Haute Ecole d'Ingéniérie et de Gestion du canton de Vaud (HEIG-VD)

Département des Technologies Industrielles (TIN)

Filière Microtechniques (MI)

Filière Systèmes industriels (SI)

Filière Génie électrique (GE)

Introduction au logiciel MATLAB

heig-vd Institut d'Automatisation Industrielle

Prof. Michel ETIQUE, mars 2012,
Yverdon-les-Bains

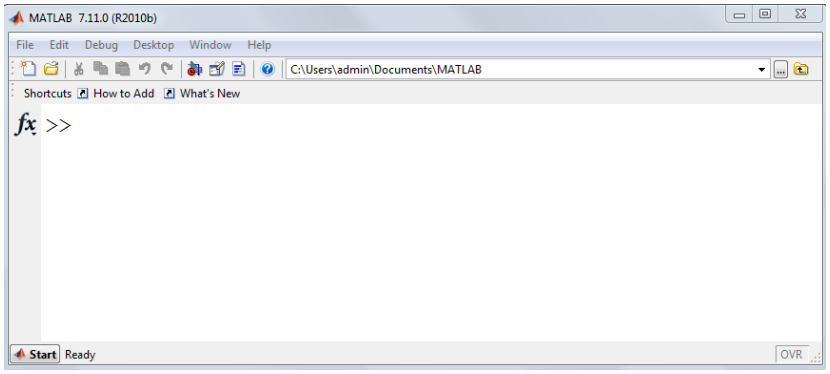
Table des matières

1 Introduction

Le présent document a pour but de fournir les bases nécessaires à l'utilisation du logiciel MATLAB, dans le cadre des cours, des exercices et du laboratoire de régulation automatique, pour lequel les boîtes à outils *Control System* et Simulink sont employées. Il ne présente donc ce logiciel que sous l'angle de son utilisation en régulation automatique.

Le logiciel MATLAB (*MATrix LABoratory*) est spécialisé dans le domaine du calcul matriciel numérique. Tous les objets définis dans MATLAB le sont donc au moyen de vecteurs et de matrices/tableaux de nombres. Un ensemble important d'opérateurs et de fonctions MATLAB de base facilitent leur manipulation et des opérations comme par exemple le produit et l'inversion matricielles (inv), la transposition (') ou encore le calcul des valeurs propres (eig) font partie de la bibliothèque standard. D'autres fonctions servant à la création et à la manipulation de matrices et de tableaux (diag, fliplr, flipud, rot90, rand, ones, zeros, linspace, logspace) sont également disponibles en nombre.

L'environnement MATLAB se présente sous la forme d'un espace de travail (*Workspace*), où un interpréteur de commandes exécute des opérations et fonctions MATLAB. Les sources de celles-ci sont disponibles, écrites en "langage" MATLAB, voire en C ou en Fortran. L'utilisateur peut à sa guise les modifier, mais en s'en inspirant, il peut surtout créer et rajouter ses propres fonctions.



MATLAB offre également plusieurs fonctions destinées à la résolution (numérique) d'équations différentielles linéaires ou non-linéaires, notamment par la méthode de Runge-Kutta (ode23 et ode45), l'intégration numérique (trapz, quad et quad8), la recherche des solutions d'équations algébriques (roots) ou transcendantes (fzero), la création et manipulation de polynômes (poly, polyder, polyval, conv, deconv), la transformée de Fourier rapide (fft, fft2, ifft).

Des fonctions propres au traitement de données (expérimentales, telles que celles obtenues au laboratoire), comme min, max, mean, cumsum, sort, std, diff, ainsi que celles relatives à l'interpolation (polyfit, interp1) sont autant d'outils très pratiques pour l'ingénieur analysant un problème pratique ou théorique.

L'interface graphique de MATLAB est sans conteste l'un des points forts du logiciel et facilite le tracé de courbes et l'obtention de graphiques 2D ou 3D de grande qualité (plot, stairs, stem, hist, mesh, surf, plot3). Le module *Handle Graphics* offre la possibilité de contrôler intégralement cette interface, permettant ainsi à l'utilisateur de mettre en forme tous éléments d'un graphique, de créer ses propres menus (uimenu) ainsi que des objets graphiques tels que sliders (ascenseurs), boutons, menu surgissants (uicontrol) avec une facilité déconcertante.

Le "langage" MATLAB contient un minimum de structures de programmation (structure itérative, structure conditionnelle, sous-routine) mais reste très rudimentaire. L'avantage est qu'il est très simple et très rapide à programmer, offrant une grande tolérance (syntaxe simple, pas de définition obligatoire de types, etc), ce qui permet un gain appréciable en temps de mise au point. L'ingénieur peut par ce moyen être plus efficace dans l'analyse d'un problème, en concentrant ses efforts sur celui-ci et non pas sur l'outil servant à le résoudre. En revanche, MATLAB ne convient pas à la programmation d'applications d'une certaine ampleur. Dans ce dernier cas, il est possible, sous certaines conditions, de programmer l'application en C et de l'exécuter à partir l'espace de travail MATLAB.

Au logiciel de base s'ajoutent, selon la configuration choisie, les fonctions provenant d'une série de boîtes à outils (*toolbox*) dédiés à des domaines techniques spécifiques, comme

- le traitement de signal (*signal processing toolbox*),
- la régulation automatique (*control system toolbox*),
- l'identification (*system identification toolbox*),
- les réseaux de neurones (*neural networks toolbox*),
- la logique floue (*fuzzy logic toolbox*),
- le calcul symbolique *(symbolic math toolbox)*,

et bien d'autres encore. Ces boîtes à outils sont simplement constituées d'un ensemble de fonctions spécialisées programmées à partir des fonctions de base de MATLAB, permettant par exemple la synthèse de filtres, le calcul de FFTs, la simulation d'algorithmes flous ou encore le calcul de réponses harmoniques.

Simulink n'est rien d'autre qu'une boîte à outils de MATLAB permettant au moyen d'une interface graphique évoluée la construction rapide et aisée ainsi que la simulation de schémas fonctionnels complexes, contenant des systèmes linéaires, non linéaires voire non-stationnaires, y compris des opérateurs logiques, des outils mathématiques d'analyse, etc.

Dans le cadre de la régulation automatique, MATLAB constitue avant tout un très puissant outil d'analyse des systèmes dynamiques linéaires, dont on peut très facilement obtenir les propriétés, comme les pôles et zéros (tf2zp, roots) ou le gain statique (dcgain) et tracer les réponses de tous types, la réponse impulsionnelle (impulse), indicielle (step), à un signal quelconque (lsim), ou harmonique (bode, nyquist). MATLAB se prête aussi bien à l'analyse des systèmes analogiques ou numériques (dcgain et ddcgain, step et dstep, impulse et dimpulse, bode et dbode, lsim et dlsim), sans qu'il soit cependant possible d'étudier des systèmes mixtes

analogiques-numériques (il faut pour cela faire usage de la boîte à outils Simulink. A noter que SysQuake permet d'analyser des systèmes mixtes analogiques numériques). Différentes commandes permettent de construire et d'analyser des systèmes linéaires de structure complexe, formés de multiples blocs connectés tantôt en série, parallèle ou étant contre-réactionnés (feedback, parallel, append, connect, blkbuild).

Pour le traitement aisé de systèmes non-linéaires, de même que pour la construction de schémas fonctionnels complexes, on aura à l'évidence intérêt à utiliser la boîte à outils Simulink, qui constitue alors un outil idéal.

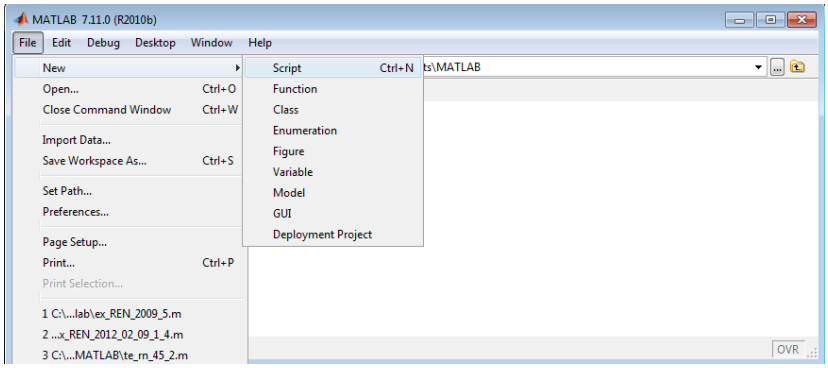
Incontestablement, MATLAB est un formidable outil pour l'ingénieur, y compris pour celui traitant des problèmes pratiques. Avec sa boîte à outils Simulink, il est une référence au niveau mondial, non seulement dans les universités et instituts de recherche, mais aussi dans le milieu industriel.

2 Méthode de travail

2.1 Edition et sauvegarde des fichiers MATLAB

2.1.1 Editeur intégré à MATLAB

Dans un premier temps, on peut se contenter d'introduire ses commandes une à une au niveau de l'espace de travail où elles sont interprétées directement. Cependant, par la suite, il est beaucoup plus pratique d'écrire sa séquence de commandes complète au moyen d'un éditeur, puis de sauvegarder le tout dans un fichier avec l'extension .M. Cette séquence pourra alors être exécutée dans MATLAB par simple introduction du nom du fichier. MATLAB possède un éditeur intégré que l'on appelle sélectionnant File-New-Script.



De cette façon, l'utilisateur peut rajouter ses propres fonctions et se créer rapidement et facilement toute une panoplie d'outils (par exemple, "synthèse automatique d'un régulateur PID par la méthode de Bode"). Les sources de MATLAB étant disponibles pour la plupart, ce n'est pas un problème que de modifier à sa guise l'une ou l'autre des routines, par exemple dans de but d'obtenir un tracé particulier du lieu de Bode.

2.2 Aide en ligne

En plus de l'aide de Windows, une aide en ligne est disponible pour chaque commande de MATLAB. Il suffit d'introduire :

help nom_de_commande

2.3 Création de fichiers de commandes et de fonctions utilisateur

2.3.1 Fichiers de commande ("script files")

Un fichier de commande (*script file*) est un fichier ASCII d'extension .M contenant une suite de commandes MATLAB. Il être exécuté directement en tapant

simplement son nom dans l'espace de travail MATLAB.

2.3.2 Fonctions

De nouvelles fonctions peuvent être ajoutées à MATLAB par l'utilisateur. Il suffit de créer un fichier de nom

nom_de_fonction.M

contenant les commandes à exécuter et dont l'entête a le format :

function [liste des arguments de sortie] = nom_de_fonction(liste des arguments d'entrée)

Exemple La fonction suivante convertit la grandeur d'entrée x en débibels et la retourne dans y.

```
function [y] = lin2db(x)
y = 20*log10(x);
```

Le fichier correspondant est sauvegardé sous le nom lin2db.m, dans un répertoire figurant dans le chemin de recherche de MATLAB.

Contrairement aux fichiers de commande, les variables intervenant dans les fonctions sont locales.

Les commentaires documentant les fonctions peuvent être insérés en les faisant précéder du symbole %.

Détail pratique intéressant, les premières lignes d'un fichier

nom_de_fichier.m

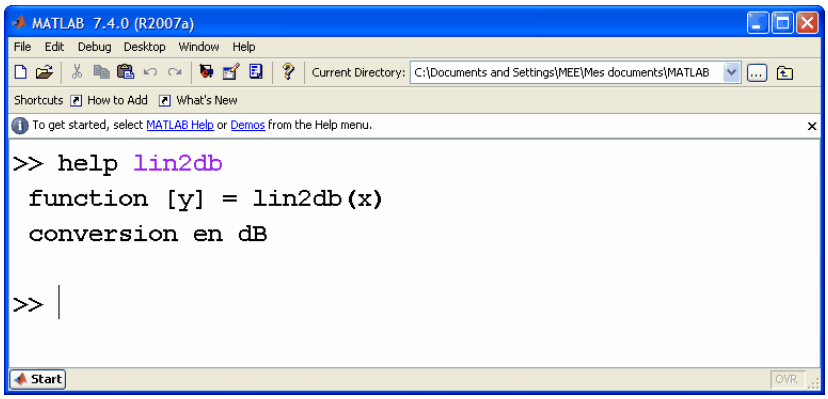
précédées du symbole % sont affichées lorsque l'on tape :

help nom_de_fichier

Ceci permet de fournir une aide en ligne pour chaque fonction utilisateur nouvellement créée. Pour la fonction lin2db, on documentera par exemple comme suit :

```
function [y] = lin2db(x)
%function [y] = lin2db(x)
%conversion en dB
y = 20*log10(x);
```

En tapant help lin2db, on obtiendra alors :



2.4 Sauvegarde de données sur disque

Les variables définies dans l'espace de travail MATLAB peuvent être sauvegardées dans des fichiers ASCII par la commande :

```
save chemin\nom_de_fichier.dat nom_de_variable -ascii
```

Un tel fichier ASCII peut être ultérieurement relu soit par MATLAB (fonction load) soit par d'autres programmes (Excel, Word, etc). Dans le premier cas, il est cependant conseillé de sauvegarder directement la variable dans un fichier binaire *.MAT de format spécifique à MATLAB :

```
save chemin\nom_de_fichier nom_de_variable
```

La fenêtre graphique peut être soit imprimée directement, soit sauvée dans un fichier, lequel peut être intégré dans un document (menu importation d'image) ou imprimé ultérieurement. Signalons, parmi les différents formats de fichiers disponibles, le format PostScript qui fournit une qualité irréprochable. Par exemple, le sauvetage de la fenêtre graphique courante s'effectue par :

```
print chemin\nom_de_fichier.eps -deps
```

pour le format PostScript et par

```
print chemin\nom_de_fichier.wmf -dmeta
```

pour le format Windows Meta File. Auparavant, il faut prendre garde à ce que la fenêtre graphique courante soit bien celle que l'on souhaite imprimer en tapant

```
figure(numero_de_la_fenetre_a_imprimer)
```

Attention : la commande print est très rudimentaire et ne donne aucun message en cas d'échec ou si le fichier existe déjà !

La copie de la fenêtre graphique est également possible par l'intermédiaire du clipboard, ce qui permet par exemple de sauver la fenêtre dans Word ou PowerPoint (choisir le format EMF plutôt que le bitmap).

3 Les bases de MATLAB

3.1 Création et calcul de vecteurs et matrices

3.1.1 Vecteurs

Un vecteur-ligne est introduit de la façon suivante :

```
v=[5,2,13,4]
```

Le vecteur v s'affiche alors à l'écran :

```
v=5  2  13  4
```

Si l'introduction est terminée par un point-virgule, on évite l'affichage du vecteur v. Un vecteur-colonne peut être introduit en remplaçant les virgules par des points-virgules ou des retours de chariot. L'accès aux composantes d'un vecteur s'effectue directement par des commandes du genre :

```
v(3)
```

ce qui donne à l'écran :

```
ans =
13
```

Si comme dans le cas présent, v(3) n'est pas affecté à une variable, par une commande de la forme x=v(3) , MATLAB copie d'office le résultat dans la variable système ans (*answer*), alors disponible pour le calcul suivant. La première composante du vecteur a l'indice 1 :

```
v(1)
```

Remarque Dans MATLAB, les indices des vecteurs et matrices doivent être des entiers positifs. L'indice zéro n'est donc pas plus admis que les indices négatifs.

3.1.2 Matrices

Une matrice peut être construite de différentes manières :

```
m=[5,2,13,4;6,8,3,10;0,1,20,9]
```

et l'affichage devient :

```
m =
5  2  13  4
6  8  3  10
0  1  20  9
```

ou encore, ayant défini préalablement les vecteurs-ligne v1, v2 et v3 :

```
v1=[5,2,13,4];
v2=[6,8,3,10];
v3=[0,1,20,9];
m=[v1;v2;v3]
```

L'accès à un élément de la matrice s'effectue par :

```
m(2,4)
```

et l'on obtient :

```
ans =
10
```

Le remplacement de 2 par : (deux points) permet d'obtenir toute la colonne 4 :

```
m(:,4)
```

et l'écran affiche le vecteur-colonne :

```
ans =
4
10
9
```

De même, l'affichage d'une sous-matrice s'obtient par :

```
m(2:3,2:4)
```

et l'écran affiche :

```
ans =
8  3  10
1  20  9
```

L'accès aux colonnes 2 et 4 de la matrice m se réalise comme suit :

```
m(:,[2,4])
```

ce qui produit :

```
ans =
2  4
8  10
1  9
```

Parmi les opérations matricielles qui ont une certaine importance pratique, signalons l'opérateur de transposition

```
m'
```

donnant dans le cas de l'exemple

```
ans =
5  6  0
2  8  1
13  3  20
4  10  9
```

et la fonction eig qui calcule les valeurs propres d'une matrice. Soit la matrice carrée 4×4 formée de composantes aléatoires (ici utilisation de la fonction rand)

```
A=rand(4,4)
A =
0.2190  0.9347  0.0346  0.0077
0.0470  0.3835  0.0535  0.3834
0.6789  0.5194  0.5297  0.0668
0.6793  0.8310  0.6711  0.4175
```

Les valeurs propres (*eigenvalues*) sont :

```
eig(A)
ans =
1.4095
0.1082 + 0.4681i
0.1082 - 0.4681i
-0.0763
```

3.1.3 Construction de matrices, vecteurs et tableaux particulièrs

Les routines ones et zeros permettent de construire des matrices dont tous les éléments sont égaux à 1 respectivement à 0. Voir également eye (matrice identité), diag (matrice diagonale), linspace (vecteur dont les composantes sont espacées linéairement entre deux limites) et logspace (vecteur dont les composantes sont espacées logarithmiquement entre deux limites).

Exemple Génération d'un vecteur de pulsations $\omega \left[\frac{\text{rad}}{\text{s}}\right]$ de 100 composantes réparties logarithmiquement entre $0.01 = 10^{-2}$ et $10 \left[\frac{\text{rad}}{\text{s}}\right] = 10^{1} \left[\frac{\text{rad}}{\text{s}}\right]$:

```
omega = logspace(-2,1,100);
```

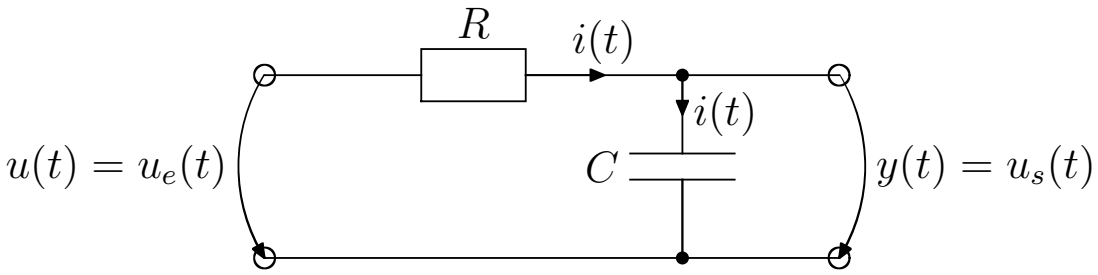


FIGURE 1 – Circuit RC (fichier source)

3.1.4 Nombres complexes

Les vecteurs et matrices peuvent avoir des composantes complexes. Les nombres complexes sont introduits comme suit :

```
x = a + j*b;
```

ou

```
x = a + i*b;
```

L'unité de l'axe imaginaire est donc indifféremment i ou j. Des fonctions sont prévues pour le calcul de la partie réelle (real(x)), de la partie imaginaire (imag(x)), du module (abs(x)), de l'argument (angle(x)) et du conjugué complexe (conj(x)).

Exemple Soit à calculer la réponse fréquentielle d'un filtre passe-bas d'ordre 1

$$G(j \cdot \omega) = \frac{Y(j \cdot \omega)}{U(j \cdot \omega)} = \frac{1}{1 + j \cdot \omega \cdot R \cdot C}$$

pour ω variant entre 10^{-2} $\left[\frac{\text{rad}}{\text{s}}\right]$ et 10 $\left[\frac{\text{rad}}{\text{s}}\right]$. Le code suivant effectue le calcul de $G(j \cdot \omega)$ ainsi que de ses module `AG` et phase `AG`

```
%vecteur de pulsations comprises entre 10^-2 et 10 rad/s
omega = logspace(-2,1,100);
R = 100e3;              %resistance
C = 10e-6;                      %capacite
G = 1./(1+j*omega*R*C);         %nombre complexe G
AG = abs(G);            %Gain de G en fonction de la pulsation
phiG = angle(G)*180/pi;%phase de G en fonction de la pulsation
```

3.2 Opérations sur les tableaux

On rappelle le slogan publicitaire de MATLAB, *"live is to short to spend time with for loops"*, pour mentionner qu'effectivement, la majeure partie des opéra-

tions peuvent s'effectuer sans boucles de type for. Par exemple, le calcul du sinus cardinal

$$\mathrm{sinc}\,(\theta) = \frac{\sin\,(\theta)}{\theta}$$

de l'angle θ pour les valeurs de suivantes

```
theta = linspace(-4*pi,4*pi,100);
```

soit 100 valeurs espacées linéairement entre $-4{\cdot}\pi$ et $+4{\cdot}\pi$, se fait avantageusement dans MATLAB par

```
sinc = sin(theta)./theta;
```

où le point précédant l'opérateur de division indique qu'il ne s'agit pas d'une opération matricielle, mais d'une opération **élément par élément** sur deux tableaux. Il faut donc bien faire la distinction entre par exemple le **produit matriciel**

```
A*B
```

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\cdot b_{11} + a_{12}\cdot b_{21} & a_{11}\cdot b_{12} + a_{12}\cdot b_{22} \\ a_{21}\cdot b_{11} + a_{22}\cdot b_{21} & a_{21}\cdot b_{12} + a_{22}\cdot b_{22} \end{bmatrix}$$

et le produit, élément par élément de deux tableaux :

```
A.*B
```

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\cdot b_{11} & a_{12}\cdot b_{12} \\ a_{21}\cdot b_{21} & a_{22}\cdot b_{22} \end{bmatrix}$$

Remarque Ici, la fonction linspace a été utilisée, une variante étant d'écrire

```
theta = -4*pi:8*pi/(100-1): 4*pi;
```

3.3 Représentation graphique 2D

Plusieurs types de représentations graphiques 2D sont disponibles :

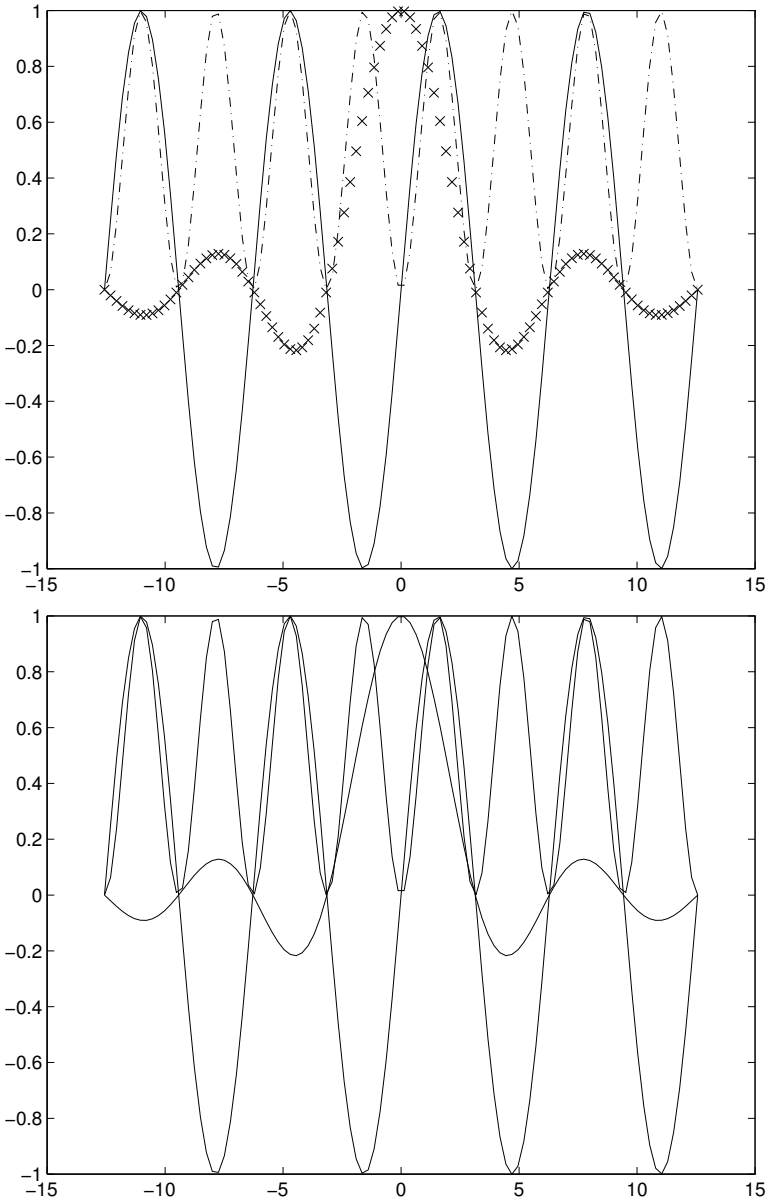
Fonction	Description
plot(y)	Représente chaque colonne de la matrice y en fonction de l'indice du rang.
plot(x,y) plot(x,cos(x))	Représente les **colonnes** de y en fonction de celles de x. Il faut que x ait soit une seule colonne, soit autant de colonnes que y, et réciproquement.
plot(t1,y1,t2,y2) plot(t1,y1,' :',t2,y2,'-.') plot(t1,y1,'o',t2,y2,'x')	Représente y1 en fonction de t1 et y2 en fonction de t2 (y1 et t1 doivent avoir le même nombre de lignes, de même que y2 et t2)
plot(t,[y1,y2])	Représente les **colonnes** y1 et y2 en fonction de t (y1, y2 et t doivent avoir le même nombre de lignes)
stairs(y)	Représente chaque élément de y en fonction de son indice. Le tracé est en escaliers, utile pour les réponses de systèmes numériques (pas d'interpolation entre les points représentés, mais plutôt une extrapolation d'ordre 0)
stem(y)	Représente chaque élément de y en fonction de son indice. Chaque point est représenté par symbole " échantillon ", ce qui est utile pour les systèmes numériques.
semilogx(w,20*log10(A))	Représente A (ici en [dB]) en fonction de w, l'échelle étant logarithmique. (A et w doivent avoir le même nombre d'éléments)
semilogx(w,phase)	Représente phase en fonction de w, l'échelle étant logarithmique. (phase et w doivent avoir le même nombre d'éléments)
loglog(x,y)	Représente y en fonction de x, les échelles étant toutes deux logarithmiques. (x et y doivent avoir le même nombre d'éléments)

Il est de plus recommandé de consulter la documentation MATLAB (ou l'aide en ligne) pour les commandes axis, hold, et subplot, title, xlabel, ylabel, grid, gtext ainsi que figure. Cette dernière instruction permet de spécifier le numéro de la fenêtre graphique dans laquelle le tracé doit être effectué. L'exécuter juste avant l'une des instructions de traçage active directement la fenêtre graphique qui apparaît alors à l'écran.

Exemple La suite des instructions ci-dessous calcule et trace le sinus, le sinus carré ainsi que le sinus cardinal de l'angle θ défini à l'aide de l'instruction linspace :

```
theta=linspace(-4*pi,4*pi,100);
sinc = sin(theta)./theta;
sinus_2 = sin(theta).^2;
figure(1),plot(theta,sin(theta),theta,sinus_2,'-.',theta,sinc,'x')
m = [sin(theta)',sinus_2',sinc'];
figure(2),plot(theta,m)
```

Le résultat est le suivant :



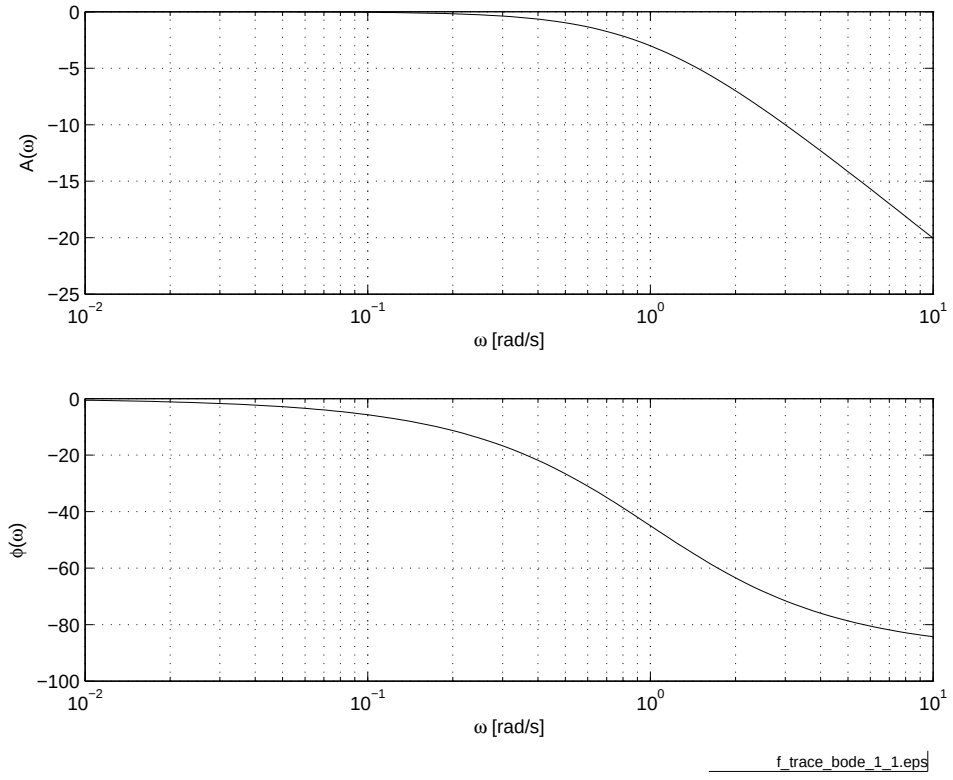
Exemple Le tracé du diagramme de Bode de la réponse harmonique de

$$G(j \cdot \omega) = \frac{Y(j \cdot \omega)}{U(j \cdot \omega)} = \frac{1}{1 + j \cdot \omega \cdot R \cdot C}$$

(§ 3.1.4 page 14) peut être fait, outre par la fonction bode, par la suite d'instructions :

```
figure
subplot(211)
semilogx(omega,20*log10(AG))
grid
xlabel('\omega [rad/s]')
ylabel('A(\omega)')
subplot(212)
semilogx(omega,phiG)
grid
xlabel('\omega [rad/s]')
ylabel('\phi(\omega)')
```

Le résultat est le suivant :



4 Analyse de systèmes dynamiques linéaires à l'aide de la boîte à outils control systems

4.1 Introduction de fonctions de transfert

4.1.1 Introduction sous forme polynômiale

L'introduction de fonctions de transfert s'effectue en deux temps, les numérateur et dénominateur devant être donnés séparément. Le principe est simple : les numérateur et dénominateur apparaissent sous forme de vecteurs-ligne, les composantes desquels étant les coefficients des puissances décroissantes de s (systèmes analogiques) ou de z (systèmes numériques).

Soit par exemple la fonction de transfert

$$G(s) = \frac{Y(s)}{U(s)} = 36 \cdot \frac{1 + s \cdot 10}{1 + s \cdot 2 + s^2 \cdot 0.1}$$

Son introduction dans MATLAB se fait par les commandes :

```
numG=36*[10,1];
denG=[0.1,2,1];
```

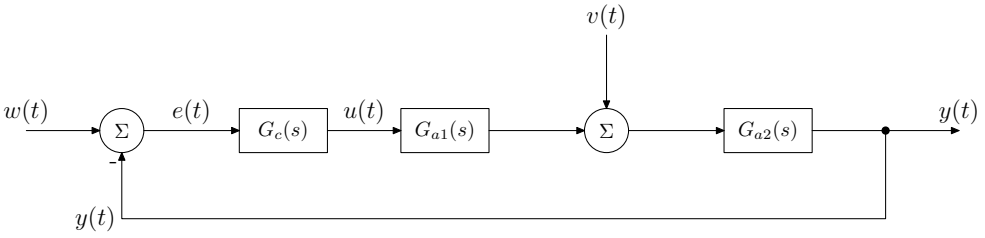
Le coefficient de s^0 (ou z^0 pour les systèmes numériques), **qu'il soit nul ou non**, doit toujours être donné. Ainsi, la fonction de transfert de l'intégrateur

$$G(s) = \frac{Y(s)}{U(s)} = \frac{2.71828}{s}$$

est introduite sous la forme :

```
numG=2.718280*[1];
denG=[1,0];
```

Les noms donnés aux numérateurs et dénominateurs sont libres. On aura toutefois intérêt à être organisé et rigoureux dans les notations employées ; par exemple, en se référant au schéma fonctionnel universel, en régulation de correspondance ($v(t) = 0$),



on ne saura que trop recommander l'emploi d'une notation semblable à la suivante

```
numGc=2*[1,20];
denGc=[1,0];
Gc=tf(numGc,denGc)
numGa1=1e-2*[1,10];
denGa1=[1,2,0.11];
Ga1=tf(numGa1,denGa1)
numGa2=36*[1,0.1];
denGa2=[1,20,10];
Ga2=tf(numGa2,denGa2)
```

pour les fonctions de transfert

$$G_c(s) = 2 \cdot \frac{s+20}{s}$$
$$G_{a1}(s) = 0.01 \cdot \frac{s+10}{s^2+2\cdot s+0.11}$$
$$G_{a2}(s) = 36 \cdot \frac{(s+0.1)}{s^2+20\cdot s+10}$$

On peut alors calculer sans autre :

```
Go=Gc*Ga1*Ga2;
```

4.1.2 Introduction sous forme de zéros, pôles et gain ("forme d'Evans")

Il est également possible d'introduire une fonction de transfert par le biais de ses zéros, pôles et de son facteur d'Evans k. Les zéros et pôles doivent apparaître sous forme de vecteurs-colonne. Soit par exemple la fonction de transfert :

$$G(s) = \frac{5.32}{s} \cdot \frac{\left((s+3)^2+7^2\right)}{(s+11)\cdot\left((s+5)^2+3^2\right)}$$
$$= \frac{5.32}{s} \cdot \frac{(s-(-3+7j))\cdot(s-(-3-7j))}{(s-(-11))\cdot(s-(-5+3j))\cdot(s-(-5-3j))}$$

La suite de commandes nécessaires est simplement :

```
zG=[-3+j*7,-3-j*7]';
sG=[0,-10,-5+j*3,-5-j*3]';
kG=5.32;
G = zpk(zG,sG,kG);
```

Il va sans dire que l'on privilégiera cette forme lorsque les numérateur et dénominateur de la fonction de transfert du système $G(s)$ ou $G(z)$ ne sont pas directement disponibles sous forme de polynômes.

4.2 Introduction de modèles d'état

Si un système est connu sous la forme de ses équations d'état (linéaires),

$$\frac{d\vec{x}}{dt} = A \cdot \vec{x} + B \cdot \vec{u}$$
$$\vec{y} = C \cdot \vec{x} + D \cdot \vec{u}$$

il suffit d'indiquer à MATLAB les quatre matrices A, B, C, et D, par exemple :

```
A=[-3,0;0,7];
B=[10,0,1]';
C=[1,2;3,4];
D=[0];
```

4.3 Passage d'un modèle à l'autre

Les pôles et zéros des fonctions de transfert sont obtenus à l'aide de la fonction tf2zp (*transfer function to zero pole*) :

```
[zG,sG,kG]=tf2zp(numG,denG);
```

MATLAB place alors les zéros et les pôles dans les vecteurs-colonne zG et sG, respectivement, et le facteur d'Evans dans la variable kG. Les noms zG, sG et kG sont arbitraires. On accède à ces valeurs par exemple en tapant leur nom :

```
sG
```

MATLAB affiche :

```
sG=
-19.4868
-0.5132
```

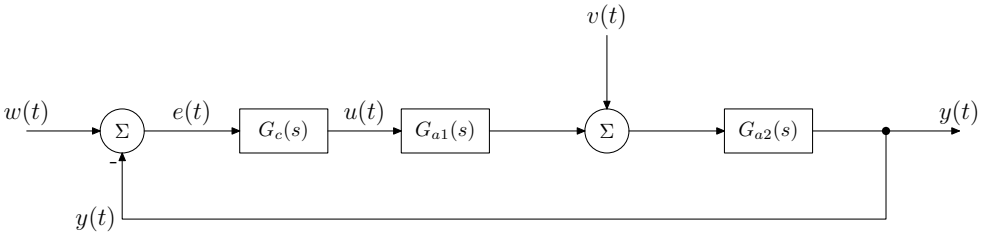
Remarque : un polynôme est introduit sous la forme d'un vecteur-ligne v dont les composantes représentent les coefficients des puissances décroissantes ; ses racines sont obtenues par la fonction roots(v)

Toute une catégorie de routines permettent de passer d'un modèle à l'autre.

4.4 Construction de schémas fonctionnels

4.4.1 Fonction multiplication et feedback

Prenons pour exemple le schéma fonctionnel universel, dont on souhaite obtenir les fonctions de transfert en boucle ouverte $G_o(s)$ et en boucle fermée $G_w(s)$. Pour ce faire, faut procéder par étapes, comme l'indique la figure ci-après :



1. Calcul de G_a(s) par la mise en série de $G_{a1}(s)$ et $G_{a2}(s)$.
   On fait usage de la fonction multiplication :

```
Ga = Ga1*Ga2;
```

2. Calcul de $G_o(s)$ par la mise en série de $G_c(s)$ et $G_a(s)$. On procède de même :

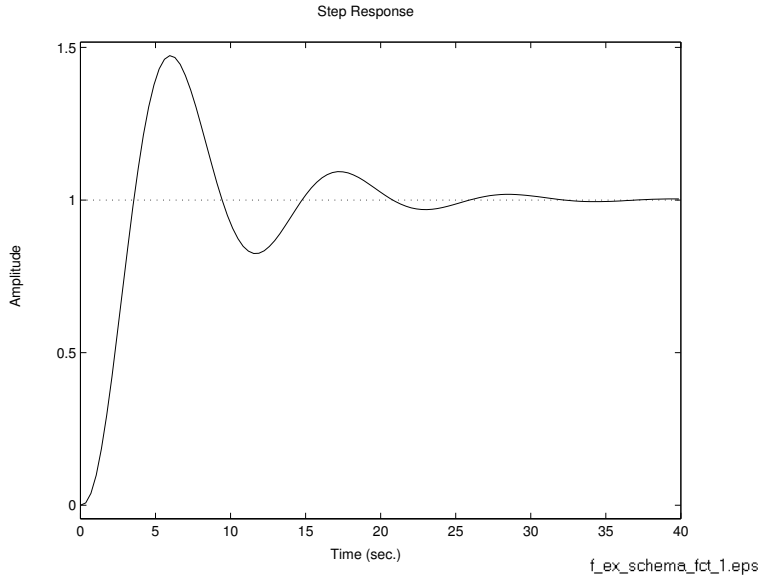
```
Go = Gc*Ga;
```

3. Calcul de $G_{yw}(s)$ par fermeture de la boucle (retour unitaire). On fait usage de la fonction feedback comme suit :

```
Gyw = feedback(Go,1)
```

4. La réponse indicielle en boucle fermée peut alors être calculée et tracée par :

```
step(Gyw)
```

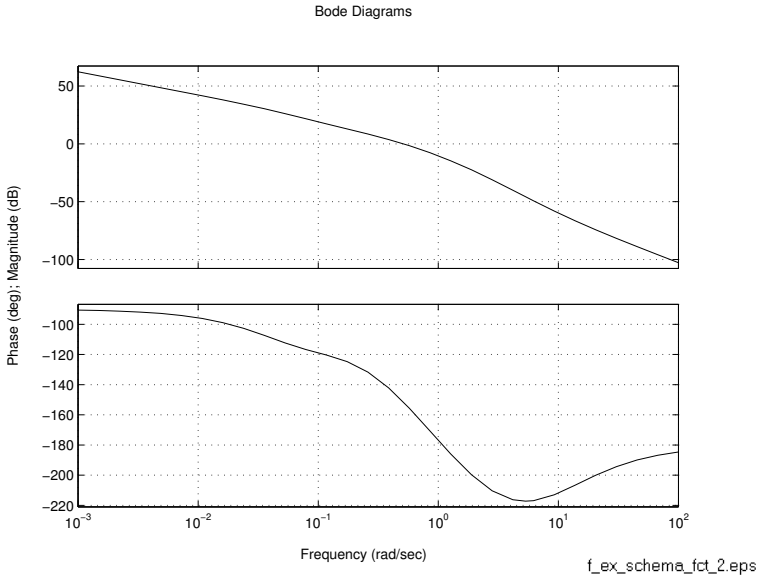


On peut au passage calculer le gain statique (gain DC) de $G_{yw}(s)$.

```
dcgain(Gyw)
```

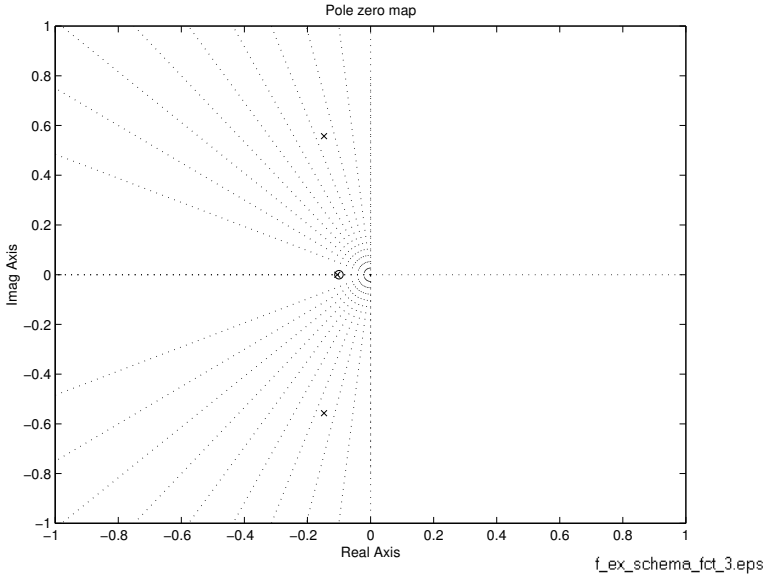
5. La réponse harmonique en boucle ouverte peut par exemple aussi être calculée :

```
bode(Go)
```



6. Finalement, on peut afficher la configuration pôle-zéro de $G_{yw}(s)$:

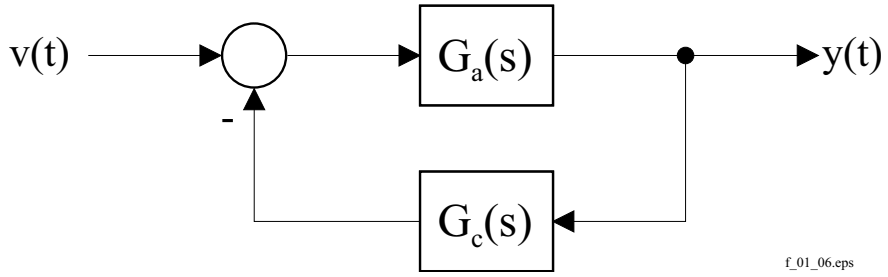
```
pzmap(Gyw)
```



La fonction feedback est utile pour calculer la fonction de transfert équivalente de systèmes ayant pour schéma fonctionnel la figure ci-dessous, soit :

$$G_{yv}(s) = \frac{Y(s)}{V(s)} = \frac{G_a(s)}{1 + G_a(s) \cdot G_a(s)}$$

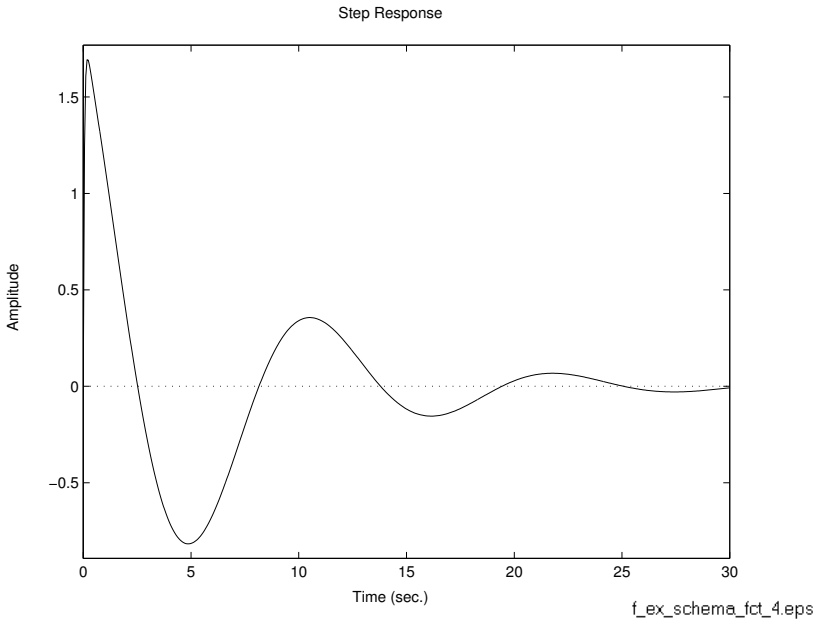
```
Gyv=feedback(Ga,Gc,signe);
```



Si le paramètre signe n'est pas spécifié ou vaut -1, la fonction de transfert $G_c(s)$ est en contre-réaction, alors qu'elle est réactionnée pour une valeur de signe égale à 1.

L'exemple ci-dessous calcule le fonction de transfert en régulation de maintien du système asservi précédemment traité.

```
G1 = Gc*Ga1;
Gyv = feedback(Ga2,G1);
```



4.5 Calcul et tracé de réponses de systèmes dynamiques linéaires

4.5.1 Réponse temporelle

Pour le calcul des réponses temporelles, les fonctions impulse, step, lsim sont disponibles. Leur signification sont données dans le tableau 4.5.1 page suivante.

Lorsqu'aucun argument de sortie n'est spécifié (*lhs, left handside argument*), le calcul et le tracé des réponses sont simultanés.

Les paramètres t, n, et w sont des optionnels et MATLAB les détermine lui-même par défaut. Si l'on souhaite les fixer, on peut par exemple le faire selon les indications du tableau 4.5.1 page 27.

il convient de spécifier la période d'échantillonnage h.

Il est souvent utile d'effectuer ces calculs de réponse sans tracer celle-ci immédiatement. Dans ce cas, on procède comme ci-dessus en spécifiant toutefois des arguments de sortie dans lesquels MATLAB sauvegardera les résultats de ses calculs :

```
[y,t]=impulse(G,t);
[y,t]=step(G,t);
[y]=lsim(G,u,t);
```

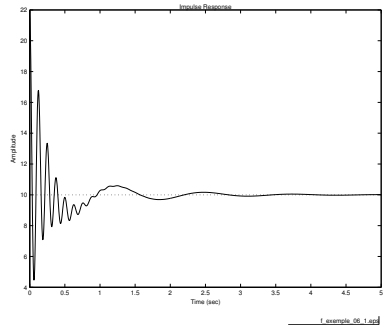
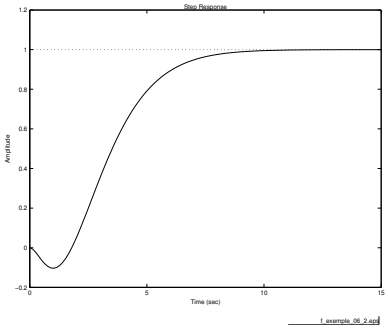
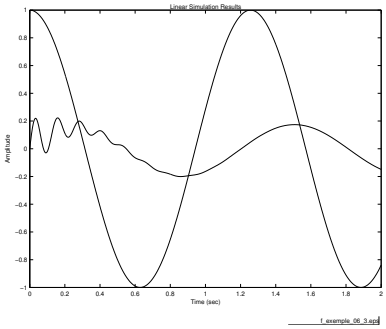
Commande	Réponse	Exemple
impulse(G)	réponse impulsionnelle	
step(G)	réponse indicielle	
lsim(G,u,t)	réponse à une entrée u quelconque définie par l'utilisateur	

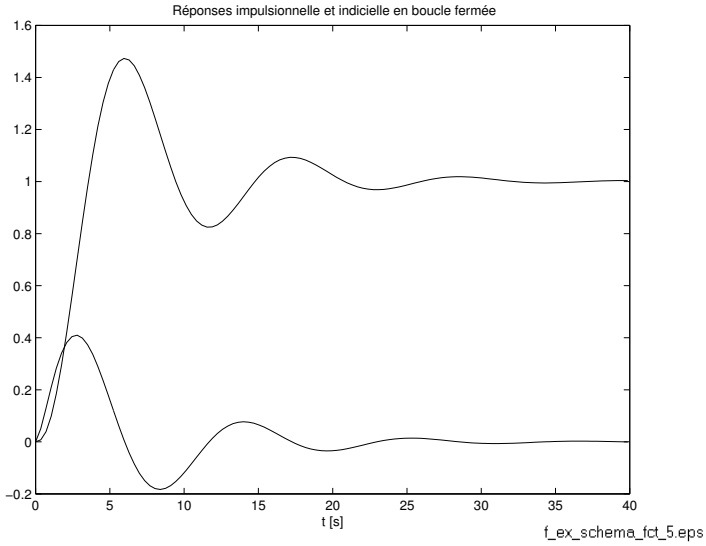
TABLE 1 –

Paramètre	Signification	Exemple
t	(facultatif) vecteur-ligne de temps, spécifiant les instants où la réponse doit être calculée (ce paramètre est obligatoire si l'on utilise lsim)	t=[0 :0.01 :0.2] ; ou t=linspace(0,0.2,201) ; (crée un vecteur temps variant entre 0 [s] et 0.2 [s] par pas de 0.01 [s]) ou $t = t_{\max}$
w	(facultatif) vecteur-ligne de pulsation, spécifiant les pulsations où la réponse doit être calculée	w=logspace(2,5,100) ; (crée un vecteur pulsation de 100 points espacés logarithmiquement et variant entre 10^2 $\left[\frac{\text{rad}}{\text{s}}\right]$ et 10^5 $\left[\frac{\text{rad}}{\text{s}}\right]$)
u	vecteur-colonne de l'entrée simulée du système (ce paramètre est obligatoire si l'on utilise lsim ou dlsim)	u=1.5*sin(2*pi*10*t)' ; (crée un vecteur u représentant une entrée sinusoïdale d'amplitude 1.5 et de fréquence 10 [Hz]. u est calculé pour chaque instant défini dans le vecteur t, par exemple celui donné ci-dessus)

TABLE 2 –

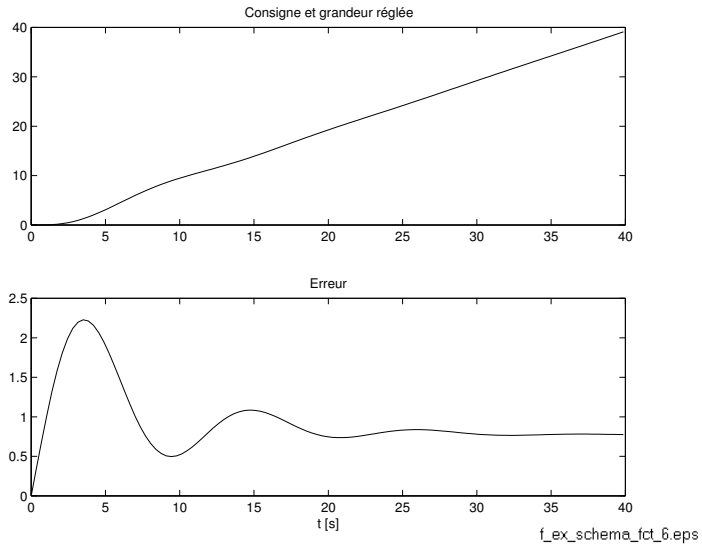
y est un vecteur-colonne contenant la réponse cherchée et t un vecteur-ligne contenant les instants auxquels elle a été calculée. La figure suivante montre le calcul préalable des réponses impulsionnelle et indicielle avant leur tracé à l'aide de la fonction **plot**.

```
[ys,t] = step(Gyw);
[yi] = impulse(Gyw,t);
figure
plot(t,[yi,ys])
title('Réponses impulsionnelle et indicielle en boucle fermée')
xlabel('t [s]')
```



De même, l'exemple suivant montre le tracé de la réponse d'un système asservi à une rampe de consigne, en faisant notamment usage de la fonction lsim.

```
wr = t';
yr = lsim(Gyw,wr,t);
figure
subplot(211),plot(t,yr)
title('Consigne et grandeur réglée')
subplot(212),plot(t,wr-yr)
title('Erreur')
xlabel('t [s]')
subplot(111)
```



4.5.2 Réponse fréquentielle

Les fonctions bode, nyquist et nychols sont disponibles. Leur utilisation est la suivante :

```
bode(G)
bode(G,w)
nyquist(G)
nyquist(G,w)
```

Si les résultats des calculs doivent être sauvegardés en vue d'un traitement ultérieur, on indique également des arguments de sortie ; dans le cas de la fonction bode, on aura par exemple :

```
[A,phi,w] = bode(G);
[A,phi] = bode(G,w);
```

A et phi sont des vecteurs-colonne contenant respectivement le gain et la phase et w un vecteur-ligne contenant les pulsations correspondantes.

4.6 Analyse des propriétés des systèmes

dcgain(G)	Gain statique d'un système.
damp(G)	Taux d'amortissement ζ, pulsation propre non-amortie ω_n et pôles d'un système

4.7 Calcul affichage des marges de gain et de phase

[Am,phim]=margin(Go)	Marges de phase et de gain d'un système.	Si les arguments de sortie ne sont pas spécifiés, trace le diagramme de Bode et montre graphiquement la valeur des marges

4.8 Tracé du lieu des pôles (ou lieu d'Evans)

rlocus(G,k)	Trace le lieu des pôles de la fonction de transfert.	k est une option ; c'est un vecteur-ligne contenant les différentes valeurs du facteur d'Evans pour lesquelles le lieu doit être tracé.
sgrid zgrid	Affiche les courbes équi-amortissement des plan de s et de z respectivement.	A exécuter immédiatement après rlocus
pzmap(G)	Graphe la configuration pôles-zéros de la fonction de transfert.	
[k,p]=rlocfind(G)	Tracé interactif du lieu des pôles.	A exécuter directement après rlocus et sgrid ; une croix est alors affichée, que l'on peut déplacer avec la souris sur un point particulier du lieu. En cliquant, on obtient alors dans k le facteur d'Evans correspondant et dans p l'expression du pôle.

4.9 Divers

minreal(G)	Force la compensation pôle-zéro.	
printsys(num,den,'s') printsys(num,den,'z')	Affiche les fonctions de transfert comme fractions ratrionnelles en s ou en z	

5 Les structures de contrôle du "langage" MATLAB

Les structures de contrôle usuelles sont disponibles :

Instruction	Exemple
for *variable=expression* *statements* end	delta_t=0.01 ; for i=1 :1 :21 ; %le pas vaut par defaut 1 y(i)=sin(2*pi*10*(i-1)*delta_t) ; end plot(y)
while *expression* *statements* end	delta_t=0.01 ; i=0 ; while i<21 i=i+1 ; y(i)=sin(2*pi*10*(i-1)*delta_t) ; end
if *expression* *statements* elseif *expression* *statements* else *statements* end	delta_t=0.01 ; i=0 ; if i<21, i=i+1 ; y(i)=sin(2*pi*10*(i-1)*delta_t) ; end

6 Simulation de systèmes dynamiques linéaires et non-linéaires avec la boîte à outils Simulink

6.1 Introduction

La boîte à outils Simulink est un complément extrêmement puissant à MATLAB. Les contributions de Simulink sont principalement :

1. La construction conviviale de schémas fonctionnels, faite à la souris ;
2. La simulation de systèmes linéaires et surtout non-linéaires, ainsi que de systèmes non-stationnaires ;
3. La simulation de systèmes mixtes analogiques et numériques.

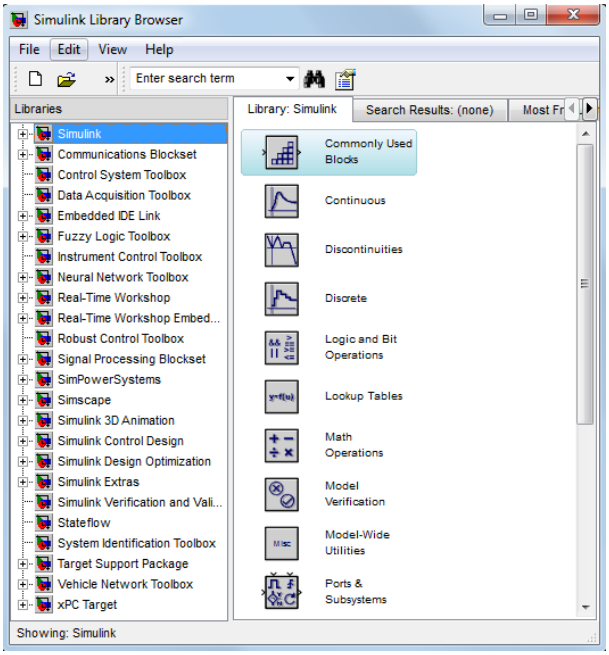


FIGURE 2 – Librairie standard de Simulink.

Pour la construction de schémas, le gain de temps se situe au-delà de 10 à 100, car il évite l'utilisation extrêmement fastidieuse des fonctions MATLAB blkbuild et connect.

Si certains reprochent à MATLAB un certain caractère rébarbatif, Simulink, en plus d'offrir de nouveaux outils très puissants, y remédie complètement en offrant une interface utilisateur extrêmement conviviale. Les deux logiciels restent toutefois complètement liés l'un à l'autre, Simulink n'étant du point de vue hiérarchique qu'une boîte à outils de MATLAB.

L'interface utilisateur de Simulink est l'outil permettant d'introduire directement les schémas fonctionnels de systèmes dynamiques. Bien que théoriquement

son usage ne soit pas indispensable, on peut dire qu'il est utilisé dans 90% des cas de problèmes à résoudre. On y accède à partir l'espace de travail MATLAB par la commande :

```
>>simulink
```

ou en appuyant sur le bouton correspondant de la barre d'outils de la fenêtre de MATLAB.

Une nouvelle fenêtre, appelée simulink apparaît. Elle n'est en fait qu'une bibliothèque de blocs fonctionnels standards, réunis ici en sept groupes, fournis avec le logiciel. La figure 2 page précédente montre ces groupes.

On remarque notamment les bibliothèques *Linear* et *Nonlinear* contenant chacune une collection de blocs linéaires et non-linéaires respectivement.

Fidèle à la philosophie de MATLAB, Simulink est ouvert et l'utilisateur peut rajouter ses propres blocs, qu'il peut organiser en une ou plusieurs bibliothèques ([2, et p.4.11 et p.3-19 et ss). Dans ce sens, les possibilités sont quasi illimitées.

Bien que cela ne paraisse pas impératif au premier abord, il se révèle assez tôt indispensable de bien connaître MATLAB pour utiliser Simulink de manière cohérente et organisée, i.e. professionnellement. Dans tous les cas, on a intérêt à utiliser Simulink en forte connection avec MATLAB ; il faut savoir que tous les blocs Simulink peuvent être paramétrés de manière symbolique. Ceci permet de définir les paramètres dans l'espace de travail MATLAB, soit manuellement, soit en exécutant un fichier de type *script*. Dans ce cas, plusieurs fichiers de paramétrisation peuvent être créés et lancés dans MATLAB avant le début de la simulation. La documentation desdits fichiers permet de rationaliser et d'organiser le grand nombre de simulations que l'on peut faire. C'est dans ce genre de situation que les compétences en programmation (organisation, décomposition et analyse du problème, structuration) de l'utilisateur peuvent s'avérer déterminantes.

Des boîtes à outils complémentaires (*Real Time Workshop*, *Matlab C Compiler*) permettent de générer le code C du schéma de simulation, ainsi que le code objet pour des processeurs de signaux Simulink peut être programmé relativement simplement en C. Dans ce cas, les performances en terme de temps de calcul seront nettement meilleures. Le fait de pouvoir tester son code source C avec MATLAB/Simulink avant de le livrer et l'exécuter sur l'application réelle permet d'accélérer et facilite considérablement le développement. Il s'agit incontestablement d'un des points forts de MATLAB/Simulink.

La programmation graphique, i.e. l'introduction directe de schémas fonctionnels, est un outil fantastique offert par Simulink. Cependant, les limites de ce genre de programmation sont rapidement atteintes lorsque le système est relativement complexe, auquel cas il sera plus facile de programmer Simulink "en texte" de manière classique, i.e. en créant des *fonctions S* (*S-function*), plutôt que d'interconnecter les blocs par de nombreuses lignes créées à la souris. Les fonctions S ont un format un peu particulier, propre à Simulink, mais s'écrivent presque

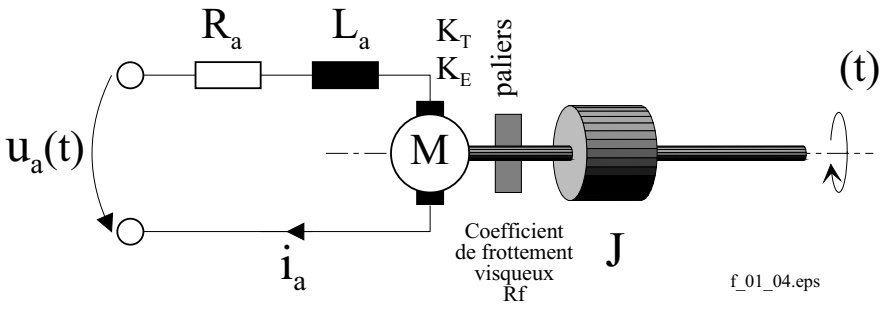


FIGURE 3 – Schéma technologique d'un entraînement DC à excitation séparée constante. Le frottement est purement visqueux, proportionnel à la vitesse, de coefficient R_f.

aussi simplement que les fonctions MATLAB ; elles consistent à programmer les équations différentielles du système à simuler.

6.2 Exemple introductif : simulation d'un système dynamique linéaire

On se propose d'illustrer l'utilisation de Simulink par l'exemple classique d'un moteur à courant continu à excitation séparée constante exploité en boucle ouverte (figure 3).

On admettra que le système est parfaitement linéaire, en faisant les hypothèses simplificatrices suivantes, ceci pour simplifier l'exemple et se concentrer sur l'essentiel de cette étude, i.e. le logiciel MATLAB/Simulink :

- on ne prend pas en compte l'effet des perturbations ;
- le frottement sur l'arbre moteur est purement visqueux ;
- le moteur à courant continu étant compensé, l'effet de réaction d'induit est négligeable.

Plusieurs démarches sont possibles pour la construction et l'organisation du schéma, de même que pour la simulation. On propose ici une façon de faire relativement laborieuse par rapport à la simplicité de l'exemple, calquée sur des problèmes concrets d'une certaine envergure, pour lesquels la méthodologie employée est déterminante. Comme on le verra, le lien avec MATLAB est sous-jacent.

6.2.1 Modélisation

Les modèles en t et en s du système sont (les conditions initiales étant supposées identiquement nulles) :

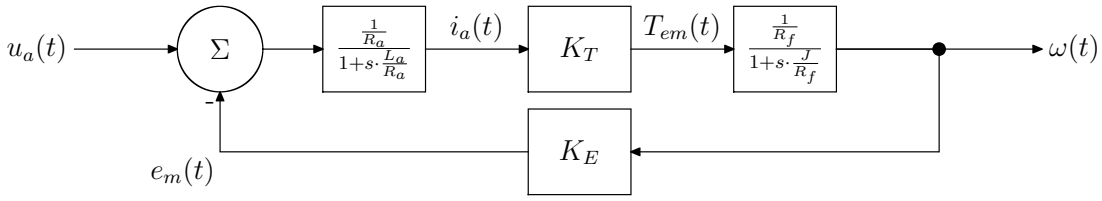


FIGURE 4 – Schéma fonctionnel correspondant au schéma technologique de la figure 3 page précédente.

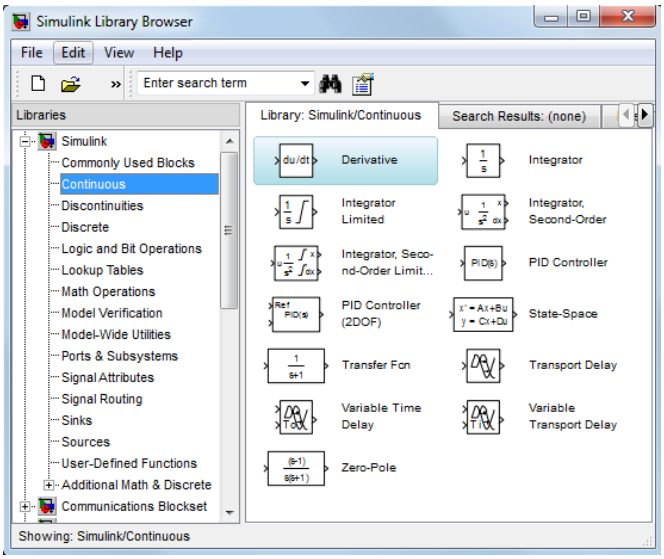


FIGURE 5 – Bibliothèque d'éléments dynamiques linéaires.

$$\left\{\begin{array}{l} u_a(t) = R_a \cdot i_a(t) + L_a \cdot \frac{di_a}{dt} + K_E \cdot \omega(t) \\ T_{em}(t) = K_T \cdot i_a(t) \\ J \cdot \frac{d\omega}{dt} = T_{em}(t) - R_f \cdot \omega(t) \end{array}\right\} \mathcal{L} \left\{\begin{array}{l} U_a(s) = R_a \cdot I_a(s) + L_a \cdot s \cdot I_a(s) + K_E \cdot \Omega(s) \\ T_{em}(s) = K_T \cdot I_a(s) \\ J \cdot s \cdot \Omega = T_{em}(s) - R_f \cdot \Omega(s) \end{array}\right\}$$

Après mise en équations, le schéma fonctionnel détaillé est celui de la figure 4.

6.2.2 Construction du schéma

Pour introduire le schéma fonctionnel de la figure 4 dans Simulink, il faut commencer par appeler Simulink à partir l'espace de travail de MATLAB, comme déjà fait précédemment.

Pour définir la fenêtre de travail, sélectionner New dans le menu File. Une nouvelle fenêtre est alors activée, nommée automatiquement Untitled. C'est dans cette même fenêtre que le schéma sera construit à partir des blocs contenus dans

les bibliothèques Simulink, voire d'autres blocs créés par l'utilisateurs et réunis dans d'autres bibliothèques. Revenant à la fenêtre Simulink et double-cliquant sur le groupe Linear, apparaissent alors certains blocs utiles à la construction (figure 5 page précédente) :

- le sommateur (Sum) ;
- le gain (Gain) ;
- la fonction de transfert (Transfer Fcn).

Sélectionnant le premier de ces blocs à l'aide de la souris dont le bouton gauche est maintenu pressé, on amène le bloc dans la fenêtre de travail en déplaçant la souris. Comme le gain et la fonction de transfert apparaissent chacun deux fois dans le schéma fonctionnel, l'opération est répétée en conséquence. La fenêtre de travail Untitled a maintenant l'allure de la figure 6 page suivante.

Pour faciliter la schématique, le bloc Gain 1 peut être changé de sens en le sélectionnant (le sélectionner avec la souris) et en choisissant Flip Horizontal dans le menu Options. On peut aussi faire usage des fonctionnalités offertes par le bouton droit de la souris.

Il faut alors réaliser les interconnections, toujours à la souris, en en maintenant le bouton gauche pressé, la souris pointant sur la flèche symbolisant l'entrée ou la sortie du bloc à connecter (le curseur prend la forme d'une fine croix, alors qu'il a l'allure d'une croix en trait épais dans le cas aucune entrée ou sortie ne sont sélectionnées). Le bouton gauche étant toujours pressé, on amène la souris sur la flèche du port à relier. On obtient alors facilement le schéma de la figure 7.

Les équations du moteur à courant continu montrent que la FEM est contre-réactionnée. Double-cliquer alors sur le bloc Sum et changer la liste des signes de ++ en +-. A titre de documentation, il vaut également la peine de renommer ces blocs, en cliquant sur les textes les accompagnant et les remplaçant par des désignations plus parlantes, comme par exemple induit, constante de couple, charge mécanique et constante de FEM (figure 8 page 39). Tant qu'à faire, pourquoi ne pas aussi nommer les signaux, en cliquant à l'endroit où le commentaire doit apparaître (ne pas cliquer l'un des blocs ou l'une des connections) et en tapant le texte. L'utilisation d'accents est aux risques et périls de l'utilisateur d'un logiciel d'origine américaine !

La paramétrisation des blocs s'effectue en double-cliquant et en remplissant les rubriques de la boîte de dialogue qui apparaît alors (figure 9 page 40). Par exemple, pour le bloc Induit, il faut spécifier les numérateur et dénominateur de la fonction de transfert. On recommande ici vivement de ne pas introduire de valeurs numériques, mais d'indiquer les noms de variables qui seront ultérieurement définies dans l'espace de travail MATLAB. On donnera donc numGa1 et denGa1, que l'on initialisera dans MATLAB par les commandes

```
numGa1 = 1/Ra;
denGa1 = [La/Ra, 1];
```

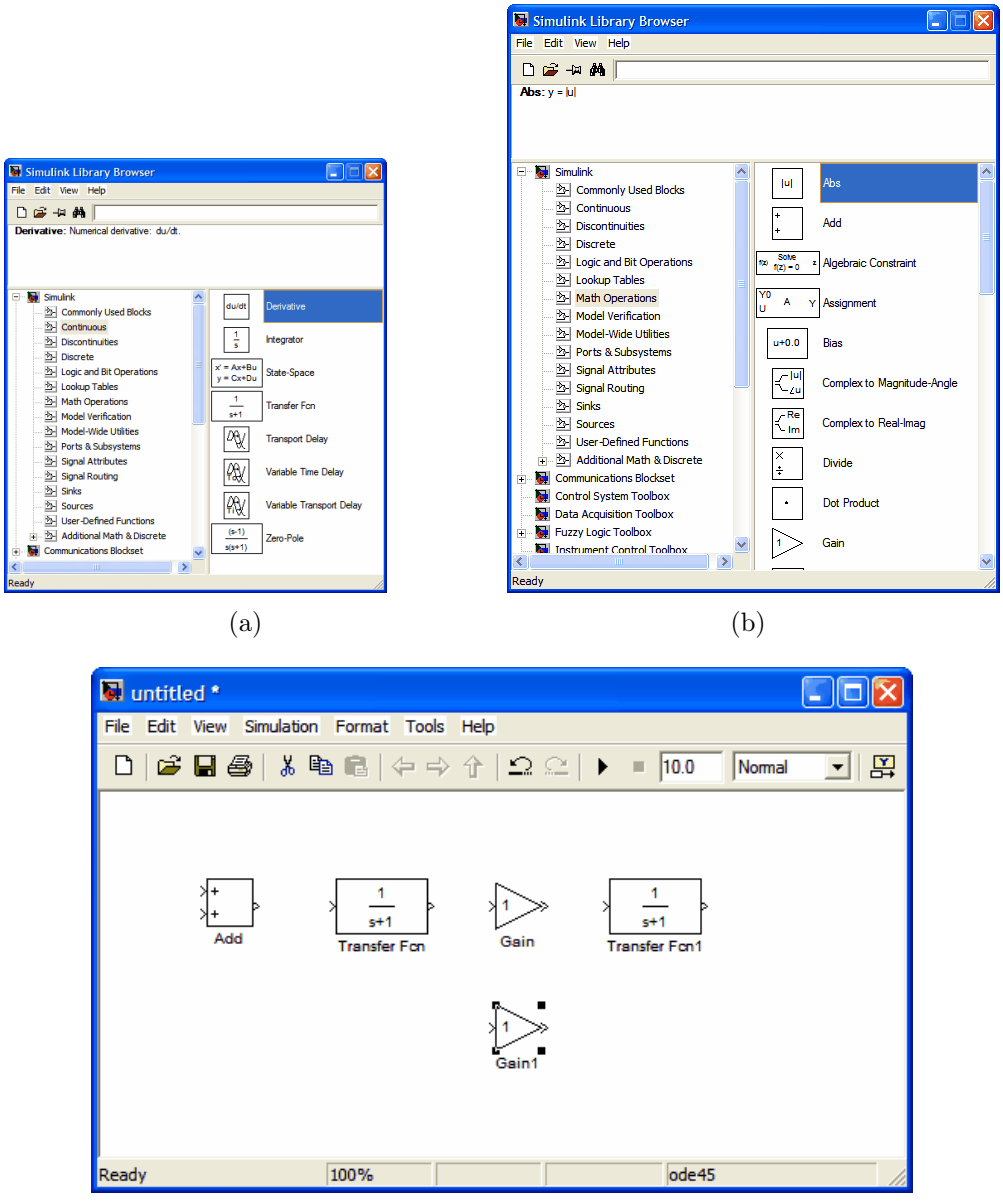


FIGURE 6 – Construction du schéma fonctionnel 6(c) du moteur DC de la figure 4 page 36 à l'aide des bibliothèques Continuous (6(a)) et Math operations 6(b).

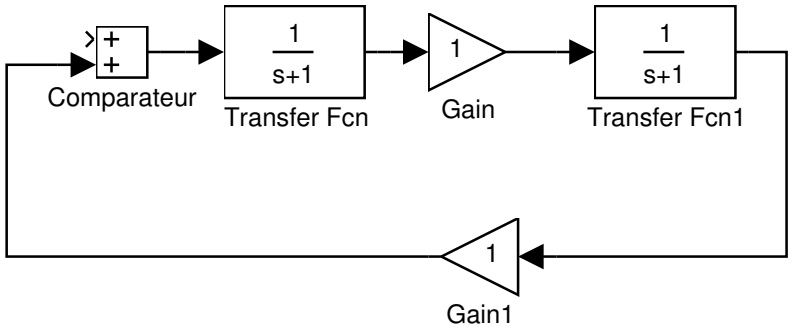


FIGURE 7 –

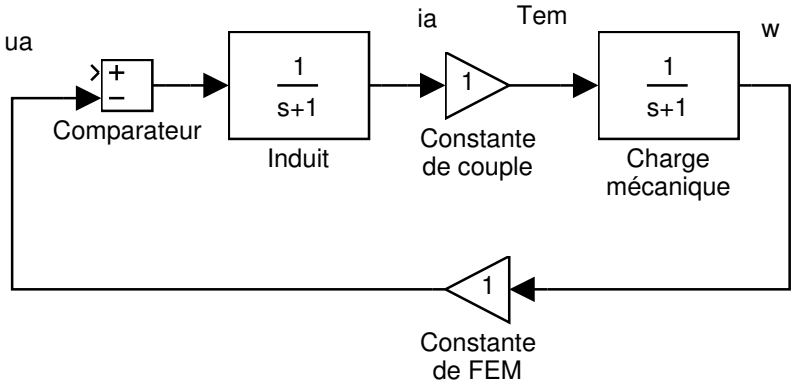


FIGURE 8 –

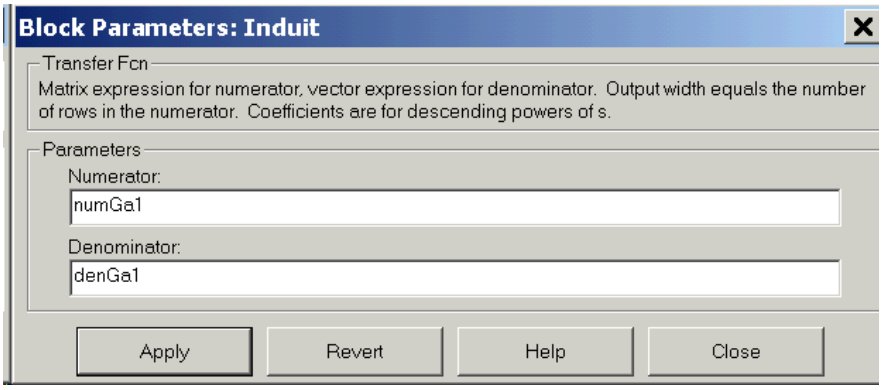


FIGURE 9 –

les valeurs numériques de R_a et L_a étant supposées connues dans l'espace de commande MATLAB.

Il reste à spécifier le signal d'entrée (la tension d'induit $u_a(t)$) et les signaux intéressants à visualiser ainsi que les moyens pour le faire. On propose pour cet exemple que $u_a(t)$ ait la forme d'un saut unité, que l'on visualise en plus de la vitesse angulaire ω(t) à l'aide d'un oscilloscope.

Pour générer la tension d'induit $u_a(t)$, on ouvre le groupe Sources de la fenêtre simulink et l'on amène le bloc Step Input dans la fenêtre de travail, en le connectant au comparateur. Pour visualiser les signaux, on ouvre le groupe Sinks de la fenêtre simulink et l'on amène également le bloc Scope. Comme l'on souhaite afficher deux signaux simultanément, il faut aller chercher dans le groupe Connections un multiplexeur (bloc Mux).

Il est encore nécessaire de paramétrer ces nouveaux blocs. En double cliquant successivement sur chacun d'eux, on peut spécifier que :

- le saut-unité s'effectue dès l'instant zéro, son amplitude étant 1 ;
- l'échelle horizontale de l'oscilloscope est 0.08 (selon § 6.2.4 page 43) ;
- le multiplexeur a deux entrées ($u_a(t)$ et $\omega(t)$).

Après avoir connecté ces blocs, on obtient le schéma suivant de la figure 10 page ci-contre.

Il est alors temps de sauver le schéma dans un fichier, ce qui se fait à partir du menu File, option SaveAs....On propose d'appeler ce schéma ex_01, à sauver dans `d:\users`. Simulink le sauve d'office avec l'extension .mdl.

6.2.3 Initialisation des paramètres

Pour initialiser les paramètres, on propose de créer un fichier MATLAB de type *script* , i.e. un fichier batch contenant la suite des commandes nécessaires. Profitant de la connaissance que l'on a de MATLAB, on obtient rapidement le fichier d'initialisation baptisé ini_01.m ci-dessous :

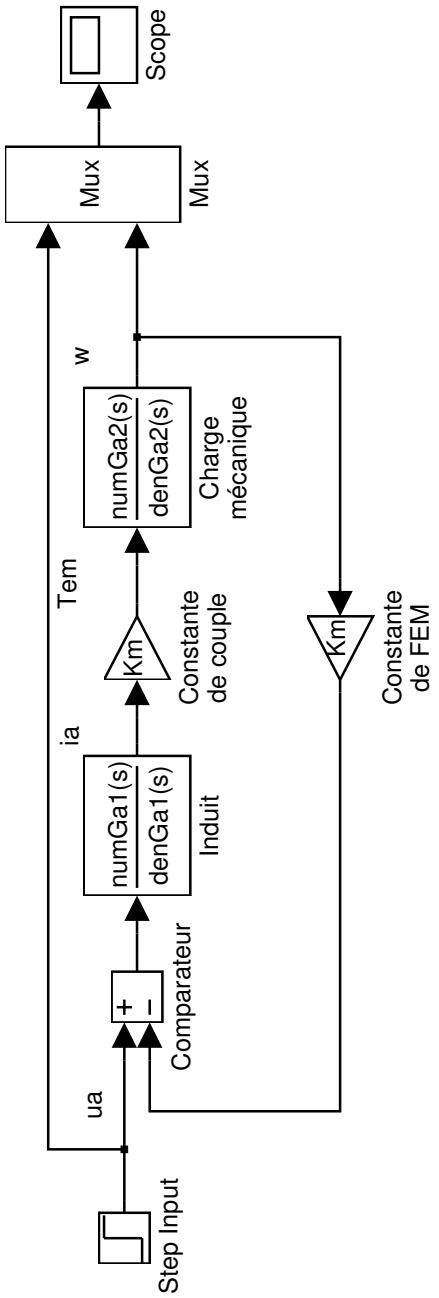

Scope
Mux
Mux
w
numGa2(s)
denGa2(s)
Charge
mécanique
Tem
Km
Constante
de couple
Km
Constante
de FEM
ia
numGa1(s)
denGa1(s)
Induit
+
–
Comparateur
ua
Step Input

FIGURE 10 –

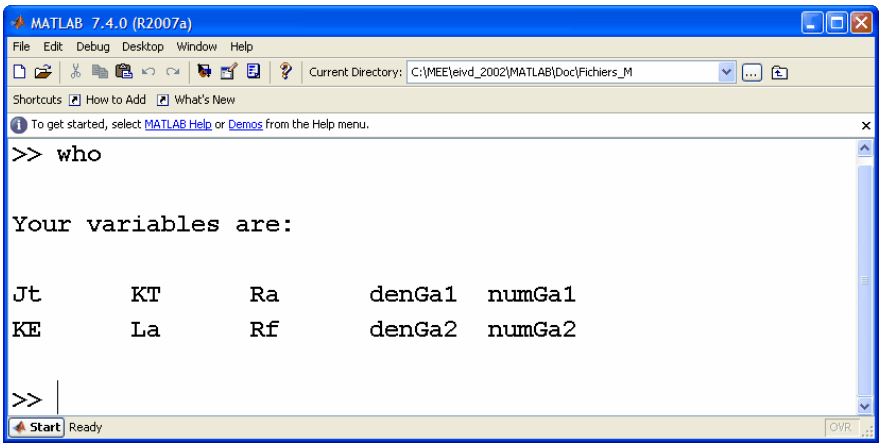


FIGURE 11 – Listage par la commande who des variables définies dans l'espace de commande MATLAB.

```
%/*
%*****************************************************************************
%                    INI_01.M
%*****************************************************************************
%          Fichier script contenant la suite des commandes
%          necessaires a l'initialisation des parametres
%          du moteur DC
%
%*****************************************************************************
%*****************************************************************************
%          Version   Date                Auteur     Motif
%          0         18/03/96  MEE       Creation
%                    21/05/97  MEE       Mise a jour notation KT et KE
%                    29/11/01  MEE       Mise a jour
%          ...       ...                 ...        (Modifications)
%*****************************************************************************
%*****************************************************************************
%*/
%          Initialisation
%          Parametres d'un moteur DC
          Ra = 1.84;                    %resistance de l'induit
          La = 0.00077;         %inductance de l'induit
          KT = 0.9;             %constante de couple
          KE = KT;              %constante de FEM
          Rf = 0.001;                   %coefficient de frottement visqueux (Nms/rad)
          J = 0.0061;                   %Moment d'inertie total

%         Fonctions de transfert
          numGa1 = 1/Ra;
          denGa1 = [La/Ra,1];

          numGa2 = 1/Rf;
          denGa2 = [J/Rf,1];
```

L'exécution de ce fichier se fait dans MATLAB en tapant son nom :

```
ini_01
```

On peut vérifier que MATLAB connaît désormais toutes les variables (vues par les blocs Simulink comme des paramètres) nécessaires à la simulation en tapant who (figure 11).

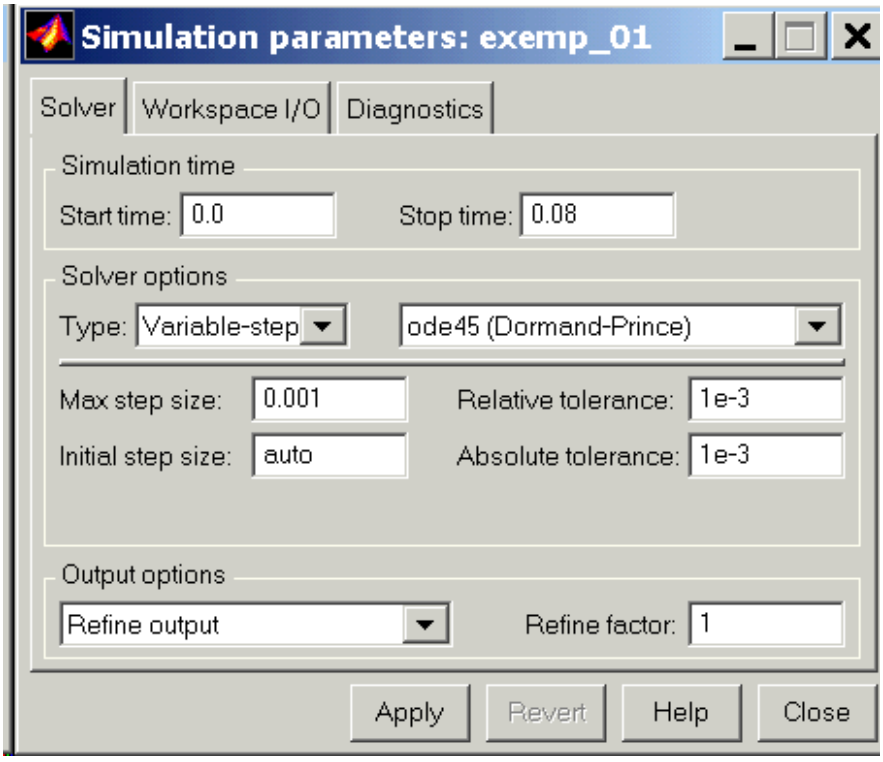


FIGURE 12 –

6.2.4 Simulation

On peut maintenant revenir à la fenêtre Simulink ex_01 (si elle n'est plus ouverte, simplement taper son nom dans MATLAB) et entrer dans le menu Simulation, option Parameters, ou plusieurs choix importants doivent être faits (12).

Il s'agit de déterminer tout d'abord la **méthode d'intégration** numérique des équations différentielles. On choisira par défaut Runge-Kutta 45.

L'**instant de départ** de la simulation ainsi que celui auquel elle se **termine** sont fixés notamment en fonction de la durée de la simulation désirée.

Le **pas d'intégration maximum** est à ajuster avec précaution, puisqu'il a une influence déterminante sur la précision et le temps de calcul. On voit qu'il est nécessaire, avant de démarrer la simulation, de se faire une idée de la rapidité des phénomènes. Dans le cas de l'exemple traité, la constante de temps mécanique τ_m et électrique τ_e

$$\tau_m = \frac{R_a \cdot J}{K_T \cdot K_E} = \frac{1.84\,[\Omega] \cdot 0.0061\,[\mathrm{kg \cdot m^2}]}{\left(0.9\left[\frac{\mathrm{N \cdot m}}{\mathrm{A}}\right]\right)^2} = 0.0139\,[\mathrm{s}]$$

$$\tau_e = \frac{L_a}{R_a} = \frac{7.7\,[\mathrm{mH}]}{1.84\,[\Omega]} = 0.0042\,[\mathrm{s}]$$

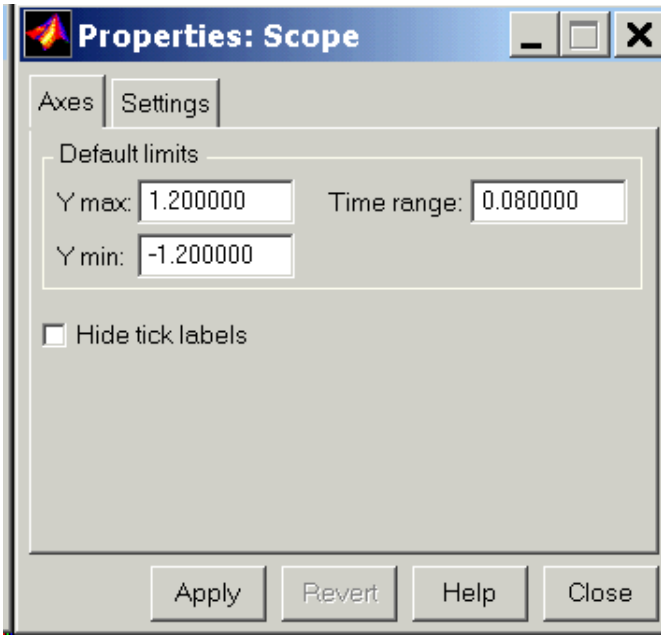


FIGURE 13 –

nous indiquent qu'avec un pas d'intégration maximum de l'ordre de 0.001 [s], la précision fixée par le paramètre tolérance devrait être atteinte facilement.

Grosso modo, le pas maximum devrait avoir une valeur de l'ordre de la durée d'un millième de la durée de simulation.

En double-cliquant sur l'oscilloscope, on peut encore ajuster sont ses échelles horizontale et verticale, que l'on choisira ici à 0.08 [s] et 1.2 $\left[\frac{\text{rad}}{\text{s}}\right]$ respectivement. A noter que 0.08 [s] représente la largeur totale de l'écran de l'oscilloscope (soit 5 divisions), 1.2 $\left[\frac{\text{rad}}{\text{s}}\right]$ étant équivalent à la moitié de la hauteur (soit 3 divisions).

Il reste maintenant à lancer la simulation, en choisissant l'option Start du menu Simulation.

Si l'on bascule (`Alt Tab`) sur l'oscilloscope, l'affichage est alors normalement conforme à la figure 14 page suivante. Au besoin, on peut appuyer sur le bouton zoom (symbole jumelles).

6.2.5 Sauvegarde des signaux puis traitement dans l'espace de travail MATLAB

Il vaut également la peine de sauvegarder dans une variable MATLAB les signaux intéressants pour effectuer un traitement ultérieur dans MATLAB. On choisit $u_a(t)$, le couple électromagnétique $T_{em}(t)$ et la vitesse angulaire $\omega(t)$. En effectuant cette sauvegarde, il ne faut pas omettre l'information temps t, que l'on peut obtenir en allant chercher le bloc Clock dans le groupe Sources de la bibliothèque. Les trois informations à sauver sont de préférence multiplexées,

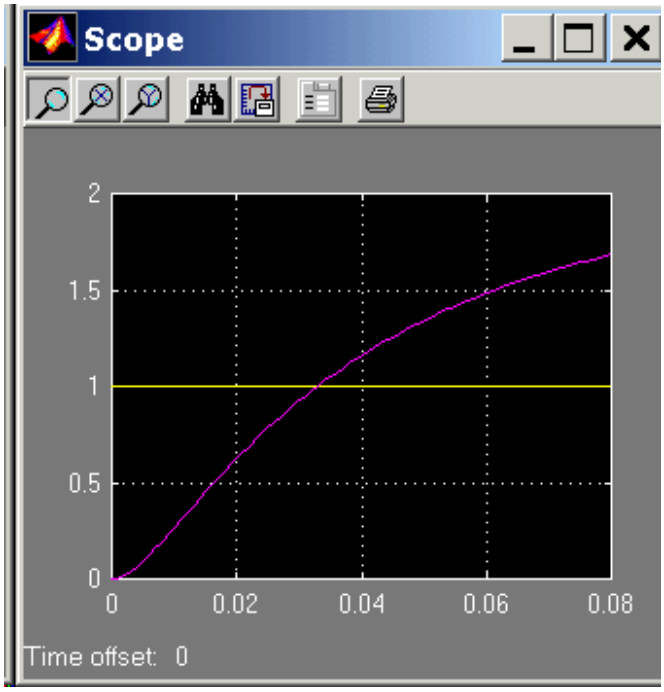


FIGURE 14 –

i.e. réunies en un signal de type vecteur, à l'aide d'un nouveau bloc Mux, du groupe Connections, que l'on paramétrise à trois entrées. Enfin, le signal issu de ce multiplexeur est transmis à un bloc assurant la liaison avec l'espace de travail MATLAB, le bloc To Workspace, que l'on trouve dans le groupe Sinks. Ce dernier bloc a pour paramètres (figure 15 page suivante) le nom de la variable MATLAB dans laquelle les résultats seront sauvés, ainsi qu'une indication relative au nombre maximum de pas de calcul à sauver. Si la durée du pas et faible et que celle de la simulation est grande, on conçoit que le nombre de données à sauver soit prohibitif et provoque une saturation de l'espace mémoire. Pour l'éviter en partie, on peut en option spécifier au bloc To Workspace de ne sauver qu'à intervalles réguliers, de durée plus grande que le pas d'intégration.

Le schéma se présente alors comme suit, sauvé dans le fichier ex_02.mdl (figure 16 page 47)

Comme précédemment, on démarre la simulation et lorsque celle-ci est terminée, on peut revenir à l'espace de travail MATLAB et découvre l'existence de la variable mesures, créée par le bloc To Workspace.

Le fichier MATLAB cal_01.m suivant en extrait les informations et trace les trois courbes $u_a(t)$, $\omega(t)$ et $T_{em}(t)$, ainsi que la puissance mécanique $P_{mec}(t)$ après avoir calculé celle-ci (figure 17 page 48).

```
%/*
%*************************************************************************
%                        CAL_01.M
```

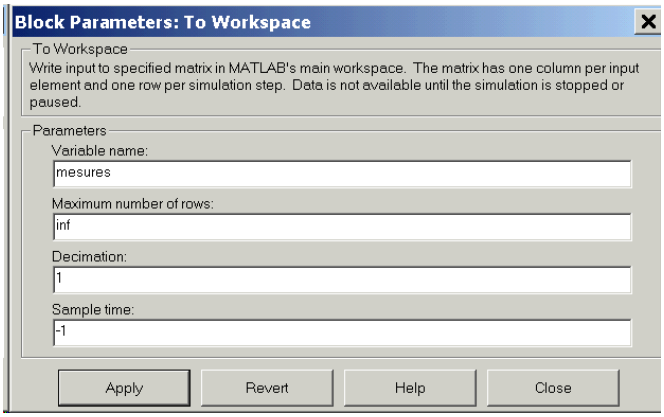


FIGURE 15 –

```
%*************************************************************************
%          Fichier script contenant la suite des commandes
%          necessaires a l'interpretation de la variable
%          mesures produite par ex_02.m
%
%*************************************************************************
%*************************************************************************
%          Version    Date                Auteur     Motif
%          0          18/03/96  MEE       Creation
%          ...        ...                 ...        (Modifications)
%*************************************************************************
%*************************************************************************
%*/

%          Extraction du temps t et des 3 signaux ua, omega et Tem

           t = mesures(:,1);
           ua = mesures(:,2);
           omega = mesures(:,3);
           Tem = mesures(:,4);

%          Calcul de la puissance mecanique instantanée

           Pmec = Tem .* omega;

%          Affichage sur un ecran form d'un colonne de 4 graphiques
           figure(gcf)
           subplot(411), plot(t,ua)
           xlabel('t [s]')
           ylabel('u_a(t)')
           title('MESURES')

           subplot(412), plot(t,omega)
           xlabel('t [s]')
           ylabel('\omega(t)')

           subplot(413), plot(t,Tem)
           xlabel('t [s]')
           ylabel('T_{em}(t)')

           subplot(414), plot(t,Pmec)
           xlabel('t [s]')
           ylabel('P_{mec}(t)')

           subplot(111)
```

6.2.6 Lancement de la simulation à partir de MATLAB

Les simulations effectuées peuvent être lancées à partir de l'espace de travail MATLAB, sans même ouvrir la fenêtre du schéma de simulation. La ligne de commande est la suivante :

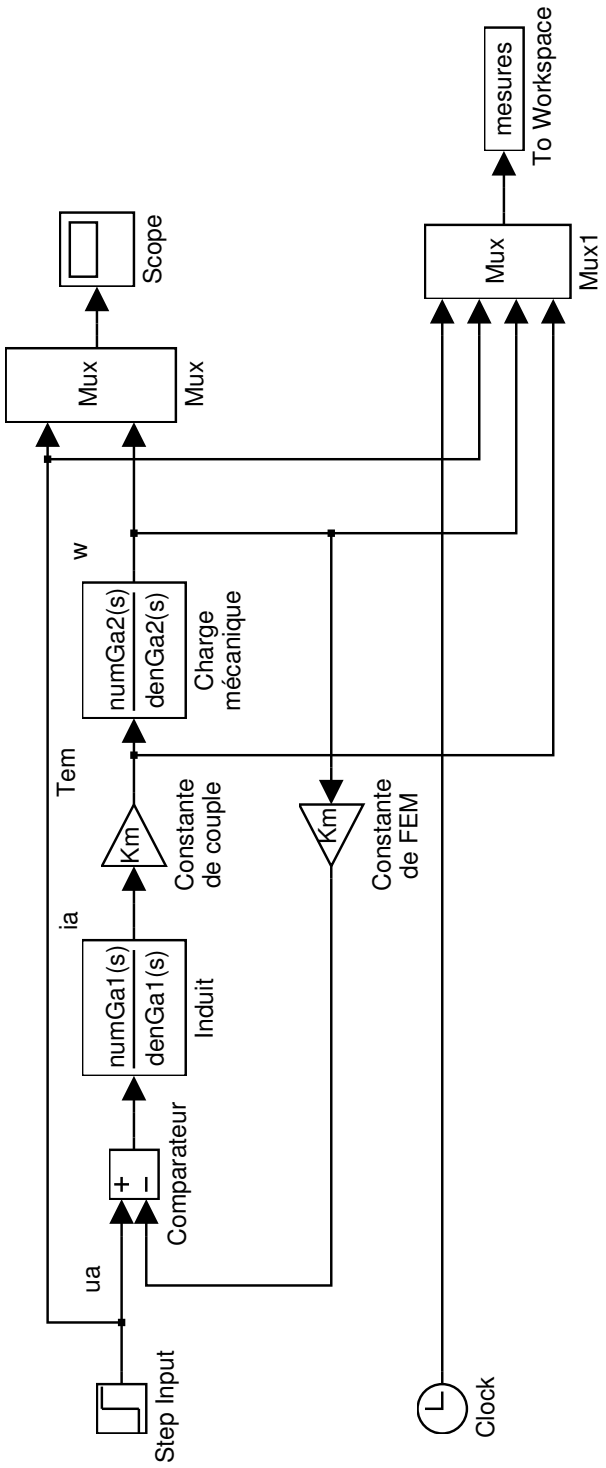

mesures
To Workspace
Scope
Mux
Mux1
Mux
Mux
w
numGa2(s)
denGa2(s)
Charge
mécanique
Tem
Km
Constante
de couple
Km
Constante
de FEM
ia
numGa1(s)
denGa1(s)
Induit
+
-
Comparateur
ua
Step Input
Clock

FIGURE 16 –

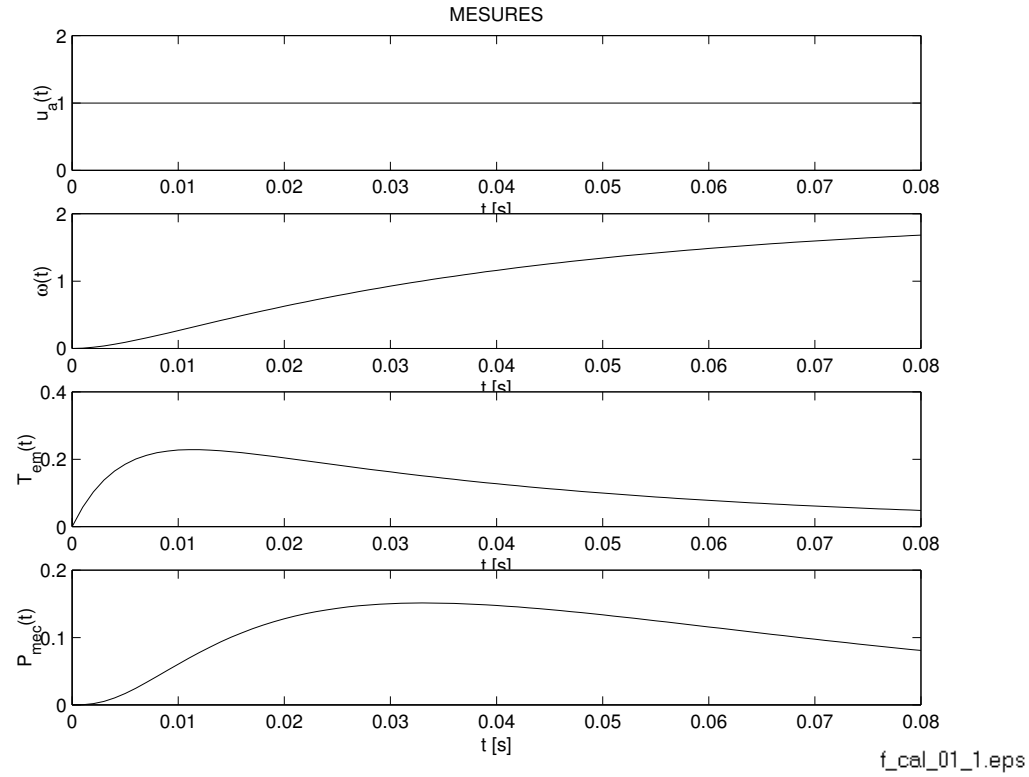


FIGURE 17 –

```
[t,x,y] = sim('ex_01',tfinal,options,u);
```

Dans notre cas :

```
ini_01
[t,x,y] = sim('exemp_01',0.08,[]);
```

t, x et y seront à la fin de la simulation des vecteurs-colonne contenant respectivement les instants de simulation, le vecteur d'état correspondant et les sorties à ces instants. Ces dernières doivent être explicitement connectées à des blocs de type Outport, groupe Connections ;

- tfinal est l'instant auquel la simulation prend fin ;
- x0 est optionnel et représente le vecteur d'état (colonne) initial, i.e. les conditions initiales ;
- options est un vecteur à plusieurs composantes. Voir l'aide de MATLAB ;
- u est un tableau spécifiant les valeurs prises par les entrées en fonction du temps, spécifié dans la première colonne. Les entrées sont amenées au schéma de simulation par des blocs de type Inport, groupe Connections.

Il est important de réaliser que cette commande peut être intégrée sans autre à tout fichier MATLAB, qu'il soit de type script ou function.

Pour des informations plus détaillées, on consultera le §4-104 de [2].

6.3 La bibliothèque standard

6.3.1 Sources

Cette bibliothèque (figure 18 page suivante) contient des éléments **générateurs de signaux** tels que saut unité, sinusoïde, fichiers de points, variable MATLAB, bruit, séquences, le temps courant de la simulation (horloge), etc, sans oublier le générateur de signal lui-même. Sans parler du fait que l'on peut soit-même créer son propre générateur de signal, on note ici qu'il y a vraiment intérêt à paramétrer ces blocs dans à l'aide de variables définies proprement dans MATLAB.

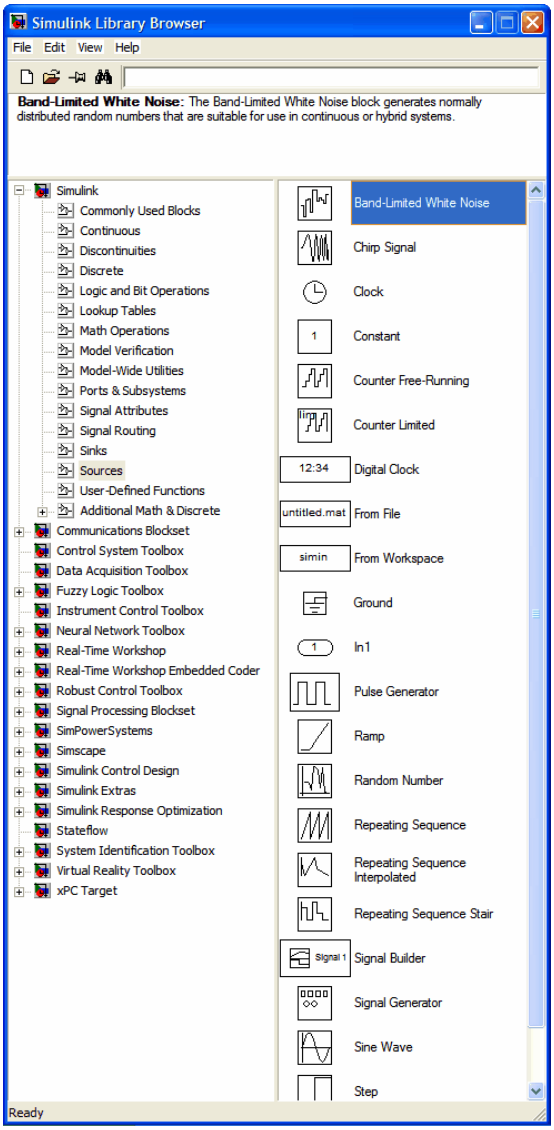

Simulink Library Browser
File Edit View Help
Band-Limited White Noise: The Band-Limited White Noise block generates normally distributed random numbers that are suitable for use in continuous or hybrid systems.
Simulink
Commonly Used Blocks
Continuous
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functions
Additional Math & Discrete
Communications Blockset
Control System Toolbox
Data Acquisition Toolbox
Fuzzy Logic Toolbox
Instrument Control Toolbox
Neural Network Toolbox
Real-Time Workshop
Real-Time Workshop Embedded Coder
Robust Control Toolbox
Signal Processing Blockset
SimPowerSystems
Simscape
Simulink Control Design
Simulink Extras
Simulink Response Optimization
Stateflow
System Identification Toolbox
Virtual Reality Toolbox
xPC Target
Band-Limited White Noise
Chirp Signal
Clock
Constant
Counter Free-Running
Counter Limited
12:34
Digital Clock
untitled.mat
From File
simin
From Workspace
Ground
In1
Pulse Generator
Ramp
Random Number
Repeating Sequence
Repeating Sequence Interpolated
Repeating Sequence Stair
Signal 1
Signal Builder
Signal Generator
Sine Wave
Step
Ready

FIGURE 18 –

6.3.2 Sinks

Cette bibliothèque (figure 19 page ci-contre) contient des éléments **récepteurs de signaux** tels que fichiers, variable MATLAB ainsi que l'oscilloscope. S'agissant de ce dernier, il est à considérer plutôt comme un gadget permettant de vérifier le bon fonctionnement de la simulation. La combinaison de l'oscilloscope avec un multiplexeur analogique (bibliothèque *Connections*) permet de grouper plusieurs signaux sur une seule ligne de créer ainsi un oscilloscope multi-traces. La présence d'un grand nombre d'oscilloscopes dans un même schéma ralentit considérablement la simulation, comme d'ailleurs la sauvegarde systématique des signaux dans des variables MATLAB.

Dans le cadre d'un projet, c'est toutefois cette dernière solution qui sera préférée, l'analyse des signaux pouvant s'effectuer dans MATLAB une fois la simulation terminée. On dispose alors de toutes les fonctions MATLAB pour procéder aux analyses et traitements (par exemple, filtrage, interpolation, transformée de Fourier, calcul de la puissance, etc). C'est une règle simple qui mérite d'être soulignée : **il faut utiliser Simulink pour la simulation, et MATLAB pour l'analyse et le traitement off line.**

A titre illustratif, si un schéma de simulation fournit les signaux de tension $u(t)$ et de courant $i(t)$, il vaut mieux sauver ces deux signaux dans l'espace de travail MATLAB à intervalles réguliers et calculer la puissance instantanée $p\left(t\right) = u\left(t\right) \cdot i\left(t\right)$ lorsque la simulation a pris fin, plutôt que de faire ce calcul dans le schéma de simulation (comme fait au § 6.2.5 page 44). Dans ce dernier cas, on augmente inutilement la quantité de calculs à faire en cours de simulation, et de plus on ajoute une non-linéarité au schéma. Globalement, la perte de temps peut être considérable.

D'une manière plus générale, il faut réserver Simulink à la simulation de la partie dynamique des systèmes et lui épargner les autres calculs. En langage de représentation des systèmes dans l'espace d'état, ceci revient à limiter le schéma de simulation à l'équation d'état différentielle,. i.e.

$$\frac{d}{dt}\vec{x} = f\left(\vec{x}\right) + g\left(\vec{u}\right) \quad ,$$

et à omettre l'équation d'observation

$$\vec{y} = h\left(\vec{x}\right) + i\left(\vec{u}\right)$$

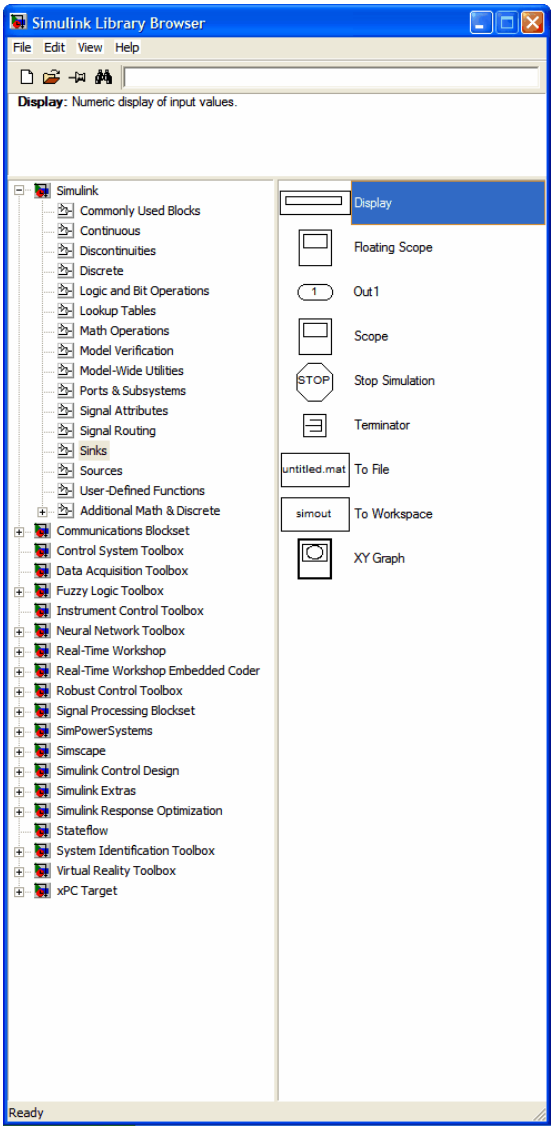

Simulink Library Browser
File Edit View Help
Display: Numeric display of input values.
Simulink
Commonly Used Blocks
Continuous
Discontinuities
Discrete
Logic and Bit Operations
Lookup Tables
Math Operations
Model Verification
Model-Wide Utilities
Ports & Subsystems
Signal Attributes
Signal Routing
Sinks
Sources
User-Defined Functions
Additional Math & Discrete
Communications Blockset
Control System Toolbox
Data Acquisition Toolbox
Fuzzy Logic Toolbox
Instrument Control Toolbox
Neural Network Toolbox
Real-Time Workshop
Real-Time Workshop Embedded Coder
Robust Control Toolbox
Signal Processing Blockset
SimPowerSystems
Simscape
Simulink Control Design
Simulink Extras
Simulink Response Optimization
Stateflow
System Identification Toolbox
Virtual Reality Toolbox
xPC Target
Display
Floating Scope
1
Out1
Scope
STOP
Stop Simulation
Terminator
untitled.mat
To File
simout
To Workspace
XY Graph
Ready

FIGURE 19 –

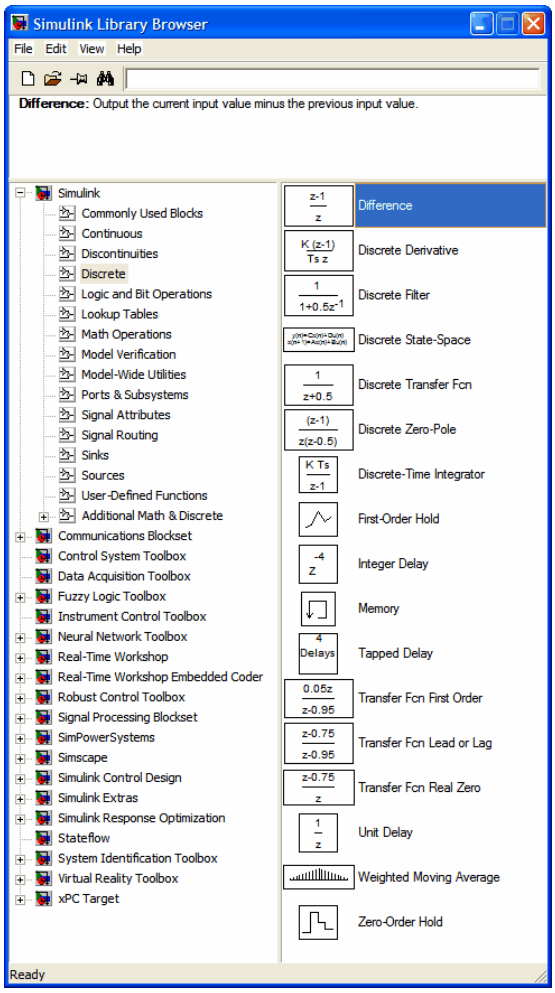


FIGURE 20 –

6.3.3 Discrete

Cette bibliothèque (figure 20) contient des éléments **discrets** linéaires (dicrétisation du temps exclusivement), tels que gains, systèmes dynamiques linéaires (transmittance isomorphe, équations d'état), élément de maintien, etc. L'intégrateur limité, curieusement présent dans cette bibliothèque, est en fait un élément non-linéaire.

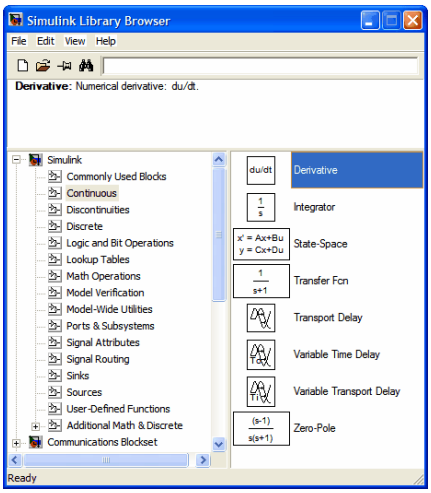


FIGURE 21 –

6.3.4 Linear

Cette bibliothèque (figure 21) contient des éléments **analogiques** effectuant tous des opérations **linéaires** sur les signaux, tels que gains, sommateurs, systèmes dynamiques linéaires représentés par leur fonction de transfert ou leurs équations d'état.

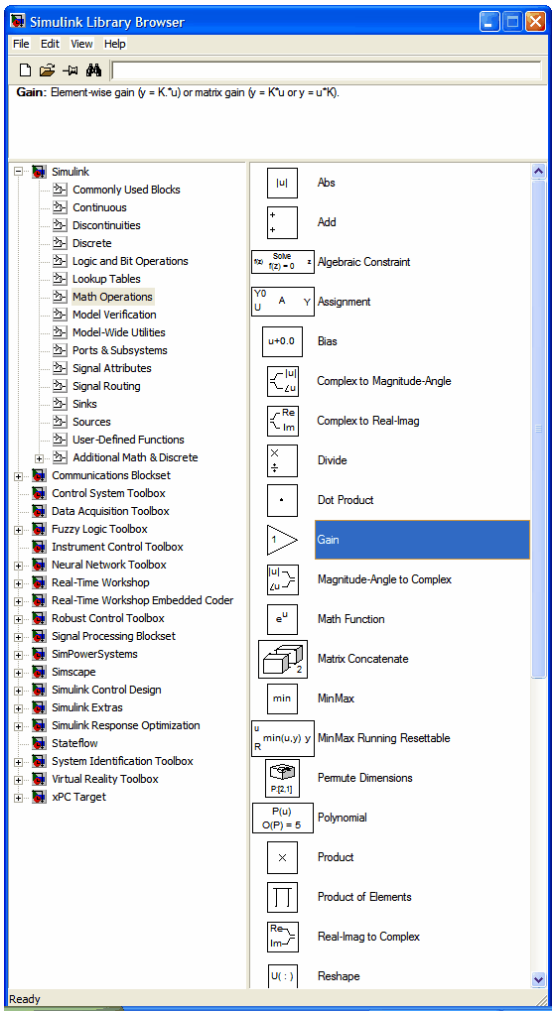


FIGURE 22 –

6.3.5 Nonlinear

Cette bibliothèque (figure 22) contient des éléments non-linéaires statiques ou dynamiques. C'est l'une des plus intéressantes, puisque grâce au différents blocs mis à disposition, on peut simuler par exemple

- un élément de type tout-ou-rien avec ou sans hystérèse (Sign, Relay) ;
- un élément introduisant une saturation (Saturation) ;
- un effet de jeu (backlash) ;
- la quantification de l'amplitude apportée par un convertisseur A/D ou D/A (Quantizer) ;
- un effet de seuil ou de zone morte (Dead zone) ;
- un effet de limitation de la variation par rapport au temps (Rate limiter) ;
- une non-linéarité de type valeur absolue (Abs) ;

– un élément effectuant le produit de deux signaux (Product) ;
– des opérations logiques simples ou complexes (AND, Combinatorial logic) ;
– un élément créant un retard pur (Transport delay) ;
– un élément dont la caractéristique statique est disponible dans une table (look up table) ;
– etc.

Là aussi, il est vivement conseillé de paramétrer ces blocs au moyen de variables définies dans MATLAB.

Cette liste non-exhaustive, bien qu'offrant déjà de nombreuses possibilités, ne doit pas faire oublier les possibilités fantastiques offertes par les blocs

– MATLAB Fcn ;
– S-Function.

Le premier de ces blocs permet d'appeler une fonction MATLAB standard ou définies par l'utilisateur (linéaire ou non). Celle-ci peut être multivariable.

Le second offre les mêmes possibilités que le premier, à ceci près qu'il s'agit d'une fonction directement exécutable par le noyau de calcul de Simulink : aucune conversion préalable n'est nécessaire, la fonction S fournissant à Simulink les informations demandées, celles-ci étant spécifiées par les paramètres d'appel. Cette fonction S peut donc être un autre schéma Simulink, voire être écrite explicitement par l'utilisateur dans un fichier respectant les conventions d'appels (simples) de Simulink. Dans ce dernier cas, la fonction peut être programmée soit en langage MATLAB ou même directement en langage C.

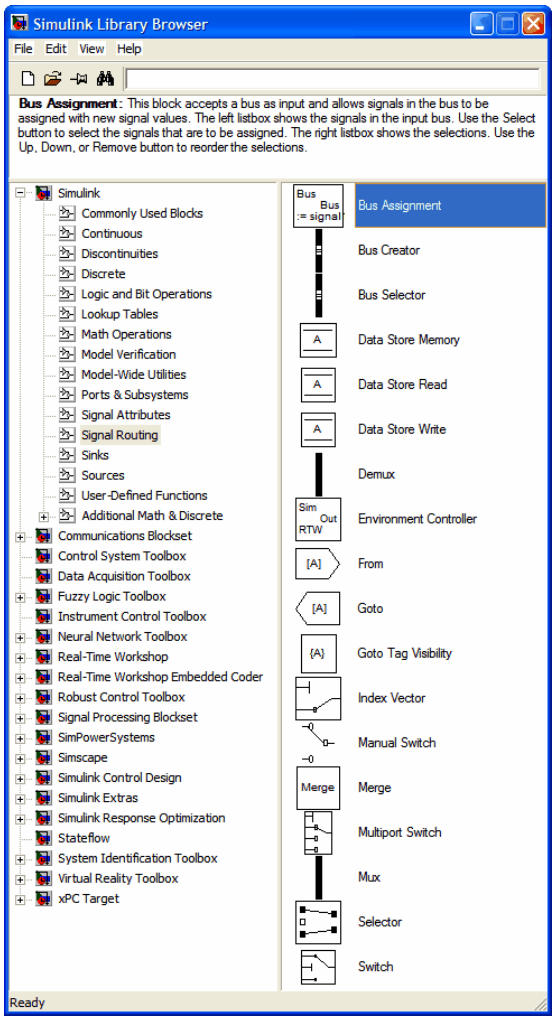


FIGURE 23 –

6.3.6 Connections

Cette bibliothèque (figure 23) contient essentiellement des éléments permettant de grouper plusieurs signaux sur une même ligne (Mux) et celui faisant l'opération inverse (Demux).

Les blocs Inport et Outport sont mono-dimensionnels et ne sont pas directement utiles, sauf dans le cas ou la simulation est effectuée depuis l'espace de travail MATLAB à l'aide d'une ligne de commande, sans ouvrir l'interface graphique de Simulink. Ils permettent alors de définir clairement les signaux d'entrée et de sortie. Simulink les rajoute automatiquement à un schéma lorsque que l'on groupe plusieurs blocs un seul, afin d'alléger le schéma ou d'identifier plus facilement les ensembles de blocs dédiés à une seule fonction globale.

6.4 Les S-functions de Simulink (version provisoire)

La boîte à outils Simulink de MATLAB est avant tout un ensemble de routines de résolution numérique des équations différentielles (ou aux différences dans le cas discret) régissant le comportement de systèmes dynamiques. Parmi celles-ci, on note par exemple ode45 (Runge-Kutta d'ordre 4 ou 5). Ces routines peuvent traiter des fichiers décrivant le système à simuler à condition que celui-ci ait un format défini : il doit apparaître comme étant une **S-function**.

Il n'en va pas autrement lorsque l'utilisateur, plutôt que de décrire son système par son modèle d'état, profite de l'éditeur de schéma de Simulink pour construire directement le schéma fonctionnel du système à la souris : en arrière-plan, l'éditeur de Simulink transforme en fait le schéma en une S-function compatible avec les routines de résolution numérique.

Si l'on connaît le format d'une S-function, il est donc possible de se passer de l'éditeur de schéma de Simulink et d'écrire directement un fichier MATLAB satisfaisant le format précité, i.e d'écrire une S-function. Dans certains cas, il est plus rapide d'implanter les équations de la dynamique sous forme texte plutôt que graphique. Outre le modèle d'état du système, le fichier de la S-function devra implanter le protocole d'échange de données entre la routine de résolution numérique et la S-function. Or, ce dernier est relativement simple ; dans les grandes lignes, on dira que la routine de résolution interroge la S-function en affectant des valeurs aux arguments (signaux d'entrée et éventuellement paramètres). Par exemple,


```
[t,x,y] = sim('S_si_02_02',tf,options,ut);
```

avec tf, options et ut représentant respectivement l'instant final, les options et le signal d'entrée. S_si_02_02.mdl est le nom d'un schéma Simulink (figure 24 page suivante) comprenant un bloc de type S-function qui fait référence à un fichier MATLAB (figure 24 : s_02_02.m, listing 1) programmé par l'utilisateur respectant le format S-function.

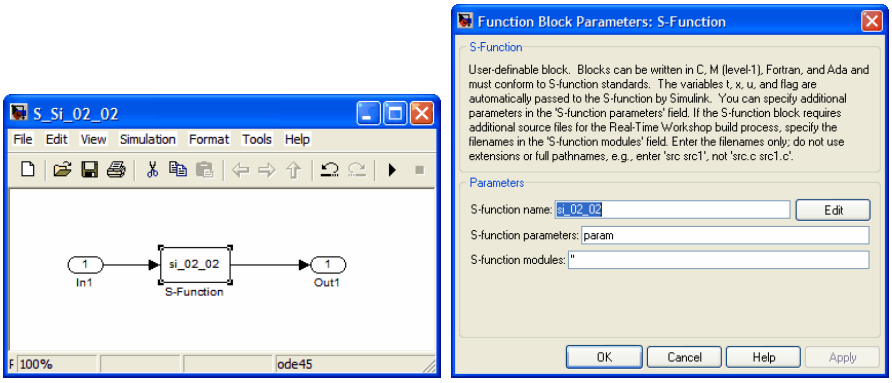
Les options de simulation prennent la forme d'une structure et se fixent à l'aide de la fonction simset :

```
options = simset('Solver','ode45','InitialState',x0);
```

Dans cet exemple, la méthode de résolution numérique est ode45 et les conditions initiales de simulation sont condensées dans le vecteur d'état x0.

La S-function peut être écrite en langage MATLAB, C, ou Fortran. La forme de la fonction "S" est très générale et s'adapte aussi bien à des systèmes continus, discrets, voir hybrides. Il en résulte que n'importe quel système dynamique peut être décrit par une "S-Function".

Le bloc "S-Function" est disponible parmi les blocs de la librairie "User-Defined Functions".



(a) Schéma Simulink contenant un bloc S-function.

(b) Paramétrisation du bloc S-function : le fichier contenant la S-function est si_02_02.m et a été programmé par l'utilisateur. Voir par exemple le listing 1.

FIGURE 24 – Schéma Simulink implantant un bloc appelant une S-function.

6.4.1 Conventions d'appel des S-functions

Informations diverses quant à la nature du système Si le noyau de calcul appelle la fonction si_02_02 en posant flag=0, il attend en retour sys qui doit alors être un vecteur contenant respectivement le nombre d'états analogiques, d'états discrets, de sorties, d'entrées, de racines discontinues ainsi qu'une indication si la matrice D est non nulle. De plus, il attend dans `x0` le vecteur d'état initial (i.e. conditions initiales).

Equation de sortie Si le noyau de calcul appelle la fonction `si_02_02` en posant flag=3, il attend en retour sys qui doit alors correspondre à la sortie du système.

Equation d'état différentielle Si le noyau de calcul appelle la fonction `si_02_02` en posant flag=1, il attend en retour sys qui doit alors correspondre à la derivée du vecteur d'état.

Equation d'observation Si le noyau de calcul appelle la fonction `si_02_02` en posant flag=3, il attend en retour sys qui doit alors correspondre au signal de sortie.

6.4.2 Exemple

Listing 1 – Exemple de code MATLAB d'une S-function

```
function [sys,x0,str,ts] = si_02_02(t,x,u,flag,param)

switch flag,
```

```
  %%%%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(param);

  %%%%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u,param);

  %%%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%%
  case { 2, 4, 9 },
    sys = [];

  %%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%%%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% end csfunc

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts]=mdlInitializeSizes(param)

sizes = simsizes;
sizes.NumContStates  = 3;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = param(8:10)';
str = [];
ts  = [0 0];

% end mdlInitializeSizes
%
%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
%
function sys=mdlDerivatives(t,x,u,param)

%                    Extraction des parametres
%                    -------------------------
                     Jn = param(1);
                     Rf = param(2);
                     La = param(3);
                     Ra = param(4);
                     KT = param(5);
                     Ke = param(6);
                     alpha = param(7);
                     temp = alpha+sin(x(3));

                     xdot(1) = -Ra/La*x(1) - Ke/La*x(2) + 1/La*u;
                     xdot(2) = KT/Jn/temp*x(1) - ...
                                                 Rf/Jn/temp*x(2);
                     xdot(3) = x(2);
                     sys = xdot;

% end mdlDerivatives
%
%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
%
function sys=mdlOutputs(t,x,u)
```

```
        sys = x(2);

% end mdlOutputs
```

6.4.3 Autres fonctionnalités de Simulink

Signalons que d'autres outils que les méthodes de résolution numérique sont disponibles avec Simulink. En particulier, la routine `linmod('si_02_02',x0)` permet de d'extraire un modèle d'état du système linéarisé autour du point de fonctionnement spécifié.