1}{50}\right)*\left(p+\frac{1}{120}\right)}$$

Créons un nouveau programme **zptfniv2.m** (passage des zéros-pôles vers la fonction de transfert)

```
% zptfniv2
% Ce programme crée une fonction de transfert à partir des pôles et des zéros
% d'un double réservoir
% H(p)=0.5/(50p+1)(120p+1)
z=inf;
% Le zéro est infini
k=0.5/(50*120);
p=[-1/50 -1/120];
[num, den]=zp2tf(z,p,k)
printsys(num,den,'p')
```

Nous obtenons en exécutant **zptfniv2** sur ***MATLAB*** le résultat suivant :

```
» zptfniv2
num =
   1.0e-004 *
        0       0       0.8333
den =
   1.0000   0.0283   0.0002
num/den =
                8.3333e-005
   ------------------------------------
   p^2 + 0.028333 p + 0.00016667
```

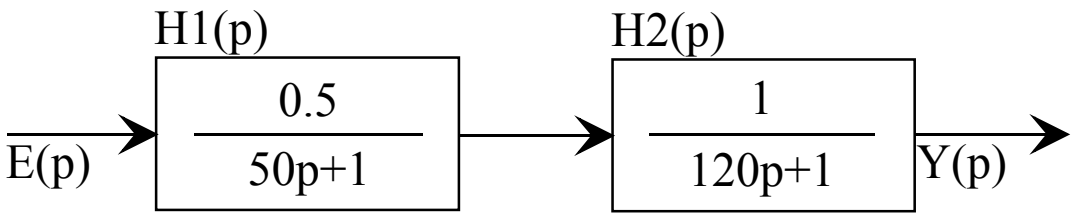
En multipliant le numérateur et le dénominateur de cette expression par **1/8.3333e-005**, nous retrouvons le même résultat que précédemment.

110.1.2 . Opérations sur des Schémas blocs sous MATLAB

Les opérations sur des schémas blocs telles que la mise en série, la mise en parallèle et la mise en boucle fermée sont représentées respectivement par **series**, **parallel** et **feedback**.

10.1.2.1 . Utilisation de series

Prenons comme exemple un système composé de deux fonctions de transfert en séries. Le but est de calculer la fonction de transfert équivalente :



Programme ***MATLAB*** :

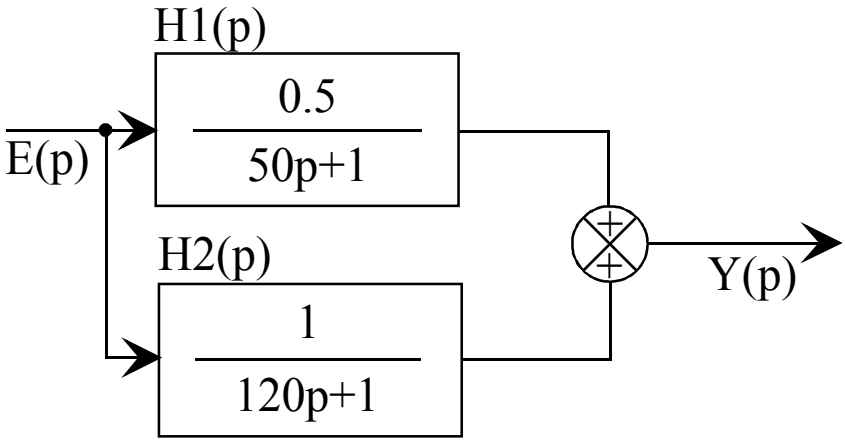
```
% niv2s
% Ce programme crée une fonction de transfert à partir des fonctions
% de transfert de chaque réservoir
% H1(p)=0.5/(50p+1) et H2(p)=1/(120p+1)
n1=0.5;
d1=[50 1];
n2=1;
d2=[120 1];
[num,den]=series(n1,d1,n2,d2)
printsys(num,den,'p')
```

Sur la fenêtre de ***MATLAB*** apparaît :

```
» niv2s
num =
    0         0         0.5000
den =
    6000      170       1
num/den =
                0.5
    ------------------------
    6000 p^2 + 170 p + 1
```

10.1.2.2 . Utilisation de parallel

Prenons comme exemple un système composé de deux fonctions de transfert en séries. Le but est de calculer la fonction de transfert équivalente :



Programme ***MATLAB*** :

```
% niv2p
% Ce programme crée une fonction de transfert à partir des fonctions
% de transfert de chaque réservoir
% H1(p)=0.5/(50p+1) et H2(p)=1/(120p+1)
n1=0.5;
d1=[50 1];
sys1=tf(n1, d1);
n2=1;
d2=[120 1];
sys2=tf(n2, d2);
parallel(sys1, sys2)
```

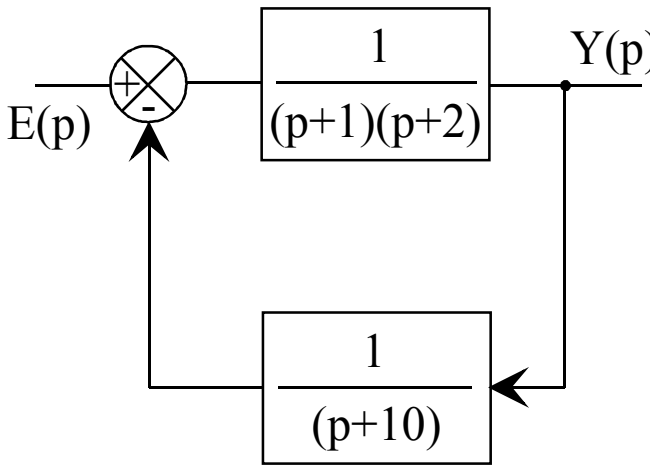
Sur la fenêtre de ***MATLAB*** apparaît :

```
» niv2p
Transfer function:
          110 s + 1.5
    ------------------------
    6000 s^2 + 170 s + 1
```

10.1.2.3 . Utilisation de feedback

La fonction **feedback** donne la fonction de transfert en boucle fermée.

Soit le schéma bloc suivant :



Calculons la fonction de transfert en boucle fermée du schéma bloc ci-dessus.

Programme ***MATLAB*** :

```
%bf.m
n1=1;
d1=conv([1 1],[1 2]);
n2=1;d2=[1 10];
[num,den]=feedback(n1,d1,n2,d2)
printsys(num,den,'p')
```

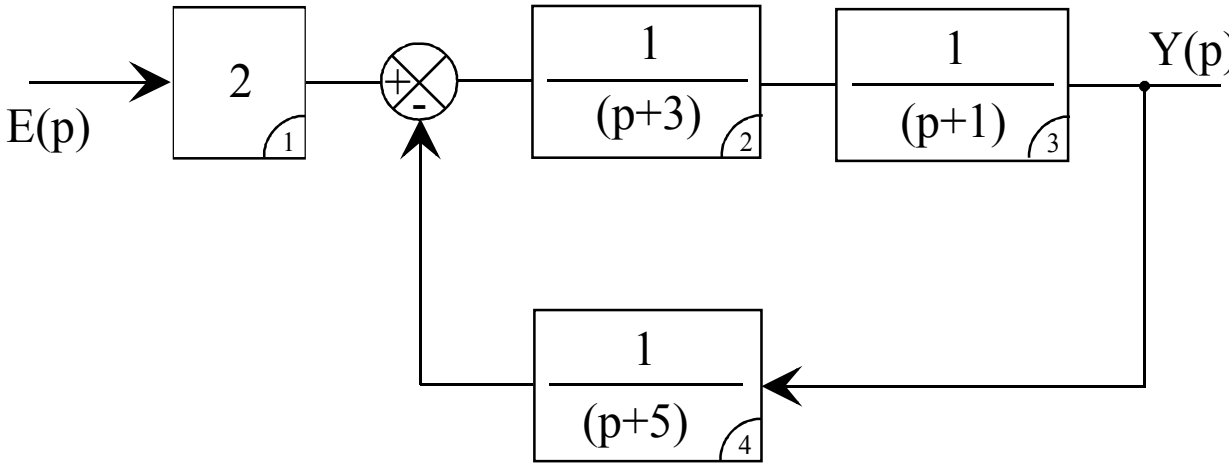
Sur la fenêtre de ***MATLAB*** apparaît :

```
» bf
num =
    0     0     1    10
den =
    1    13    32    21
num/den =
                 p + 10
  -----------------------------
  p^3 + 13 p^2 + 32 p + 21
```

Pour réduire les schémas blocs, nous pouvons utiliser dans le même programme les fonctions **feedback, series et parallel** mais il est beaucoup plus intéressant d'utiliser la fonction "**blkbuild**".

10.1.2.4 . Utilisation de blkbuild

Soit le schéma bloc suivant :



Nous pouvons remarquer que chaque bloc a un numéro d'identification dans le bas à droite du bloc cela est très utile pour la définition des liaisons.

```
%redshema.m
%Définition du bloc numéro 1

n1=2;
d1=1;

%définition du bloc numéro 2

n2=1;
d2=[1 3];

%définition du bloc numéro 3

n3=1;
d3=[1 1];

%définition du bloc numéro 4
```

```
n4=1:
d4=[1 5];

%nombre de blocs dans le schéma blocs

nblocks=4;

%conversion du schéma blocs dans l'espace d'état

blkbuild

%Définition des liaisons
%A l'entrée du bloc numéro 2 nous trouvons la sortie du
%bloc numéro 1 et la sortie du bloc numéro 4 mais le bloc numéro 4 passe
%dans un sommateur avec un signe -. Donc, dans la première colonne de la matrice
%nous trouvons
%le numéro du bloc que nous allons définir la liaison ici "2"
%à l'entrée de ce bloc nous trouvons le bloc "1" qui correspond à la deuxième colonne
%et dans la dernière colonne nous trouvons la deuxième connection ici "-4" à cause du
%sommateur
%Ensuite nous effectuons la même démarche. A l'entrée du bloc numéro "3" nous
%trouvons
%la sortie du bloc numéro "2". Et à l'entrée du bloc numéro "4" nous trouvons la
%sortie du bloc numéro "3"

q=[2 1 -4 ; 3 2 0 ; 4 3 0];

%L'entrée se fait sur le bloc numéro "1"

entree=1;

%La sortie se fait sur le bloc numéro "3"

sortie=3;

%CONNECT permet de calculer la fonction de transfert dans l'espace d'état du schéma
%bloc défini ci-dessus

[a,b,c,d]=connect(a,b,c,d,q,entree,sortie);

%SS2TF fait la conversion d'une fonction de transfert définie dans l'espace d'état
%à une fonction de transfert en Laplace

[num,den]=ss2tf(a,b,c,d);

%affichage du résultat sous forme d'une fonction de transfert

printsys(num,den,'p')
```

Si nous exécutons ce programme, nous obtenons :

```
» redshema
State model [a,b,c,d] of the block diagram has 4 inputs and 4 outputs
num/den =
    3.5527e-015 p^2 + 2 p + 10
    ------------------------------
      p^3 + 9 p^2 + 23 p + 16
```

210.1.3 . Calcul et tracé des réponses temporelles à partir de la fonction de transfert

A partir d'une fonction de transfert, ***MATLAB*** permet de tracer des réponses indicielles (**step**), réponse impulsionnelle (**impulse**), réponse indicielle avec conditions initiales (**initial**), réponse à un signal quelconque (**lsim**).

Traçons la réponse indicielle d'un système du deuxième ordre avec un coefficient d'amortissement $\xi = 0.3$, une pulsation propre égale à $\omega_n 10$ rad/s et un gain statique égal à 1.

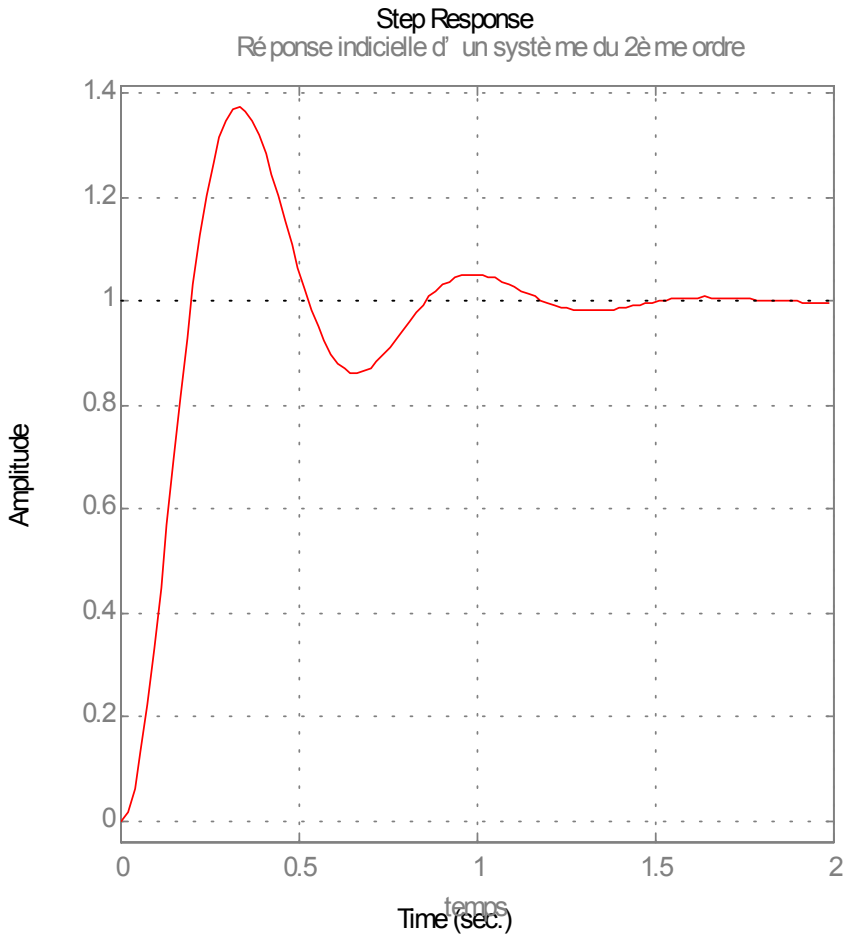
Appelons le programme **riord2.m**

```
% riord2.m

Xi=0.3 ;
wn=10 ;
[num,den]=ord2(wn,Xi)

% Nous multiplions le numérateur par wn^2 pour avoir la fonction sous la forme
% H(p)=Wn^2/(P^2+2*Xi*Wn*P+ Wn^2)

num=num*wn^2;
Systeme=tf(num,den)
step(Systeme);
title('Réponse indicielle d'un système du 2ème ordre');
xlabel('temps');
grid;
```

Si nous exécutons ce programme, nous obtenons :

```
» riord2
num =
    1

den =
    1     6     100
Transfer function:
          100
    ------------------
    s^2 + 6 s + 100
```

Et nous obtenons la figure suivante :



Nous pouvons analyser cette réponse indicielle.

Grâce à cette courbe nous pouvons calculer la valeur finale, l'instant du premier dépassement, le dépassement en %, le temps de réponse et le temps de montée par exemple.

```
%Analord2.m
%appel du programme de la F.T du 2ème ordre.
riord2
%Calcul de la valeur finale
valeurfinale=polyval(num,0)/polyval(den,0)
%Calcul de l'instant du premier dépassement
[y,x,t]=step(num,den);
[Y,k]=max(y);
tempsdepass=t(k);
%Calcul du dépassement
Depassement=100*(Y-valeurfinale)/valeurfinale
%Calcul du temps de montée entre 10% et 90% de la valeur finale
n=1;
while   y(n)<0.9*valeurfinale
        n=n+1;
end
m=1;
```

```
while    y(m)<0.1*valeurfinale
         m=m+1;
end
tempsmontee=t(n)-t(m)
```

Nous obtenons sur ***MATLAB*** les résultats suivant :

```
» analord2
num =
    1
den =
    1      6      100
Transfer function:
          100
    ------------------
    s^2 + 6 s + 100
valeurfinale =
    1
Depassement =
    37.2255
tempsmontee =
    0.1288
```

310.1.4 . Calcul et tracé des réponses fréquentielles à partir d'une fonction de transfert

Le module de Contrôle permet de générer les diagrammes classiques tels que :

- Diagramme de Bode (**bode**),

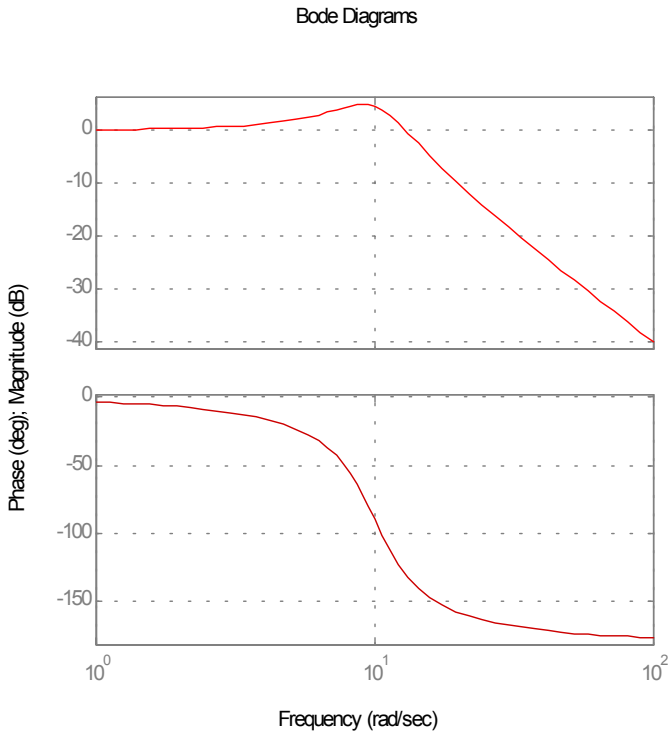
- Diagramme de Nyquist (**nyquist**),

- Diagramme de Black-Nichols (**nichols**).

Ainsi pour tracer le diagramme de Bode de la fonction de transfert de deuxième ordre, nous écrivons :

```
» % Le deuxième ordre est placé dans la variable Systeme
» bode(Systeme)
```

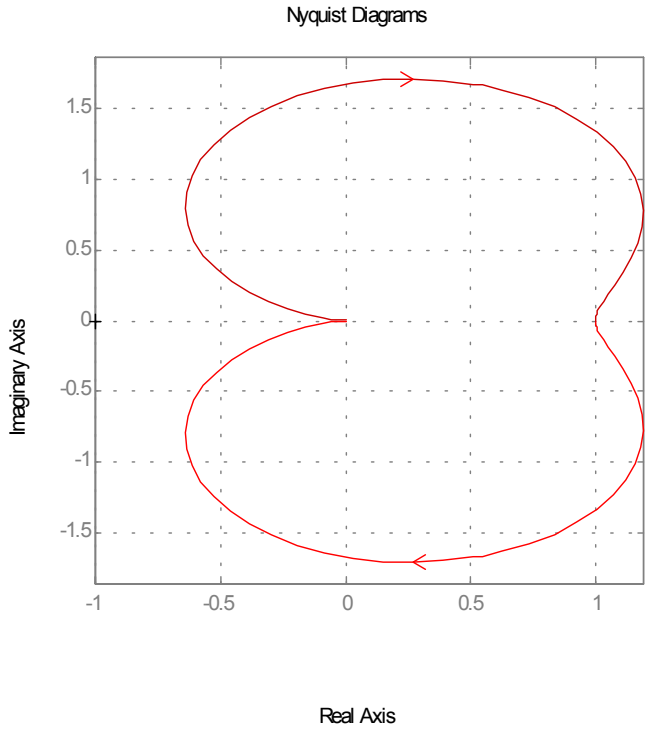
Nous obtenons la figure suivante :



Nous pouvons aussi tracer le diagramme de Nyquist par la fonction suivante :

```
» nyquist(Systeme)
» grid
```

Nous obtenons la figure suivante :

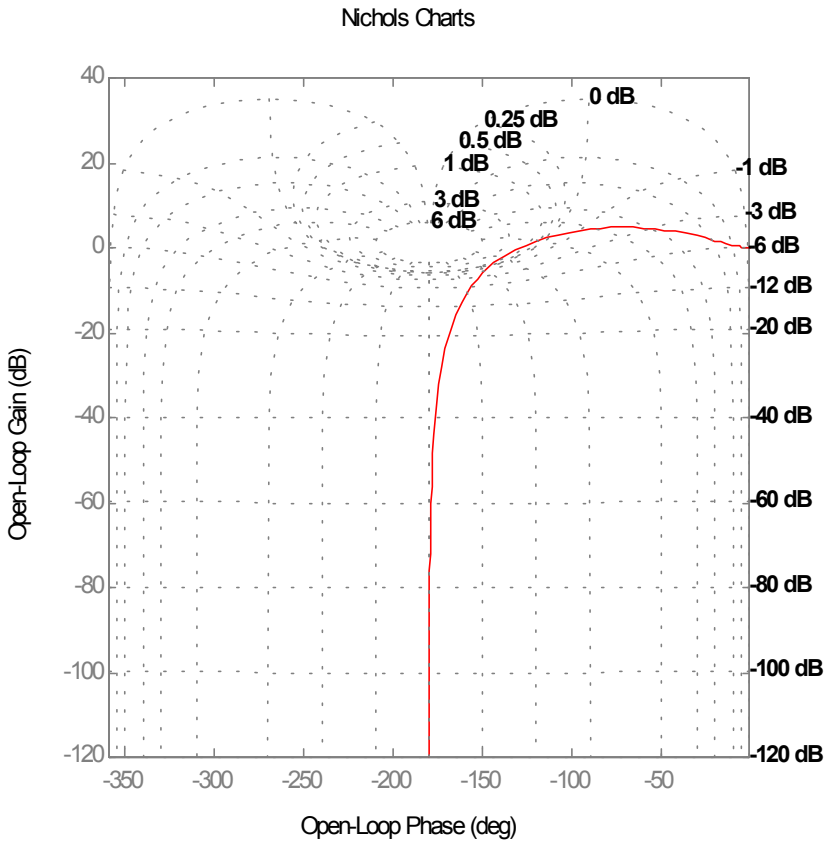


Il est à remarquer que la fonction **nyquist** trace aussi le lieu complémentaire donc pour les fréquences négatives.

Nous pouvons tracer aussi un diagramme de Black-Nichols avec la fonction suivante :

```
» nichols(Systeme)
» ngrid
```

Nous obtenons la figure suivante :

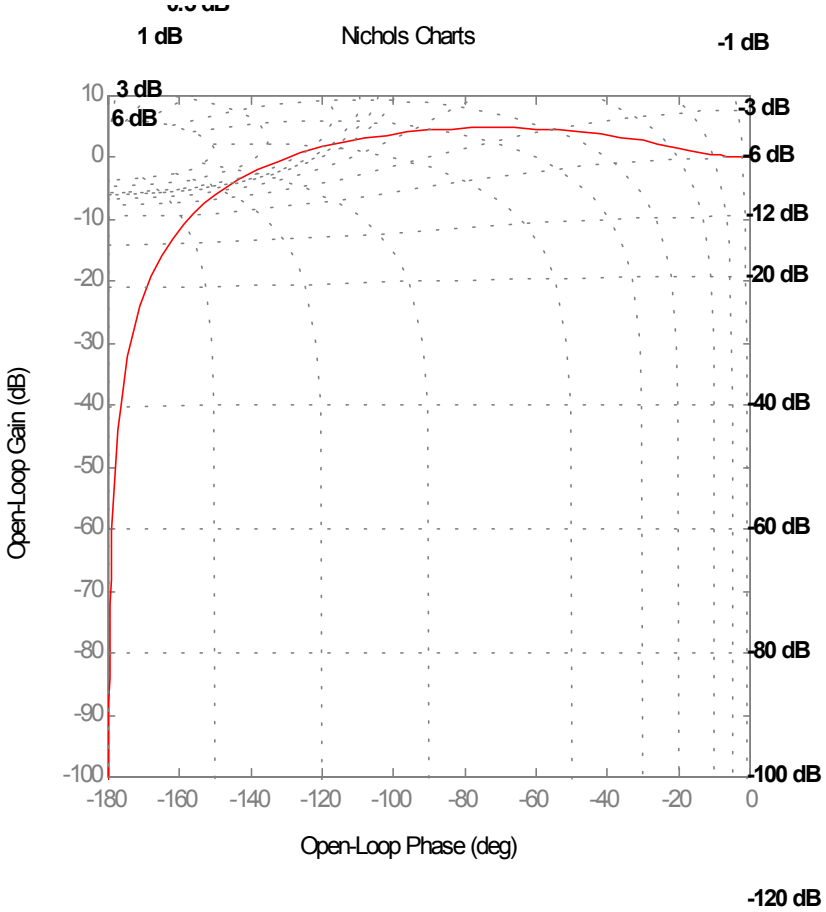


La commande **ngrid** génère l'abaque de Black-Nichols.

Il arrive souvent que les axes générés automatiquement par ***MATLAB*** ne conviennent pas, il faut alors redéfinir de nouveaux axes appropriés cela est possible à l'aide de la fonction **axis**.

```
» nichols(Systeme)
» ngrid
» axis([-180 0 -100 10])
```

Nous obtenons la figure suivante :



Il est intéressant de calculer la bande passante d'un système. Créons le programme bandord2.m.

```
%bandord2.m
riord2
[amplitude,phase,w]=bode(Systeme);
[amplitudemaxi,k]=max(amplitude);
wmaxi=w(k);
n=1;
while 20*log10(amplitude(n))>=-3,
  n=n+1;
end
banpass=w(n)
```

Nous obtenons sur ***MATLAB*** :

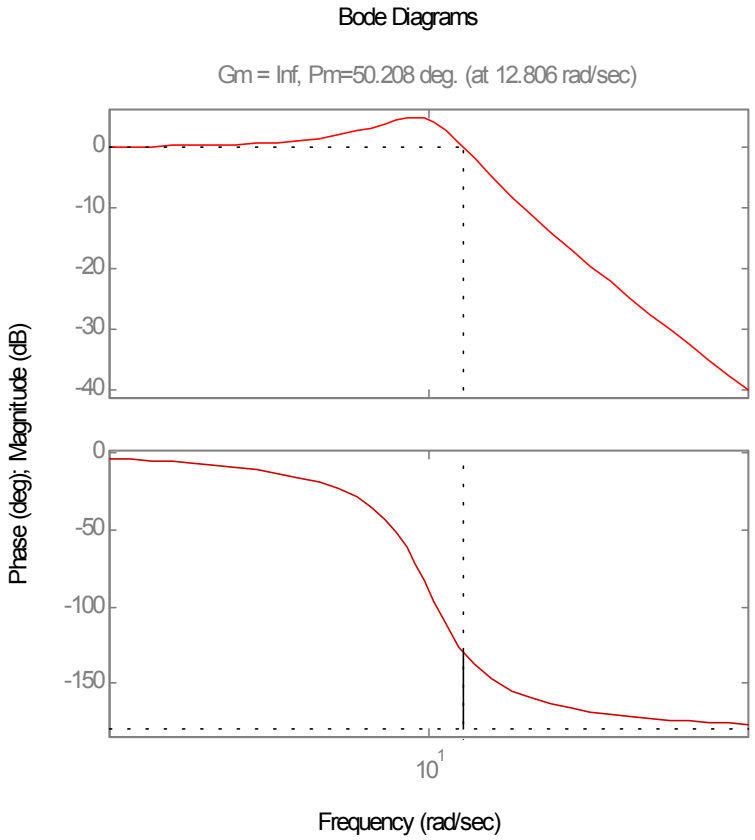
```
» bandord2
num =
    1
den =
    1     6     100
Transfer function:
        100
    ------------------
    s^2 + 6 s + 100
banpass =
```

15.7085

Nous pouvons calculer la marge de gain, la marge de phase, la pulsation qui donne un gain unitaire (w_c) et la pulsation qui donne une phase de -180° (w_g) par la fonction **margin**.

```
» % Le deuxième ordre est placé dans la variable Systeme
» margin(Systeme)
```

Nous obtenons sur ***MATLAB*** la figure suivante :



Le Superviseur de contrôle de ***MATLAB*** : **LTIVIEW** (linear time variant visualisation) permet d'effectuer tous les types des réponses d'une fonction de transfert.

Une fois que nous avons défini des fonctions de transfert, nous pouvons visualiser et tracer les différentes réponses temporelles et fréquentielles.

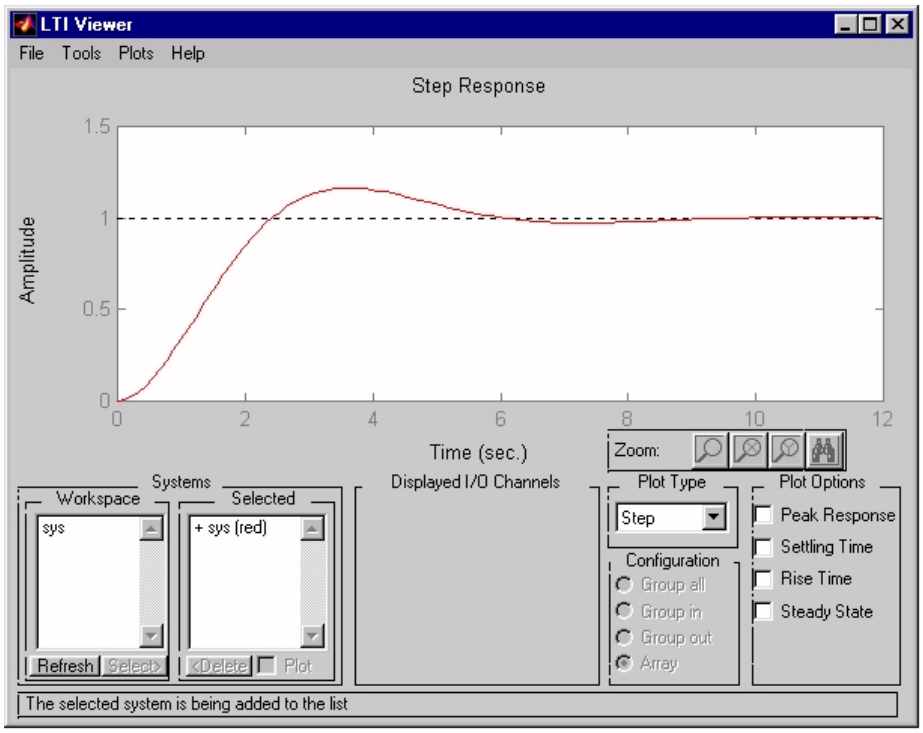
LTIVIEW permet de calculer les différentes grandeurs comme le temps de réponse, les marges de stabilité, etc…

Exemple :

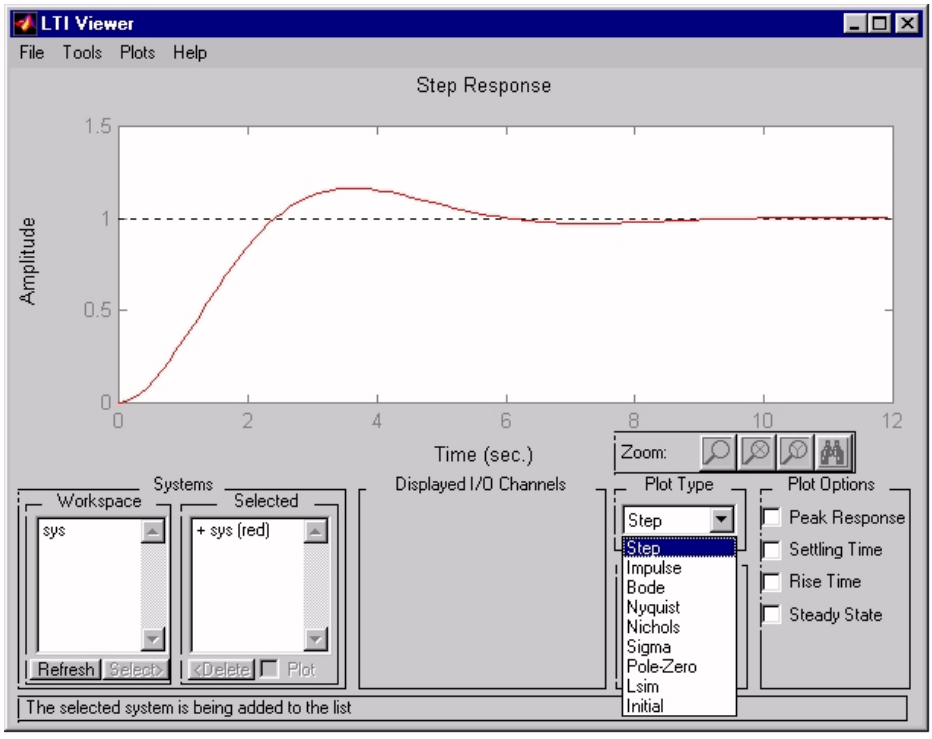
```
» num=1
num =
    1
» den=[1 1 1]
den =
    1     1     1
» sys=tf(num, den)
Transfer function:
```

```
          1
    -------------
    s^2 + s + 1
» ltiview
```

Nous obtenons sur ***MATLAB*** la fenêtre suivante :



Le choix de la réponse se fait sur le menu suivant :



410.1.4 . Calcul et tracé des réponses fréquentielles et temporelles à partir d'une fonction de transfert avec retard pur

Nous voulons simuler le comportement de la fonction de transfert suivante : $H(p) = \frac{0.5 * e^{-2p}}{(50p + 1)}$

Il faut définir la fonction de transfert avec le retard pur. Ce retard sera une valeur estimée.

Dans ce cas, le programme doit se décomposer en deux parties, nous devons déjà définir la fonction de transfert équivalent au retard pur :

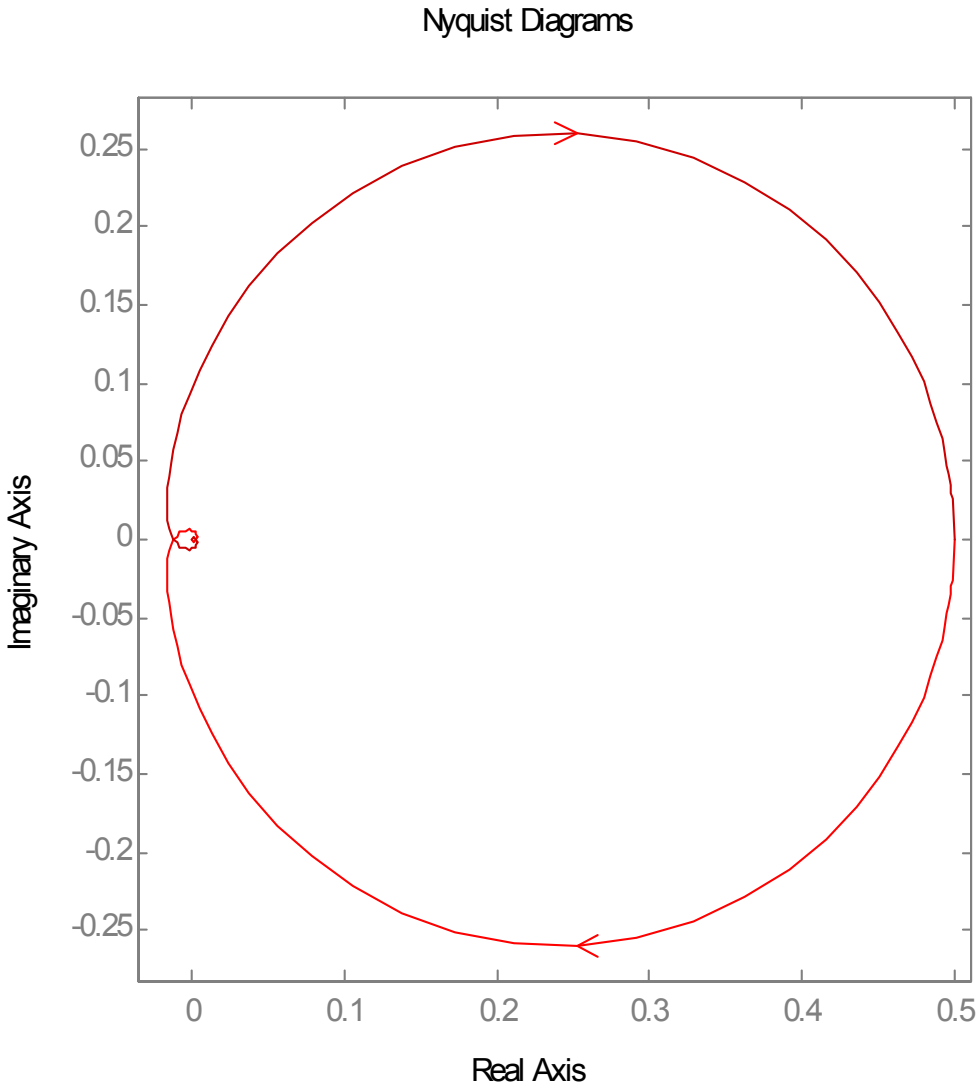
```
» [num1, den1]=pade(2,2)
num1 =
    1     -3     3
den1 =
    1     3     3
```

La fonction **pade(T, N)** permet de calculer la fonction de transfert équivalente au retard pur e^{-Tp} avec **T** le retard pur et **N** le degré estimation.

Ensuite nous définissons la fonction de transfert générale : $\frac{0.5}{(50p + 1)}$

```
» [num1, den1]=pade(2,2)
num1 =
    1     -3     3
den1 =
    1     3     3
» num2=0.5
num2 =
    0.5
» den2=[50 1]
den2 =
    50     1
» num=conv(num1, num2)
num =
    0.5    -1.5   1.5
» den=conv(den1, den2)
den =
    50     151    153    3
» nyquist(num, den)
```

Et nous obtenons la figure suivante :



3410.2 . MODELISATION PAR ESPACE D'ETAT

Les équations d'état d'un système continu s'écrivent :

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

La représentation par espace d'état peut travailler avec les systèmes multi-entrées multi-sorties (MIMO). Sur ***MATLAB***, nous travaillons directement avec les matrices A, B, C et D.

510.2.1 . Passage d'une représentation à l'autre

Nous pouvons passer d'un modèle représenté par fonction de transfert à un modèle représenté dans l'espace d'état par la fonction suivante **tf2ss** et vice-versa par la fonction **ss2ft**. De même nous pouvons passer de la représentation par pôles et zéros à la représentation d'état par la fonction **zp2ss** et vice-versa par la fonction **ss2zp**.

Reprenons l'exemple du système du deuxième ordre.

```
» Xi=0.3 ;
wn=10 ;
[num,den]=ord2(wn,Xi)
% Nous multiplions le numérateur par wn^2 pour avoir la fonction sur la forme
% H(p)=Wn^2/(P^2+2*Xi*Wn*P+ Wn^2)
num=num*wn^2;
num =
    1
den =
    1    6    100
» [A,B,C,D]=tf2ss(num, den)
A =
   -6   -100
    1    0
B =
    1
    0
C =
    0    100
D =
    0
```

610.2.2 . Commandabilité et observabilité d'un système

10.2.2.1 . Commandabilité

L'opération de commandabilité est assurée par **ctrb** et l'observabilité du système par **obsv**.

```
» Co=ctrb(A,B)
Co =
    1    -6
    0    1
```

Si le rang de la matrice Co est égal à la dimension de la matrice A, alors le système est commandable. Nous pouvons effectuer le test suivant.

```
%Co1.m
Xi=0.3 ;
wn=10 ;
[num,den]=ord2(wn,Xi);
% Nous multiplions le numérateur par wn^2 pour avoir la fonction sous la forme
% H(p)=Wn^2/(P^2+2*Xi*Wn*P+ Wn^2)
num=num*wn^2;
[A,B,C,D]=tf2ss(num, den);
Co =ctrb(A,B);
n=length(A);
if rank(Co)==n
disp('système commandable')
else disp('système non commandable')
end
```

Sur ***MATLAB*** nous observons le résultat suivant :

```
» Co1
système commandable
```

10.2.2.2 . Observabilité

De même, nous pouvons calculer la matrice d'observabilité.

```
» Xi=0.3 ;
wn=10 ;
[num,den]=ord2(wn,Xi)
% Nous multiplions le numérateur par wn^2 pour avoir la fonction sur la forme
% H(p)=Wn^2/(P^2+2*Xi*Wn*P+ Wn^2)
num=num*wn^2;
num =
    1
den =
    1      6      100
» [A,B,C,D]=tf2ss(num, den)
A =
   -6     -100
    1      0
B =
    1
    0
C =
    0      100
D =
0
» Ob=obsv(A,C)
Ob =
    0      100
   100     0
```

Si le rang de la matrice Ob est égal à la dimension de la matrice A, alors le système est observable.

```
%Ob1.m
Xi=0.3 ;
wn=10 ;
[num,den]=ord2(wn,Xi);
% Nous multiplions le numérateur par wn^2 pour avoir la fonction sous la forme
% H(p)=Wn^2/(P^2+2*Xi*Wn*P+ Wn^2)
num=num*wn^2;

[A,B,C,D]=tf2ss(num, den);

Ob =obsv(A,C);
n=length(A);

if rank(Ob)==n
disp('système observable')
else disp('système non observable')
end
```

Sur ***MATLAB*** nous observons le résultat suivant :

```
» Ob1
système observable
```

710.2.3 . Calcul des coefficients d'amortissement à partir de la matrice d'état

Nous pouvons calculer le coefficient d'amortissement (ξ) et la pulsation propre non amortie (ω_n) en utilisant la fonction **damp** à partir de la matrice A du système modélisé dans l'espace d'état.

```
» damp(A)
   Eigenvalue                  Damping      Freq. (rad/s)
   -3.00e+000 + 9.54e+000i     3.00e-001    1.00e+001
   -3.00e+000 - 9.54e+000i     3.00e-001    1.00e+001
```

1011 . L'OUTIL SIMULINK

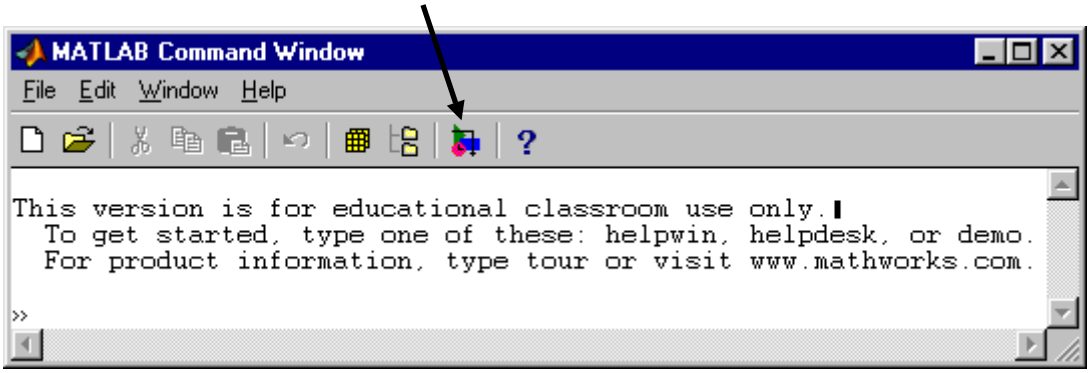
Ce logiciel est un programme de simulation destiné à traiter les systèmes dynamiques.

Comme extension de ***MATLAB***, ***SIMULINK*** comprend plusieurs caractéristiques spécifiques aux systèmes dynamiques. Sur ***SIMULINK*** toutes les fonctions de ***MATLAB*** restent valable. Nous pouvons considérer ***SIMULINK*** comme une version graphique de ce logiciel. Ainsi, avec ***SIMULINK***, pour intégrer un modèle, il suffit de dessiner le schéma bloc correspondant au système et suivre l'évolution de la simulation directement sur des blocs appelés "**Scope**", "**Graph**", etc ...

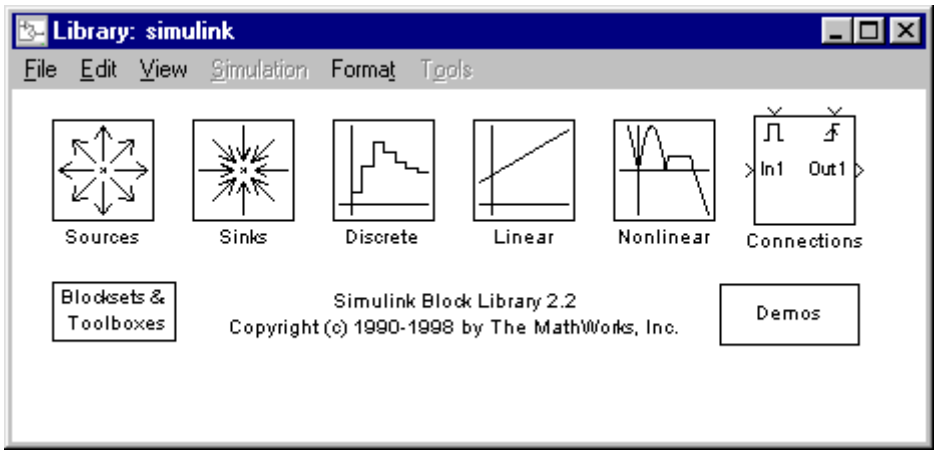
Tous les composants nécessaires à la réalisation du schéma bloc d'un système se trouvent dans la bibliothèque standard de ***SIMULINK***. En fait cette bibliothèque est composée de sous systèmes dans lesquels sont groupés les différents éléments (sommateur, fonction de transfert, intégrateur, limiteur,...). Des blocs réalisant de nouvelles fonctions peuvent être créés et rajoutés soit à la bibliothèque standard ou bien à une bibliothèque.

3511.1 PRESENTATION

Pour démarrer **SIMULINK**, il suffit de taper la commande "**simulink**" sous environnement ***MATLAB*** ou cliquer sur l'icône suivante  dans la fenêtre de ***MATLAB.***



La fenêtre suivante apparaît :

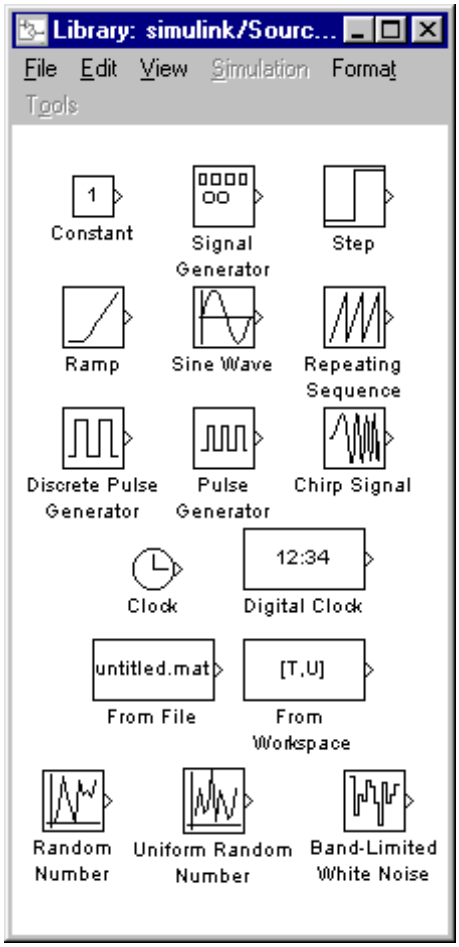


Cette fenêtre contient les 7 familles de blocs (**Sources**, **Sinks**, **Discrete**, **Linear**, **Nonlinear**, **Connections** et **Toolboxes**) utilisés lors des simulations. Pour avoir accès à ces blocs, il suffit de cliquer sur l'un d'entre eux.

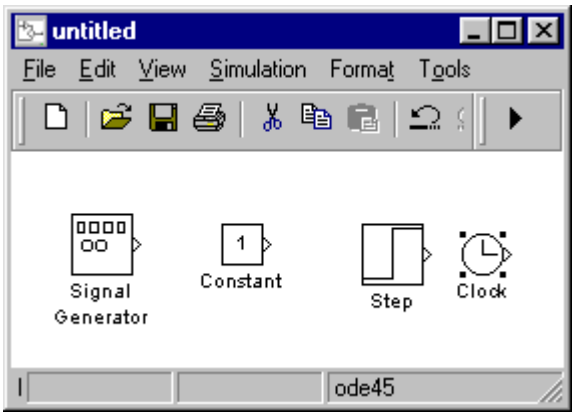
3611.2 . BLOCS UTILISES DANS LES SIMULATIONS

811.2.1 . Famille Sources

Cette famille contient des signaux d'entrée pour les simulations que nous réaliserons.



Les blocs usuels sont :



Les paramètres du bloc "**signal generator**" se règlent par l'intermédiaire du mask : nous choisissons le type du signal (sinus, carré ou triangle), la fréquence et l'amplitude. La fréquence du signal est donnée en rad/s ce qui correspond à une pulsation.

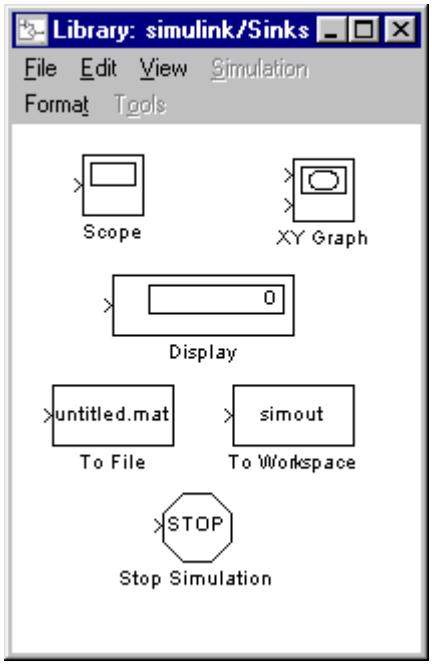
Le bloc "**constant**" donne une valeur d'entrée constante.

Le bloc "**step input**" permet d'appliquer un échelon.

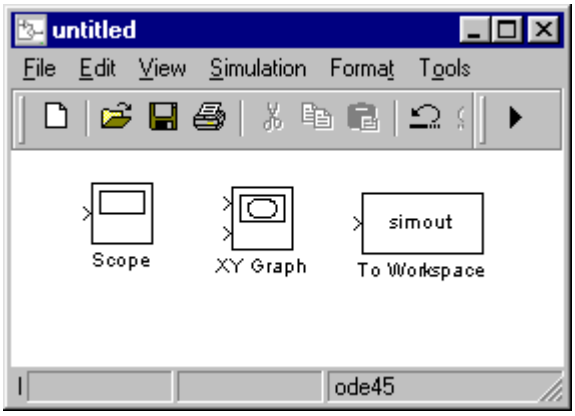
Le bloc "**clock**" permet de connaître le temps qui s'écoule pendant la simulation.

911.2.2 . Famille Sinks

Cette famille contient les blocs de visualisation des résultats.



Les blocs usuels sont :



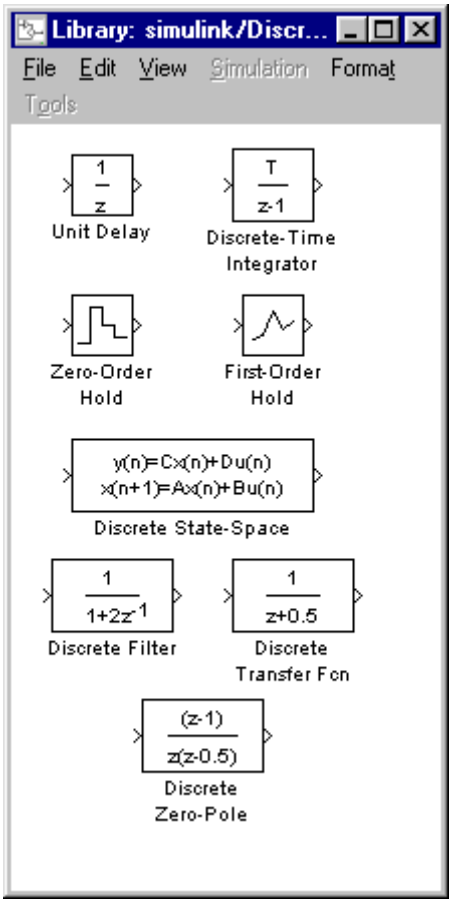
Le bloc "**scope**" est un oscilloscope qui permet de visualiser les sorties sous **SIMULINK**. La plage de visualisation est réglée en abscisses et en ordonnées.

Le bloc "**XY graph**" permet aussi de visualiser les sorties. Nous réglons les amplitudes maximum et minimum, le temps de visualisation, le nombre de points et la couleur des courbes.

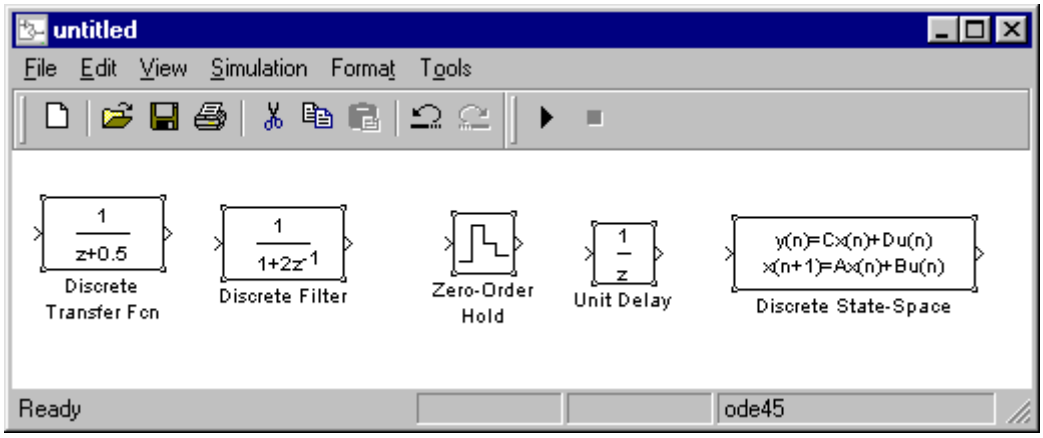
Le bloc "**simout**" permet de stocker les résultats des sorties sous l'environnement ***MATLAB*** dans une variable nommée **yout**.

1011.2.3 . Famille Discrete

Cette famille contient des blocs qui fonctionnent en mode linéaire discret.



Les blocs usuels sont :



Le bloc "**discrete transfert fcn**" permet de modéliser un système en transformée en z.

Le bloc "**filter**" permet de modéliser un système en z^{-1}.

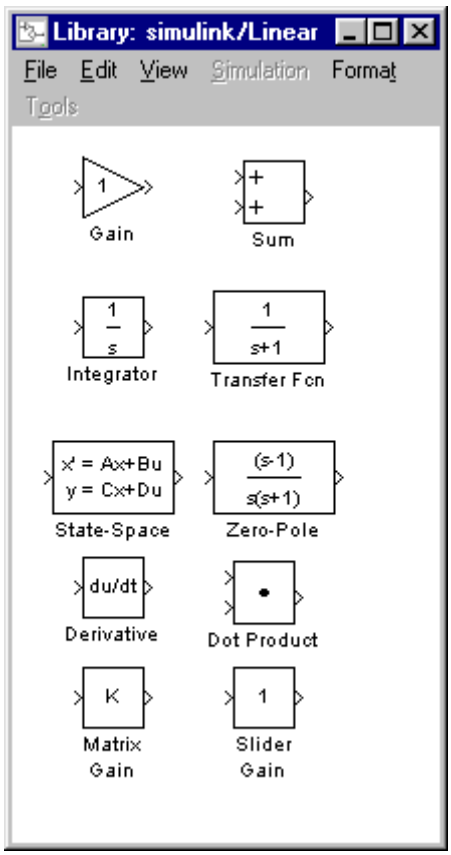
Le bloc "**zero-order hold**" est un bloqueur d'ordre zéro qui maintient une valeur constante entre deux périodes d'échantillonnage.

Le bloc "**unit delay**" est un bloc qui retarde une variable d'une période d'échantillonnage.

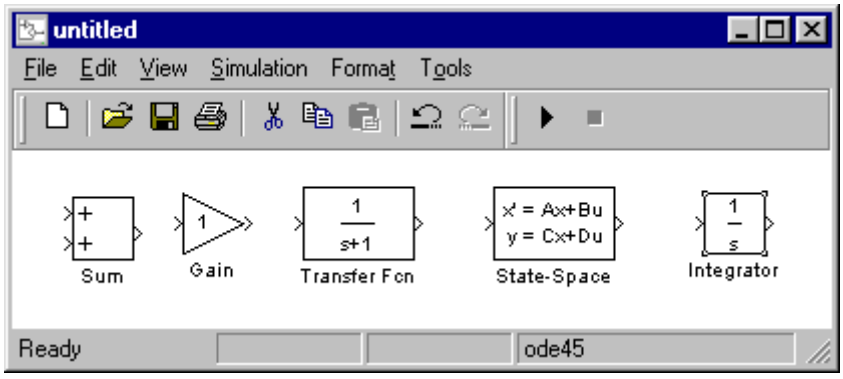
Le bloc "**dicrete state space**" permet de modéliser un système sous forme de variables d'état discrètes.

1111.2.4 . Famille Linear

Cette famille contient des blocs qui fonctionnent en mode linéaire continu.



Les blocs usuels sont :



Le bloc "**sum**" permet de faire la somme ou la différence de deux entrées. Ce bloc sera utile pour l'association des actions d'un régulateur.

Le bloc "**gain**" amplifie la valeur de l'entrée. Ce bloc peut jouer le rôle d'un régulateur proportionnel.

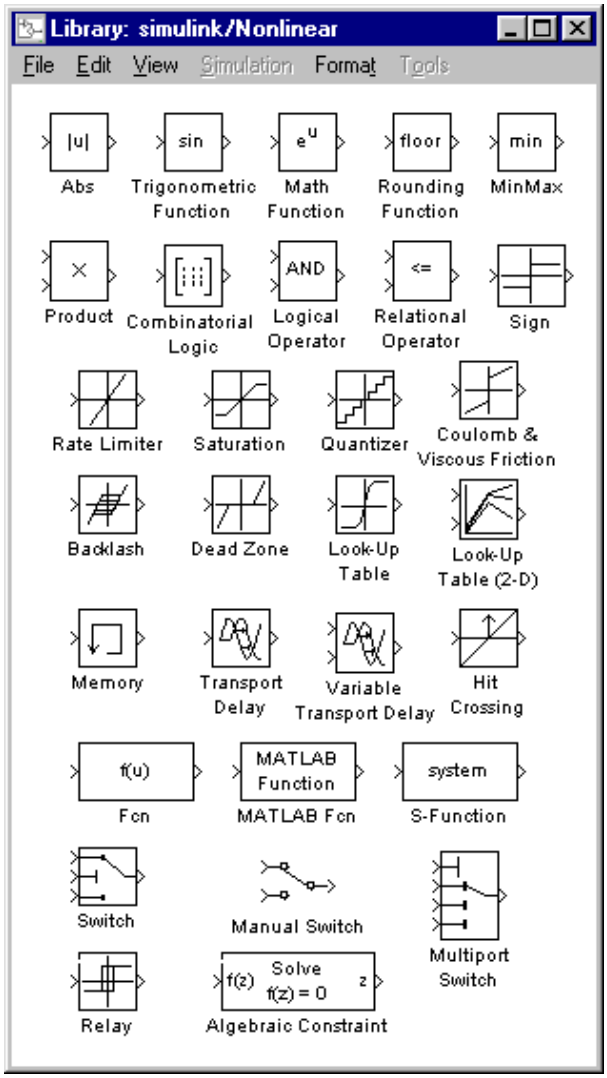
Le bloc "**transfert fcn**" est utilisé pour modéliser un système en transformées de Laplace sous la forme d'un numérateur et d'un dénominateur.

Le bloc "**state space**" modélise un système sous forme de variables d'état. Nous introduisons dans le masque du bloc, les matrices A, B, C et D et les conditions initiales du vecteur d'état s'il y' en a.

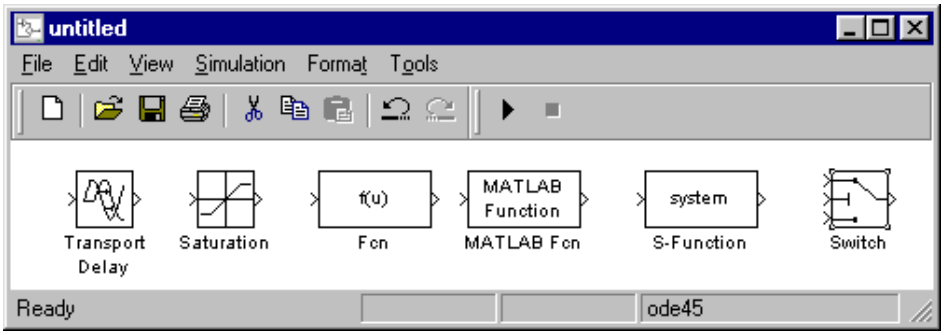
Le bloc "**integrator**" donne l'intégrale de l'entrée dont la valeur de la constante d'intégration doit être fixée par l'intermédiaire du masque.

1211.2.5 . Famille Non Linear

Cette famille contient des blocs non linéaires.



Les blocs les plus utilisés sont :



Le bloc "**transport delay**" permet de modéliser un retard pur continu.

Le bloc "**saturation**" modélise une non linéarité de type limiteur. Nous introduisons les valeurs maximales et minimales de l'entrée.

Le bloc "**fcn**" permet de générer une fonction dépendant de l'entrée.

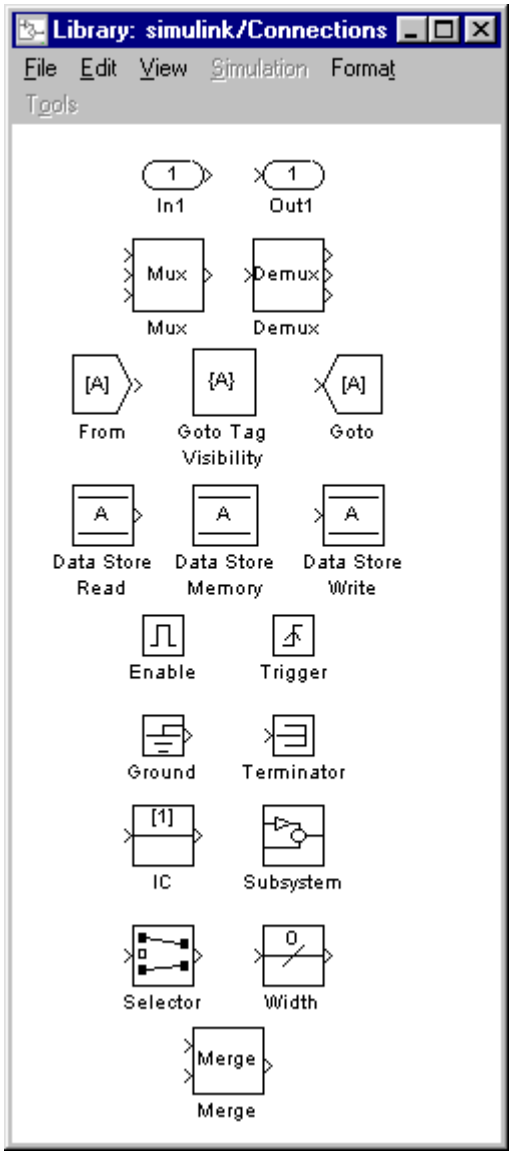
Le bloc "**matlab fcn**" appelle une fonction ***MATLAB***.

Le bloc "**S-function**" génère un bloc fonction à partir d'un algorithme développé sous ***MATLAB*** ou un autre langage évolué.

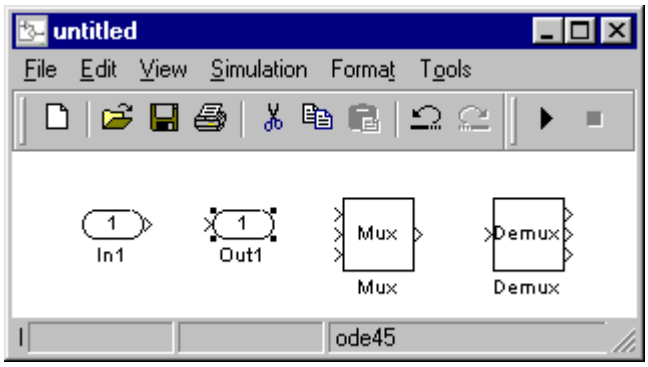
Le bloc "**Switch**" permet de faire un choix entre la première et la troisième entrée en fonction de la valeur de la seconde. Si cette dernière est inférieure à zéro, la troisième entrée est reconduite en sortie, sinon la première est reconduite.

1311.2.6 . Famille Connections

Cette famille contient des divers blocs de connexions de variables.



Les blocs suivant sont très utilisés.



Le bloc "**Mux**" transforme un ensemble d'entrées scalaires en un vecteur. Chaque entrée de ce bloc représente une composante de ce vecteur. Le nombre de composantes se règle par l'intermédiaire du masque de saisie.

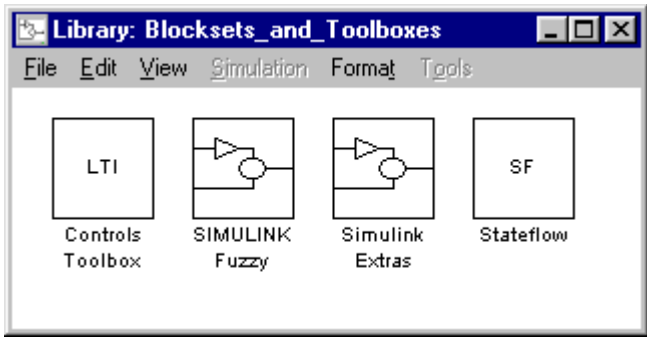
Le bloc "**Demux**" transforme un vecteur d'entrée en composantes scalaires. Le nombre de composantes se règle aussi par l'intermédiaire du masque de saisie.

Le bloc "**Inport**" matérialise une entrée externe.

Le bloc "**Outport**" matérialise une sortie externe.

1411.2.7 . Famille Blocksets and Toolboxes

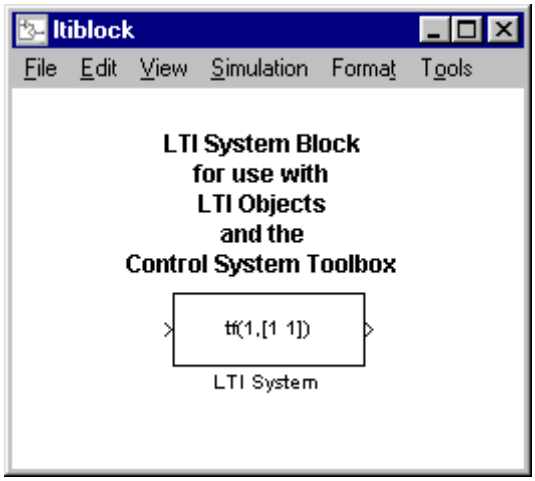
Cette famille est composée de plusieurs blocs additionnels utilisés en Automatique.



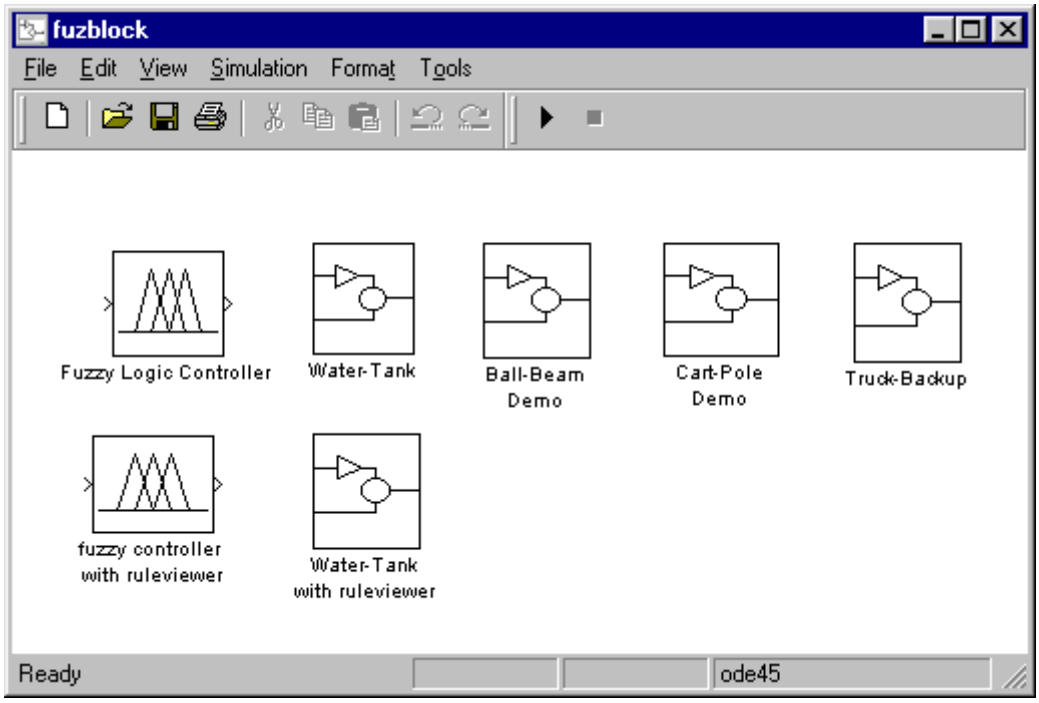
Nous remarquons que cette famille est découpée en sous-famille ("**Controls Toolbox**", "**Simulink Fuzzy**", "**Simulink Extras**" et "**Stateflow**").

11.2.7.1 . Sous-famille Controls toolbox

Cette famille est composé d'un bloc



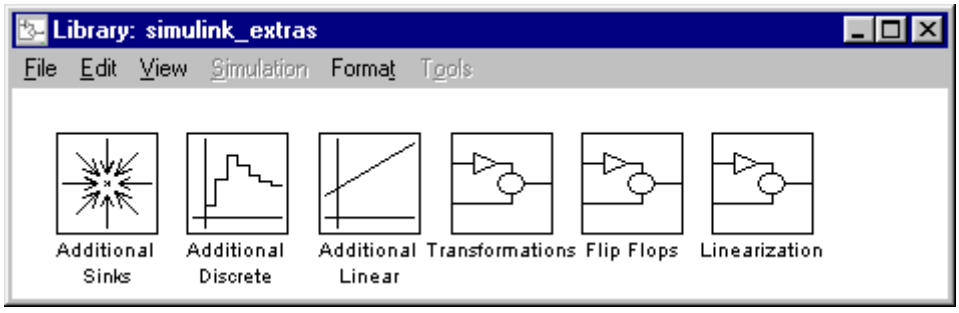
11.2.7.2 . Sous-famille "Simulink FUZZY"



Généralement ces blocs sont utilisés pour réaliser de la commande ou de la logique floue.

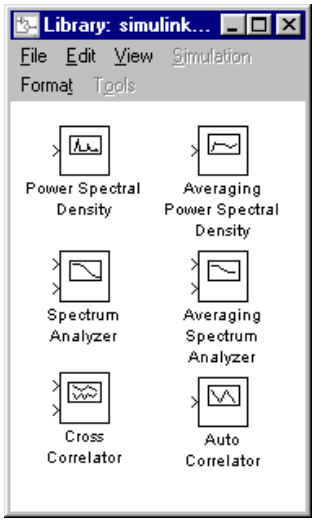
11.2.7.3 . Sous-famille "Simulink EXTRAS"

Cette famille est composée de plusieurs blocs.

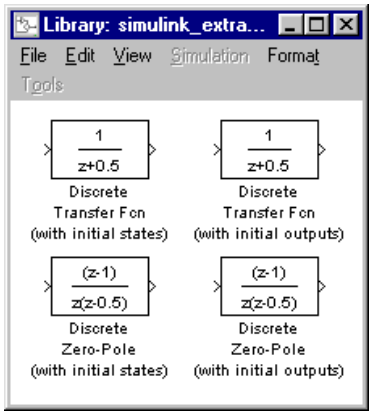


Cette famille est décomposée en sous-familles ("**Additional Sinks**", "**Additional Discrete**", "**Additional Linear**", "**Transformations**", "**Flip Flops**" et "**Linerization**")

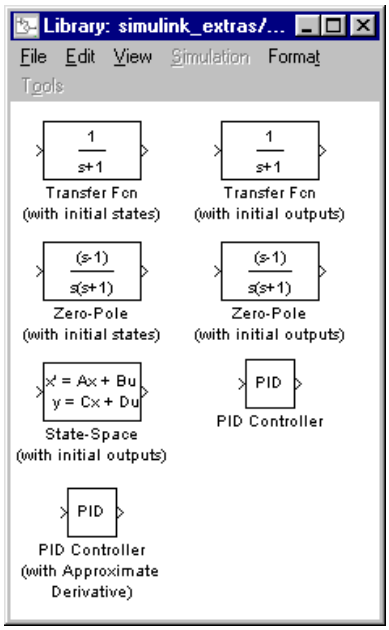
1 / Additional Sinks



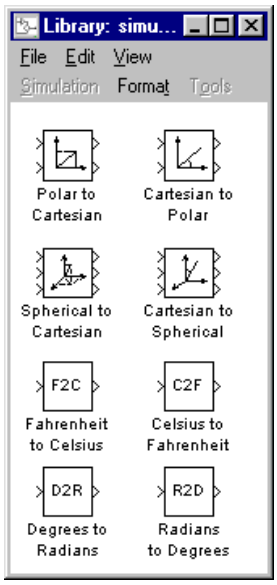
2 / Additional Discrete



3 / Additional Linear

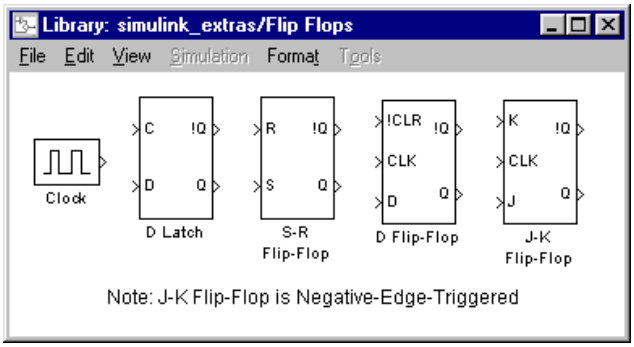


4 / Transformations

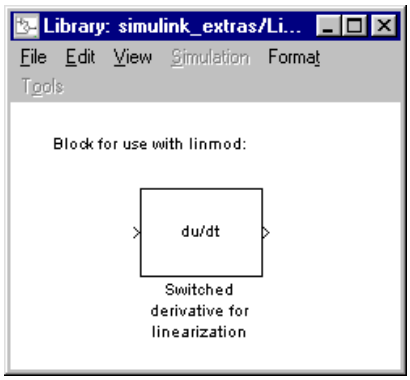


5 / Flip Flops

La sous famille "**Flip Flops**" contient des blocs de logique séquentielle tels que les bascules synchrones RS, JK et D.

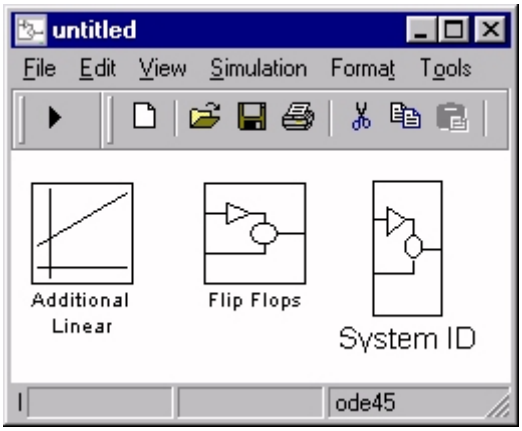


6 / Linerization

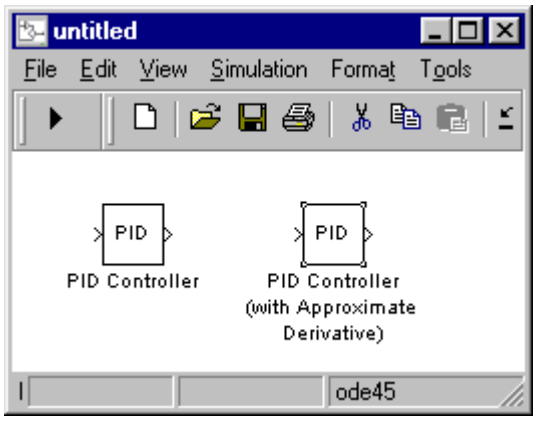


Les blocs qui nous intéressent sont essentiellement :

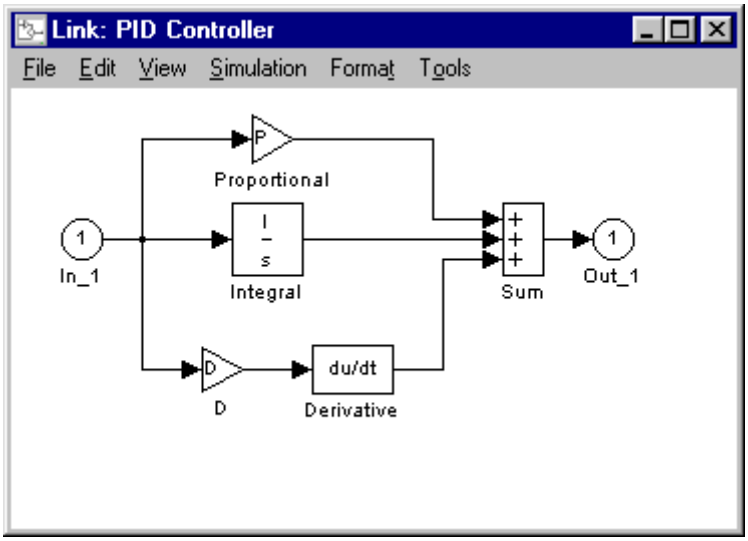
Régulateurs PID (Additional linear),
Régulateurs (Additional linear),
Identification,
Logique séquentielle (Flip Flops).



La sous famille "**PID Controllers**" contient deux types de PID :

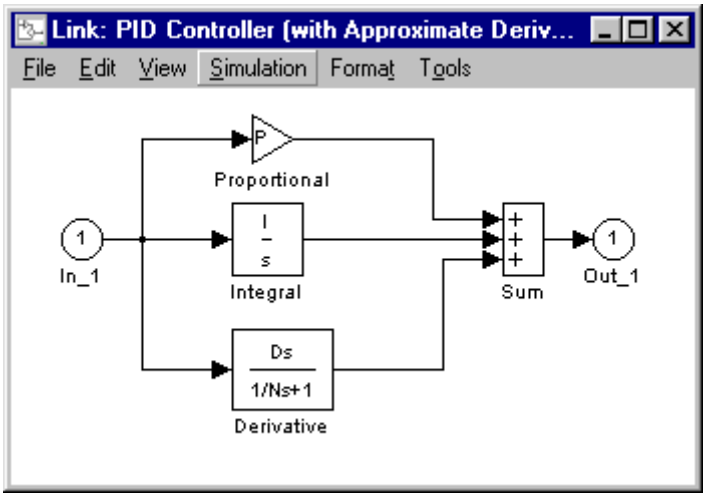


Le bloc "**PID Controller**" est un bloc PID parallèle. Pour connaître la structure du bloc "**PID**", sélectionnez le bloc, choisissez le menu "**edit**" et faites glisser la souris sur "**Look Under Mask**". Le schéma interne du bloc PID apparaît :



La commande d'un tel PID s'écrit : $\mathrm{Out\_1}(s) = (P + \frac{I}{s} + Ds).\mathrm{In\_1}(s)$

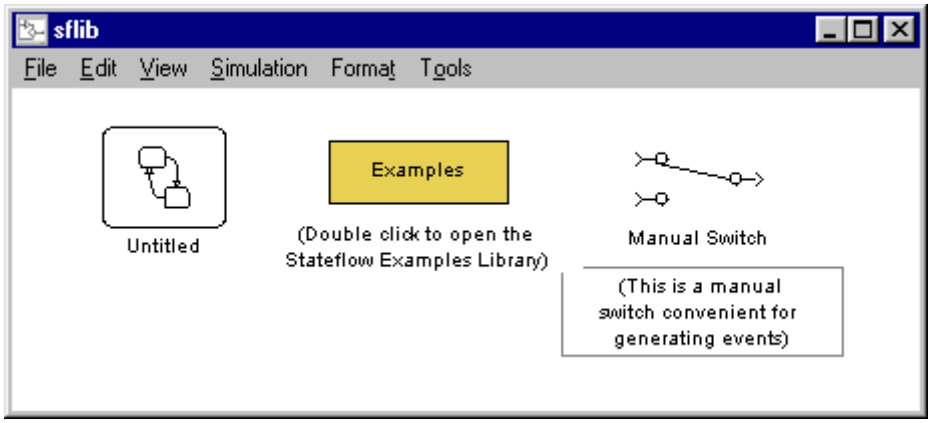
Le bloc "**PID with Approximate Derivative**" est un bloc PID avec une fonction dérivée en Laplace approximer. Il y a donc 4 paramètres à régler (P,I,D,N).



La commande PID à dérivée filtrée s'écrit : $\mathrm{Out\_1}(s) = \left(P + \frac{I}{s} + \frac{Ds}{1 + \frac{1}{N}s} \right).\mathrm{In\_1}(s)$.

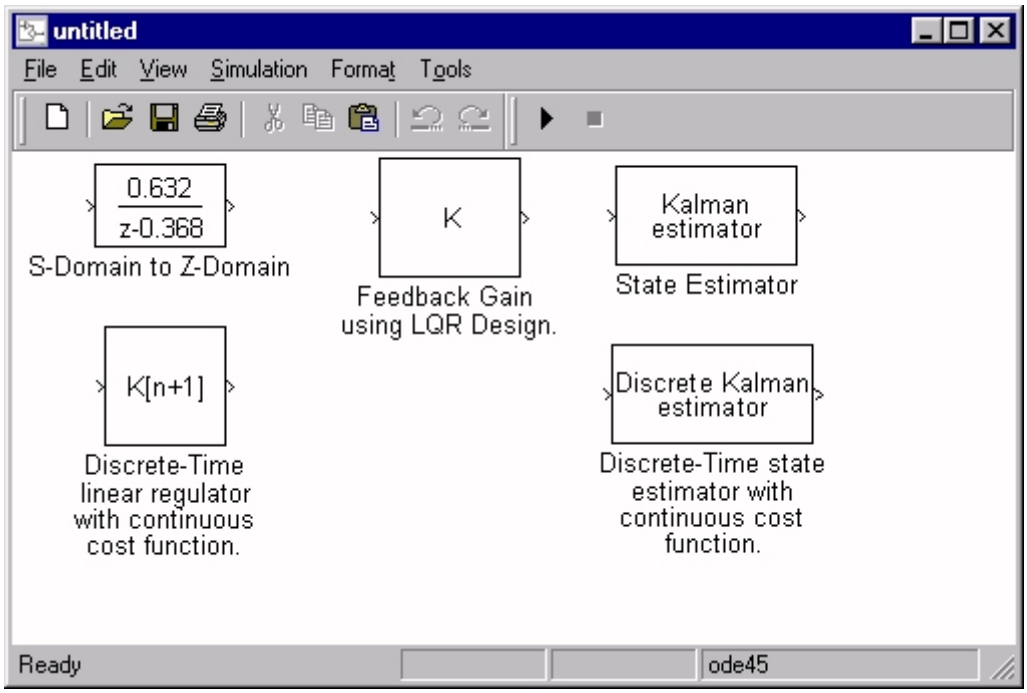
11.2.7.4 . Sous-famille "Stateflow "

Cette famille est composée de plusieurs blocs.



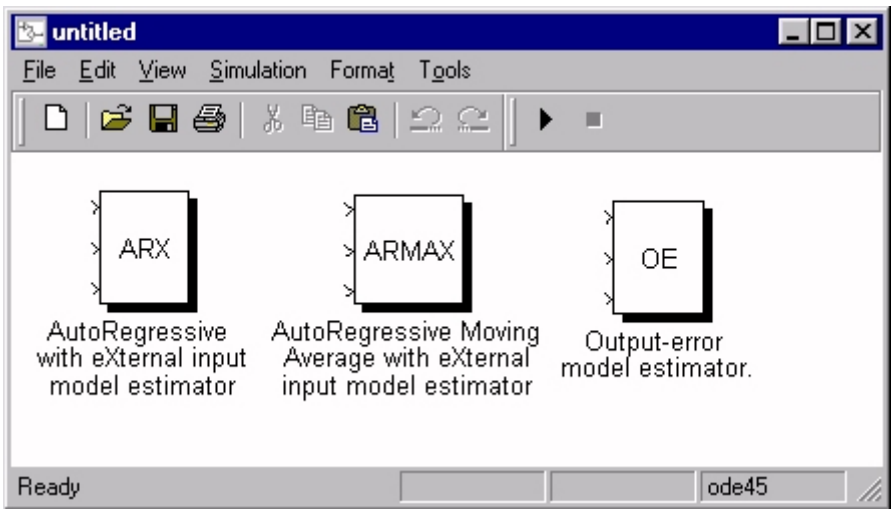
11.2.7.5 . Sous-famille "Controllers"

La sous famille "**Controllers**" contient un filtre de Kalman discret, un régulateur optimal quadratique discret, un filtre de Kalman continu, un régulateur optimal quadratique continu, une conversion continu-discret avec un bloqueur d'ordre zéro.



11.2.7.6 . Sous-famille "System ID"

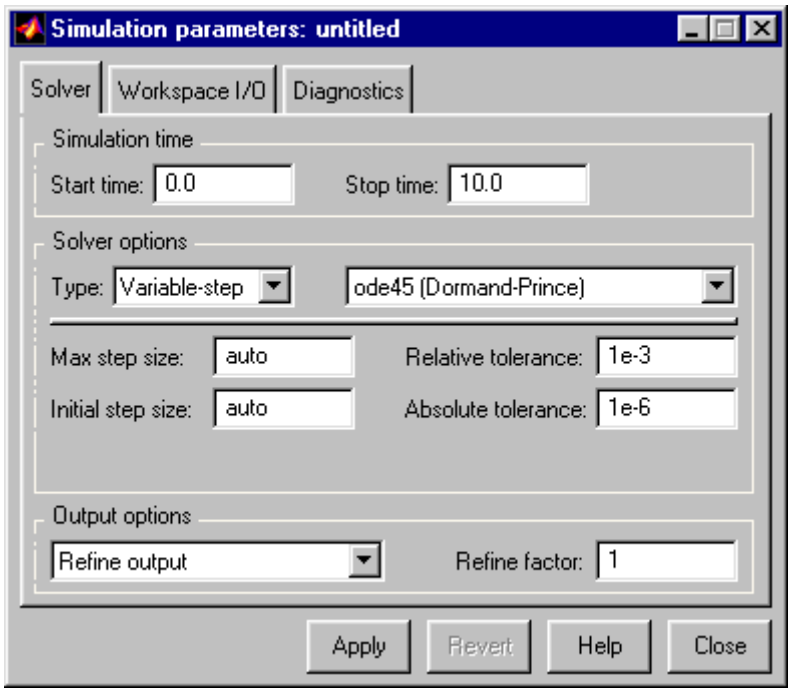
La sous famille "**System ID**" contient des blocs d'identification de systèmes monovariables. Nous utiliserons essentiellement le bloc "**ARX**" basé sur la méthode des moindres carrés ordinaires, le bloc "**ARMAX**" basé sur la méthode des moindres carrés étendues et le bloc "**OE**", basé sur la méthode dite d'erreur de sortie.



3711.3 . PARAMETRES DE SIMULATION

Avant de mettre en marche une simulation avec **SIMULINK**, il faut établir l'environnement de simulation. Celui-ci se décompose en sept parties :

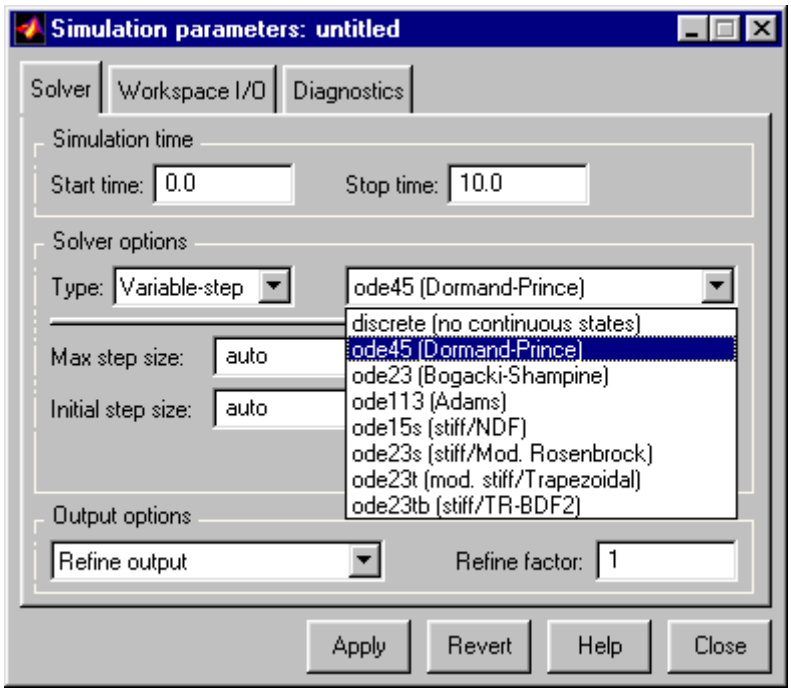
- Mode de simulation (Integration Algorithm),
- Instant de départ (Start Time),
- Instant final (Stop Time),
- Pas minimum de simulation (Min Step Size),
- Pas maximum de simulation (Max Step Size),
- La tolérance.
- Les variables retournées.



1511.3.1 . Mode de simulation

Simuler un système sous **SIMULINK,** revient à résoudre une équation différentielle théorique. Or un ordinateur ne fonctionne pas en continu mais de façon échantillonné. En effet, toutes les opérations effectuées par ce dernier sont cadencées par une horloge interne à une certaine fréquence.

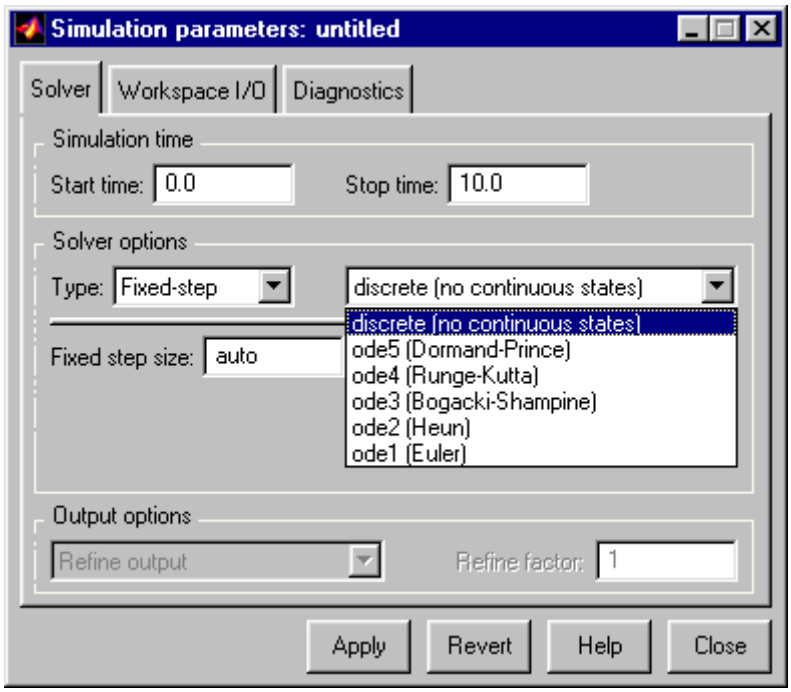
Pour effectuer la résolution d'une équation différentielle en mode échantillonné, deux modes sont proposés soit en pas échantillon variable("**Variable-Step**") où en pas d'échantillonnage fixe ("**Fixed-Step**"). Dans le premier mode huit méthodes de résolutions sont proposées :



La résolution "**ode45(Domand-Prince)**" est utilisée pour les systèmes linéaires composés de blocs tels que fonction de transfert, espace d'état, sommateur, gain, etc.... Cette résolution reste valable si le système contient un peu de blocs non linéaires et elle est adaptée aussi aux systèmes ayant des parties dynamiques rapides et lentes. ("dynamique raide").

La résolution "**adams**" est utilisée pour des systèmes lisses non linéaires et quasi-stationnaires. (paramètres ne variant pas beaucoup dans le temps).

Dans le deuxième mode cinq méthodes de résolutions sont proposées :



La résolution "**ode4(Runge-Kutta)**" est basée sur Runge-Kutta du 4ème ordre. Cette résolution est valable pour des systèmes non linéaires et/ou discontinus. Cette résolution n'est pas valable pour des systèmes ayant une dynamique rapide mais reste valable pour des systèmes ayant des parties continues et discrètes.

La résolution "**euler**" est utilisée pour vérifier les résultats seulement.

1611.3.2 . Instant de départ et d'arrivée

La simulation débute à partir de la valeur donnée dans "**Start Time**" et s'achève à celle du "**Stop Time**".

1711.3.3 . Pas initial et maximum de simulation

Les limites de la valeur du pas d'échantillonnage de la simulation sont données par le "**Initial Step Size**" et le "**Maximum Step Size**".

Le "**Initial Step Size**" est le pas d'intégration utilisé en début de simulation. Généralement, sa valeur est mise en mode automatique. Le "**Maximum Step Size**" doit être choisi prudemment il faut généralement le laisser en mode automatique.

1811.3.4 . Tolérance relative et la Tolérance Absolue

La tolérance relative est l'erreur relative de l'intégration à chaque pas de calcul. Cette erreur est fixée à 1e-3. La tolérance absolue est l'erreur absolue de l'intégration à chaque pas de calcul. Cette erreur est fixée à 1e-6.

3811.4 . FONCTIONNEMENT INTERNE DE SIMULINK

Le mode de représentation des systèmes dynamiques sous **SIMULINK** a été développé pour faciliter la création et l'exploitation de ceux-ci.

Ainsi, pour créer un diagramme, nous plaçons des blocs et nous les relions entre eux pour former des schémas blocs.

Sur **SIMULINK**, la simulation fonctionne de la façon suivante :

- Construction des structures de données

* Évaluation des paramètres de chaque bloc par ***MATLAB***.
* Les blocs sont classés dans l'ordre dans lequel, ils doivent être mis à jour.
* Les connexions entre les blocs sont vérifiées.

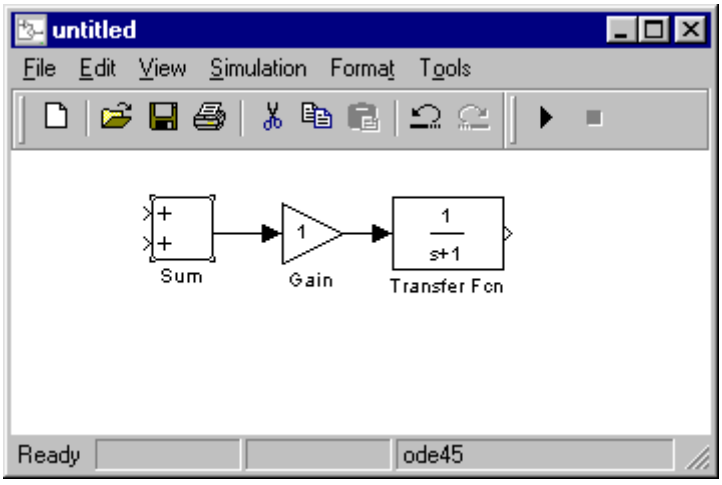
- Intégration numérique des modèles

* Calcul des sorties de chaque bloc lors de la phase d'initialisation
* Calcul du vecteur dérivé, basé sur l'instant présent, vecteur d'état et du vecteur d'entrée.
* Le vecteur dérivé est renvoyé à l'algorithme d'intégration pour calculer le nouveau vecteur d'état
* Le nouveau vecteur d'état est immédiatement calculé, les données échantillonnées et graphiques sont mise à jour.

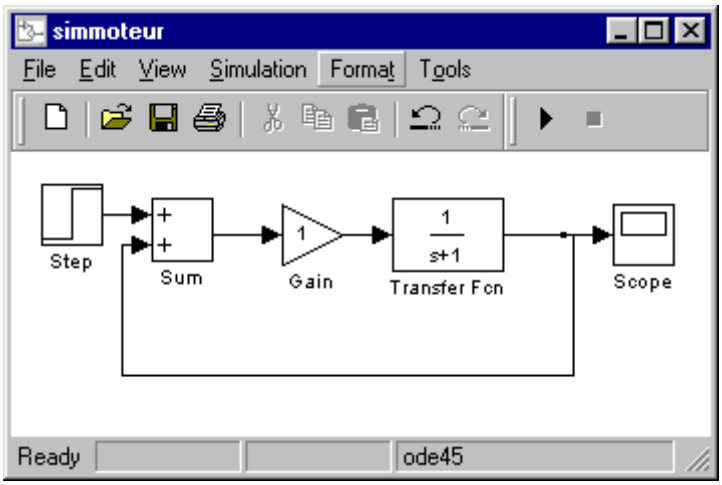
3911.5 . SIMULATION D'UN SYSTEME

Pour expliquer le fonctionnement de la simulation sur **SIMULINK**, prenons l'exemple de régulation de vitesse d'un moteur à courant continu dont la fonction de transfert est : $H(s) = \frac{0.6}{0.05s + 1}$.

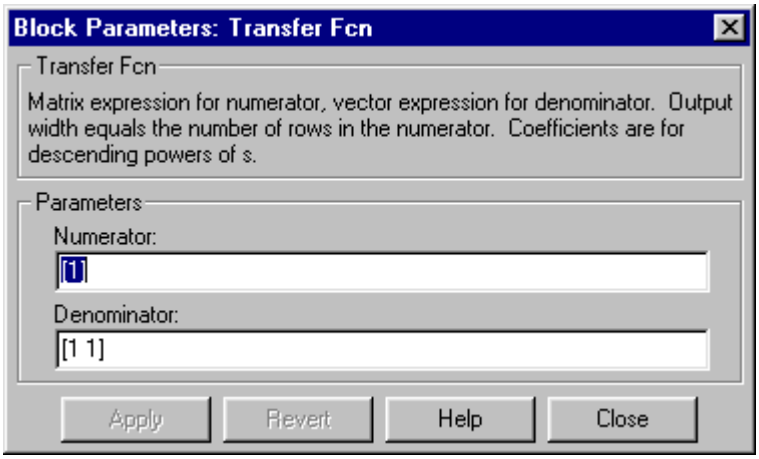
Pour cela, sur la fenêtre "**simulink**", choisissez "**file**" et cliquez sur "**new model**". Une nouvelle fenêtre apparaît intitulée "**untitled**". Pour construire notre système, il faut revenir à la fenêtre de base "**simulink**" et double cliquez sur le bloc "**Linear**". A l'aide de la souris, tirez le bloc "**Transfer Fcn**" et transportez-le dans la fenêtre "**untiled**". Refaite la même opération pour les blocs "**sum**" et "**gain**". Ensuite reliez les différents blocs entre eux de préférence à l'aide du bouton droit ou gauche de la souris. Nous obtenons la figure suivante :



Fermez la fenêtre "**Linear**" et double cliquez sur le bloc "**Sources**" situé dans la fenêtre "**Simulink**". Cliquez sur "**Step**" et transportez le dans la fenêtre "**untitled**" et fermez ensuite la fenêtre "**Sources**". Double cliquez sur le bloc "**Sinks**", cliquez sur "**Scope**" et transportez le dans la fenêtre "**untitled**" et fermez ensuite la fenêtre "**Sinks**". Reliez les différents blocs entre eux. Sauvez le fichier en ouvrant "**File**" de la fenêtre "**untitled**" et cliquez sur "**Save As**". Une boite de dialogue apparaît et sauvez le fichier par exemple sous **simmoteur.mdl**. Nous obtenons la fenêtre suivante :



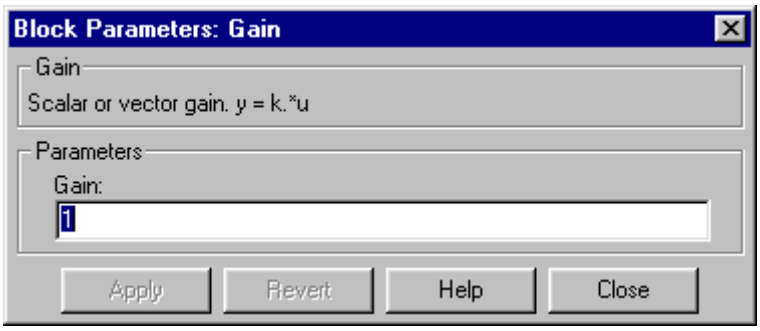
Il nous reste à configurer les différents blocs. Ainsi, double cliquez sur le centre du bloc "**Transfert Fcn**" dans simmoteur. La fenêtre suivante apparaît :



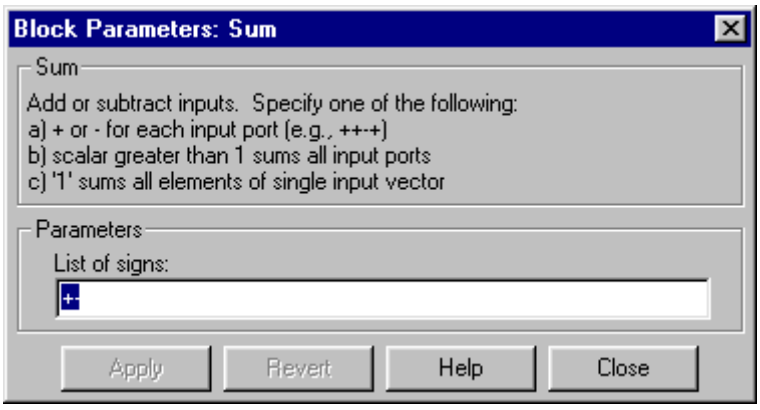
Cliquez sur le rectangle au dessous de "**Numerator**" et taper 0.6 au numérateur. Cliquez ensuite dans le rectangle "**Denominator**" et tapez [0.05 1] au dénominateur et validez ensuite par "**OK**".

Attention! Pour modifier les paramètres, il faut que le pointeur de la souris soit toujours à l'intérieur du rectangle.

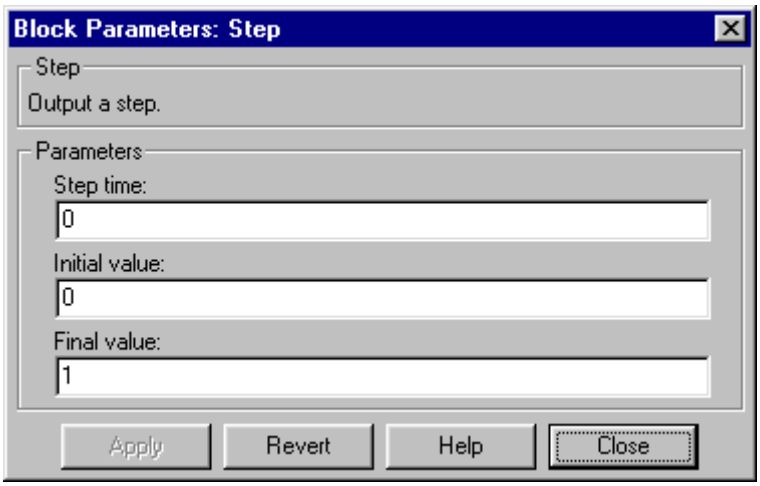
Sélectionner le gain en double cliquant sur celui-ci et mettre la valeur à 1.



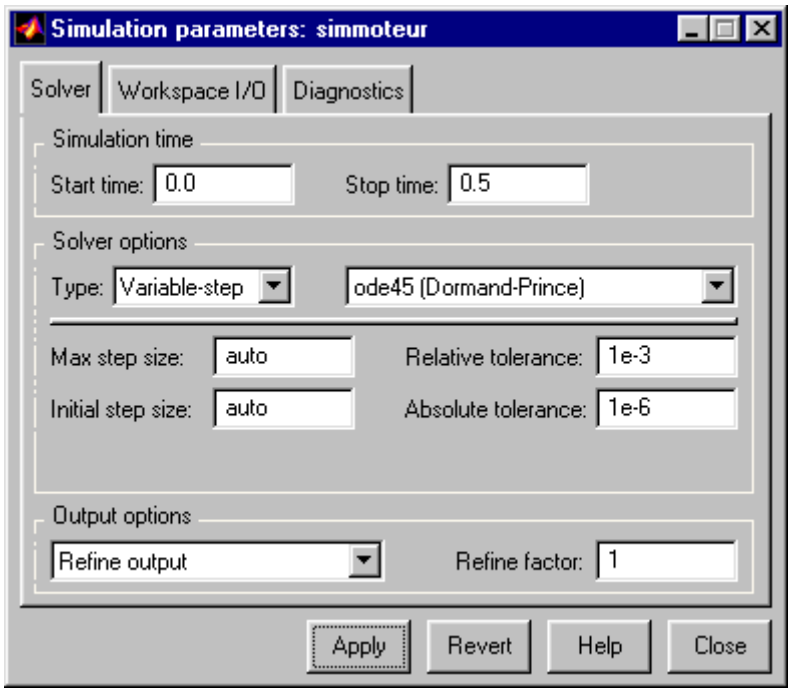
Double cliquez sur le bloc "**sum**" et mettre les signes **+ -**.



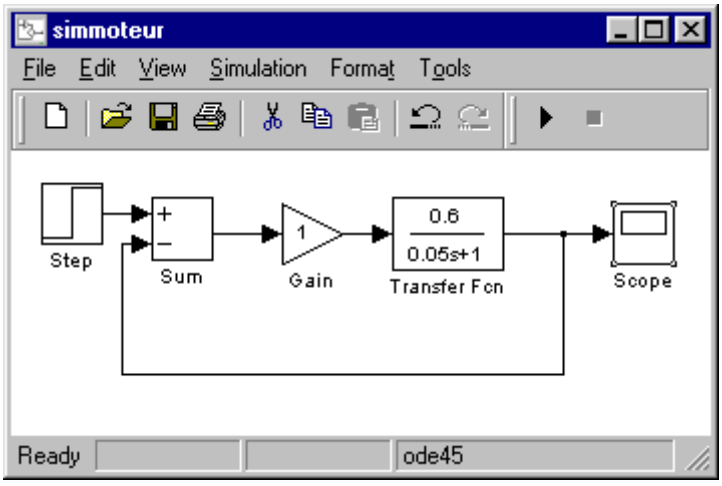
Double cliquez sur le bloc "**Step**" et mettre l'instant de départ à 0, la valeur initiale à 0 et la valeur finale à 1.



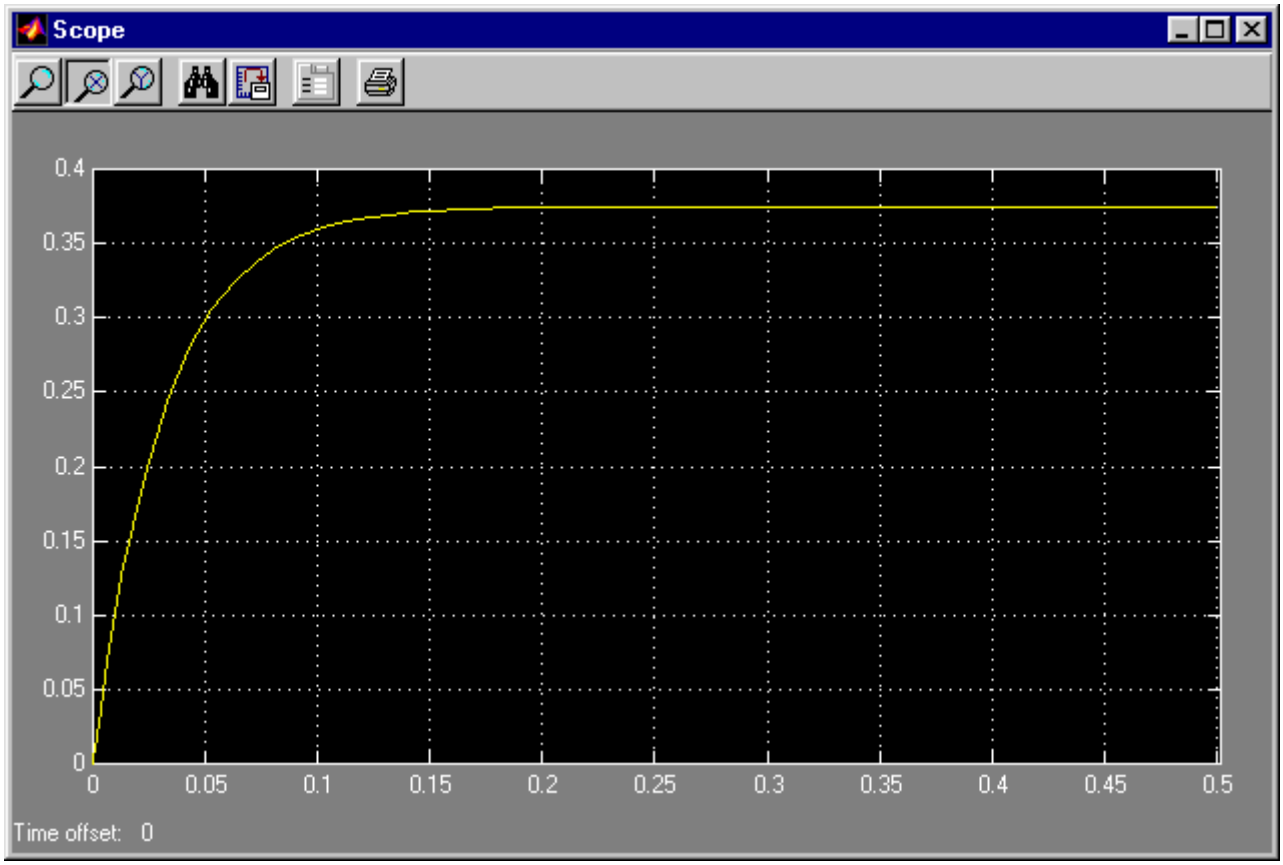
Sélectionnez "**Simulation**" et mettre la souris sur "**Parameters**" et changez les paramètres suivant : start time à 0, stop time à 0.5 et laissez les autres paramètres par défaut.



Nous obtenons la figure suivante :



Nous démarrons ensuite la simulation en appuyant sur "**Start**" :  . Nous obtenons alors, la réponse indicielle suivante :



4011.6 . CREATION D'UNE S-FUNCTION

Nous allons créer une S-function sur ***MATLAB*** qui permet de modéliser un système continu sous la forme de variables d'états :

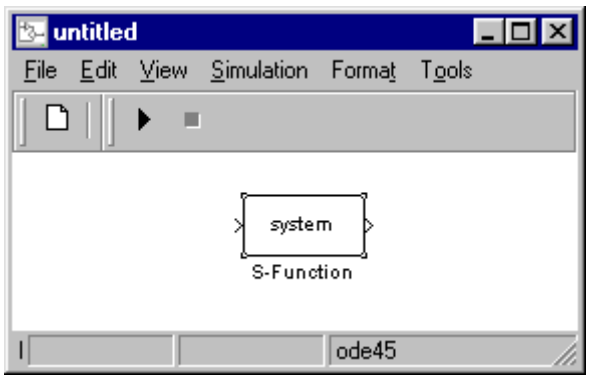
$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Appelons la fonction **sfonct1.m**. Le programme s'écrit :

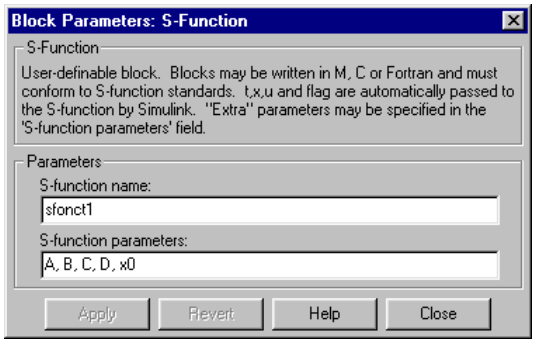
```
function [sys, x0] = sfonct1(t,x,u,flag,A,B,C,D,x0)

% t : temps, x: vecteur d'état, u : vecteur d'entrées
% flag : indique le type de sorties a envoyé dans sys

% Paramètres
% A : matrice d'état
% B : matrice d'entrées
% C : matrice de sorties
% D : matrice de transmission

% x0 : vecteur d'état initial
% flag   sys description
% ---------------------------------
%    1    dx          vecteur dérivé dx/dt.
%    2    de          états discrets, x(k+1).
%    3    y           sorties du système.
```

```
%    4    tsuivant   prochain intervalle de temps (système discret).
%    5    r          valeurs des racines

[n,m]=size(B);
[p,n]=size(C);
if nargin==0,
        sys=[n,0,p,m,0,1];
        return,
end
if abs(flag) == 1
        sys = A*x + B*u;
elseif flag == 3
        sys = C*x+D*u;
elseif abs(flag) == 0
        sys=[n,0,p,m,0,1] ;
else sys = [];
end
```

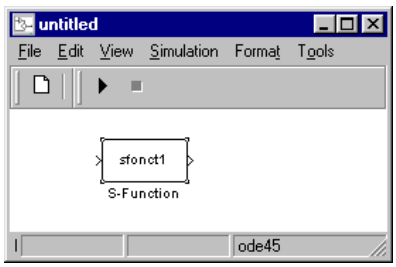
Dans le bloc "**NonLinear**", nous prenons un bloc "**S-Function**".



Nous introduisons le nom de la S-function et les paramètres A, B, C, D et x0 :

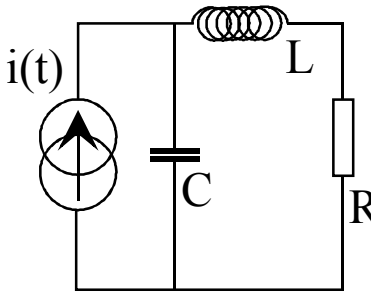


Sur **SIMULINK**, nous visualisons le bloc comme suit :



Application

Soit un circuit RLC matérialisé par la figure suivante :



En choisissant comme variables d'état, la tension aux bornes du condensateur (x_1), le courant qui traverse la self (x_2), et comme variable d'entrée la source de courant i(t), le système s'écrit :

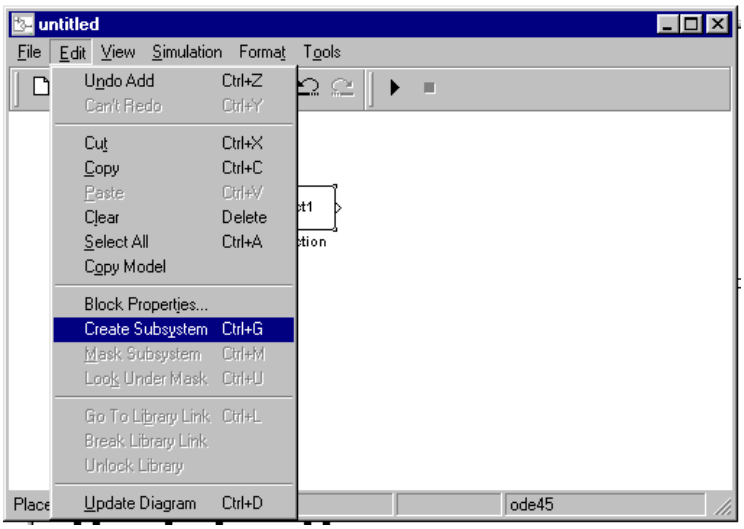
$$\dot{x}_1(t) = -2x_2(t) + 2i(t)$$
$$\dot{x}_2(t) = x_1(t) - 3x_2(t)$$

Nous identifions les matrices d'état, d'entrées, de sortie et de transmission :

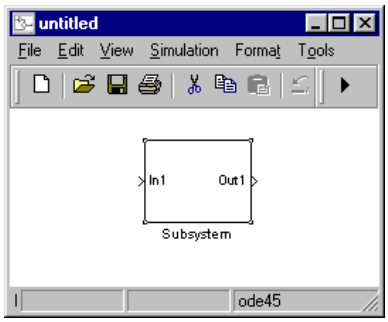
$$A = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix},\ B = \begin{bmatrix} 2 \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ et } D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Les conditions initiales sont : $x_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$.

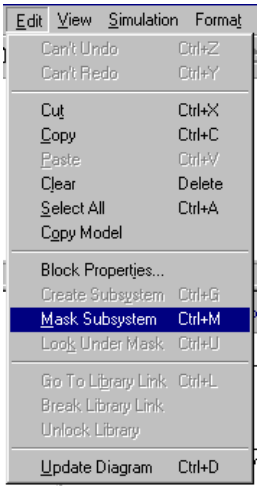
Pour définir les paramètres A, B, C, D et x0, il faut définir "**subsystem**". Pour cela nous sélectionnons le bloc S-Function et nous allons dans le menu "edit/Create subsystem" pour créer le sous système:



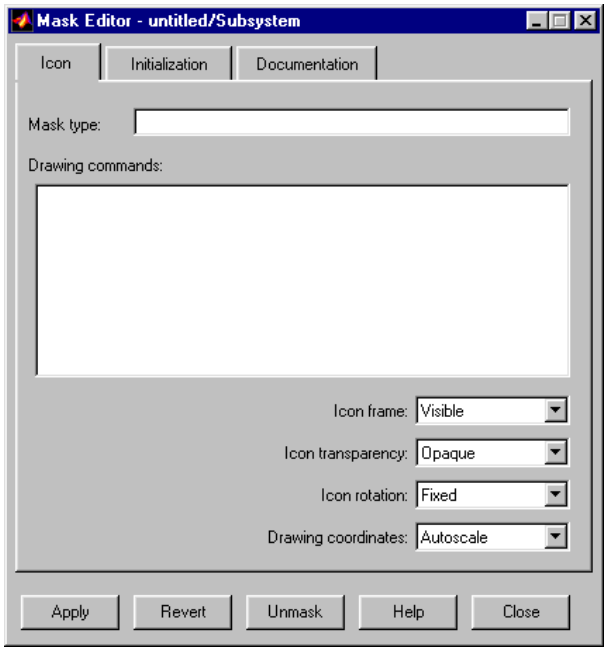
Dans ce cas nous obtenons la fenêtre suivante :



Il nous reste à entrer les paramètres. Pour cela nous sélectionnons le bloc **Subsystem** et nous allons dans le menu "edit/Mask subsystem" pour définir les paramètres du sous système:



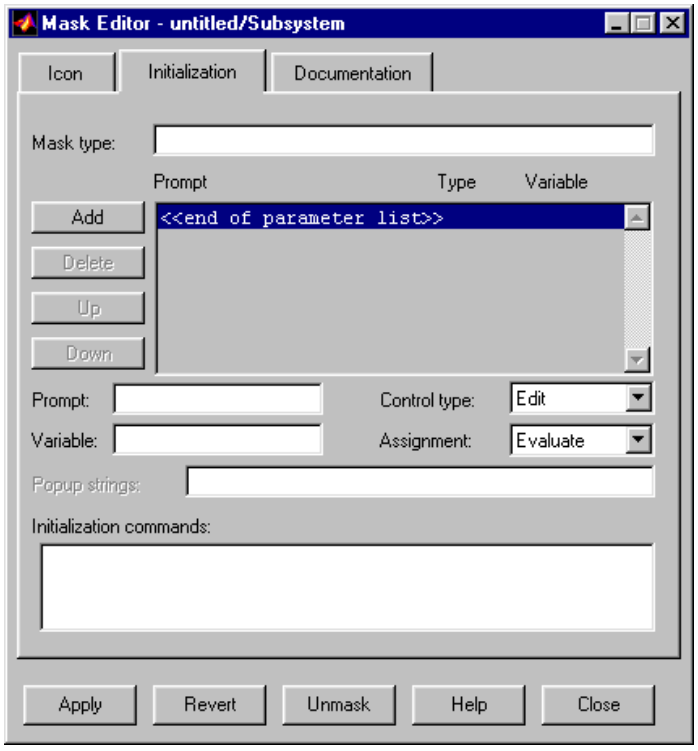
Nous obtenons la fenêtre suivante :



Cette fenêtre permet de définir l'icône, l'initialisation, les paramètres demandés lorsque ; nous cliquons sur le bloc "**subsystem**" et la documentation associée à cette fenêtre.

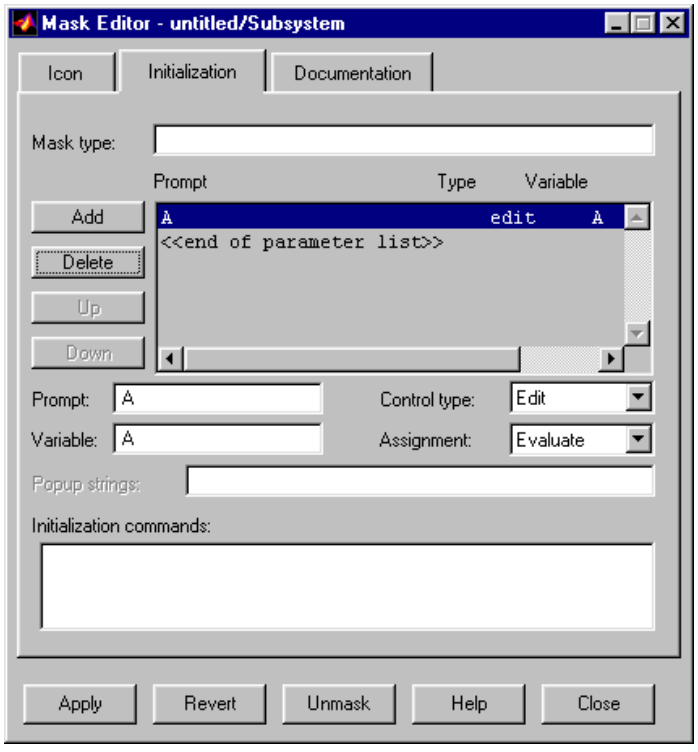
Nous allons nous intéresser aux paramètres demandés lorsque nous cliquons sur le bloc "subsystem".

Pour rentrer ces paramètres il faut cliquer sur "**Initialization**" :



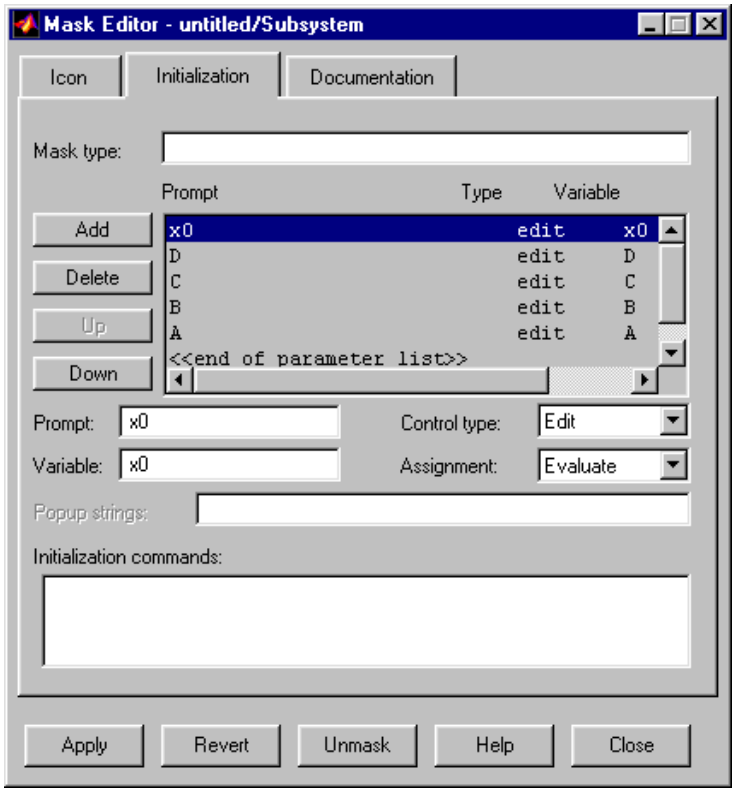
Nous allons définir la variable A et ensuite les variables B, C, D, et x0.

Dans ce cas nous cliquons sur le bouton "**ADD**" et ensuite nous entrons les paramètres **Prompt** et **variable** et nous rentrons les paramètres suivants :

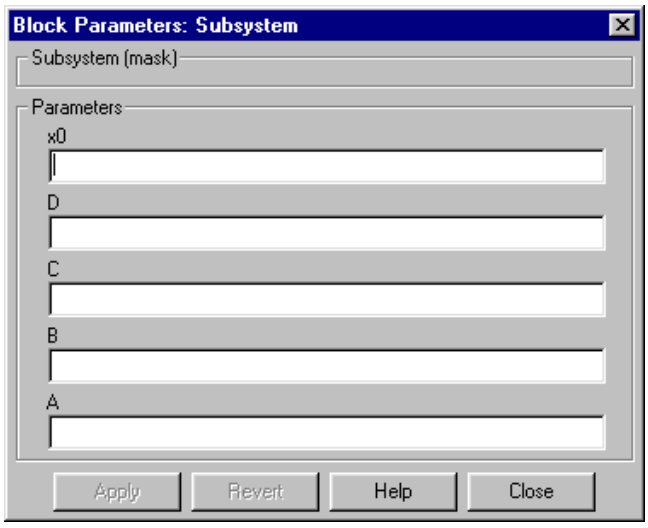


Et ensuite nous appuyons sur "**Apply**".

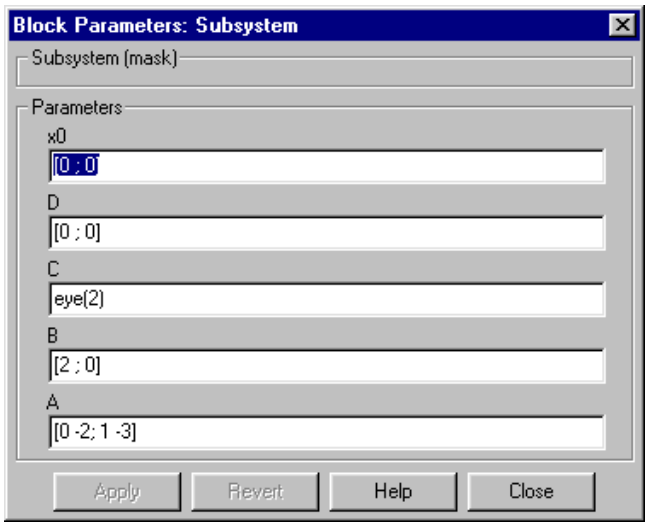
Nous effectuons la même chose pour B, C, D et x0. Nous obtenons dans ce cas la figure suivante :



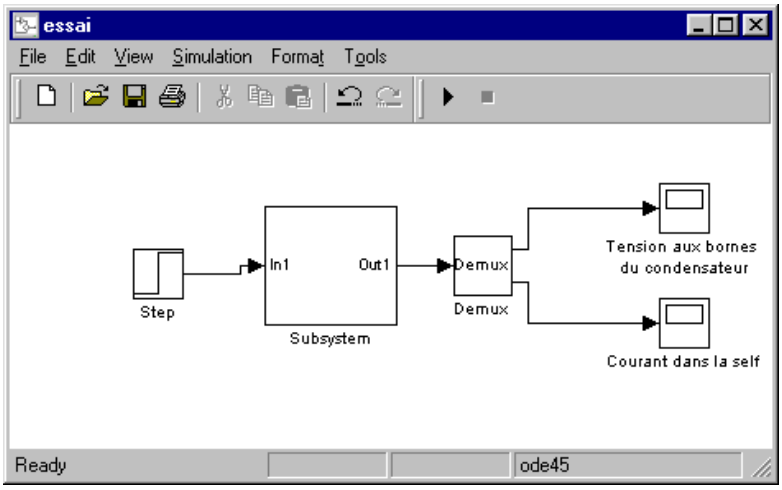
Maintenant si nous cliquons sur le bloc "**subsystyem**" nous obtenons la figure suivante :



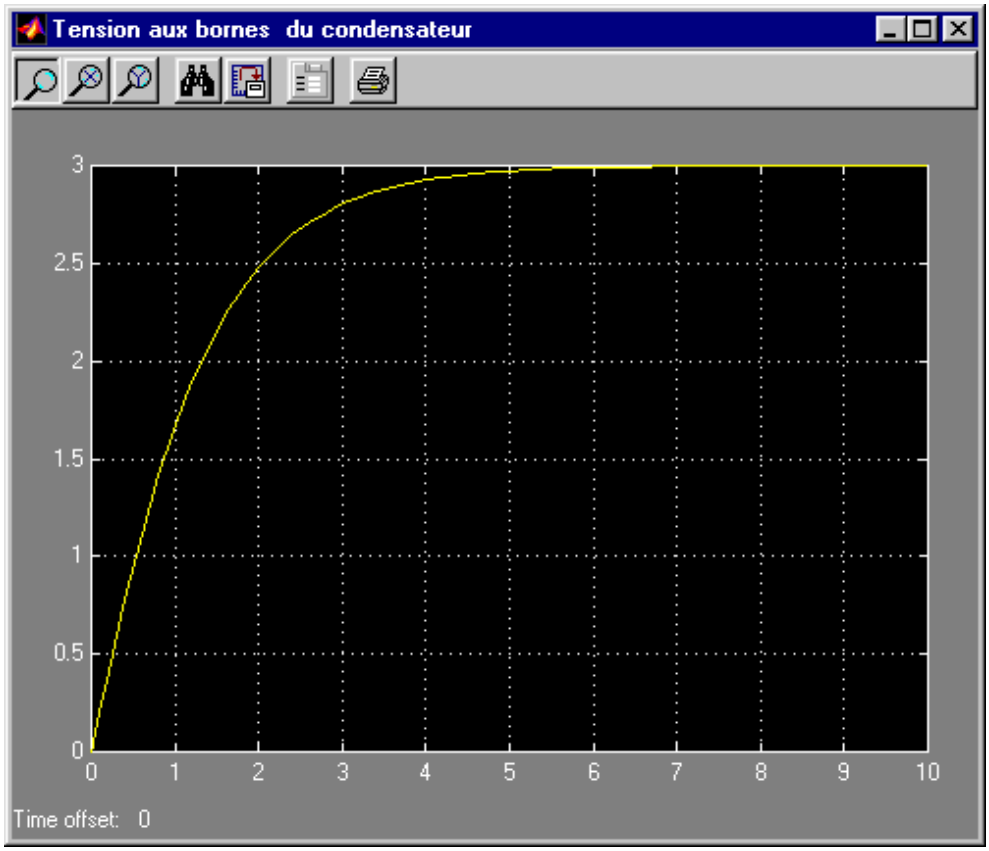
Dans cette figure nous allons entrer les paramètres A, B, C, D et x0 :



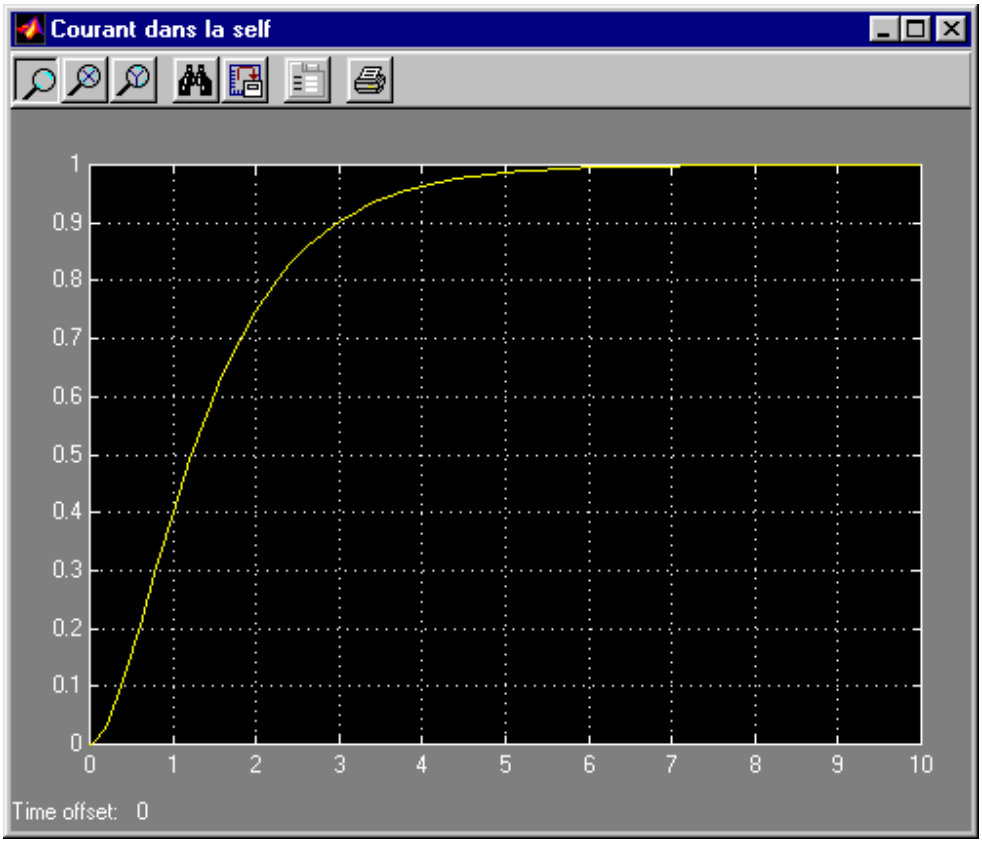
Par exemple, nous simulons la réponse indicielle de ce circuit. Sur **SIMULINK**, nous connectons la S-function aux blocs de visualisation et nous obtenons :



Après avoir réglé les paramètres de simulation, nous la démarrons et nous obtenons les courbes suivantes :



Tension aux bornes du condensateur



Courant dans la self

1112 . ANNEXES

QUICK REFERENCE TABLES

MATLAB provides 20 main categories of functions. Some of ***MATLAB***'s functions are built into the interpreter, while others take the form of M-files. The M-file functions, and in the case of the built-in functions, M-files containing only help text, are organized into 20 directories, each containing the files associated with a category. The ***MATLAB*** command help displays an online table of these main categories.

***MATLAB*'s Main categories of functions**	
color	Color control and lighting model functions
datafun	Data analysis and Fourier transform functions
demos	Demonstrations and samples
elfun	Elementary math functions
elmat	Elementary matrices and matrix manipulation
funfun	Function functions - nonlinear numerical methods
general	General purpose commands
graphics	General purpose graphics functions
iofun	Low-level file 1,0 functions
lang	Language constructs and debugging
matfun	Matrix functions - numerical linear algebra
ops	Operators and special characters
plotxy	Two dimensional graphics
plotxyz	Three dimensional graphics
polyfun	Polynomial and interpolation functions
sparfun	Sparse matrix functions
specfun	Specialized math functions
specmat	Specialized matrices
sounds	Sound processing functions
strfun	Character string functions

The following pages contain tables of functions within each of these specific areas. If you execute help on one of the directory names listed on the left side of thîs table, MATLAB displays an online version of the tables within that area.

4112.1 . GENERAL PURPOSE COMMANDS

Managing Commands and Functions	
demo	Run demos
help	Online documentation
info	Information about MATLAB and The MathWorks
lookfor	Keyword search through the help entries
path	Control MATLAB's search path.
type	List M-file
what	Directory listing of M-, MAT- and MEX-files
which	Locate functions and files

Managing Variables and the Workspace	
clear	Clear variables and functions from memory
disp	Display matrix or text
length	Length of vector
load	Retrieve variables from disk
pack	Consolidate workspace memory
save	Save workspace variables to disk
size	Size of matrix
who	List current variables
whos	List current variables, long form.
Working with Files and the Operating System	
cd	Change current working directory
delete	Delete file
diary	Save text of MATLAB session
dir	Directory listing
getenv	Get environment value
unix	Execute operating system command; return result
!	Execute operating system command
Controlling the Command Window	
clc	Clear command window
echo	Echo commands inside script files
format	Set output format
home	Send cursor home
more	Control paged output in command window
Starting and Quitting from MATLAB	
matlabrc	Master startup M-file
quit	Terminate MATLAB
startup	M-file executed when MATLAB is invoked

4212.2 . OPERATORS AND SPECIAL CHARACTERS

Operators and Special Characters	
+	Plus
-	Minus
*	Matrix multiplication
.*	Array multiplication
^	Matrix power
.^	Array power
Kron	Kronecker tensor product
\	Backslash or left division
/	Slash or right division
./	Array division
:	Colon
( )	Parentheses
[ ]	Brackets
.	Decimal point
..	Parent directory
...	Continuation
,	Comma
;	Semicolon
%	Comment
!	Exclamation point
'	Transpose and quote

.'	Nonconjugated transpose
=	Assignment
==	Equality
<>	Relational operators
&	Logical AND
|	Logical OR
~	Logical NOT
xor	Logical EXCLUSIVE OR

Logical Functions	
all	True if all elements of vector are true
any	True if any element of vector is true
exist	Check if variables or functions exist
find	Find indices of non-zero elements
finite	True for finite elements
isempty	True for empty matrix
isieee	True for IEEE floating point arithmetic
isinf	True for infinite elements
isnan	True for Not-A-Number
issparse	True for sparse matrix
isstr	True for text string

4312.3 . Language Constructs and Debugging

MATLAB as a Programming Language	
eval	Execute string with MATLAB expression
feval	Execute function specified by string
function	Add new function
global	Define global variable
nargchk	Validate number of input arguments.
Control Flow	
break	Terminate execution of loop
else	Used with if
elseif	Used with if
end	Terminate the scope of for, while and if statements
error	Display message and abort function
for	Repeat statements a specific number of times
if	Conditionally execute statements
return	Return to invoking function
while	Repeat statements an indefinite number of times

Interactive Input	
input	Prompt for user input
keyboard	Invoke keyboard as if it were a script-file
menu	Generate menu of choices for user input
pause	Wait for user response

Debugging	
dbclear	Remove breakpoint
dbcont	Resume execution
dbdown	Change local workspace context
dbquit	Quit debug mode
dbstack	List who called whom
dbstatus	List ail breakpoints
dbstep	Execute one or more lines
dbstop	Set breakpoint
dbtype	List M-file with line numbers
dbup	Change local workspace context

4412.4 . ELEMENTARY MATRICES AND MATRIX MANIPULATION

Elementary Matrices	
eye	Identity matrix
linspace	Linearly spaced vector
logspace	Logarithmically spaced vector
meshgrid	X and Y arrays for 3-D plots
ones	Ones matrix
Rand	Uniformly distributed random numbers
Randn	Normally distributed random numbers
zeros	Zeros matrix
:	Regularly spaced vector
Special Variables and Constants	
ans	Most recent answer
computer	Computer type.
eps	Floating point relative accuracy.
flops	Count of floating point operation
i, j	Imaginary unit.
inf	Infinity
NaN	Not-a-Number
nargin	Number of function input arguments
nargout	Number of function output arguments
pi	3.1415926535897
realmax	Largest floating point number
realmin	Smallest floating point number
Time and Dates	
clock	Wall clock
cputime	Elapsed CPU time
date	Calendar
etime	Elapsed time function
tic, toc	Stopwatch timer functions

Matrix Manipulation	
diag	Create or extract diagonals
flipir	Flip matrix in the left/light direction
flipud	Flip matrix in the up/down direction
reshape	Change size
rot90	Rotate matrix 90 degrees
tril	Extract lower triangular part
trilu	Extract upper triangular part
:	Index into matrix, rearrange matrix

4512.5 . SPECIALIZED MATRICES

Specialized Matrices	
compan	Companion matrix
hadamard	Hadamard matrix
hankel	Hankel matrix
hilb	Hilbert matrix
invhilb	Inverse Hilbert matrix
magic	Magic square
pascal	Pascal matrix (see online help)
rosser	Classic symmetric eigenvalue test problem
toeplitz	Toeplitz matrix.
vander	Vandermonde matrix
wilkinson	Wilkinson's eigenvalue test matrix

4612.6 . ELEMENTARY FUNCTIONS

Elementary Math Functions	
abs	Absolute value
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
angle	Phase angle.
asin	Inverse sine
asinh	Inverse hyperbolic sine
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atanh	Inverse hyperbolic tangent
ceil	Round towards plus infinity
coni	Complex conjugate
cos	Cosine.
cosh	Hyperbolic cosine
exp	Exponential
fix	Round towards zero
floor	Round towards minus infinity
imag	Complex imaginary part.
log	Natural logarithm
10gb	Common logarithm
real	Complex real part
rem	Remainder after division
round	Round towards nearest integer
sign	Signum function
sin	Sine

sinh	Hyperbolic sine
sqrt	Square root
tan	Tangent
tanh	Hyperbolic tangent.

4712.7 . SPECIALIZED MATH FUNCTIONS

Specialized Math Functions	
bessel	Bessel function
besselh	Hankel function
beta	Beta function
betaine	Incomplete beta function
betain	Logarithm of beta function
ellipj	Jacobi elliptic functions
ellipke	Complete elliptic integral
erf	Error function
erfe	Complementary error function
erfcx	Scaled complementary error function
erfinv	Inverse error function
gamma	Gamma function
gammainc	Incomplete gamma function
gammain	Logarithm of gamma function
log2	Dissect floating point numbers
pow2	Scale floating point numbers
rat	Rational approximation
rats	Rational output

4812.8 . MATRIX FUNCTIONS - NUMERICAL LINEAR ALGEBRA

Matrix Analysis	
cono	Matrix condition number
det	Determinant
norm	Matrix or vector norm
nuil	Null space
orth	Orthogonalization
rcond	LINPACK reciprocal condition estimator
rank	Number of linearly independent rows or columns
rref	Reduced row echelon form
trace	Sum of diagonal elements
Linear Equations	
chol	Cholesky factorization.
mv	Matrix inverse.
lscov	Least squares in the presence of known covariance.
lu	Factors from Gaussian elimination.
nnls	Nn-negative least- squares.
pinv	Pseudo inverse.
qr	Orthogonal-triangular decomposition.
\ and /	Near equation solution.

Eigenvalues and Singular Values	
balance	Diagonal scaling to improve eigenvalue accuracy
cdf2rdf	Complex diagonal form to real block diagonal form
eig	Eigenvalues and eigenvectors
hess	Hessenberg form
poly	Characteristic polynomial
qz	Generalized eigenvalues
rsf2csf	Real block diagonal form to complex diagonal form
sehur	Schur decomposition
svd	Singular value decomposition

Matrix Functions	
expm	Matrix exponential
expml	M-file implementation of expm
expm2	Matrix exponential via Taylor series
expm3	Matrix exponential via eigenvalues and eigenvectors
funm	Evaluate general matrix function
logm	Matrix logarithm
sqrtm	Matrix square root

4912.9 . DATA ANALYSIS AND FOURIER TRANSFORM FUNCTIONS

Basic Operations	
cumprod	Cumulative product of elements
cumsum	Cumulative sum of elements
max	Largest component
mean	Average or mean value
median	Median value
min	Smallest component
prod	Product of elements
sort	Sort in ascending order
std	Standard deviation
sum	Sum of elements
trapz	Numerical integration using trapezoidal method
Finite Differences	
del2	Five-point discrete Laplacian
diff	Difference function and approximate derivative
gradient	Approximate gradient (see online help)
Correlation	
corrcoef	Correlation coefficients
cov	Covariance matrix.

Filtering and Convolution	
conv	Convolution and polynomial multiplication
conv2	Two-dimensional convolution (see online help)
deconv	Deconvolution and polynomial division
filter	One-dimensional digital filter (see online help)
filter2	Two-dimensional digital filter (see online help)

Fourier Transforms	
abs	Magnitude
angle	Phase angle
cplxpair	Sort numbers into complex conjugate pairs
fft	Discrete Fourier transform
fft2	Two-dimensional discrete Fourier transform
fftshift	Move zeroth lag to center of spectrum
ifft	Inverse discrete Fourier transform
ifft2	Two-dimensional inverse discrete Fourier transform
nextpow2	Next higher power of 2
unwrap	Remove phase angle jumps across 3600 boundaries

5012.10 . POLYNOMIAL AND INTERPOLATION FUNCTIONS

Polynomials	
conv	Multiply polynomials
deconv	Divide polynomials
poly	Construct polynomial with specified roots
polyder	Differentiate polynomial (see online help)
polyfit	Fit polynomial to data
polyval	Evaluate polynomial
polyvalm	Evaluate polynomial with matrix argument
residue	Partial-fraction expansion (residues)
roots	Find polynomial roots
Data Interpolation	
griddata	Data gridding
interp1	1-D interpolation (1-D table lookup)
interp2	2-D interpolation (2-D table lookup).
interpft	1-D interpolation using FFT method.

5112.11 . FUNCTION FUNCTIONS

Function Functions - Nonlinear Numerical Methods	
fmin	Minimize function of one variable
fmins	Minimize function of several variables
fplot	Plot function
fzero	Find zero of function of one variable
ode23	Solve differential equations, low order method
ode45	Solve differential equations, high order method
quad	Numerically evaluate integral, low order method
quad8	Numerically evaluate integral, high order method

5212.12 . SPARSE MATRIX FUNCTIONS

Elementary Sparse Matrices	
spdiags	Sparse matrix formed from diagonals
speye	Sparse identity matrix
sprandn	Sparse random matrix
sprandsym	Sparse symmetric random matrix

Full to Sparse Conversion	
find	Find indices of nonzero entries
full	Convert sparse matrix to full matrix
sparse	Create sparse matrix from nonzero and indices
spconvert	Convert from sparse matrix external format
Working with Nonzero Entries of Sparse Matrices	
issparse	True if matrix is sparse
nnz	Number of nonzero entries
nonzeros	Nonzero entries
nzmax	Amount of storage allocated for nonzero entries
spalloc	Allocate memory for nonzero entries
spfun	Apply function to nonzero entries
spones	Replace nonzero entries with ones
Visualizing Sparse Matrices	
gplot	Plot graph, as in "graph theory"
spy	Visualize sparsity structure
Reordering Algorithms	
colmmd	Column minimum degree
colperm	Order columns based on nonzero count
dmperm	Dulmage-Mendelsohn decomposition
randperm	Random permutation vector
symmmd	Symmetric minimum degree
symrcm	Reverse Cuthill-McKee ordering
Norm, Condition Number, and Rank	
condest	Estimate 1-norm condition
normest	Estimate 2-norm
sprank	Structural rank
Miscellaneous	
spaugment	Form least squares augmented system
spparms	Set parameters for sparse matrix routines
symbfact	Symbolic factorisation analysis

5312.13 . TWO DIMENSIONAL GRAPHICS

Elementary X-Y Graphs	
f111	Draw filled 2-D polygons
loglog	Log-log scale plot
plot	Linear plot
semllogx	Semi-log scale plot
semilogy	Semi-log scale plot
Specialized X-Y Graphs	
bar	Bar graph
compass	Compass plot
errorbar	Error bar plot
feather	Feather plot
fplot	Plot function
hist	Histogram plot
polar	Polar coordinate plot
rose	Angle histogram plot
stairs	Stairstep plot

Graph Annotation	
grid	Grid lines
gtext	Mouse placement of text
text	Text annotation
title	Graph title
xlabel	X-axis label
ylabel	Y-axis label

5412.14 . THREE DIMENSIONAL GRAPHICS

Line and Area Fill Commands	
fill3	Draw filled 3 D polygons in 3 D space
plot3	Plot lines and points in 3 D space
Contour and Other 2-D Plots of 3-D Data	
clabel	Contour plot elevation labels
contour	contour plot
contour3	3-D contour plot
contourc	Contour plot computation (used by contour)
image	Display image
pcolor	Pseudocolor (checkerboard) plot
quiver	Quiver plot.
Surface and Mesh Plots	
mesh	3-D mesh surface
meshc	Combination mesh/contour plot
meshz	3-D Mesh with zero plane
sîlce	Volumetric visualization plot (see online help)
surf	3-D shaded surface
surfc	Combination surf/contour plot
surfl	3-D shaded surface with lighting
waterfall	Waterfall plot (see online help).
Graph Appearance	
axis	Axis scaling and appearance
caxis	Pseudocolor axis scaling
colormap	Color lookup table.
hidden	Mesh hidden line removal mode.
shading	Color shading mode
view	3-D graph viewpoint specification
viewmtx	View transformation matrices
Graph Annotation	
grid	Grid lines
gtext	Mouse placement of text
text	Text annotation
title	Graph title
xlabel	X-axis label
ylabel	Y-axis label
zlabel	Z-axis label for 3-D plots
3-D Objects	
cylinder	Generate cylinder
sphere	Generate sphere

5512.15 . General Purpose Graphics Functions

Figure Window Creation and Control	
clf	Clear current figure
close	Close figure
figure	Create Figure (graph window)
gcf	Get handle to current figure

Handle Graphics Operations	
delete	Delete object
drawnow	Flush pending graphics events
get	Get object properties
reset	Reset object properties
set	Set object properties.

Axis Creation and Control	
axes	Create axes in arbitrary positions
axis	Control axis scaling and appearance
caxis	Control pseudocolor axis scaling
cla	Clear current axes
gca	Get handle to current axes
hold	Hold current graph
subplot	Create axes in tiled positions.
Handle Graphics Objects	
axes	Create axes
figure	Create figure window
image	Create image
line	Create line
patch	Create patch
surface	Create surface
text	Create text
uicontrol	Create user interface control
uimenu	Create user interface menu
Hardcopy and Storage	
orient	Set paper orientation
print	Print graph or save graph to file
printopt	Configure local printer defaults
Miscellaneous	
ginput	Graphical input from mouse
ishold	Return hold state

Movies and Animation	
getframe	Get movie frame
	movie frames
moviein	Initialize movie frame memory.

5612.16 . Color Control and Lighting Model Functions

Color Controls	
caxis	Pseudocolor axis scaling
colormap	Color lookup table
shading	Color shading mode
Color Maps (see online help)	
bone	Gray-scale with a tinge of blue color map
cool	Shades of cyan and magenta color map
copper	Linear copper-tone color map
f lag	Alternating red, white, blue, and black color map
gray	Linear gray-scale color map
hsv	Hue-saturation-value color map
hot	Black-red-yellow-white color map
pink	Pastel shades of pink color map
Color Map Related Functions	
brighten	Brighten or darken color map
hsv2rgb	Hue-saturation-value to red-green-blue conversion
rgb2h 5V	Red-green-blue to hue-saturation-value conversion
rgbplot	Plot color map
spinmap	Spin color map
Lighting Models	
diffuse	Diffuse reflectance
specular	Specular reflectance
surfl	3D shaded surface with lighting
surfnorm	Surface normals

5712.17 . Sound Processing Functions

General Sound Functions	
saxis	Sound axis scaling
sound	Convert vector into sound
SPARCstation-specific Sound Functions	
auread	Read Sun audio file (see online help)
auwrite	Write Sun audio file (see online help)
lin2mu	Linear to mu-law conversion (see online help)
mu2iin	Mu-law to linear conversion (see online help)

5812.18 . Character String Functions

General	
abs	Convert string to numeric values
eval	Execute string with MATLAB expression
isstr	True for string
setstr	Convert numeric values to string
str2mat	Form text matrix from individual strings
string	About character strings in MATLAB

String Comparison	
lower	Convert string to lowercase
strcmp	Compare strings
upper	Convert string to uppercase
String to Number Conversion	
int2str	Convert integer to string
num2st r	Convert number to string
sprintf	Convert number to string under format control
sscanf	Convert string to number under format control
str2num	Convert string to number.
Hexadecimal to Number Conversion	
dec2hex	Convert decimal integer to hex string
hex2dec	Convert hex string to decimal integer
hex2num	Convert hex string to IEEE floating point number
String Conversion	
sprintf	Write formatted data to string
sscanf	Read string under format control

5912.19 . LOW-LEVEL FILE I/O FUNCTIONS

File Opening and Closing	
fclose	Close file
fopen	Open file
Unformatted I/O	
fread	Read binary data from file
fwrite	Write binary data to file
Formatted I/O	
fgetl	Read line from file, discard new line character
fgets	Read line from file, keep new line character
fprintf	Write formatted data to file
fscanf	Read formatted data from file
File Positioning	
ferror	Inquire file I/O error status
frewind	Rewind file
fseek	Set file position indicator
ftell	Get file position indicator

6012.20 . CONTROL SYSTEM

Model Building	
append	Append system dynamics
augstate	Augment states as outputs
blkbuild	Build state-space system from block diagram
cloop	Close loops of system
connect	Block diagram modelling
conv	Convolution of two polynomials (see *MATLAB Reference Guide)*
destim	Form discrete state estimator from gain matrix
dreg	Form discrete controller/estimator from gain matrices
drmodel	Generate random discrete model
estim	Form continuous state estimator from gain matrix
feedback	Feedback system connection
ord2	Generate A,B,C,D for a second-order system
pade	Padé approximation to time delay
parallel	Parallel system connection
reg	Form continuous controller/estimator from gain matrices
rmodel	Generate random continuous model
series	Series system connection
ssdelete	Delete inputs, outputs or states from model
ssselect	Select subsystem from larger system
Model Reduction	
balreal	Balanced realization
dbalreal	Discrete balanced realization
dmodred	Discrete model order reduction
minreal	Minimal realization and pole zero cance1lation
rnodred	Model order reduction
Model Conversions	
c2d	Continuous to discrete-time conversion
c2dm	Continuous to discrete-time conversion with method
c2dt	Continuous to discrete conversion with delay
d2c	discrete to continuous-time conversion
d2cm	Discrete to continuous-time conversion with method
poly	Roots to polynomial conversion (see *MATLAB Reference Guide)*
residue	Partial fraction expansion (see MATLAB Reference Guide)
ss2tf	State-space to transfer function conversion
ss2zp	State-space to zero-pole conversion.
tf2ss	Transfer function to state-space conversion
tf2zp	Transfer function to zero-pole conversion
zp2tf	Zero-pole to transfer function conversion
zp2ss	Zero-pole to state-space conversion
Model Realizations	
canon	Conversion of system to canonical forin
ctrbf	Controllability staircase form
obsvf	Observability staircase form.
ss2ss	Similarity transform.

Model Properties	
covar	Continuous covariance response to white noise
ctrb	Controllability matrix.
damp	Damping factors and natural frequencies
dcgain	Continuous steady state (D.C.) gain
dcovar	Discrete covariance response to white noise
ddamp	Discrete damping factors and natural frequencies
ddcgain	Discrete steady state (D.C.) gain
dg ram	Discrete controllability and observability gramians
dsort	Sort discrete eigen values by magnitude
eig	System eigen values (see MATLAB Reference Guide)
esort	Sort continuous eigen values by real part
gram	Controllability and observability gramians
obsv	Observability matrix
printsys	Special formatted print of system
roots	Roots of polynomial (see MATLAB Reference Guide)
tzero	Transmission zeros.

Equation Solution	
are	Algebraic Riccati equation solution
dlyap	Discrete Lyapunov equation solution
lyap	Continuous Lyapunov equation solution
lyap2	Lyapunov equation solution using diagonalization.
Time Response	
dimpulse	Discrete unit sample response
dinitial	Discrete initial condition response
disin	Discrete simulation to arbitrary inputs
dstep	Discrete step response
filter	SISO z-transform simulation (see MATLAB Reference Guide).
Impulse	Impulse response
initial	Continuous initial condition response
lsim	Continuous simulation to arbitrary inputs
ltitr	low level time response function
step	Step response
Frequency Response	
bode	Bode plots
dbode	Discrete Bode plots
dnichols	Discrete Nichols plots
dnyquist	Discrete Nyquist plots
dsigma	Discrete singular value frequency plots
fbode	Fast Bode response of continuous systems
Freqs	Laplace-transform (see the Signal Processing Toolbox)
freqz	Z-transform (see the Signal Processing Toolbox)
ltifr	Low level frequency response function
margin	Gain and phase margins
nichols	Nichols plots
ngrid	Grid lines for Nichols plot
nyquist	Nyquist plots
sigma	Continuous singular value frequency plots

Root Locus	
pzmap	Pole-zero map
rlocfind	Interactive root locus gain determination
rlocus	Evans root-locus
sgrid	Continuous root locus ω_n, ξ grid
zgrid	Discrete root locus ω_n, ξ grid.