Introduction à MATLAB et SIMULINK

Un guide pour les élèves de l'École Nationale Supérieure d'Ingenieurs Electriciens de Grenoble

Paolino Tona
Laboratoire d'Automatique de Grenoble

Ce document couvre les aspects principaux du logiciel MATLAB et de son extension SIMULINK.

Il a été élaboré à partir de l'homonyme guide préparé par Hoang Le-Huy, Professeur à l'Université Laval, Quebec, Canada.

Par rapport au document du Prof. Le-Huy qu'on peut télécharger à l'URL

`http://www.gel.ulaval.ca/~lehuy/intromatlab`

ce guide propose une mise à jour (version 5.3 de MATLAB et 3.0 de SIMULINK), des corrections et du nouveau matériel.

Plus particulièrement, une annexe sur la CONTROL SYSTEM TOOLBOX a été ajoutée. Pour la préparation de cette annexe, le « Tutorial for Control System Toolbox » de Finn Haugen (Telemark College, Porsgrunn, Norvège), s'est avéré très utile. Ce dernier document peut être consulté à l'URL :

`http://www-pors.hit.no/~finnh/contoolb.html`

Commentaires et suggestions sont les bienvenus à l'adresse mél :

`Paolino.Tona@lag.ensieg.inpg.fr`

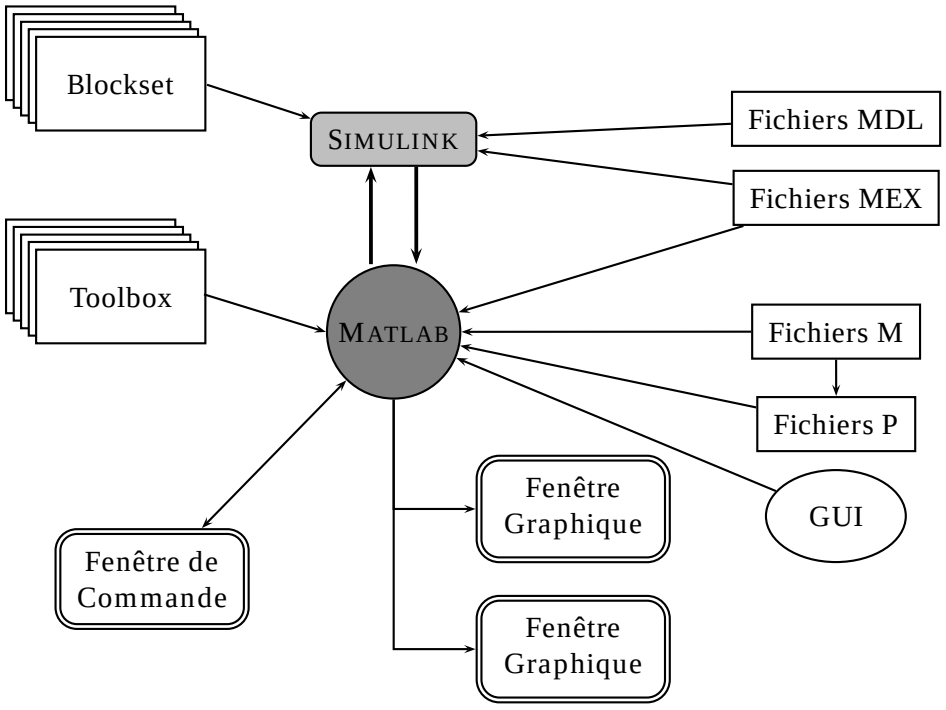
Table des matières

Introduction à MATLAB

Introduction à MATLAB

MATLAB

- est un logiciel de calcul matriciel à syntaxe simple ;
- peut être considéré comme un langage de programmation adapté pour les problèmes scientifiques, grâce à ses fonctions spécialisées ;
- est un interpréteur, car ses instructions sont interprétées et exécutées ligne par ligne ;
- possède des bonnes capacités graphiques pour présenter des résultats ou pour créer des applications ;
- peut être intégré avec du code C ou FORTRAN ;
- fonctionne dans plusieurs environnements tels que UNIX / X-Windows, Windows, Macintosh.



Fenêtre de Commande : dans cette fenêtre, l'utilisateur donne les instructions et MATLAB retourne les résultats ;

Fenêtres Graphiques : MATLAB trace les graphiques dans ces fenêtres ;

Fichiers M : ce sont des programmes en langage MATLAB (écrits par l'utilisateur) ;

Fichiers P : version pré-interprétée des fichiers M ;

Toolboxes : (« boîtes à outils») ce sont des collections de fichiers M développés pour des domaines d'application spécifiques (SIGNAL PROCESSING TOOLBOX, SYSTEM IDENTIFICATION TOOLBOX, CONTROL SYSTEM TOOLBOX, μ-SYNTHESIS AND ANALYSIS TOOLBOX, ROBUST CONTROL TOOLBOX, OPTIMIZATION TOOLBOX, NEURAL NETWORK TOOLBOX, SPLINE TOOLBOX, SYMBOLIC MATH TOOLBOX, FUZZY LOGIC TOOLBOX, etc.) ;

Simulink : c'est l'extension graphique de MATLAB permettant de travailler avec des schéma en blocs, pour modéliser et simuler des systèmes ;

Blocksets : ce sont des collections de blocs SIMULINK développés pour des domaines d'application spécifiques (DSP BLOCKSET, POWER SYSTEM BLOCKSET, etc.)

Fichiers MDL : ce sont des fichiers représentant des modèles SIMULINK ;

Fichier MEX : modules executables créés à partir de sources en C ou FORTRAN ;

GUI : interface graphique utilisateur pour créer des applications basées sur MATLAB ;

En MATLAB, il existe deux modes de fonctionnement :

mode interactif : MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'utilisateur.

mode exécutif : MATLAB exécute ligne par ligne un programme en langage MATLAB écrit dans un fichier M (ou P), ou un fichier exécutable MEX.

Une session de travail MATLAB

Démarrer MATLAB

Pour activer la Fenêtre de Commande MATLAB

sous UNIX : dans une fenêtre `cmdtool`, taper **`matlab`** ;

sous WINDOWS : cliquer sur l'icône `Matlab` sur le bureau ou dans le groupe `Matlab` du menu `Démarrer/Programmes`.

Dans la Fenêtre de Commande, l'invité de commande '**»** ' permet de taper les instructions une ligne à la fois, chaque ligne etant exécutée immédiatement après la touche `Entrée` (ou `Return`). Les instructions de contrôle (**`for`**, **`while`**, **`if`** ... **`else`**), aussi bien que le définitions de matrices et vecteurs peuvent prendre plusieurs lignes, avec une exécution différée. Une ligne peut contenir plusieurs instructions séparées par des virgules (**`,`**).

Aide

Pour obtenir de l'aide sur un sujet, une instruction ou une fonction, on tape **help** suivi par le sujet, l'instruction ou la fonction désirée (en minuscule ! ! !).

Exemple

```
» help log10
 LOG10  Common (base 10) logarithm.
   LOG10(X) is the base 10 logarithm of the elements of X.
   Complex results are produced if X is not positive.

   See also LOG, LOG2, EXP, LOGM.
```

D'autres commandes utiles sont :

`helpwin`	accéder à l'aide à travers une fenêtre de navigation
`helpdesk`	accéder à la documentation MATLAB
`lookfor xyz`	chercher la chaîne de caractères `xyz` dans les descriptions des toutes les fonctions disponibles
`demo`	lancer les demos MATLAB

Commandes système

`pwd`	nom du répertoire courant
`cd`	changer de répertoire
`dir, ls`	contenu du répertoire courant
`mkdir`	créer un nouveau répertoire
`delete`	effacer un fichier
`copyfile`	copier un fichier

Variables et espace de travail (*Workspace*)

En MATLAB il n'y a pas d'instructions pour déclarer ou dimensionner une variable. Une nouvelle variable est définie en donnant son nom et sa valeur numérique

```
» ts=0.03
ts =
    0.0300
» C=[1 1 0 0]
C =
   1   1   0   0
```

ou son expression mathématique

```
» ws=2*pi/ts
ws =
    209.4395
```

Si la variable existe déjà, MATLAB en change le contenu, et, si nécessaire, alloue plus de place en mémoire

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

Les variables ainsi définies sont stockées dans l'espace de travail et peuvent être utilisées dans les calculs subséquents.

AFFICHAGE

Pour afficher la valeur d'une variable il suffit de taper son nom

```
» ts
ts =
    0.0300
```

Le résultat d'une instruction est affiché par défaut. Si l'on veut que l'instruction soit exécutée sans afficher de résultat, il faut ajouter **;** à la fin de la ligne

```
» ws=2*pi/ts;
»
```

CLASSES

Par la syntaxe introduite ci-dessus, MATLAB définit des variables qui appartiennent à la classe **double array**, c'est à dire des tableaux de réels qui peuvent correspondre à des scalaires, des vecteurs ou des matrices. Mise à part cette classe fondamentale, il faut signaler qu'il existe d'autres classes MATLAB pré-définies. La plus importante est certainement **char array**, à laquelle appartiennent les chaînes de caractères, définies en utilisant **' '**

```
» hi = 'salut'
hi =
salut
```

INFORMATION SUR L'ESPACE DE TRAVAIL

who	affichage des variables dans l'espace de travail
whos	affichage détaillé des variables dans l'espace de travail

Exemples

```
» who
Your variables are:

C       hi       ts       ws


» whos
   Name      Size      Bytes      Class
   C          3x3         72      double array
   hi         1x5         10      char array
   ts         1x1          8      double array
   ws         1x1          8      double array
Grand total is 16 elements using 98 bytes
```

EFFACER DES VARIABLES DE L'ESPACE DE TRAVAIL

clear var1 var2	effacer les variables **var1** et **var2**
clear (all)	effacer toutes les variables

ENREGISTRER DES VARIABLES DANS UN FICHIER

Pour enregistrer les variables de l'espace de travail dans un fichier, on utilise les instructions suivantes :

`save`	enregistrer toutes les variables dans un fichier `matlab.mat`
`load`	ramène les variables enregistrées dans le fichier `matlab.mat` dans l'espace de travail
`save fichier1.mat x Y z`	enregistrer les variables **`x, Y, z`** dans un fichier `fichier1.mat`
`load fichier1`	ramène les variables enregistrées dans le fichier `fichier1.mat` dans l'espace de travail

Clore une session de travail

Pour clore une session de travail, taper **`quit`**.

Opérations mathématiques

Nombres et opérations arithmétiques

Nombres

Les nombres réels peuvent être sous différents formats :

```
5    1.0237    0.5245E-12    12.78e6    0.001234    -235.087
```

Les nombres complexes peuvent être écrits sous forme cartésienne ou polaire :

forme cartésienne `0.5 + i*2.7    -1.2 + j*0.789    2.5 + 9.7i`

forme polaire `1.25*exp(j*0.246)`

Formats d'affichage

Pour choisir le format d'affichage pour les nombres, on utilise l'instruction **`format`** :

```
format short       0.1234 (format par défaut)
format long        0.12345678901234
format short e     1.2341E+002
format long e      0.123456789012345E+002
format hex         ABCDEF0123456789
```

Opérations arithmétiques

- **+** Addition
- **-** Soustraction
- ***** Multiplication
- **/** Division à droite
- \ Division à gauche
- ^ Puissance

En demandant l'aide sur l'un de ces opérateurs

```
» help +
```

ou

```
» help /
```

on obtient la liste de tous les opérateurs (arithmétiques et non) disponibles.

Vecteurs et matrices

Vecteurs

On peut définir un vecteur `x` en donnant la liste de ses éléments :

```
» x=[3 0.15 -1.5 2.42 -0.42]
x =
   3.0000   0.1500   -1.5000   2.4200   -0.4200
```

ou en donnant la suite qui forme le vecteur :

```
» x=1:0.5:3
x =
   1.0000   1.5000    2.0000   2.5000   3.0000
```

ou en utilisant une fonction qui génère un vecteur à espacement linéaire :

```
» x=linspace(1,10,6)
x =
   1.0000   2.8000   4.6000   6.4000   8.2000   10.0000
```

ou exponentiel :

```
» x=logspace(0,2,7)
x =
   1.0000   2.1544   4.6416   10.0000   21.5443   46.4159   100.0000
```

Matrices

On définit une matrice A en donnant ses éléments :

```
» A=[0.3 2.7 4.1;3.1 -0.74 2.1;6.7 -4.3 -2.1]
A =
    0.3000    2.7000    4.1000
    3.1000   -0.7400    2.1000
    6.7000   -4.3000   -2.1000
```

Matrice unitaire :

```
» B=eye(4)
B =
    1   0   0   0
    0   1   0   0
    0   0   1   0
    0   0   0   1
```

Emploi des indices

Les éléments d'un vecteur ou d'une matrice peuvent être adressés en utilisant les indices sous la forme suivante :

`t(10)` élément n^o 10 du vecteur t
`A(2,9)` élément se trouvant à ligne 2, colonne 9 de la matrice A
`B(:,7)` la colonne 7 de la matrice B
`C(3,:)` la ligne 3 de la matrice B

Opérations matricielles

Les opérations matricielles exécutées par MATLAB sont illustrées dans le tableau suivant :

B = A'	la matrice B est égale à la matrice A transposée
E = inv(A)	la matrice E est égale à la matrice A inversée
D = A-B	soustraction
Z = X*Y	multiplication
X = A\B	résout le système d'équations $\mathbf{Ax} = \mathbf{B}$
X = A/B	équivaut à **(B'/A')'**(**help slash** pour plus de details)

Opérations "élément par élément"

Les opérations « élément par élément » des vecteurs et des matrices sont effectuées en ajoutant un point (**.**) avant les opérations *** \ / ^ '**

Exemple.

```
» A=[1 2 3 4 5];
» B=[6 7 8 9 10];
» C=A.*B;
C =
    6   14   24   36   50
» D=A./B;
D =
    0.1667   0.2857   0.3750   0.4444   0.5000
```

Effacer des lignes et des colonnes

t=[]	créer un tableaux vide
X(:,2)=[]	effacer la deuxième colonne de **X**
Y(3,:)=[]	effacer la troisième ligne de **Y**

Concaténation

L'opérateur **[...]** permet d'enchaîner des matrices ou des vecteurs entre eux pour former des matrices ou des vecteurs plus grands

```
» A = ones(2)
A =
     1   1
     1   1
» B = zeros(2)
B =
     0   0
     0   0
» C = [A B]
C =
     1   1   0   0
     1   1   0   0
» D = [A ; B]
D =
```

```
     1   1
     1   1
     0   0
     0   0
```

Il agit aussi sur des chaînes de caractères

```
» poli = 'merci';
» tres_poli = [poli ' beaucoup'];
tres_poli =
merci beaucoup
```

Polynômes

MATLAB ne fournit pas de types ou classes pré-définis pour représenter directement des polynômes. Si l'on utilise un vecteur **P** contenant les coefficients du polynôme (en ordre décroissant), les fonctions suivantes sont disponibles

roots(P)	racines de P ;
polyval(P,x)	évaluation de P en x ;
conv(P1,P2)	multiplication polynomiale
deconv(P1,P2)	division polynomiale

Si **R** est un vecteur contenant les racines d'un polynôme P, la fonction **poly(R)** reconstruit les coefficients du polynôme. Si **A** est une matrice, **poly(A)** calcule son polynôme caracteristique.

Variables et fonctions

Variables

On définit une variable en donnant son nom et sa valeur numérique ou son expression mathématique :

```
a = 1.25;
x = 0:0.5:10;
y = a*x;
z = y.^2;
```

Expressions mathématiques

On écrit les expressions mathématiques de la façon habituelle :

```
z = 5*exp(-0.4*x).*sin(7.5*y);
```

Fonctions mathématiques

Les *fonctions mathématiques de base* sont données dans le tableau suivant :

abs valeur absolue module (nb. complexe)	**angle** argument (nb. complexe)	**sqrt** racine carrée	**real** partie réelle	**imag** partie imaginaire
conj conjuguée (nb. complexe)	**round** arrondir	**fix** arrondir (vers zéro)	**floor** arrondir (vers $-\infty$)	**ceil** arrondir (vers $+\infty$)
sign signe	**rem** reste	**exp** exponentielle	**log** logarithme base e	**log10** logarithme base 10

Les *fonctions trigonométriques* sont données dans le tableau suivant :

sin	**cos**	**tan**	**asin**	**acos**	**atan**	**atan2**
sinh	**cosh**	**tanh**	**asinh**	**acosh**	**atanh**	

Exemple

```
» x=-2+5i
x =
    -2.0000 + 5.0000i
» a=real(x);
a =
    -2
» b=imag(x);
b =
    5
» m=abs(x);
m =
    5.3852
» alpha=angle(x);
alpha =
    1.9513
```

Exemple

```
» w=50;
» t=0.5e-3;
» y=25*exp(-4*t)*cos(w*t)
y =
    24.9423
```

Création de fonctions

L'utilisateur peut créer des fonctions particulières pour ses applications. Voir la partie « Programmation avec MATLAB ».

Graphiques

Introduction

S'il n'y a pas de fenêtres graphiques actives, appeler une fonction graphique suffit pour en ouvrir une. Les instructions graphiques suivantes continueront à agir sur la même fenêtre à moins d'en ouvrir d'autres (fonction **`figure`**).

Les fonctions graphiques disponibles peuvent s'afficher en demandant l'aide sur **`graph2D`**, **`graph3D`** et **`specgraph`** (graphiques et fonctions spécialisés).

Graphiques 2D

Tracé de courbes

On utilise l'instruction **`plot`** pour tracer un graphique 2D :

`plot(x,y)`	tracer le vecteur **`y`** en fonction du vecteur **`x`**
`plot(t,x,t,y,t,z)`	tracer $x(t)$, $y(t)$ et $z(t)$ sur le même graphique
`plot(t,x,'r:')`	tracer $x(t)$ en trait pointillé rouge

Taper **`helpplot`** pour explorer toutes les possibilités de cette commande.

Format du graphique

On peut choisir le format du graphique :

`plot(x,y)`	tracer $y(x)$ avec échelles linéaires
`semilogx(f,A)`	tracer $A(f)$ avec échelle $log(f)$
`semilogy(w,B)`	tracer $B(w)$ avec échelle $log(B)$
`polar(theta,r)`	tracer $r(theta)$ en coordonnées polaires
`bar(x,y)`	tracer $y(x)$ sous forme de barres

Ajout de texte au graphique

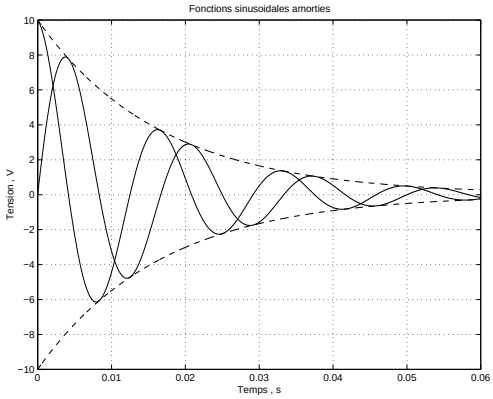
`title('Titre du graphique')`	donner un titre au graphique
`xlabel('Temps')`	étiquette de l'axe x
`ylabel('Amplitude')`	étiquette de l'axe x
`gtext('Valeur absolue')`	ajouter du texte au graphique avec la souris
`legend('sortie réelle','sortie simulée')`	ajouter une légende

Manipulation de graphiques et de fenêtres

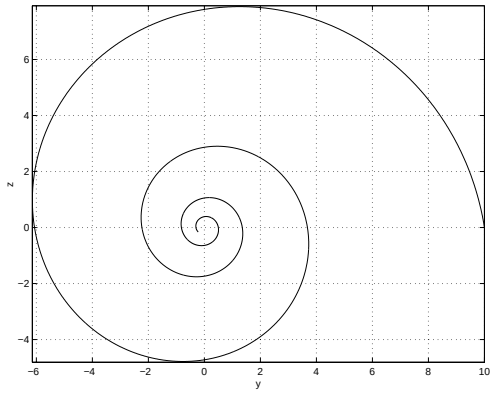
grid, **grid on**, **grid off**	ajouter une grille
axis([-1 5 -10 10])	choix des échelles $x = (-1, 5)$ et $y = (-10, 10)$
hold, **hold on**, **hold off**	garder le graphique sur l'écran (pour tracer plusieurs courbes sur le même graphique)
figure	ouvre une nouvelle fenêtre graphique
close all	ferme toutes les fenêtres graphiques

Un exemple complet

```
» t=0:0.01e-3:0.06;
» y=10*exp(-60*t).*cos(120*pi*t);
» z=10*exp(-60*t).*sin(120*pi*t);
» plot(t,y,'r',t,z,'g'),grid
» a=10*exp(-60*t);
» hold
Current plot held
» plot(t,a,'b-')
» plot(t,-a,'b-')
» title('Fonctions sinusoidales amorties')
» xlabel('Temps , s'),ylabel('Tension , V')
```



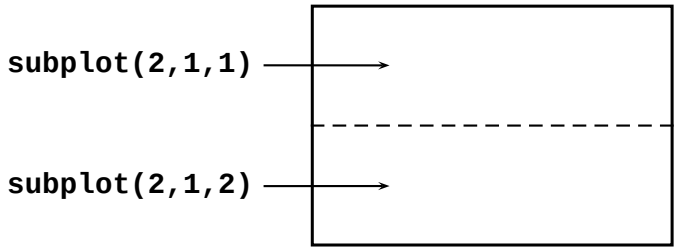

```
» hold off
» plot(y,z),grid
» axis equal
» xlabel('y'),ylabel('z')
```



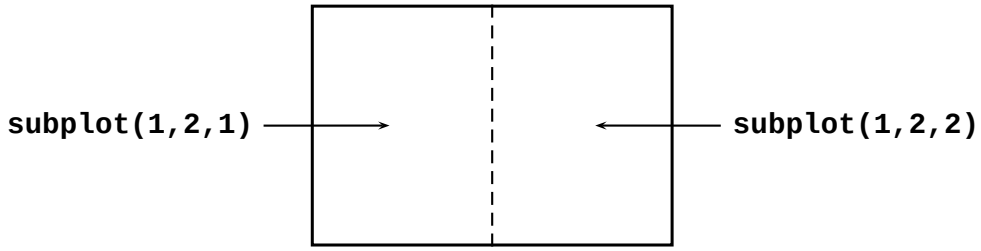
Graphique multiple

On peut tracer plusieurs graphiques dans la même fenêtre en utilisant l'instruction **subplot** pour diviser la fenêtre en plusieurs parties (le tracés se feront en suite avec **plot**, **semilogx**, etc.) :

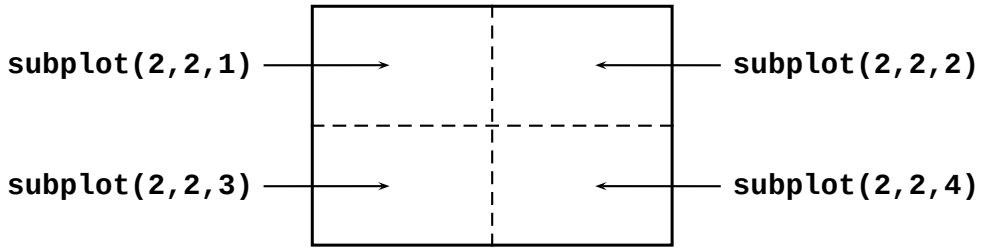
– diviser la fenêtre en deux parties (2 x 1)



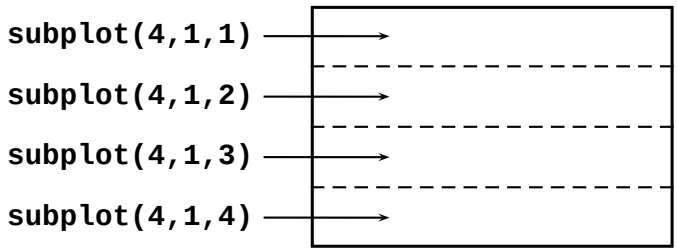
– diviser la fenêtre en deux parties (1 x 2)



– diviser la fenêtre en quatre parties (2 x 2)

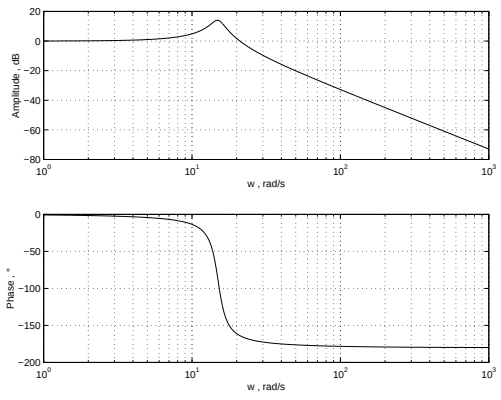


– diviser la fenêtre en quatre parties (4 x 1)



Exemple

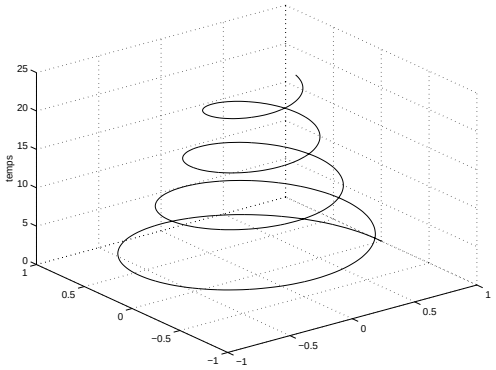
```
» w=logspace(0,3,1000);
» s=j*w;
» H=225./(s.*s+3*s+225);
» AdB=20*log10(abs(H));
» phase=angle(H)*(180/pi);
» subplot(2,1,1),semilogx(w,AdB),grid
» xlabel('w , rad/s'),ylabel('Amplitude , dB')
» subplot(2,1,2),semilogx(w,phase),grid
» xlabel('w , rad/s'),ylabel('Phase , °')
```



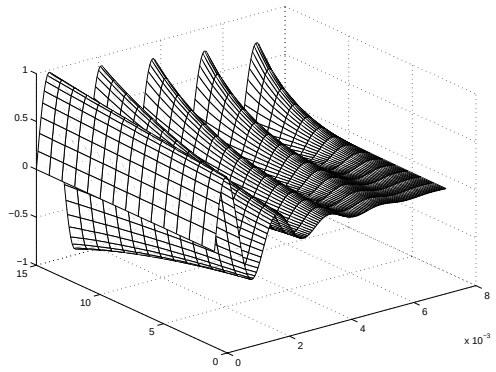
Graphiques 3D

Les deux exemples suivants montrent comment tracer des graphiques 3D.

```
» t = 0:0.05:25;
» x = exp(-0.05*t).*cos(t);
» y = exp(-0.05*t).*sin(t);
» z = t;
» plot3(x,y,z), grid
» zlabel('temps')
```




```
» b=1200*pi;
» dt=50e-6;
» for j=1:15
» for i=1:150
» k(j)=j;
» a=(16-j)*50;
» t(i)=(i-1)*dt;
» y(i,j)=exp(-a*t(i)).*sin(b*t(i));
» end
» end
» [K,T]=meshgrid(k,t);
» mesh(T,K,y)
```



Impression et enregistrement de graphiques

L'impression de graphiques se fait normalement à partir des menus de la fenêtre graphique. Plusieurs possibilités sont offertes en terme de positionnement de la fenêtre dans la feuille.

L'impression peut aussi se faire par la fonction **`print`**, en passant par l'un des pilotes d'imprimante disponibles. Toutefois, cette fonction est principalement utilisée pour enregistrer le contenu d'une fenêtres dans un fichier graphique. Par exemple, **`print -deps`** enregistre en format *Encapsulated PostScript*.

Nouvelles fonctionnalités

L'environnement **`propedit`** permet de changer les *propriétés* d'un graphique. Dans la version 5.3 la plupart de ces propriétés sont directement accessibles à partir de la fenêtre graphique. Donc, pratiquement toutes les manipulations faites par ligne de commande sont disponibles dans les menus, avec davantage de possibilités. Malheureusement, cet environnement est à l'heure actuelle un peu fragile (= bogues fréquents).

Programmation avec MATLAB

Communication avec l'utilisateur

On peut afficher un message, une valeur à l'écran avec **`disp`** :

`disp('Ceci est un test')` afficher "Ceci est un test" sur l'écran

On peut saisir une valeur avec **`input`** :

`x = input('Valeur de x = ')` afficher sur l'écran "Valeur de x = "
en attendant qu'un nombre soit tapé

Contrôle de l'execution

Boucle FOR

On peut créer une boucle, ou des boucles imbriquées en utilisant **`for`** ... **`end`**.

Exemple
– boucle FOR simple :

```
for i=1:100
  wt=24*i*0.01;
  x(i)=12.5*cos(wt+pi/6);
end
```

– boucles imbriquées :

```
for i=1:5   for j=1:20
   amp=i*1.2;
   wt=j*0.05;
   v(i,j)=amp*sin(wt);
  end
end
```

Des increment négatifs (**`for i=5:-1:1`** ... **`end`**) ou non entiers (**`for i=1:0.2:4`** ... **`end`**) sont aussi possibles.

Boucle WHILE

On peut créer une boucle en utilisant **`while`** ... **`end`**.

```
n=1;
while n<100
  x=n*0.05
  y(n)=5.75*cos(x);
  z(n)=-3.4*sin(x);
  n=n+1;
end
```

Exécution conditionnelle

Les instructions **if** ... **end** , **if** ... **else** ... **end**, **if** ... **elseif** ... **else** ... **end**, **switch** ... **case** ... **case** ... **end** permettent de choisir entre plusieurs options.

Exemple

```
n=input('Donner un nombre positif:  ');
if rem(n,3)==0
  disp('Ce nombre est divisible par 3')
elseif rem(n,5)==0
  disp('Ce nombre est divisible par 5')
else
  disp('Ce nombre n"est pas divisible par 3 ou par 5')
end
```

Fichiers M

Les fichiers M sont des fichiers ASCII (pur texte) contenant des suites d'instructions MATLAB et ayant extension `.m`, par exemple « test1.m ». Si l'on tape **`test1`**, les instructions contenues dans le fichier `test1.m` seront exécutées une par une. Pour créer des fichiers M on peut utiliser n'importe quel éditeur de texte, ou bien l'Éditeur MATLAB intégré (commande **`edit`**).

Exemple

```
% ceci est un exemple de fichier M
% les lignes de commentaire commencent par "%"
for i=1:10   for j=1:4
   x=0.005*i;
   x=30+j;
   z(i,j)=10*exp(y*x)*cos(120*pi*x);
  end
end
```

Création de fonctions MATLAB

Des nouvelles fonctions peuvent être ajoutée aux fonctions MATLAB préexistantes. Une nouvelle fonction n'est autre qu'un fichier M particulier dont la première ligne contient la définition syntaxique de la fonction, à travers le mot clé **`function`** :

`function y = mafonct1(x)`	fonction qui retourne **`y`** à partir de **`x`**
`function z = mafonct2(x,z)`	fonction qui retourne **`z`** à partir de **`x`** et de **`y`**
`function [y1,y2] = mafonct3(x1,x2)`	fonction qui retourne **`y1`** et **`y2`** à partir de **`x1`** et de **`x2`**

Le nom de la fonction est déterminé par le nom du fichier `.m` qui la contient, et non pas par le nom qui apparaît dans la définition syntaxique. Dans tous les cas, éviter, si possible, d'utiliser deux noms différents, et, surtout, ne jamais utiliser un nom de fonction qui correspond à une fonction déjà existante (en vérifier l'existence avec **`which nomfonct`**).

Exemple

```
function y = eff(x)

% eff  Calcul de la valeur efficace
%      Pour un vecteur eff(x) donne la valeur efficace
%      Pour une matrice, eff(x) donne un vecteur contenant
%      la valeur efficace de chaque colonne.

m = length(x);
y=sqrt(sum(x.*x)/m);
```

Si la fonction est contenue dans un fichier `eff.m` on pourra l'appeler par **`eff`** dans la ligne de commande, dans un autre fichier M ou dans une autre fonction « utilisateur » :

```
» v = [1; 2; 3];
» val_effy = eff(v)
» val_eff =
     2.1602
```

Remarques

- les commentaires d'entête donnés dans la fonction **`eff`** seront affichés à l'écran lorsqu'on tape **`help eff`** ;
- la fonction n'est accessible que si le fichier qui la définit se trouve dans le repertoire courant ou dans un repertoire inclus dans le « path » (**`help addpath`** ou **`help path`** pour plus de details).

Programmation orientée-objets

Pour le courageux, des fonctions telles que **`class`**, **`superiorto`**, **`inferiorto`** permettent d'accéder à un semblant de programmation orientée-objets. En tous cas, la programmation fonctionnelle suffit largement pour résoudre les problèmes plus communs.

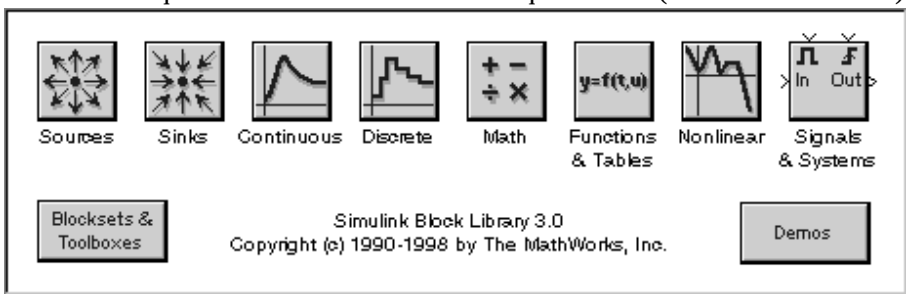
Introduction à SIMULINK

SIMULINK est l'extension graphique de MATLAB permettant de représenter les fonctions mathématiques et les systèmes dynamiques sous forme de schémas en blocs.

Démarrer SIMULINK

Dans la fenêtre Commande de MATLAB, taper **`simulink`**. La fenêtre contenant la *bibliothèque* SIMULINK va s'ouvrir.

⇐ bibliothéque Windows bibliothéque UNIX (et vielles versions)



Bien que l'interface de la bibliothèque ne soit pas homogène entre différentes versions et différentes machine, le concept de base est le même : la bibliothèque contient des collections de blocs simples qu'on peut connecter pour former des diagrammes.

Collections de blocs

Dans la version 3.0 de SIMULINK les principaux blocs sont organisés comme suit

COLLECTION	CONTENU	BLOCS LES PLUS UTILISÉS
`Sources`	sources de signaux	générateurs de signaux, horloges, chargement de données d'un fichier ou du *workspace*
`Sinks`	affichage, stockage	afficheurs de signaux, stockage de données dans un fichier ou dans le *workspace*
`Continuous`	blocs continus	intégrateur, fonction de transfert, représentation d'état, retards
`Discrete`	blocs discrets	intégrateur, fonction de transfert, représentation d'état, bloqueurs, filtres
`Math`	opérateurs mathématiques	fonctions trigonométriques, signe, valeur absolue, gains, somme, produit
`Function & Tables`	fonctions, interpolation	expression générique, tableaux d'interpolation, fonction MATLAB, *S-fonctions*

COLLECTION	CONTENU	BLOCS LES PLUS UTILISÉS
Nonlinear	blocs non-linéaires	frottements, jeux, saturations, commutateurs
Signals & Systems	portes, connections	sous-système, porte d'entrée et de sortie pour un modèle ou un sous-système, signal vectoriel→signaux scalaires (**demux**) et vice versa (**mux**)

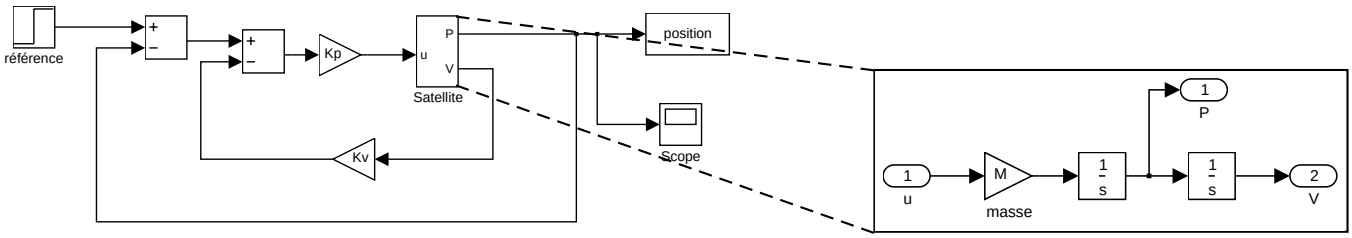
D'autres blocs utiles se trouvent dans les collections des différentes *toolbox*.

Construction d'un diagramme SIMULINK

Les étapes de base sont les suivantes

- pour commencer, dans le menu `File`, sélectionner `New Model` (pour la version 3.0 sur Windows, cliquer sur l'cône « feuille blanche ») : une fenêtre de travail `Untitled` s'ouvrira ;
- choisir les blocs dont on a besoin pour construire le diagramme dans les collections de la bibliothèque, et les faire glisser un par un dans la fenêtre de travail,
- faire des liaisons entre les blocs à l'aide de la souris ;
- cliquer sur les blocs dont on veut changer les paramètres : une fenêtre de dialogue s'ouvrira pour permettre cette opération ;
- sauvegarder le schéma ainsi obtenu dans un fichier **`*.mdl`**.

Un exemple très simple



D'habitude, on commence par mettre ensemble des sous-systèmes plus simples. Ces sous-systèmes doivent avoir des portes d'entrée (bloc **`In`**) et de sortie (bloc **`Out`**), pour relier les sous-systèmes au schéma principal. On crée des sous-systèmes par l'entrée `Create subsystem` du menu `Edit`, après avoir cadré les blocs qu'on veut regrouper à l'aide de la souris. Une deuxième façon consiste à copier un groupe de blocs à l'intérieur d'un bloc **`Subsystem`** dans la fenêtre du schéma principal.

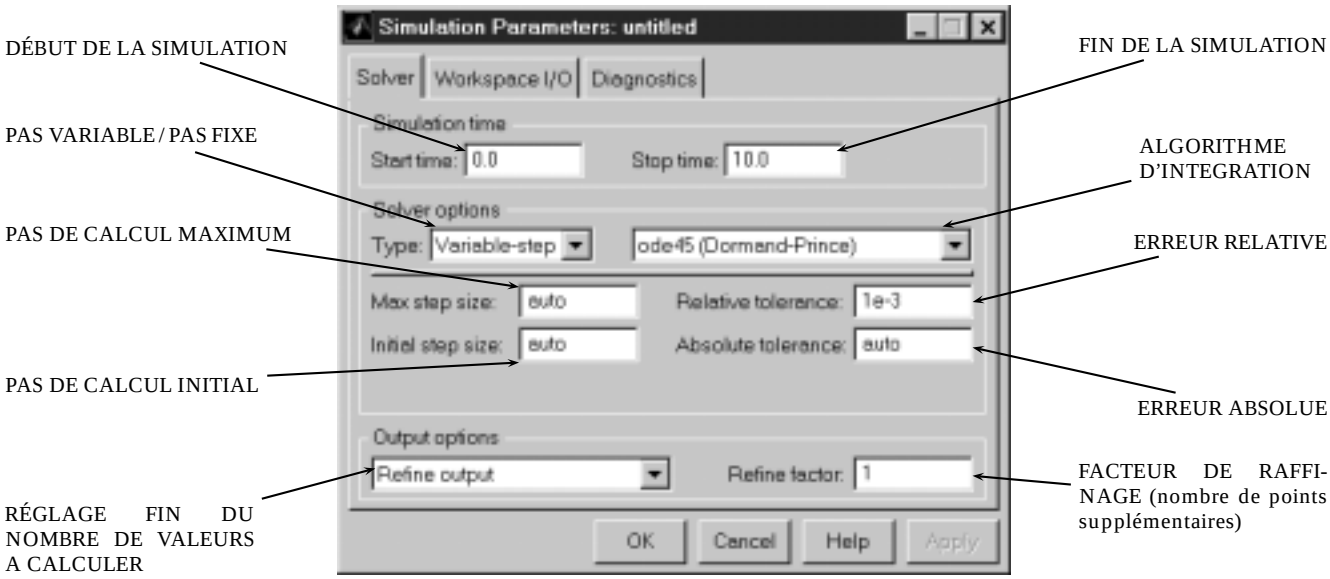
Dans le schéma principal, il faut se poser la question de comment gérer le passage des données de et vers l'espace de travail. Plusieurs solutions s'offrent : on peut utiliser des blocs des collections **`Sources`** et **`Sinks`** ou bien encore des portes d'entrée-sortie.

Simulation d'un diagramme SIMULINK

Une fois le modèle composé, il faut ajuster les paramètres de la simulation dans le menu `Simulation Parameters`.

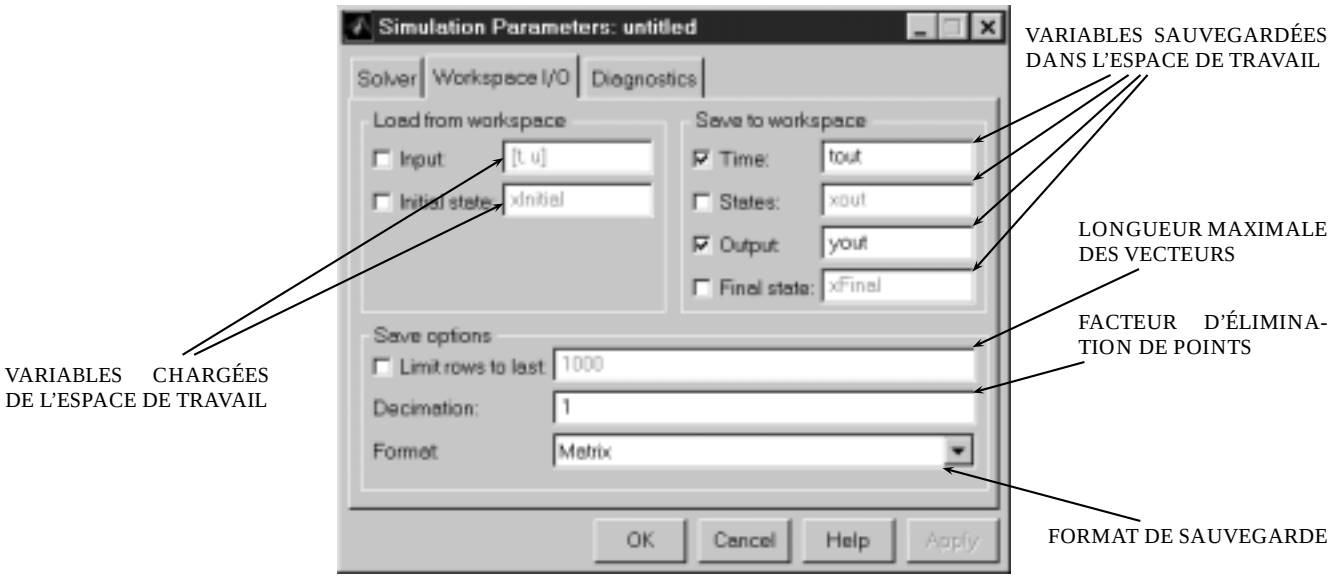
Paramètres du solveur

Pour faire des premiers essais on peut se contenter des paramètres proposés par défaut, en choisissant juste la durée désirée de la simulation. Cependant, ce menu a une influence énorme sur la qualité de la simulation pour des systèmes à peine compliqués et il faut apprendre à bien s'en servir.



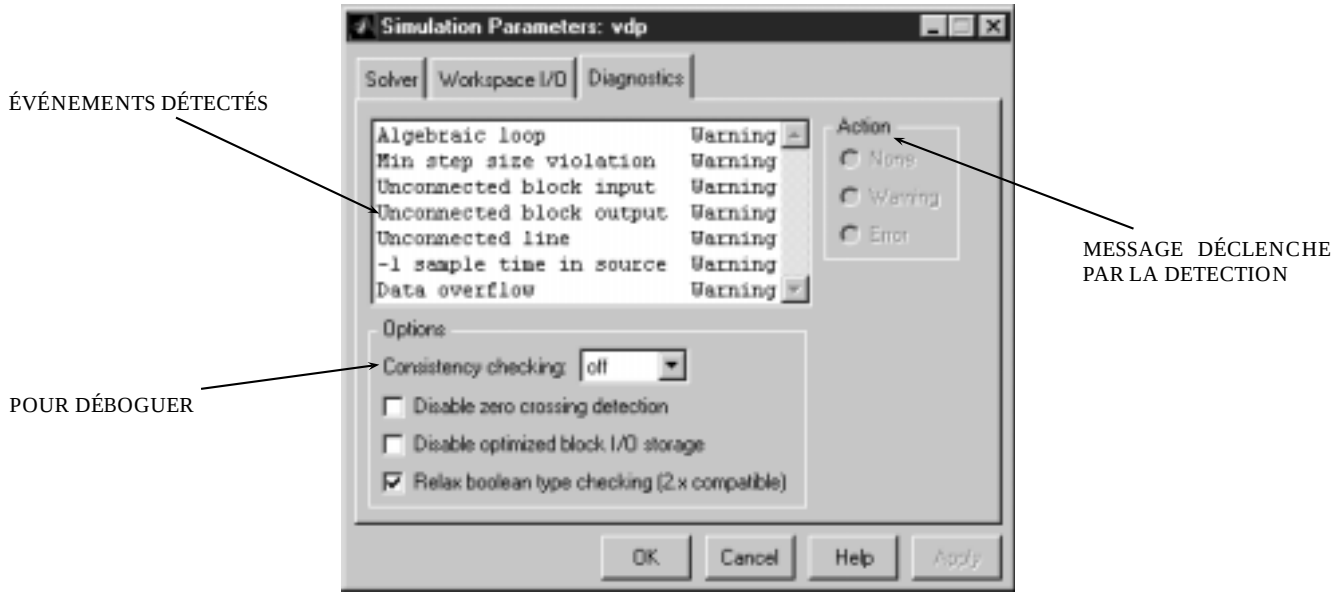
Paramètres d'entrée-sortie par rapport à l'espace de travail

Par défaut, le vecteur de temps de la simulation, qu'on utilise dans la plupart des graphiques, est enregistré dans la variable **`tout`**, qui sera disponible dans l'espace de travail. On peut aussi utiliser ce menu pour définir les entrées du modèle et pour en enregistrer les sorties, si on s'est servi de portes d'entrés-sortie dans le schéma principal.



Paramètres du diagnostic

Cette fenêtre permet de régler le diagnostic pour la simulation et fournit aussi des mécanismes pour déboguer le schéma.



Initialisation d'un modèle

Dans les schémas il est souvent préférable de rentrer des noms de variables, plutôt que des valeurs numériques. Pour que la simulation puisse avoir lieu, il faut que ces paramètres soit présents dans l'espace de travail. Normalement, on écrit les paramètres du système dans un fichier M d'initialisation, qu'il suffit d'appeler une fois par session de travail (à moins de vouloir changer les paramètres, évidemment).

La CONTROL SYSTEM TOOLBOX

Introduction

La CONTROL SYSTEM TOOLBOX (CST) est la « boîte à outils » MATLAB dédiée à l'Automatique de base. Elle fournit un support pour

- créer et manipuler des modèles linéaires de systèmes ;
- analyser ces modèles avec les outils classiques de l'Automatique ;
- synthétiser des régulateurs.

La liste complète des fonctions de cette *toolbox* s'obtient en tapant **`help control`**.

Représentation des systèmes linéaires

Dans la Control System Toolbox, les systèmes linéaires (à paramètres constants dans le temps) sont représentés par des *objets*[1] appartenant à la classe **`LTI`** (comme *Linear Time-Invariant*).

Trois différentes représentations des systèmes sont disponibles, à travers trois sous-classes de **`LTI`** :

- la classe **`tf`** correspond à des systèmes sous forme de *fonctions de transfert*, comme

$$\frac{s+2}{s^2+3s+1}e^{-2s},\quad \frac{0.7869z^{-1}-0.4773z^{-2}}{1-1.213z^{-1}+0.3679z^{-2}},\quad \left[\begin{array}{cc} \dfrac{-5}{s+1} & \dfrac{s-1}{s+2} \\ \dfrac{1}{s+3} & \dfrac{2}{s+2} \end{array}\right]$$

- la classe **`ss`** correspond à des systèmes sous forme de *variables d'état*, comme

$$\left\{\begin{array}{l} \left[\begin{array}{c}\dot{x}_1\\ \dot{x}_2\end{array}\right] = \left[\begin{array}{cc}-1 & 1\\ 0 & -2\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] + \left[\begin{array}{c}0\\ 2\end{array}\right]u \\ \qquad y = \left[\begin{array}{cc}1 & 1\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] \end{array}\right.,\quad \left\{\begin{array}{l} \left[\begin{array}{c}\Delta x_1\\ \Delta x_2\end{array}\right] = \left[\begin{array}{cc}-0.5 & 0\\ 0 & -0.8\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] + \left[\begin{array}{cc}0 & 1\\ 2 & 1\end{array}\right]\left[\begin{array}{c}u_1\\ u_2\end{array}\right] \\ \left[\begin{array}{c}y_1\\ y_2\end{array}\right] = \left[\begin{array}{cc}1 & 1\\ 1 & 0\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] \end{array}\right.$$

- la classe **`zpk`**, moins utilisée que les autres, correspond à des systèmes sous la forme *zéros-pôles-gain*, comme

$$100\frac{(s-1)(s+2)}{s(s+1)(s+3)},\quad \left[\begin{array}{cc} -5 & 1 \end{array}\right]\left[\begin{array}{c} \dfrac{(s+3)(s+5)}{(s-1)(s+1)} \\ \dfrac{s+2}{(s-1)(s+1)} \end{array}\right]$$

[1] La plupart des fonctions de la CST peuvent s'appliquer à des vecteurs et à des matrices sans passer forcément par les objets LTI, comme dans les anciennes versions de cette *toolbox* (utiliser **`help`** pour plus de details)

Une classe ultérieure (**FRD**) a été mise à disposition dans la version 5.3 pour représenter des réponses en fréquence, qui peuvent aussi provenir de données expérimentales.

Fonctions de transfert (classe `tf`)

Les fonctions de transfert se créent à l'aide de la fonction **`tf`**. Les premièrs deux paramètres de **`tf`** sont des vecteurs MATLAB contenant respectivement les coefficients du numérateurs et les coefficients du dénominateur, tandis que les paramètres suivants (facultatifs) spécifient des propriétés ultérieures de la fonction de transfert :

`tf(num,den)`	créer une f.d.t. continue $\frac{num}{den}$;
`tf(num,den,'Td',tr)`	créer une f.d.t. continue avec retard **`tr`** ;
`tf(num,den,te)`	créer une f.d.t. en z avec période d'échantillonnage **`te`** ;
`tf(num,den,te,'Variable','z^-1')`	créer une f.d.t. en z^{-1}

Exemples

```
» nc = [1 1];
» dc = [1 2 1];
» Gc = tf(nc,dc)
Transfer function:
    s + 1
-------------
s^2 + 2 s + 1


» nd = [1 0.985];
» dd = = conv([1 -1],[1 -0.999]);
» ts = 0.03;
» Gd = tf(nd,dd,ts)
Transfer function:
      z + 0.985
---------------------
z^2 - 1.999z + 0.999
Sampling time:  0.03
```

La fonction **`conv(a,b)`** utilisée ci-dessus permet d'obtenir la multiplication polynomiale entre les polynômes dont les coefficients sont définis par les vecteurs **`a`** et **`b`**.

La fonction **`tfdata`** permet de récupérer (entre autres) le numérateur et le dénominateur d'une fonction de transfert :

`[num,den]=tfdata(fdt,'v')`	extraire le numérateur et le dénominateur ;
`[num,den,te]=tfdata(fdt,'v')`	extraire aussi la période d'échantillonnage ;
`[num,den,te,retard]=tfdata(fdt,'v')`	extraire encore le retard.

Exemple

```
» [num,den,ts]=tfdata(Gd,'v');
num =
       0   1.0000   0.9850
```

```
den =
       1.0000   -1.9990   0.9990

ts =
       0.03
```

Forme zéros-pôles-gain (classe `zpk`)

La forme zéros-pôles-gain est une représentation alternative à la fonction de transfert $\frac{num}{den}$. On crée des systèmes dans cette forme à l'aide de la fonction **`zpk`**, qui prend en paramètres deux vecteurs contenant respectivement les zéros et les pôles, plus un scalaire qui indique un gain

```
» zeros=[];
» poles=[-1,-2];
» gain=-10;
» G_zpk=zpk(zeros,poles,gain)
Zero/pole/gain:
   -10
------------
(s+1) (s+2)
```

La fonction **`zpkdata`** a un rôle analogue à celui de **`tfdata`**, considérée précédemment.

Représentation d'état (classe `ss`)

Les systèmes en variables d'état se créent à l'aide de la fonction **`ss`**, en spécifiant les quatre matrices (ou vecteurs) de la représentation $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$:

`ss(A,B,C,D)`	créer un système continu en variables d'état ;
`ss(A,B,C,D,te)`	créer un système discret avec période d'échantillonnage **`te`**.

Exemple

```
» A=[0,1;-4,-4];B=[0;2];C=[1,0];D=0;
» syst_ss=ss(A,B,C,D);
a =
                    x1           x2
         x1          0            1
         x2         -4           -2
b =
                    u1
         x1          0
         x2          2
c =
                    x1           x2
         y1          1            0
a =
                    u1
         y1          0
Continuous-time system.
```

On peut extraire le quadruplé A, B, C, D d'un objet **ss** par la fonction **ssdata**

```
» [A1,B1,C1,D1]=ssdata(syst_ss)
A1 =
     0    1
    -4   -4

B1 =
     0
     2

C1 =
     1   0


D1 =
     0
```

[A,B,C,D,te]=ssdata récupère aussi la période d'échantillonnage ($= 0$ pour un système continu).

Passage d'une représentation à l'autre

Soit **syst** un objet LTI appartenant à l'une des trois classes, introduites ci-dessus. Alors :

ss(syst)	met le système **syst** en représentation d'état ;
tf(syst)	met le système **syst** sous forme de fonction de transfert ;
zpk(syst)	met le système **syst** sous forme zéros-pôles-gain.

Exemple

```
» syst_tf = tf(syst_ss);
Transfer function:
   2
-------------
s^2 + 4 s + 4
» syst_zpk = zpk(syst_tf);
Zero/pole/gain:
   2
-------
(s+2)^2
```

Les fonctions **tfdata**, **zpkdata**, **ssdata** peuvent en fait s'appliquer à n'importe quel objet LTI, car elles se chargent aussi de la conversion de représentation.

Conversion continu⇒discret

Un modèle LTI se discrétise au moyen de la fonction **c2d**, avec la syntaxe

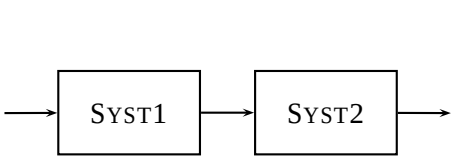
syst_disc = c2d(syst_cont, te, methode),

où **te** est la période d'échantillonnage et **methode** est la méthode de discrétisation à utiliser : bloqueur d'ordre 0 (**'zoh'**), bloqueur d'ordre 1 (**'foh'**), transformation bi-linéaire de Tustin (**'tustin'**), etc. Si l'on ne spécifie pas de méthode, le bloqueur d'ordre 0 est utilisé.

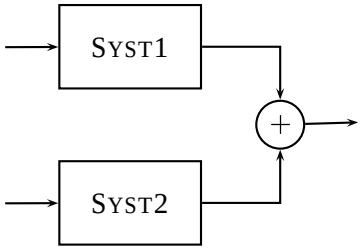
```
» syst_tf_z=c2d(syst_tf,0.1)
Transfer function:
0.008762 z + 0.007668
---------------------
z^2 - 1.637 z + 0.6703
Sampling time:  0.1
```

Manipulation de schémas en blocs

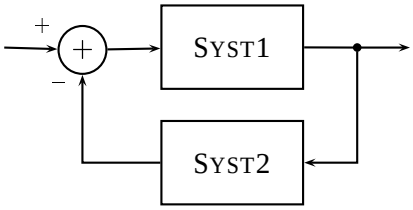
Les systèmes LTI peuvent s'interconnecter pour former des systèmes plus grands



Syst2∗Syst1
series(Syst1,Syst2)



Syst2+Syst1
parallel(Syst1,Syst2)



feedback(Syst1,Syst2)

Les fonctions **series**, **parallel**, **feedback** possèdent des options pour obtenir des systèmes plus complexes (**help** sur les noms de ces fonctions pour plus de détails). Toutefois, à partir d'un certain degré de complexité il faut prendre en considération l'option de dessiner les blocs dans un schéma SIMULINK. **TRUC :** la fonction **linmod**, conçue pour linéariser un modèle (non linéaire) SIMULINK, peut être aussi utilisée pour récupérer la représentation d'état globale d'un ensemble de blocs *linéaires* connectés.

Propriétés des modèles LTI

Pour une utilisation plus avancée il peut être utile de connaître les *propriétés* les plus importantes que les objets MATLAB représentant des modèles LTI possèdent

PROPRIÉTÉS COMMUNES À TOUS LES OBJETS LTI					
Ts	**Td**	**InputName**	**OutputName**	**Notes**	**Userdata**
période d'éch.	retard(s)	noms des entrées	noms des sorties	notes perso	autres données / info

PROPRIÉTÉS **TF**		PROPRIÉTÉS **ZPK**			PROPRIÉTÉS **SS**			
num	**den**	**z**	**p**	**k**	**a**	**b**	**c**	**d**
numérateur	dénominateur	zéros	pôles	gain	matrice A	matrice B	matrice C	matrice D
Variable					**StateName**			
nom de la variable (**'s'**, **'z'**, **'z^-1'**, etc.)					noms des variables d'état			

On peut lire ou modifier ces propriétés très facilement. Soit **syst** un objet TF discret :

syst.Ts=nouveauTe change la période d'échantillonnage (sans changer les coefficients) ;
n=syst.num{1} récupère le numérateur sous forme de vecteur ;
syst.num=nouveauNum change le numérateur ;

D'une façon analogue

zeros=syst.z{1}	récupère les zéros sous forme de vecteur ;
A=syst.a	récupère la matrice **A** ;

Une partie des propriétés n'a pas de signification mathématique, mais sert à mieux documenter le modèle

```
» G.InputName='tension (V)';G.OutputName='vitesse (rad/s)'
Transfer function from input "tension (V)" to output "vitesse (rad/s)":
   1
-------
s^2 + s
```

Outils d'analyse

Les fonctions mentionnées dans la suite sont applicables indifféremment à des objets **tf**, **zpk** et **ss**. Vue la flexibilité de ces fonctions, l'utilisation de l'aide est chaleureusement recommandée, pour en explorer toutes les possibilités.

Analyse temporelle

La fonction **step** permet de tracer la réponse indicielle d'un système

step(syst)	tracer la réponse indicielle, intervalle de temps et nombre de points choisis par MATLAB ;
step(syst,tfin)	**tfin** scalaire détermine le temps final (continu) ou le nombre total d'échantillons (discret) ;
step(syst,T)	**T** vecteur de temps fourni par l'utilisateur ;
step(syst1,syst2,...)	**T** tracer les réponses indicielles superposées de plusieurs systèmes ;
[y,t]=step(syst,...)	stocke la réponse indicielle sans la tracer.

Exemple

```
» T=0:0.1:10;
» step(syst_tf,T)
```

La réponse impulsionnelle s'obtient par la fonction **impulse**, dont la syntaxe est identique à **step**. Pour la réponse initiale d'un système en variables d'états on utilise la fonction **initial(sys,x0)** avec des options similaires à celles vue pour les fonctions ci-dessus.

Pour obtenir des réponses à des signaux d'entrées plus complexe on a deux choix :

– utiliser la fonction **lsim** avec un signal généré par **gensig**, **square** ou **sawtooth** ;
– utiliser SIMULINK et le « blockset » LTI.

Pôles, zéros, valeurs propres, ...

`[vp]=eig(syst)`	calculer les valeurs propres de **`syst`** ;
`p=pole(syst)`	calculer des pôles de **`syst`** ;
`z=tzero(syst)`	calculer les zéros (de transmission, pour un système multi-variable) de **`syst`** ;
`[p,z]=pzmap(syst)`	calculer les pôles *et* les zéros de **`syst`** ;
`[w,csi]=damp(syst)`	calculer pulsation naturelle et amortissement des pôles de **`syst`** ;
`gs=dcgain(syst)`	calculer le gain statique de **`syst`**.

En tapant **`pzmap(syst)`** tout court, les pôles et le zéros du système sont tracés sur le plan s ou z. Des grilles correspondantes à des fréquences et des amortissements constants peuvent être tracés à l'aide de **`sgrid`** et **`zgrid`**, respectivement.

Analyse fréquentielle

La fonction **`bode`** permet de tracer le diagramme de Bode d'un système

`bode(syst)`	tracer le diagramme de Bode, intervalle de fréquences et nombre de points choisis par MATLAB ;
`bode(syst,wi,wf)`	tracer le diagramme de Bode entre les fréquences **`wi`** et **`wf`** ;
`bode(syst,W)`	tracer le diagramme de Bode avec un vecteur de fréquences généré par l'utilisateur (d'habitude avec **`logspace`**) ;
`bode(syst1,syst2,...)`	tracer les diagrammes de Bode indicielles superposés de plusieurs systèmes ;
`[G,P,W]=bode(syst,...)`	stocke le diagramme de Bode (gain, phase et fréquence) sans le tracer.

Une syntaxe similaire est prévue pour les fonctions **`nyquist`** et **`nichols`**. La grille pour le diagramme de Nichols se dessine avec **`ngrid`**.

Pour un système stable, la fonction

`[mg,mp,wmg,wmp]=margin(syst)`,

calcule le marge de gain et phase, aussi bien que les fréquences qui leur sont associées.

Le *LTIVIEWER*

En tapant **`ltiview`** ou **`ltiview(syst)`**, on rentre dans un environnement graphique interactif pour l'analyse des modèles LTI, qui est très performant, mis à part des bogues et une tendance fâcheuse à se planter, qu'on espère voir disparaître dans une prochaine version du logiciel.

Normes

`norm(sys,2)`	norme H_2
`norm(sys,inf)`	norme H_∞

Analyse et manipulation des systèmes en variables d'état

La plupart de fonctions suivantes n'ont du sens que si le système considéré est en forme d'état :

syst=ss2ss(syst,T)	effectuer la transformation $\mathbf{z} = \mathbf{T}\mathbf{x}$ sur le vecteur d'état $\mathbf{x}$ de **syst**
systc=canon(syst,type)	calculer la forme compagnon de **syst** (si **type** égal à **'companion'**)
CO=ctrb(syst)	calculer la matrice de commandabilité de **syst**
OB=obsv(syst)	calculer la matrice d'observabilité de **syst**
asys=augstate(sys))	ajouter les états de **syst** à ses sorties.

Pour mieux comprendre (et contrôler) ce que ces fonctions font, il est probablement plus raisonnable de les utiliser sur les matrices et / ou les vecteurs de la représentation d'état, plutôt que sur l'objet **ss** :

CO=ctrb(syst.A,syst.B)	calculer la matrice de commandabilité
OB=obsv(syst.A,syst.C)	calculer la matrice d'observabilité

Outils de synthèse

Pour un système mono-variable, il est possible de tracer le *lieu de racines* par la fonction **rlocus(syst)**. La fonction **rlocfind(syst)** permet de déterminer le gain sur lieu de racines de façon interactive. Un environnement graphique complet pour la synthèse par cette méthode est accessible en tapant **rltool(syst)**.

Le placement de pôles en variables d'état s'obtient par les fonctions **acker** et **place**, cette dernière étant mieux conditionnée et applicable aussi aux systèmes multi-variables. Les deux fonctions sont à appliquer à des matrices et non pas à des objets LTI.

La fonction **estim(syst,L)** permet de former un observateur à partir d'un gain $\mathbf{L}$, et la fonction **estim(syst,K,L)** permet de créer un ensemble régulateur-observateur à partis des gains $\mathbf{K}$ et $\mathbf{L}$.

Autres outils

D'autres outils de synthèse sont disponibles :

- pour la commande LQR (**lqr, dlqr, lqry, lqrd**);
- pour la commande LQG (**kalman, kalmd, lqgreg**);
- pour la résolution d'équations de Lyapounov (**lyap, dlyap**);
- pour la résolution d'équations de Riccati (**care, dare**).