Initiation à Matlab
Appliqué à l'automatique et au traitement du signal

TABLE DES MATIERES

1. Introduction

1.1. Qu'est ce que Matlab ?

Matlab est un langage de programmation de haut niveau destiné au calcul scientifique. On le trouve dans les applications de :

- calcul,
- développement d'algorithmes,
- modélisation et simulation,
- analyse et visualisation de données,
- création de graphiques scientifiques,
- création d'application avec interfaces utilisateurs.

Il existe un grand nombre de *toolboxes*, familles de fonctions étendant les fonctions de base de Matlab à un certain type de problème. On trouve ainsi des *toolboxes* dans les domaines du traitement du signal, de la commande des systèmes, des réseaux de neurones, de la logique floue, des ondelettes, de la simulation, etc…

1.2. Démarrer Matlab

1. Trouvez Matlab sur votre ordinateur.
2. Démarrez le programme. Un certain nombre de fenêtres apparaissent. Les plus importante sont *l'interpréteur en ligne* et la fenêtre *Help*. Si cette dernière n'apparaît pas, dans l'onglet *Help* sélectionnez *Matlab Help*.

C'est tout ; l'aide en ligne est extrêmement conviviale et permet d'acquérir tout seul la maîtrise du logiciel…

Ce manuel se propose tout de même de décrire l'utilisation du logiciel pour une utilisation en automatique et en traitement du signal.

1.3. Arrêter Matlab

Taper *quit* ou *exit* ou fermez les fenêtre.

1.4. A propos des versions

Le présent manuel a été rédigé avec Matlab 6.0 sous les yeux. La plupart de ce qui est écrit dans ce manuel s'applique néanmoins à Matlab 5.3 et versions antérieures.

2. Matlab comme interpréteur en ligne

2.1. Premiers pas

Au lancement de Matlab, une fenêtre s'ouvre et propose le prompt >> : on est en présence de l'interpréteur en ligne.

Premier essai

Taper *2+2* et appuyez sur *envoi*. L'affichage propose alors :

```
>> 2+2
ans =
        4
>>
```

Bravo : Matlab sait résoudre des opéra tions

Deuxième essai

Taper *a=2* et *envoi*. L'affichage propose :

```
>> a=2
a =
        2
>>
```

Bravo : Matlab sait manipuler des objets.

Pour travailler en silence : taper *a=2;* L'affichage propose :

```
>>a=2;
>>
```

Troisième essai

Taper *i^2*. La réponse est :

```
>> i^2
ans =
        -1
>>
```

Ouf. Matlab sait manipuler des nombres complexes voire compliqués.

2.2. Un peu de méthode

Choisir un répertoire de travail

Avant toutes choses, choisir un répertoire de travail : c'est dans celui -ci que vous stockerez vos fichiers personnels. Dans la barre de menu, utilisez *File/Set Path*…

Les expressions

La programmation en Matlab consiste en l'enchaînement *d'expressions* composées de *variables*, de *nombres*, de *opérateurs* et de *fonctions*.

- Variables

Il n'est pas nécessaire de déclarer le type ou la dimension de s variables : Matlab choisit selon le contexte.

Par exemple :

```
a=2.37
```

Crée la variable a de dimension 1 et lui attribue la valeur 2.37.

Autre exemple :

```
>> a(2)=1.5
a =
    2.37    1.5
>>
```

reprend la variable a et modifie sa dimension : c'est maintenant un vecteur de dimension 2 dont la deuxième valeur vaut 1.5

- Nombres

Exemples de nombres :

```
3          -99        0.001
1.6e-20    1i         -3.14j
3e5i
```

Les nombres ont jusqu'à 16 chiffres significatifs et doivent rester entre 10^{-308} et 10^{+308}.

- Matrices

Exemple :

```
>> mamatrice=[1 2 3;4,5,6;7 8,9]
mamatrice =
    1    2    3
    4    5    6
    7    8    9
>>
```

Les données sont entrées en ligne, séparées par des virgules ou des espaces. Les lignes sont séparées par des point virgules.

- Opérateurs

+	Addition	-	Soustraction
*	Multiplication	/	Division
^	Puissance	./	Division élément par élément (matrices)
'	Conjugué et transpose	.*	Multiplication élément par élément (matrices)
.'	Transposée		

Pour la liste des opérateurs disponibles, tapez *help ops*.

- Fonctions

Pour obtenir une liste des fonctions mathématiques élémentaires taper :

```
>>help elfun
```

Pour obtenir une liste des fonctions matricielles élémentaires taper :

```
>>help elmat
```

Pour obtenir la liste des fonctions d'une *toolbox* spécifique, par exemple *signal toolbox* (traitement du signal) taper :

```
>>help signal
```

Bien sûr il est aussi possible de définir ses propres fonctions, voir le chapitre 3.4 page 14.

- Constantes

Un certain nombre de fonctions prédéfinies (sans arguments) r envoient les valeurs de constantes usuelles :

pi	3.14159265358979
i	unité imaginaire $\sqrt{-1}$
j	pareil que i
eps	précision relative en virgule flottante : 2^{-52}
realmin	plus petit nombre en virgule flottante : 2^{-1022}
realmax	plus grand nombre en virgule flottante : 2^{+1022}
Inf	Infini (résultat de 1/0 par exemple)
NaN	Not A Number (résultat de 0/0 par exemple)

Attention : il est possible de réaffecter les valeurs de ces constantes (éviter d'utiliser i et j comme indice dans les boucles de calcul, notamment).

Un petit jeu amusant : essayer *Inf+Inf, Inf/0, Inf-Inf, realmax+eps, realmax*(1+eps)*, etc…

Un peu plus sur les matrices

- Opérations sur les matrices et fonctions matricielles

Les additions, soustractions transposées et autres inversions de matr ices (taper *A^-1*) sont triviales.

Un élément de la matrice s'appelle avec les parenthèses. Par exemple *A(1,3)* renvoie l'élément de la première ligne, troisième colonne. Attention : pas de colonne 0.

La concaténation de matrice est triviale, attention cepen dant à respecter les dimensions. (Exemple : *Z=[A,B;C,D]*)

L'opérateur : s'utilise de différentes manières :

- pour créer des listes. Par exemple `1:4` renvoie `[1 2 3 4]` et `1:2:5` renvoie `[1 3 5]`.
- Pour faire appel à des sous-parties de matrices. Par exemple `A[:,3]` renvoie la troisième colonne de A et `A[1:2:k,3]` renvoie les k premiers éléments impairs de la troisième colonne de A…
- Pour supprimer une ligne ou une colonne. Par exemple `X(2,:)=[]` supprime la deuxième ligne (et redimensionne donc la matrice)

- Stocker des matrices dans des fichiers

On utilise souvent des matrices de grandes dimensions (résultats de mesures par exemple). Les données doivent alors être écrites sous forme de tableau, séparées par des blancs et des fins de lignes dans un fichier.dat

Par exemple un fichier *donnees.dat* crée par n'importe quel éditeur de texte contient :

```
1.001     2.34    0.12
2.44431   2.2     2.0091
3.1       47      4.31112
```

La commande *load donnees.dat* charge le fichier sous matlab et crée la matrice *donnees*.

A l'inverse, disposant sous Matlab d'un tableau *résultat* par exemple, il est possible de le stocker dans un fichier par la commande *save fichier_resultat resultat*. Il est alors stocké sous forme binaire dans un fichier *fichier_resultat.mat* et récupérable par la comm ande *load*. Pour créer un fichier texte[1] (ressemblant au fichier *donnees.dat* par exemple) utiliser la commande *fwrite*.

Les structures

Matlab permet de manipuler des données sous formes de *structures* : les données sont rangées dans une arborescence et access ibles par des *champs*. L'exemple suivant permet de comprendre comment créer une structure simple :

```
>> toto.tata=[1 2 3 4]
toto =
    tata: [1 2 3 4]
>> toto.titi=2
toto =
    tata: [1 2 3 4]
```

[1] Par exemple pour être traité par un autre logiciel comme Excel…

```
    titi: 2
>> toto
toto =
    tata: [1 2 3 4]
    titi: 2
>>
```

La structure toto contient deux champs : *tata* (qui est un vecteur de dimension 4) et *titi* (qui est un entier). Pour accéder à un élément de la structure, on utilise le point :

```
>> toto.tata
ans =
     1     2     3     4
>>
```

Bien sûr, on peut imbriquer plus de deux champs et inclure des variables de nature différentes. Ainsi le "papa du papa du papa de mon papa était un fameux pioupiou" [2] donne :

```
>>papa.papa.papa.papa='fameux pioupiou';
```

[2] Paroles et musique Boby Lapointe

3. Ctrl-C Ctrl-V, M-files, fonctions : rationalisons notre travail

3.1. Historique

Les flèches de direction (←↑→↓) permettent de retrouver et modifier les lignes de commandes précédentes.

La version 6 de Matlab propose une fenêtre *command history* qui garde en mémoire les commandes passées dans la session courante et dans les sessi ons précédentes.

3.2. Copier-Coller (Ctrl-C Ctrl-V)

En pratique on tape les commandes directement dans la fenêtre de l'interpréteur pour de petits calculs. Lorsque cela devient plus sérieux il est beaucoup plus rentable d'utiliser un éditeur de texte pour entrer les expressions (séparées par des retours à la ligne) puis de copier-coller dans la fenêtre Matlab. L'éditeur de texte le plus commode est alors celui de Matlab : dans la barre de menu *File / New / M-file*.

3.3. Scripts

L'étape suivante consiste à créer des *scripts* : il s'agit de fichiers textes comprenant un ensemble de commandes. Pour cela le plus simple consiste à utiliser l'éditeur/Debugger de Matlab.

Créons par exemple un nouveau script :

File / New / M-file

Puis écrivons la séquence de commande qui nous intéresse :

```
%Mon premier script
a=2;
b=3;
a+b
```

Sauvegardons ce script dans un fichier : File / Save As puis choisir un nom de fichier, par exemple *essai*. Notons que Matlab ajoute l'extension *.m* par défaut.

Pour lancer le script depuis Matlab il suffit de taper son nom *essai* sans l'extension *.m*. L'affichage propose alors :

```
>> essai
ans=
```

```
        5
>>
```

Deux remarques :

1. La ligne précédée du caractère % n'a pas été interprétée : il s'agit d'une ligne de commentaires.
2. Seule les expressions qui ne sont pas terminées par un point virgule (;) sont visibles dans la fenêtre de l'interpréteur.

3.4. Fonctions

Les M-files permettent aussi de définir des fonctions. A la différence des scripts, les variables définies dans une fonction sont locales. Examinons par exemple le m -file comportant les lignes d'instruction suivantes :

```
function r = rank(A,tol)
%   RANK Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
%cas où il n'y qu'un argument
if nargin==1
  tol = max(size(A)) * max(s) * eps;
end
%cas où il y a deux arguments
r = sum(s > tol);
```

La première ligne définit le nom de la fonction, le nombre maximum et l'ordre des arguments d'entrée et de sortie.

Les lignes suivantes, précédées du caractère %, sont des lignes de commentaire particulières : elles sont renvoyées lorsqu'on tape *help rank*.

Les lignes suivantes sont le corps du programme. Les variables *r*, *A*, *tol*, *s* sont *locales* à la fonction.

La fonction *rank* peut être appelée de différentes manières (on suppose que la matrice B a été définie ultérieurement) :

```
>> rank(B)
ans=
        2
>> rang=rank(B);
>>rang=rank(B, 1.e-6);
```

Dans le premier cas, la variable de sortie n'est pas nommée : le résultat est stocké dans la variable *ans*.

Dans le second cas, on ne fournit qu'un argument. Le corps du programme a prévu ce cas : la variable *nargin* prend la valeur 1 et on attribue une valeur par défaut à *tol*.

3.5. Instructions et structures de contrôle

Matlab dispose d'un certain nombre d'instructions de contrôle :

- if
- switch
- for
- while
- continue
- break

Pour la syntaxe exacte et pour des exemples, utiliser l'aide en ligne : *help if* (par exemple).

Il est parfois possible de réduire le temps de calcul en utilisant la *vectorisation*. Examinons par exemple les lignes suivantes :

```
A=1:1:100;
for ii=1:1:100
   B(ii)=sin(A(ii));
end;
```

Ces lignes sont strictement équivalentes à :

```
A=1:1:100;
B=sin(A);
```

Le résultat produit est le même que dans la première version, le temps de calcul plus court et la lisibilité améliorée.

3.6. Un peu plus sur la programmation

Le programmeur en Java ou C++ sera rassuré de savoir que le langage de programmation de Matlab est aussi orienté-objet. Le lecteur qui vient de lire ces lignes et qui n'a rien compris est invité à attendre les cours d'informatique.

4. Les figures

Matlab permet de créer des graphiques complexes et d'en contrôler les moindres détails. Ce chapitre donne l'essentiel à savoir ainsi que quelques clefs pour aller plus loin.

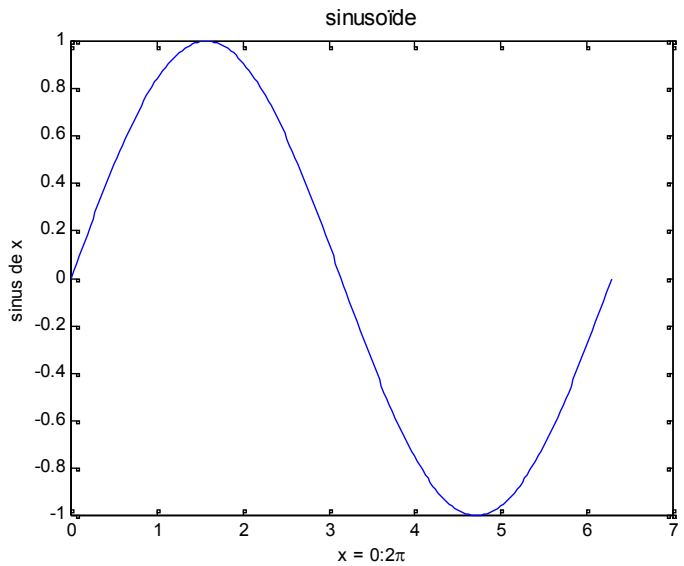
4.1. Tracer des courbes

Créer une figure

La séquence d'instruction suivante crée d'abord une figure puis place les commentaires sur les deux axes et enfin une ligne de titre :

```
>> x=0:pi/100:pi;
>>y=sin(x);
>>plot(x,y);
>>xlabel('x = 0:2\pi');
>>ylabel('sinus de x');
>>title('sinusoïde','Fontsize',12);
```

La figure suivante apparaît dans une nouvelle fenêtre :



Style et couleurs des lignes

Il est possible de choisir le style et la couleur des lignes avec la commande *plot* : *plot(x,y,'color_style_marker')*. *'color_style_marker'* est une chaîne de caractère comportant jusqu'à quatre caractères :

- Couleur : 'c', 'm', 'y', 'r', 'g', 'b', 'w'
- Style de ligne : '-', '--', ':', '-.', 'none'
- Type de ligne : '+', 'o', '*', 'x', 's' (square), 'd' (diamond), etc…

La variable *'color_style_marker'* peut par exemple prendre la valeur *'y-o'* pour que le tracé soit jaune et constitué de ronds reliés par un trait continu.

Superposer des courbes

La fonction plot admet un nombre indéfini d'arguments et permet de superposer dans une même figure plusieurs courbes.

Essayer par exemple : `plot(x,y,x,y*2)`

Variables complexes

Si Z est un vecteur de nombre complexes la fonction `plot(Z)` est équivalente à `plot(real(Z),imag(Z))`.

Ajouter des courbes à une figure existante

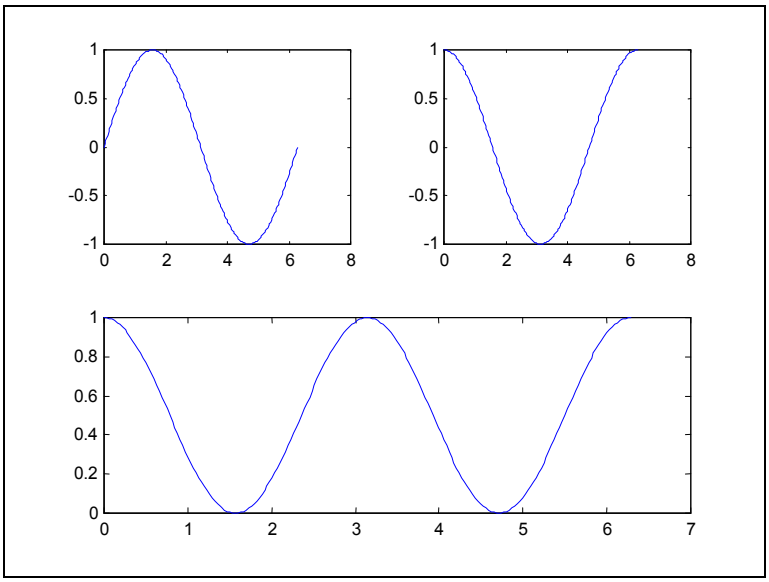
La commande `hold on` permet de conserver la figure courante : le tracé suivant ne se fait pas dans une nouvelle fenêtre mais se superpose au tracé précédent (en adaptant éventuellement les échelles).

Plusieurs courbes dans une même fenêtre

La commande subplot permet de partitionner la fenêtre de la figure et de tracer plusieurs courbes. Considérons l'exemple suivant :

```
» x=0:pi/100:pi*2;
» subplot(2,2,1);plot(x,sin(x);
» subplot(2,2,1);plot(x,sin(x));
» subplot(2,2,2);plot(x,cos(x));
» subplot(2,1,2);plot(x,cos(x).*cos(x));
```

La figure créée est reproduite ci-après :



Le premier subplot partitionne en deux lignes et deux colonnes et place le pointeur sur la première case. Le second subplot partitionne de la même manière et place le pointeur sur la deuxième case. Le troisième subplot partitionne en deux lignes et une seule colonne et place le pointeur sur la deuxième case (la deuxième ligne).

Placer des titres et des commentaires

Les commandes xlabel, ylabel, title, text permettent de commenter les axes x, y, de placer un titre au graphique ou de placer un commentaire n'importe où sur le graphique.

4.2. Un peu plus sur les graphiques

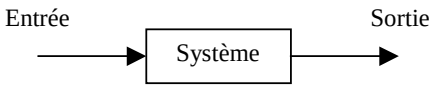
Une fois le graphique créé, il est possible de rajouter des titres, des comment aires, etc... en utilisant la fonction Tools / Edit Plot de la barre de menu de la fenêtre graphique.

De manière plus générale, et cela sort du cadre de ce manuel, les données décrivant l'ensemble d'un graphique sont stockées sous la forme d'une *structure*. Il est donc possible de pointer directement sur un élément d'un graphique (axe, titre, donnée, etc…) et de le modifier en utilisant des fonctions spécialisées. Pour plus de détails consulter l'aide de Matlab.

5. Matlab pour l'automaticien[3]

5.1. Rappels

Un *système* au sens de l'automaticien est composé de deux flèches et d'un carré :



Une grandeur physique d'entrée a un effet sur un système et provoque l'évolution d'une grandeur de sortie.

Pour que tout soit plus simple, on suppose que :

- la grandeur d'entrée est une grandeur physique continue e(t),
- idem pour la grandeur de sortie s(t),
- le système est *continu linéaire invariant*, c'est à dire qu'une équation différentielle à coefficients constants relie les grandeurs d'entrée et de sortie[4].

Le système est représenté par sa fonction de transfert F(p). (F(p) est une fraction rationnelle de la variable complexe p).

Quelle que soit l'entrée e(t) on peut écrire :

$$S(p) = F(p).E(p)$$

où E(p) et S(p) sont respectivement la transformée de Laplace de l a grandeur d'entrée e(t) et de la grandeur de sortie s(t).

Une autre représentation est l'équation d'état : le système est entièrement décrit par la connaissance de quatre matrices A, B, C et D et par l'utilisation d'une variable supplémentaire X (de dimension 1 ou plus) appelé *vecteur d'état*. La relation liant l'entrée à la sortie est alors :

$$\begin{cases} \dot{X}(t) = A \cdot X(t) + B \cdot e(t) \\ s(t) = C \cdot X(t) + D \cdot e(t) \end{cases}$$

5.2. Les objets : description des systèmes linéaires

Matlab permet de décrire des systèmes linéaires sous forme de représentation d'état ou sous forme de fonction de transfert et de passer d'une forme à l'autre.

---

[3] Les fonctions décrites dans ce chapitre font partie de *control toolbox* et de *simulink* tous deux installés sur la plupart des ordinateurs du département DAS.

[4] Bien sûr il est possible de représenter des systèmes beaucoup plus compliqués, le cours d'automatique s'en chargera.

Fonction de transfert

Soit par exemple le système représenté par la fonction de transfert suivante :

$$F(p) = \frac{p^2 + 3 \cdot p + 4}{p^3 + 2 \cdot p + 1}$$

La fonction *tf* de *control system toolbox* permet de créer l'objet matlab associé à cette fonction de transfert :

```
>> SYStf=tf([1 3 4],[1 0 2 1])
 Transfer function:
s^2 + 3 s + 4
-------------
s^3 + 2 s + 1
```

(On notera que conformément à l'usage anglo-saxon la variable de Laplace est notée s)

Une autre manière est de créer la variable de Laplace s et de l'utiliser comme selon la séquence ci-dessous :

```
>> s=tf('s')
 Transfer function:
s
 >> SYStf=(s^2+3*s+4)/(s^3+2*s+1)
 Transfer function:
s^2 + 3 s + 4
-------------
s^3 + 2 s + 1
```

Les fonctions de transfert, une fois créées se manipulent simplement comme le montrent les exemples suivants :

```
>>FTBF=SYStf/(1+SYStf);
>>S=SYStf*1/s;
etc…
```

Représentation d'état

Une fois créées les matrices A, B, C et D du modèle d'état, on crée l'objet *Matlab* associé au système par la fonction ss (comme *state space*) :

```
>> SYSss=ss(A,B,C,D);
```

Passage d'une représentation à l'autre

Les fonctions ss et tf s'utilisent aussi pour passer d'une représentation à l'autre. Par exemple :

```
>>[A,B,C,D]=ss(SYStf);
```

calcule la représentation d'état à partir de la fonction de transfert.

Systèmes discrets

Cela sort un peu du cadre de ces rappels mais il est bien évidemment possible de créer des modèles pour des systèmes discrets ou de discrétiser des modèles continus, etc… Pour plus de détails voir l'aide en ligne de Matlab.

5.3. Analyse des systèmes et simulation

Analyse d'un système

Un système linéaire étant représenté sous Matlab il peut être intéressant d'en étudier les propriétés.

La fonction *bode* trace le diagramme de Bode (gain et phase) du système.

La fonction *nichols* trace le diagramme de Black-Nichols. La fonction *ngrid* superpose l'abaque de Black (gain et fréquence en boucle fermée) sur le diagramme.

La fonction *pole* calcule les pôles du système.

La fonction *zero* calcule les zéros du système.

La fonction *damp* calcule les pôles et, pour chacun, son amortissement et sa fréquence.

La fonction *pzmap* trace les pôles et zéros dans le plan complexe.

Réponse d'un système

La fonction *impulse* trace la réponse impulsionnelle d'un système.

La fonction *step* trace la réponse à une entrée en échelon unitaire.

Un outil avancé : ltiview

La fonction *ltiview* (exemple : ltiview(SYStf)) ouvre une fenêtre permettant de tracer les diagrammes de Bode, de Black, les réponses impulsionneles ou indicielles, etc…

Un outil avancé : sisotool

La fonction *sisotool* (pour Single Input Single Output) ouvre une fenêtre permettant d'effectuer diverses manipulations sur le système bouclé :

- manipulation de la boucle fermée en utilisant les techniques du lieu des racines,
- choix de correcteur par manipulation des diagrammes de Bode,
- addition de pôles et de zéros au compensateur,
- analyser les réponses en boucles fermées,
- ajuster les marges de gaine et de phase,
- effectuer les conversions entre modèle continu et modèle discret.

5.4. Simulink

Introduction

Simulink est un outil pour *Matlab* (*toolbox*) possédant une interface graphique puissante permettant de modéliser, simuler et analyser des systèmes dynamiques.

Pour lancer Simulink taper *simulink* dans la fenêtre de l'interpréteur Matlab. Une liste de tous les blocks disponibles apparaît dans une fenêtre *Simulink Library Browser*.

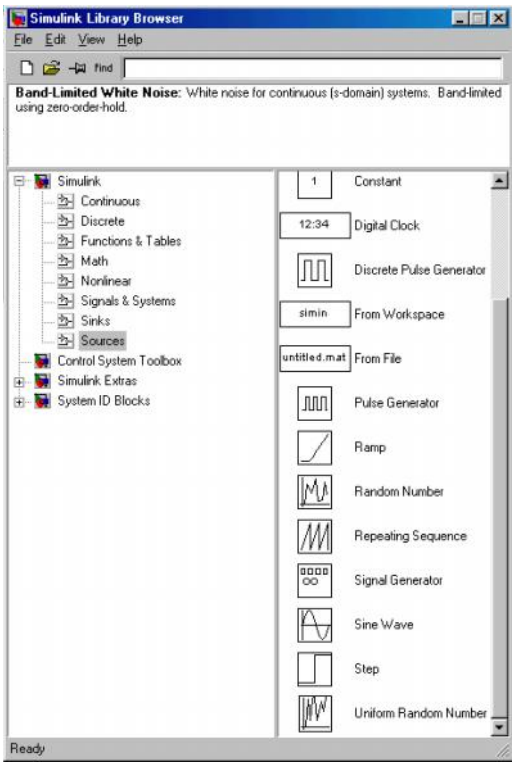
Pour créer un nouveau modèle choisir *File / New / Model* dans la barre de menu. Une nouvelle fenêtre *Untiteld* apparaît.

On construit le modèle en sélectionnant des blocs depuis le *Library Browser* et en les collant dans la fenêtre du modèle.

Un exemple très simple

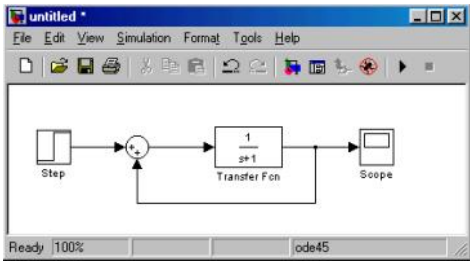
L'exemple qui suit permet de se familiariser avec Simulink. On souhaite simuler un système du premier ordre bouclé avec un retour unitaire et excité par un échelon.

1. Lancer Matlab
2. Choisir le bon répertoire de travail, par exemple *C:/etudiants/toto*
3. Dans Matlab taper *simulink*, une fenêtre *Simulink Library Browser* apparaît



4. Ouvrir un nouveau modèle : *File / New / Model*. Une nouvelle fenêtre *Untiteld* apparaît.

5. Dans la fenêtre *Simulink Library Browser*, sélectionner le bloc *Step* de la famille *Sources*. Le faire glisser dans la fenêtre *Untitled*.
6. Répéter l'opération avec un bloc *Sum* de la famille *Math*, puis un bloc *Transfer Fcn* de la famille *Linear* et enfin un bloc *Scope* de la famille *Sinks*.
7. Reliez les blocs par des flèches en utilisant le glisser-déplacer de la souris. Vous obtenez alors le modèle suivant :



8. Pour préciser les paramètres des blocs double-cliquez sur ceux-ci et utiliser les menus.
9. Pour définir les paramètres de la simulation (durée, méthode d'intégration, etc…), choisissez *Simulation / Simulation Parameters* depuis la barre d'outils.
10. Pour lancer la simulation, choisissez *Simulation / Start* depuis la barre d'outils.

6. Matlab pour le traitement du signal

6.1. Les fonctions de base du traitement du signal

On ne rappellera pas ici les fondamentaux du traitement du signal qui sont supposés connus. Rappelons uniquement que le traiteur du signal s'occupe de *filtrer* des *signaux*. Les fonctions de base du traitement du signal sont donc, du moins avec *Matlab* :

- la fonction *fft* qui calcule la transformée de Fourier d'une séquence,
- la fonction *filter* qui applique un filtre numérique à une séquence.

6.2. Les signaux numériques

Représentation des signaux

Les signaux sous Matlab sont constitués de séries de chiffres stockés sous forme de vecteurs. Par exemple :

```
>> x=1:1:20;
>>x=x';
>>y=[x 2*x x/pi];
```

x est un vecteur colonne composé de vingt éléments. y est composé des signaux x, 2*x e t x/pi.

Signaux courants

Généralement les signaux dépendent du temps. Il est alors commode de commencer par générer un signal représentant le temps. Par exemple pour une durée de 0 à 1 seconde, par intervalles de 1ms :

```
>> t=0:0.001:1;
```

Ci-après les signaux les plus courants et les manières de les obtenir :

- sinusoïde :

```
>>f=50;
>>signal=sin(2*pi*f*t);
```

- sinusoïde bruitée (bruit blanc) :

```
>>signal_bruite = signal + 0.5*randn(size(t));
```

- impulsion :

```
>>y = [1; zeros(99,1)];
```

- échelon unitaire :

```
>>y = ones(100,1);
```

- rampe :

```
>>y = t;
```

- etc…

```
>>y = t.^2;
```

- signal carré (période : 2π)

```
>>y = square(4*t);
```

Importer des signaux

Dans la pratique il est courant de travailler avec des signaux provenant de l'extérieur (résultats de mesures, enregistrements, etc…)

- Si les signaux proviennent déjà de Matlab (extension .m) ou sont sous forme texte (extension .txt) utiliser la fonction *load*.
- Si les signaux sont des sons : utiliser *auread*, *auwrite* (extension .au), *wavread* et *wavwrite* (format .wav)
- Si les signaux sont des images : utiliser *imread* et *imwrite* (exemple pour une image en format bitmap : *imread(image.bmp,'bmp'*). Les formats d'image les plus courants sont reconnus.
- De manière plus générale, un "assistant d'importation" facilite le travail pour importer des données, sons, image, etc... Il est accessible depuis la barre d'outils de Matlab : *File / Import data*.

Analyser les signaux

Plusieurs manières d'analyser les signaux (par exemple des sons) :

- les écouter (*wavplay*, *soundsc*),
- les voir si c'est des images (*image*),
- visualiser leurs évolutions temporelles (*plot*),
- calculer leurs transformées de Fourier (fonction *fft*)
- visualiser directement leurs densités spectrales de puissance (*psd* pour power spectral density, *psdplot* pour tracer la densité spectrale de puissance)

6.3. Les filtres numériques

La convolution de signaux

La fonction de base du filtrage est la convolution. La fonction Matlab qui réalise la convolution est *conv*. Voici un exemple d'application :

```
>>conv([1 1 1],[1 1 1])
ans =
```

```
    1    2    3    2    1
```

Filtres

La sortie y(n) d'un filtre h en fonction de l'entrée x(n) est généralement calculée à partir des transformées en z :

$$Y(z) = \frac{b(1) + b(2)\cdot z^{-1} + ... + b(n_b)\cdot z^{n_b-1}}{a(1) + a(2)\cdot z^{-1} + ... + a(n_a)\cdot z^{n_a-1}} X(z)$$

La fonction filter

C'est la fonction qui permet, à partir d'un signal d'entrée de calculer la sortie d'un filtre donné. L'exemple suivant est suffisamment explicite :

```
>>B=[1 2 ];       %polynome dénominateur (ordre croissant des coefficients)
>>A=[1 2 3];      %polynome numérateur (ordre croissant des coefficients)
>>x=[1 2 3 4 5 6 7 8 9 10];        %signal d'entrée
>>y=filter(B,A,x)          %calcul de la sortie du filtre
y =
   1    2    0    4    5   -6   16    8  -39   82
>>
```

Le filtre réalise l'algorithme suivant :

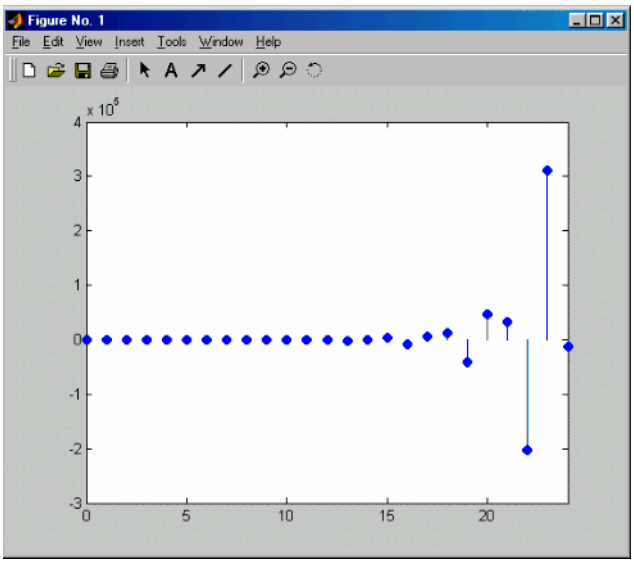
$$A(1)*y(n) = B(1)*x(n) + B(2)*x(n-1) + ... + B(n_b+1)*x(n-n_b)$$
$$- A(2)*y(n-1) - ... - A(n_a+1)*y(n-n_a)$$

Les conditions initiales sont supposées nulles dans cet exemple. Pour des conditions initiales non nulles, consulter l'aide en ligne.

Réponse impulsionelle d'un filtre

Pour obtenir la réponse impulsionelle d'un filtre il est possible d'utiliser la fonction *filter* déjà rencontrée ou bien d'utiliser la fonction *impz* qui calcule et trace la réponse impulsionelle. Avec l'exemple précédent :

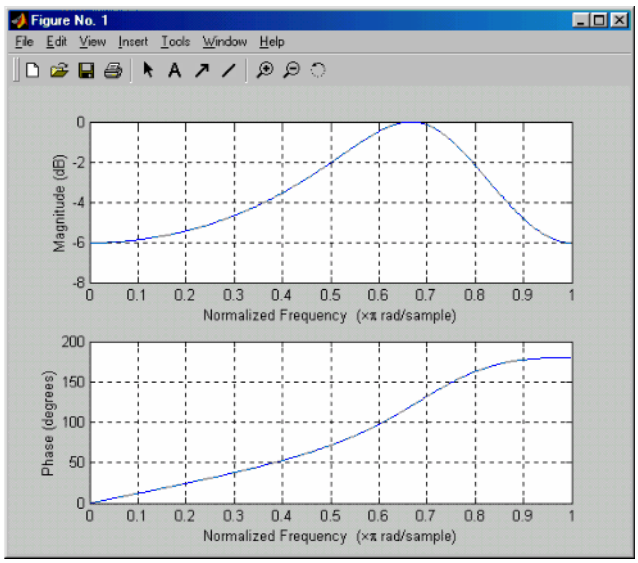
```
>>imp(z(A,B);
```



Réponse fréquentielle d'un filtre

La réponse fréquentielle d'un filtre s'obtient avec la fonction *freqz* (fréquences normalisées).

```
>>freqz(B,A)
```



Pôles et zéros d'un filtre

La fonction *zpoles* trace les pôles et les zéros d'un filtre numérique. (En l'appliquant à notre exemple on comprend pourquoi sa réponse impulsionelle diverge…).

6.4. Les signaux et les filtres continus

Se reporter au chapitre 5.2.

6.5. Les outils avancés

Se reporter à l'aide de *Matlab Signal Toolbox*…

Références

http://www.mathworks.com/ LA société Mathworks, qui développe et commerci alise Matlab