Introduction à MATLAB & SIMULINK

Environnement et programmation avec MATLAB,

Introduction à Simulink et Toolbox,

Création d'Interfaces Graphiques Utilisateur (GUIs) avec MATLAB.

Ridha BOUALLEGUE
Sup'Com
2004

Environnement MATLAB

MATLAB est un environnement dédié à l'Automatique et au Traitement du Signal existant sous Windows et Unix. Il inclut un langage interprété, un éditeur de scripts, un éditeur de schémas blocs, et la possibilité de construire un interface graphique interactif simple (fenêtres, souris, zones éditables, labels, boutons, …).

1. Fonctions et scripts MATLAB

Dans un M-file (extension .m) de Matlab, on range des scripts exécutables par l'interpréteur, ou un ensemble de '*functions*' Matlab. Le nom du fichier doit reprendre le nom de la fonction principale placée en début de fichier, les autres fonctions sont locales.
Voici par exemple le contenu d'un fichier nommé **newstats.m** :

```
function [avg,med] = newstats(u)
% newstats finds mean and median with internal
% functions.
n = length(u) ;
avg= mean(u,n) ;
med=   median(u,n) ;

function a = mean(v,n)
%calculate average.
a=sum(v)/n ;

function m = median(v,n)
%calculate median.
w = sort(v) ;
if rem(n,2) == 1
   m = w((n+1)/2) ;
else
   m = (w(n/2)+w(n/2+1))/2 ;
end
```

En utilisant la fonction Editor/Debugger, MATLAB permet à l'utilisateur de créer de nouvelles fonctions à partir des opérateurs arithmétiques et logiques du langage, et des fonctions déjà intégrées et parfois compilées. Pour la plupart des fonctions déjà utilisées, telles que `step, bode,` il existe des M-files step.m, bode.m que l'on peut consulter à titre d'exemples avec l'éditeur MATLAB.

Quelques commandes :

- **edit newstats** démarre l'éditeur debugger pour newstats.m
- **edit** ouvre l'éditeur
- **debugger :** depuis le menu de l'éditeur, il est possible de placer des breakpoints dans les functions et scripts et d'exécuter pas à pas. On visualise également la valeur actuelle des variables.
- **profiler :** voir menu également.
- **path :** consultable et modifiable depuis MATLAB
- **type bode** liste le contenu du M-file bode.m
- **type abs** retourne « *built-in function* »
- **what** liste les M-files chargés dans l'espace de travail.
- **who,** les variables du workspace de Matlab

Ainsi, une méthode de travail consiste à créer un M-file au moyen de la commande `'edit'`, après avoir placé le répertoire de travail dans la variable `path`, puis à exécuter (commande Run de l'éditeur, ou commande Debug) la nouvelle fonction programmée.

Afin de programmer en « Matlab », on énumère maintenant les types de données disponibles, les opérateurs élémentaires, les structures de contrôle, le passage d'arguments ,etc ...

2. Les types de données

Il existe 6 types de données dans MATLAB, tous organisés en tableaux multidimensionnels (les tableaux à deux dimensions sont appelés matrices, d'où le nom MATLAB ou MATrix LABoratory) :

double, char, sparse, uint8, cell et struct.

Les deux premiers types double **et** char **sont les plus utilisés.**
La classe numeric **regroupe** sparse**,** uint8 **et** double**, il existe une classe suppléméntaire** UserObject **où l'utilisateur peut créer son type.**

Class	Example	Description
: double	**[1 2 7 ; 3 4 pi]** **5+6i, ou 5+6*i** **ou 5+6j,** **10 :-0.5 :-sqrt(2)**	**Scalaires, vecteurs, et matrices. Complexes : faire i^2, ou k=sqrt(-1)**
: char	**'Hello'** **'\leftarrow flêche'** **gtext(ans)**	**Chaînes de caractères**
: sparse	**speye(5)**	**Matrices creuses**
: cell	**{17 'hello' eye(2)}** **ans{2}** **'hello'**	**Structures rangeant des données de taille et format variable**
: struct	**a.day=12 ;** **a.color='red' ;** **a.mat=magic(3) ;** **a**	**Structure**
: uint8	**uint8(magic(3))**	**Stockage sur 8 bits**
: UserObject	**G=inline('sin(x)')**	**G=Fonction sinus**

Interpréter les exemples suivants

```
tableau = debut :pas :fin ;
tableau = debut :fin ; %(pas =1 par défaut)
tableau = [a b c  d ; e f g h]
```

```
» m = magic(3) % matrice magique 3x3
m =  8     1     6
     3     5     7
     4     9     2
» u=uint8(magic(3)) ;
» whos
  Name      Size         Bytes  Class
  m         3x3             72  double array
  u         3x3              9  uint8 array
 Grand total is 103 elements using 1121 bytes
```

```
G=inline('exp(-pi*m/sqrt(1-m^2))','m') ;
G(0.2) → 0.526
G=inline('exp(-pi*m./sqrt(ones(size(m)-m.^2))','m') ;
X=0 :.1 :1 ;
plot(X,G(X))
rang = inline('rank(x)')  % francise la commande rank
>> rang([0 1 ;-2 -3]) → 2
```

```
Cellule={'Bonjour,' 'nous vivons une époque moderne'};
Cellule{2} → ans= nous vivons une époque moderne
Str=Cellule{2} ;
Str(5 :11) → ans= vivons
Str(11 :-1 :5) → ans = snoviv
length(Cellule{1})→ ans = 8
gtext(Cellule) %imprime sur deux lignes dans la figure
```

3. Opérateurs Arithmétiques et Logiques

MATLAB propose trois types d'opérateurs :

- **Les opérateurs arithmétiques permettent de spécifier les traitements sur les scalaires, vecteurs et matrices :**
  - **+ , - , * , / , ^ , et ' sont les opérateurs de base,**
  - **on trouve aussi : .*, ./ et .\ , .^, .' .**
  - **l'opérateur : des tableaux (A(1, :) = première ligne de A)**
  - **Un ensemble de fonctions utilitaires sont associées aux tableaux : `size`, `length`, `max`, `min`, `sum`, `prod` ,`abs`, … elles sont généralement compilées (built-in) le source n'est plus visible (essayer type abs par exemple)**
  - **Les tableaux sont stockés linéairement colonne par colonne, dans la mémoire sous la forme d'un vecteur. Ainsi, pour la matrice A=magic(3), A(1,2) et A(4) désignent la même composante (valant 1).**

- **Les 6 opérateurs de relation :** **> , >= , < , <= , == , ~= s'appliquent sur deux scalaires, sur les éléments correspondants de deux tableaux de même dimension, sur un tableau et un scalaire. Un tableau booléen résultat est élaboré, avec la valeur 1 là où la relation est vraie, 0 ailleurs.**

```
A= [0 ,1 ;-2 ,-3] ;
B= [1 ,0 ; 0   ,1] ;
A<B retourne [1 ,0 ; 1, 1]
```

- **Fonction `find` :** **appliquée à un tableau, en spécifiant une condition, cette fonction retourne la liste des indices pour lesquels la condition est vérifiée :**

Exemple : Analyser le calcul suivant du temps de réponse à 5 % sur la réponse indicielle de `process`.

```
[y,t] = step(process) ;
vfri = y(length(y)) ;
index = find(abs(y- ... vfri*ones(size(y)))>0.05*vfri) ;
tr5 = t(max(index))
```

`Remarquer` `que ... est le continuateur de ligne`

- **Les 3 opérateurs logiques** **permettent de constituer les expressions logiques évaluées à 0 pour FAUX et 1 pour VRAI.**
Ce sont : & AND , | OR , et ~ NOT.

Une variable numérique non nulle est évaluée à VRAI, une variable nulle à FAUX (comme en C):

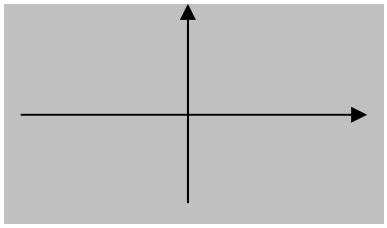
```
U=[1 0 2 3 0 5] ;
V=[5 6 1 0 0 7] ;
U & V → 1 0 1 0 0 1
U | V → 1 1 1 1 0 1
et ~U = 0 1 0 0 1 0
```

- **il existe un ensemble de fonctions logiques built-in, telles que**
  - ✓ **xor (a, b)**
  - ✓ **all (U < 3) retourne 1 si vrai pour toutes les composantes de U**
  - ✓ **any (U < 3)**
  - ✓ **A=[0 1 2 ;3 5 0]**
    all (A) retourne quoi ?
  - ✓ **A=[ ] → isempty (A) retourne vrai**
  - ✓ **B=2 ; →isa(B , 'double') est vrai**

EXEMPLES D'OPERATIONS SUR LES TABLEAUX :

Quel est le résultat des instructions suivantes ?

- `A=magic(3)`
- `B=[A,A]`
- `C=[A ;2*A]`
- `A( :,1)`
- `size(A), size(A,2)`
- `A(2,1), et A(2)`
- `A(1:2,2:3)`
- `A(2 , :  ) = []`
- `D=A([1 3 2],[2 1 3])`



4. Flow Control (Structures de contrôle)

if logical expression
Statements
else…ou elseif … end

```
function [s]=seuil(e)% fichier seuil.m
% réalisation de la N.L. seuil plus saturation
global VSAT
global SEUIL
if abs(e)>VSAT+SEUIL
   s=sign(e)*VSAT
elseif abs(e)<SEUIL
   s=0
else s=e-sign(e)*SEUIL
end
```

Remarque : VSAT et SEUIL sont définies comme des variables globales, cette déclaration doit être répétée dans le script appelant.

```
for i=start :increment :end,
    Statements
end
```

```
global VSAT % dans la fenêtre MATLAB
global SEUIL
VSAT=15
SEUIL=1
X= -20 :.1 :20 ;
s=[] ;
for i = X,
 s=[s seuil(i)] ;
end
plot(X,s)
```

Est valide également :

```
    for  i = A, abs(i), end % A matrice
    for  i = [1 3  5 7 11], i, end
```

```
while expression
  Statements
end
```

```
% Lecture de la position x,y de la souris
Button=1 ;
while Button==1,
  [x,y,Button]=ginput(1) ;
end
```

```
switch expression (scalar or boolean)
case value1
  Statements
case value2
  Statements
otherwise
  Statements
end
```

```
switch var
        case 1
        disp('1')
        case{2,3,4}
        disp('2 ou 3 ou 4')
        otherwise
        disp('autre chose')
end
```

Remarque : break permet de sortir du bloc

5. String Evaluation

Il est possible d'exécuter des chaînes de caractères entrées par l'utilisateur durant l'exécution « on the fly » :

- **eval(string) : évalue une chaîne, par exemple**

```
% génération d'une matrice de Hilbert
t='1/(i+j-1)' ;
for i = 1:n
      for j=1:n
            a(i,j)=eval(t) ;
      end
end
```

ou bien

```
>>eval('t=clock')
```
% exécute une instruction complète

- **feval(string,value) : évalue une fonction dont le nom est dans la chaîne 'string'**

Exemple :

```
fun = ['sin' ;'cos' ;'log'] ;
k=input('Choose function number : ') ;
x=input('Enter value : ') ;
feval(fun(k,: ),x)
```

ou encore, génération des variables v1,v2,…v10, vi=i^2

```
for i=1 :10
     eval(['v',int2str(i),'=i^2'])
end
```

6. Dualité commande/function

Pour les commandes MATLAB telles que load, help, edit, etc … les syntaxes suivantes sont équivalentes :

```
load reponse, ou load('reponse')
edit newstats ou edit('newstats')
type bode, ou type(str) avec str='bode'
etc ...
```

Cela permet de construire les arguments des commandes dans des chaînes de caractères.
Exemple :

```
for d = 1 :31  s=['fich',int2str(d),'.mat'] ;
load(s)
end
```

Chargement et Sauvegarde de données

```
P1=tf(2,[1 1]) ;
[y,t]=impulse(P1) ;
save('fich','y','t')
clear y,t
load fich ou load('fich') retrouve y et t
```

Conversion de données numériques en chaînes :

```
A=input('entrer une donnée numérique')
disp(['la donnée est : ',num2str(A)])
```

(voir également dans la même famille int2str, str2int et str2num)

EXERCICE

Le théorème de Cayley-Hamilton assure qu'une matrice carrée A vérifie elle même son équation caractéristique $\det(\lambda I - A) = 0$**. La fonction** `cayley` **suivante utilise le théorème de Cayley Hamilton pour calculer** A^n **, A est une matrice carrée de dimensions (n,n).**
On génère un message d'erreur en l'absence d'argument d'entrée, et un avertissement si la matrice A donnée n'est pas carrée.

Analyser la solution proposée ci-dessous. Comment lancer la fonction `cayley` **? pour déclencher une erreur ? un warning ? Comment calculerait-on** M^5**, M matrice magique d'ordre 5 ?**

```
function [r]=cayley(a)
%calcule A^n en utilisant Cayley Hamilton
if ~nargin, error('pas d''argument'), end
if ~isempty(a),
   [m,n]=size(a);
   if (m==n),     % il existe abcdchk
      p=poly(a)
      c=zeros(n,n)
      for i=1:n, c=c+p(i+1)*a^(n-i); end
   else ,
      disp('la matrice doit être carrée'),
      return
   end
   disp(['Calcul de A^',int2str(n)]),c
end
```

EXERCICE PERSONNEL AVEC MATLAB

Rechercher et analyser la fonction `abcdchk` **de MATLAB. Quelle en est la fonction ? Comment procède t'on ? Quelles sont les autres fonctions MATLAB appelées ? Faire help abcdchk, d'où provient le texte de l'aide sur abcdchk ?**

» type abcdchk ou >> edit abcdchk.m

Vérification du nombre d'arguments d'entrée et de sortie dans les scripts MATLAB

Les fonctions `nargin` et `nargout` de MATLAB sont évaluées au nombre d'entrées et au nombre de sorties précisées lors de l'appel d'une fonction MATLAB.

ANALYSER LA FONCTION STRTOK SUIVANTE :

```
% Fichier strtok.m du répertoire strfun
%
function [token, remainder] = strok(string,delimiters)
% nargin, nargout, error, voir également warning('xxx')
if nargin <1, error('Not enough input arguments.') ;
end
token= [] ; remainder = [] ;
len = length(string) ;
if len == 0
  return
end
if (nargin == 1)
   delimiters = [9:13 32] ; % white space characters
end
i = 1 ;
while (any(string(i) == delimiters))
      i = i + 1 ;
   if (i  > len), return, end
end
 start = i ;
 while (~any(string(i) == delimiters))
   i = i+1 ;
   if (i > len), break, end
 end
 finish = i-1 ;
token = string(start :finish);
if (nargout == 2), Remainder = string(finish + 1 :end) ;
end
```

Passage d'un nombre variable d'arguments

Le manuel de MATLAB conseille alors d'utiliser les tableaux de cellules (*cell arrays*) vus plus haut, parce que les enregistrements sont de taille de composition et de longueur quelconques :

Dans cet ordre d'idées, analyser les deux fonctions suivantes, testvar.m et testvar2.m

```
function testvar(varargin)
% tester avec testvar([2 3],[1 5],[4 8],[6 5]) etc...
for i = 1:length(varargin)
  x(i) = varargin{i}(1);
  y(i) = varargin{i}(2);
end
xmin= min(0,min(x));
ymin= min(0,min(y));
axis([xmin fix(max(x))+3 ymin fix(max(y))+3])
plot(x,y,'r*')
```

```
function [varargout] = testvar2(arrayin)
% tester avec a= [1 2 3 4 5; 6 7 8 9 10]'
% [p1,p2,p3,p4,p5] = testvar2(a)
for i = 1:nargout
   varargout{i} = arrayin(i,:);
end
```

EXERCICE : EXPLICITER EN UTILISANT L'AIDE DE MATLAB LES FONCTIONS ET OPERATEURS SUIVANTS :

- max, min, mean, median :
- length, size :
- sum, prod,
- rem, mod
- fix, ceil, round, floor :
- sort
- break, return :
- step, impulse, lsim, bode, nichols, nyquist :
- plot, subplot, :
- grid, axis, hold, xlabel, ylabel, title

- `gtext, text, legend`
- `input`
- `pause(n)`
- `disp`
- `ginput`
- `tf, zpk,`
- `ss, ss2tf, tf2ss`
- `dcgain :`
- `rlocus, rlocfind`
- `find`
- `num2str, str2num, int2str`
- `fft`

EXEMPLES DE SCRIPTS :

Le premier script crée deux fichiers de données (extension .mat) contenant chacun une réponse indicielle. Le second applique à une réponse indicielle lue dans un fichier .mat les méthodes d'identifi-cation de base vues en cours pour les ordres 1 et 2:

```
function creation(fich1,fich2)
% Function 'creation' pour TD 9- ESSI1
% dans le répertoire courant de deux fichiers
% nom1.mat et nom2.mat fich1='nom1' et fich2='nom2'
% contenant les réponses indicielles des processus
% de ft f1 et f2 dans les vecteurs [s,t]
f1=tf(2,[1 1]);
[s,t]=step(f1);
save(fich1,'s','t')
f2=tf(4,[1 .5 1]);
```

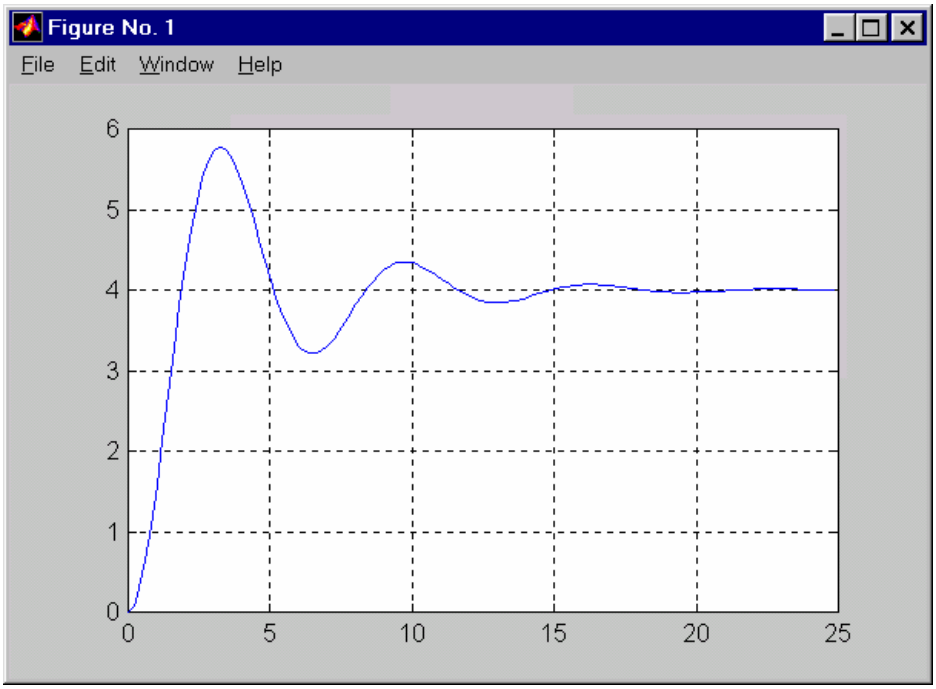
```
[s,t]=step(f2);
save(fich2,'s','t')
% relecture des réponses sous MATLAB
% >> load nom1 ou load('nom1') idem pour nom2
% relire creation: >> type creation

function transfert=identification(fichier)
% retrouve la fonction de transfert associée à une
% réponse indicielle du premier ou du second ordre
% s'il y a dépassement, on choisit le seconde ordre,
% sinon, premier ordre.
% la réponse est dans les vecteurs t, et s de fichier
load(fichier)
plot(t,s)
grid
[m,i]=max(s);
vf=s(length(s));
if vf==m
   tr=t(max(find(s<=(1-exp(-1))*vf)));
   disp(['premier ordre, vf=',num2str(vf),...
         'tr5%= ',num2str(tr)])
   transfert=tf(vf,[tr 1]);
else
   disp('second ordre')
   d1=(m-vf)/vf;
   ep=log(d1)/pi;
   m=sqrt(ep^2/(1+ep^2));
   omeg=pi/(sqrt(1-m^2)*t(i));
   transfert=tf(vf*omeg^2,[1 2*m*omeg omeg^2]);
end
```

Résultat second ordre

4.077

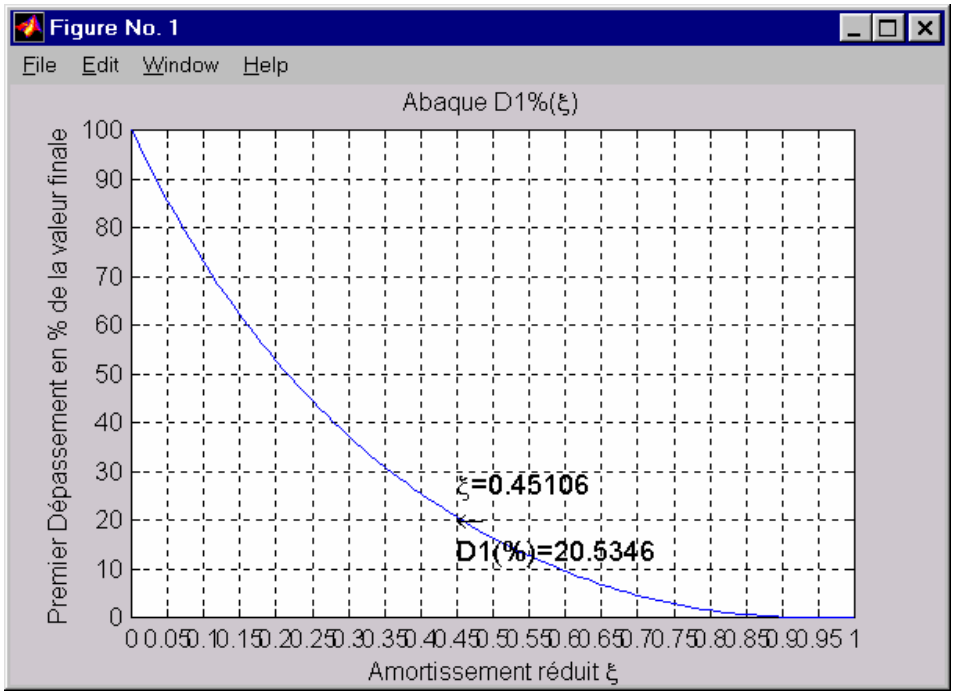
---

s^2 + 0.5041 s + 1.02



Ce script trace et exploite l'abaque D1(m) du premier dépassement d'un second ordre type en fonction de l'amortissement réduit :

```
function [d]=d1dem(m)
% Abaque du premier dépassement fonction de
% l'amortissement réduit pour un second ordre
% L'argument m est forcé à .01:.01:.707 par défaut.
% En l'absence de sortie, tracé de l'abaque.
% Cliquer sur une abscisse m affichage le point          % correspondant

if ~nargin,
   m = 0:0.01:1;
   warning('Attention, pas d''entrée !!!')
end
d = 100*exp(-pi*m./sqrt(ones(size(m))-m.^2));
if ~nargout,
   plot(m,d) %on fixe les propriétés du tracé
   set(gca,'Xgrid','on','XTick',0:.05:1,'ygrid','on')
   title('Abaque D1%(\xi)')
   ylabel('Premier Dépassement en % de la valeur finale')
   xlabel(' Amortissement réduit \xi')
   bouton=1;
   h=[]; % création d'un handler vide
   while bouton==1,
      [x,y,bouton]=ginput(1);
      if bouton~=1, break, end
      %replacement sur la courbe
      [mini,imin]= min(abs(m-ones(size(m))*x));
      %effacement du texte précédent s'il y a lieu
      if ~isempty(h), delete(h),end
      h=text(x,d(imin),{['\xi=',num2str(x)] ...
       ['\leftarrow'] ...
       ['D1(%)=',num2str(d(imin))]},'fontsize',11);
   end
end
```



Ce script Matlab constitue un signal carré à 100 Hz et le « joue » sur les hauts parleurs du PC après avoir affiché son spectre obtenu par transformée de Fourier:

```
[y,fs,bit]=wavread('chord');
[N,C]=size(y);
t=(1:2*N)/fs;
f=100;
s= sign(sin(2*pi*f*t));
wavwrite(s,fs,bit,'son1');
spectre=fft(s,2048)/2048;
freq=0:2047;
freq=freq*fs/2047;
subplot(2,1,1)
plot(freq,abs(spectre))
subplot(2,1,2)
tvisu = (0:2047)/fs;
plot(tvisu,s(1:2048))
axis([0,0.1,-2,+2]);
! C:\windows\media\sndrec32.exe /play /close son1.wav
```

Celui ci enregistre un son en utilisant le programme magnétophone de Windows soit sndrec.exe

```
! del titi.wav
! sndrec32.exe /record /close
! C:\windows\media\sndrec32.exe /play /close titi.wav
[siffle,fs,bit]=wavread('titi');
[N,C]=size(siffle);
t =(0:N-1)/fs;
plot(t,siffle,'y')
```

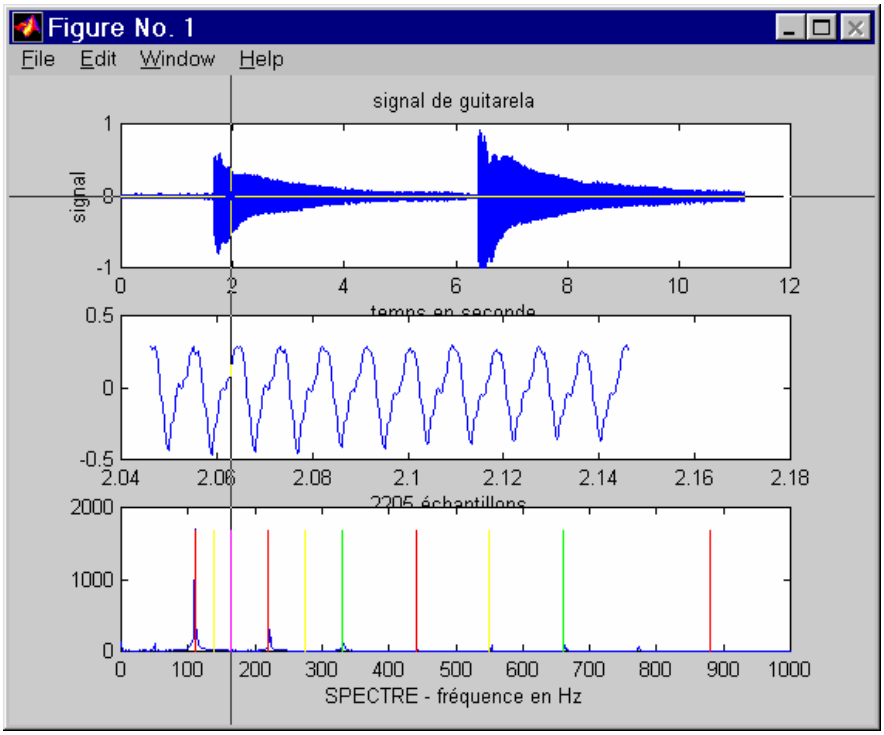
Et ici, on joue un petit air en utilisant l'instruction sound de Matlab

```
f=444;
fs=8000;
t1=0:1/fs:0.5;
t2=0:1/fs:1;
blanche=size(t2,2)
noire=fix(blanche/2)
sil=zeros(1,fix(noire/4));
la=sin(2*pi*f*t1);
lad=sin(2*pi*f*1.06*t1);
si=sin(2*pi*f*power(2,6)*t1);
do=sin(2*pi*f*power(2,1/4)*t1);
dod=sin(2*pi*f*power(2,1/3)*t1);
re=sin(2*pi*f*(1.06^5)*t1);
red=sin(2*pi*f*sqrt(2)*t1);
mi=sin(2*pi*f*(1.06^7)*t1);
fa=sin(2*pi*f*power(2,2/3)*t1);
fad=sin(2*pi*f*power(2,3/4)*t1);
sol=sin(2*pi*f*(1.06^10)*t1);
sold=sin(2*pi*f*(1.06^11)*t1);
la=sin(2*pi*2*f*t1);
acdo=(do+mi+sol)/3;
acfa=(fa+la+do)/6;
acsol=(sol+si+re)/2;
```

```
sound([do sil do sil do sil re sil mi mi sil ...
      re re sil do sil mi sil re sil re sil do do ...
      acdo acsol acfa acdo acsol acdo],fs)
```

Le script suivant analyse un son enregistré dans un fichier wav : il isole une fenêtre temporelle
et en calcule la transformée de Fourier

```
fichierwav='guitarela';
[son,fs]=wavread(fichierwav);
sound(son,fs)
[Nb,C]=size(son);
if C>1, son=son(:,1); end; %cas de la stéréo
S=2^14; % Nbre de points de la fft
FHz=1500; %fenêtre de fréquence observée
dt=0.1; %seconde
T=fix(dt*fs);
F=fix(FHz*S/fs);
```




```
% on trace le signal dans son entier
subplot(3,1,1)
plot([0:Nb-1]/fs,son)
title(['signal de ', fichierwav])
xlabel('temps en seconde')
ylabel('signal')
bouton=1;
% choix d'un instant de début pour l'analyse
while bouton==1,
   [x,y,bouton]=ginput(1);
   In=fix(x(1)*fs)+1;
   if Nb-In<S, In=Nb-S; end;
   subplot(3,1,2)
   t=[In:In+T-1]/fs;
```

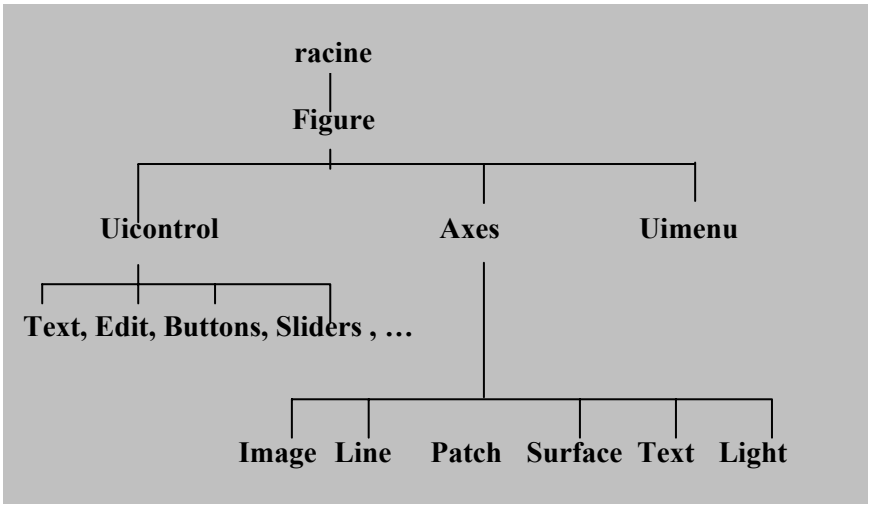
```
    plot(t,son(In:In+T-1))
    xlabel([int2str(T),' échantillons '])
    subplot(3,1,3)
    %calcul et tracé du spectre
    spectre=fft(son(In:In+S-1));
    spec=spectre(1:F);
    interv=0:F-1;
    freq=interv*fs/S;
    plot(freq,abs(spec)')
    xlabel('SPECTRE - fréquence en Hz')
    M=max(abs(spec)); %marque le LA 440 Hz
    line(2*[440,440],[0,M],'color','red')
    line(1.5*[440,440],[0,M],'color','green')
    line(1.25*[440,440],[0,M],'color','yellow')
    line([440,440],[0,M],'color','red')
    line(0.75*[440,440],[0,M],'color','green')
    line(0.625*[440,440],[0,M],'color','yellow')
    line(0.5*[440,440],[0,M],'color','red')
    line(0.375*[440,440],[0,M],'color','green')
    line(0.3125*[440,440],[0,M],'color','yellow')
    line(0.25*[440,440],[0,M],'color','red')
    axis([0 1000,0 2000])
end
```

Création d'Interfaces Graphiques Utilisateur (GUIs) avec MATLAB

Matlab permet à l'utilisateur de programmer des interfaces graphiques interactifs afin de présenter ses résultats. Le chapitre *GUI Implementation* de la notice *Building GUIs with MATLAB* est peu détaillé, les interfaces graphiques réalisables restent relativement simples. On précise donc les notions et les composants permettant de comprendre le fonctionnement des GUIs de MATLAB et on illustre par un exemple.

Structure : arbre

Un GUI se présente comme une structure arborescente (ci-dessous) composée d'objets d'interface



Construire un GUI c'est donc construire une telle structure**.**

Objets d'interface

On utilise les objets suivants dans la suite, ce n'est pas une liste exhaustive**:**

figure, text, axes, line, edit, slider, button, checkbox, …

Propriétés **:**

Chaque **objet** possède un **ensemble de propriétés** généralement programmables qui fixent l'apparence graphique et les réactions de l'objet aux sollicitations de l'utilisateur.

Les propriétés peuvent être des chaînes de caractères, des vecteurs de valeurs numériques, spécifiés selon le format courant :
Par exemple, `Xtick = [0 :0.2 :1]`, ou `Xgrid = 'on'`

Si chaque type d'objet possède des propriétés propres, certaines propriétés sont communes à tous les objets : un objet `text` a un nom (propriété `Tag`), une chaîne de caractères qu'il affiche (propriété String), des couleurs `Foregroundcolor` et `Backgroundcolor`, avec une police de caractères FontName de taille `Fontsize` … Pour retrouver la valeur d'une propriété, il faudra en spécifier le

nom dans une chaîne de caractères. L'éditeur d'interfaces `guide` ne teste pas les majuscules et les trois premières lettres suffisent pour retrouver une propriété: on pourra taper '`str`' pour la propriété '`String`' par exemple.

Callbacks : Réactivité de l'interface

Parmi les propriétés des objets de l'interface, les Callbacks contiennent des scripts ou des fonctions MATLAB pour programmer les réactions de l'interface aux commandes de l'utilisateur.
Ainsi, imaginons un PushButton nommé '`Bouton`' auquel on a associe le Callback : `grid on`. Cliquer sur Bouton provoque le tracé d'une grille sur les axes de tracé courants. Le Callback `close(gcf)` fermerait l'interface graphique, `cla` effacerait les tracés, etc …

Handlers : Identificateurs des objets

Les objets étant créés lors de la constitution de l'interface, ou dynamiquement durant l'exécution, on leur associe lors de la création un identificateur unique, qu'on appelle le `Handler` et qui permet de les manipuler. Certains handlers sont réservés et mis à jour en permanence :

gcf **: attaché à la figure courante**
gca **: axes de tracé courants**
gcbf **: figure activée (dans laquelle on clique)**
gcbo **: objet activé actuellement à l'aide de la souris**

Pour retrouver dans un script le handler d'un objet de l'interface dont on connaît une propriété qui le caractérise, on peut utiliser la fonction `findobj` :

h = findobj(gcf, 'Tag', 'Fig1') par exemple, h handler de Fig1

Outils d'aide : (propedit, et GUI Layout Tool)

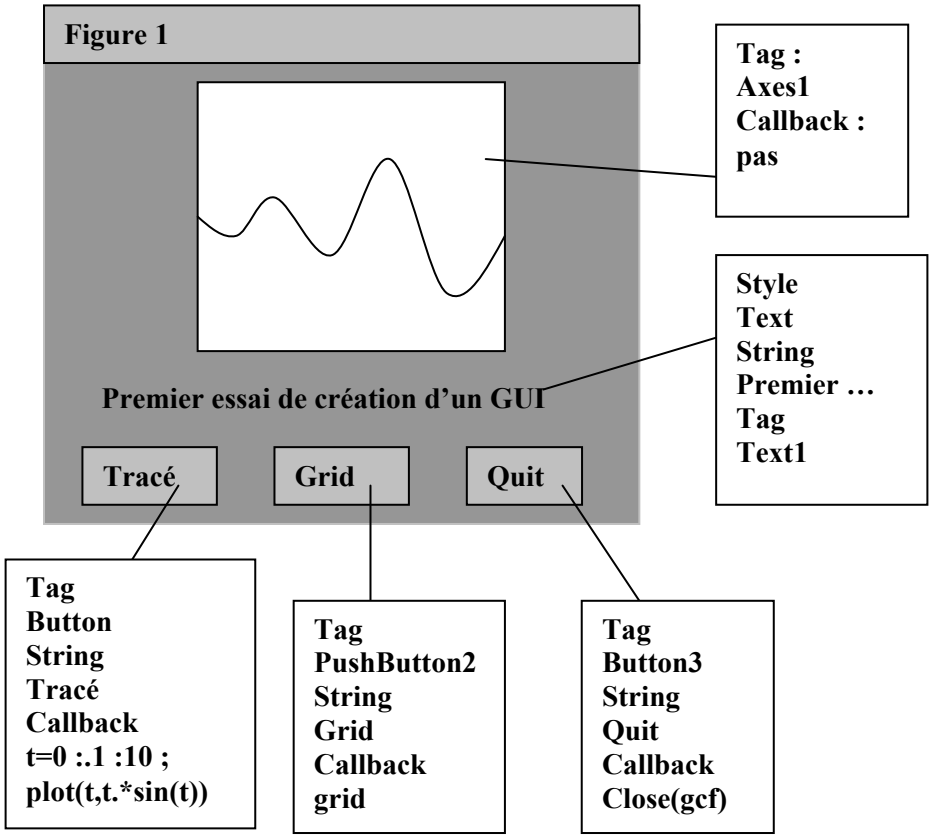
Depuis la barre de menu de Matlab, ou depuis la ligne de commande, on lance l'éditeur de propriétés, soit propedit, ou même l'éditeur d'interfaces graphiques : `guide`.

```
>> propedit       % ou menu File de Matlab
>> guide          % ou menu File
```

Les fonctions get **et** set **associées aux handlers, permettent de modifier par programme dans les callbacks les propriétés des figures et objets graphiques:**

```
>> get(0)          % liste les propriétés de 'root'
>> h= plot(t,y)  % retourne le handler h du plot
>> propedit(h)
>> delete(h)      %efface l'objet h, le tracé
>> get(h)
>> set(h,'Color',[0.5 0 0]) % ou 'color'→ rouge sombre
>> set(gca,'Xgrid','on', …
              'XTick',[0 1 3 4 4.5 5])
>> close(gcf)    % équivalent à close
```

Premier exemple simple



Création et Gestion d'Objets

En fait, il n'est pas besoin de constituer un GUI pour créer des objets et spécifier des valeurs pour leurs propriétés:

```
>> h=text(x,y,'Ceci est un objet')
```

crée un objet text **et une fenêtre graphique, et retourne le handler** h **;**

```
>>set(h,'FontSize',12, …
        'Color', [1 0 0], …
        'Font','arial')
```

en modifie les propriétés graphiques. Pour effacer l'objet, il suffira de faire:

```
>> delete(h)
```

De même, sont licites les créations d'objets telles que :

```
>> h= plot(t,sin(t),t,cos(t),'r*') ou
>> hl = line(t,sin(t),'LineWidth','thick') ou encore
>> hf = figure('position',[100 100 400 200], …
                'pointer','crosshair', …
                'color',[1 0 0], ...
                'Name','Mon premier interface')
```

qui trace une figure dont le fond est rouge, le curseur devient une croix +, le handler hf retourné vaut 1

```
>>delete(hf) ou delete(1) est équivalent à close
>> ha= axes(…)
```

Pour créer un objet d'interface on peut utiliser la fonction uicontrol :

```
>>h=uicontrol('style','pushbutton','string','Terminer', …
                'callback','close')
```

Exécuter un plot crée une arborescence:

root → figure → (axes→ line), → title → xlabel → ylabel

L'éditeur de propriétés propedit **permet de parcourir les arborescences, et de visualiser ou de modifier les propriétés des objets existants.**

L'éditeur d'interface guide **permet de définir graphiquement les objets d'interface et leurs propriétés**

EXERCICE – ILLUSTRATION :

Afin de mieux appréhender la création et l'utilisation des objets d'interface, on réalisera les manipulations suivantes avec Matlab :

```
t=0 :.1 :100 ;
figure
axes
line(t,t.*sin(t))
close
```

puis `plot(t,t.*sin(t))`

conclusion ? L'instruction `plot` **crée une figure, un axe (zone de tracé), et une (ou des) lignes**

```
h= plot( t, t.*sin(t), t, t)
```

L'instruction retourne maintenant deux handlers associées aux duex lignes tracées

```
get(h(1),'color'))
set(h(2),'Tag','ligne 2', …
                'color',[0.1,0.8,0.5], …
  'ButtondownFcn','close')
```

Puis, cliquer sur la ligne h(2). La figure est fermée en réaction. Remarquer que les noms des propriétés graphiques peuvent être écrits en minuscules ou en majuscules, indifféremment) . Enfin, reprendre les lignes précédentes, et ajouter une troisième fois

```
h= plot( t, t.*sin(t), t, t)
get(h(2),'buttondownfcn')
```

donne une chaîne vide. Le nouveau dessin a crée deux nouvelles lignes dans lesquelles les propriétés ont été remises à leur valeur défaut, en particulier ButtonDownFcn = ' ', bien que le tracé soit identique.

FICHIERS DE CREATION D'UNE INTERFACE :

Il faut noter qu'une interface MonGUI créé à partir de l'éditeur d'interfaces guide tient dans deux fichiers. Le script MonGUI.m contient la description en langage Matlab de l'interface, la première instruction exécutable en est d'ailleurs :

```
load MonGUI
```

Cette instruction Matlab permet de charger des données numériques placées dans un fichier de données (extension .mat dans Matlab) ici MonGUI.mat.

MonGUI.m peut être édité mais il est déconseillé d'en modifier le texte à moins d'être un spécialiste averti.

```
edit MonGUI
```

et on lance l'exécution de l'interface en lançant le script MonGUI

```
>> MonGUI
```

DUPLICATION D'OBJETS D'INTERFACE :

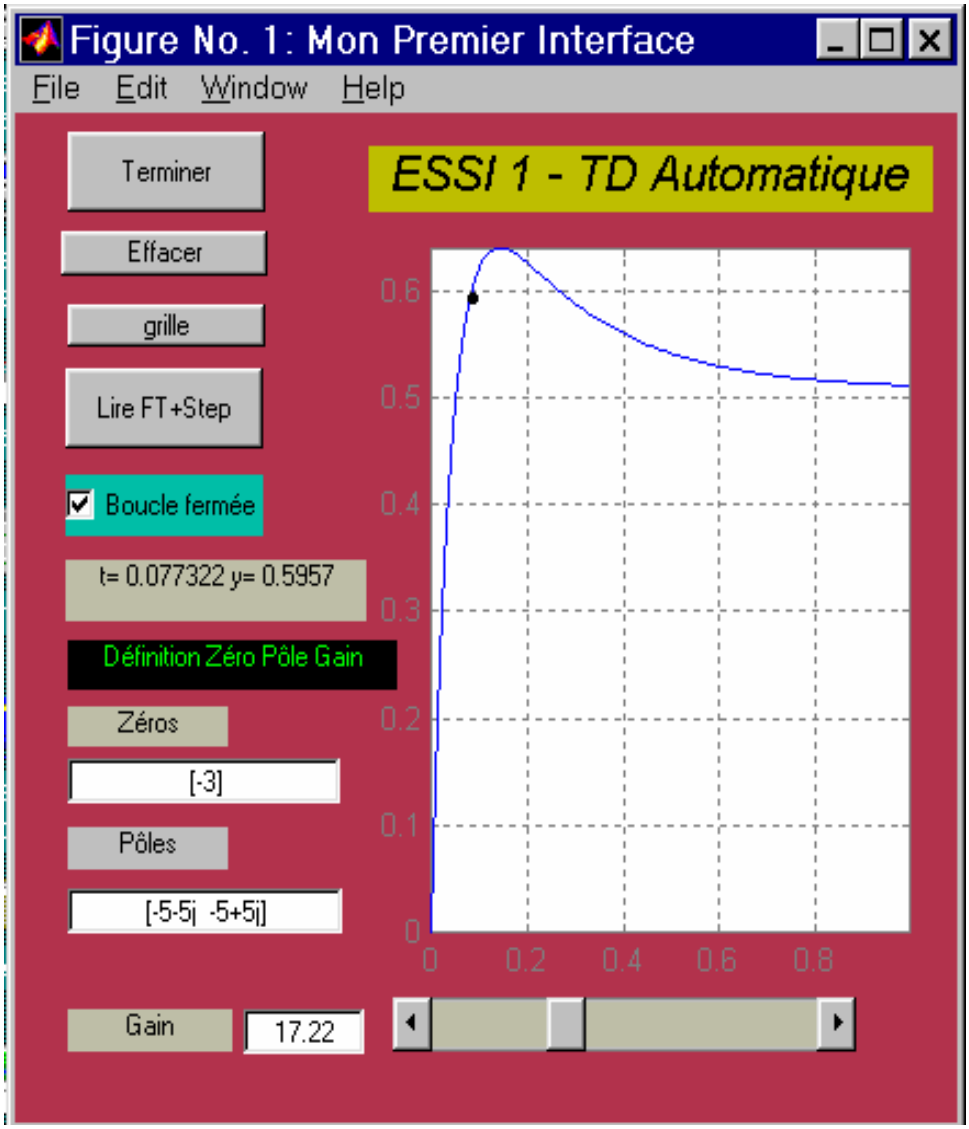
Dupliquer un objet graphique avec l'éditeur guide (sélection et bouton droit) crée un second objet de même nom jusqu'à la modification de la propriété 'Tag' de l'objet nouveau né. Ne pas modifier le nom 'Tag' posera un problème si l'on souhaite lire ou modifier les propriétés de l'objet dans le fonctionnement ultérieur de l'interface. Typiquement, on modifie un objet en croyant modifier un autre. Prendre des précautions. D'ailleurs 'lireFichier' et ' lireFichier' sont des noms différents bien sûr pour des objets différents.

Quelques propriétés des objets d'interface

On énumère un ensemble d'entre elles qui sont utilisées dans l'exemple de la page suivante:

Tag :	'Text1', 'Edit2', 'Slider1', 'Axes1'
String :	'LireFT+Step', ou 'Terminer', ou 'Boucle fermée'
Callback :	'close(gcf)' ou 'grid on', …
ButtonDownFcn	'animate start'
WindowButtonDownFcn, WindowButtonMotionFcn	
ForegroundColor, BackGroundColor, Color: [Rouge Vert Bleu]	
Value :	1 : Checkbox cochée, 0 : non cochée
FontAngle	'italic' (text)
FontName	'Brush Script'
FontSize	16
FontWeight	'bold'
Max, Min	50, et 1 (slider)
SliderStep	[0.01 0.1] (slider)
Position	[100 100 200 400] soit xBG, yBG, Lx, Ly
Name	'Mon premier interface'
Pointer	'arrow', ou 'fullcrosshair' (figure)
CurrentPoint	Currpt=get(gca,'currrentpoint') → x et y souris
NextPlot	'add'

Création d'un GUI plus élaboré :



On explicite l'exemple d'interface ci-contre, crée à l'aide du script `gui` **dans** `gui.m`. **Ce script utilise la fonction** animator **de** animator.m**). En fait, trois fichiers sont nécessaires:** `gui.m`, `gui.mat`, `animator.m` **dans le répertoire** *Matlab*.

On a utilisé l'éditeur d'interfaces : >> guide,
pour définir et placer les éléments de l'interface et pour en ajuster les propriétés.
On lance par la commande : `>> guide`

Fonctions de l'interface :

(1) tracer la réponse indicielle d'un processus donné sous la forme zéros , pôles, gain
(2) mesurer les points de la réponse en pointant avec la souris,
(3) prévoir la réponse du système bouclé à retour unitaire et avec un gain unité.

On utilise 5 objets text, 3 zones edit, un slider, 4 buttons, un axe, une checkbox.
De plus,

- **Cliquer sur la fenêtre provoque le tracé**
- **Les listes de zéros et de pôles sont entrées dans des zones de texte éditables**
- **Le gain est spécifié à l'aide d'un scrollbar**
- **Le curseur devient une croix, 'crosshair'**
- **La position de la souris est symbolisée par un point, et apparaît dans une fenêtre texte**
- **On superpose les tracés (Nextplot= 'add')**
- **Un bouton permet d'effacer les tracés, un autre ajoute une grille, un troisième ferme l'interface et termine.**

Callbacks et fonctions utilisés dans l'interface

Cliquer sur le bouton LireFT+Step provoque l'exécution du callback suivant écrit sous la forme d'un script MATLAB, ou simplement de l'appel à une fonction accessible dans le path (voir ici l'appel à la fonction animator un peu plus loin).

LireFT + Step : est donc un pushbutton dans voici le 'callback'

```
h=findobj(gcbf,'Tag','Edit1');
a=str2num(get(h,'String'));
h=findobj(gcbf,'Tag','Edit2');
b=str2num(get(h,'S tring'));
h=findobj(gcbf,'Tag','Edit3');
g=str2num(get(h,'String'));
proc=zpk(a,b,g); % Create zero-pole-gain models or convert to zero-pole-gain format.
if get(findobj(gcf,'Tag','Checkbox1'), 'Value'),
  [y,t] = step(proc/(1+proc)); %plot result of zpk function
else,
  [y,t]=step(proc);
end
plot(t,y) ;
axis([0 t(length(t)) min(y) max(y)])
grid on
```

Cliquer dans la zone du checkbox renvoie 0 ou 1 dans la variable v ; selon le cas, le texte associé est modifié et mis à jour

CheckBox : callback

```
v=get(gcbo,'Value')
if v==1,
  set(gcbo,'String','Boucle fermée')
else  set(gcbo,'String','Boucle ouverte'),
end
```

La position du curseur du slider (échelle) est placée dans v, puis affichée dans la zone de texte de nom Edit3. C'est le gain désiré, que l'on peut également taper dans la zone éditable Edit3.

Slider : callback

```
v=get(gcbo,'value');
h=findobj('tag','Edit3');
set(h,'string',num2str(v))
```

Il faudrait ajouter le couplage inverse Edit3 → Slider

Deux callbacks sont définis ici selon les actions de l'utilisateur sur la Figure, fenêtre principale de tracé. Appuyer sur un bouton de la souris démarre la fonction animator qui affiche la position de la souris sur les tracés pour faciliter les mesures.

Figure : (la figure englobe l'ensemble de l'interface)

Buttondownfcn

animator start **(équivalent à la syntaxe animator('start')**

WindowButtonDownFcn

```
h=findobj(gcbf,'Tag','Edit1');
a=str2num(get(h,'String'));
h=findobj(gcbf,'Tag','Edit2');
b=str2num(get(h,'String'));
h=findobj(gcbf,'Tag','Edit3');
g=str2num(get(h,'String'));
proc=zpk(a,b,g);
if get(findobj(gcf,'Tag','Checkbox1'), 'Value'),
  [y,t] = step(proc/(1+proc));
else,
  [y,t]=step(proc);
end
line(t,y); (noter la différence avec plot(t,y) qui recrée l'objet Axes1)
axis([0 t(length(t)) min(y) max(y)])
grid on
```

Fonction pour la lecture de la position de la souris sur les axes des tracés

```
function [x,y]=animator(action)
switch(action)
case 'start',
   set(gcbf,'WindowButtonMotionFcn', ...
            'animator move;')
   set(gcbf,'WindowButtonUpFcn', ...
            'animator stop')
case 'move'
   if ~isempty(findobj(gca,'Tag','toto')),
      delete(findobj(gca,'Tag','toto'))
   end
   currpt=get(gca,'CurrentPoint');
   h=findobj(gcbf,'Tag','Text1');
   set(h,'string',['t= ', ...
```

```
          num2str(currpt(1,1))...
        ,' y= ',num2str(currpt(1,2))])
   x=currpt(1,1);
   y=currpt(1,2);
   text(x,y,'\bullet','Tag','toto');
case 'stop'
   set(gcbf,'WindowButtonMotionFcn','')
   set(gcbf,'WindowButtonUpFcn','')
end
```

Cette fonction est un bloc switch .. case que l'on appelle avec un argument avec les trois possibilités 'start', 'move' et 'stop'.
'start' modifie la réactivité de l'interface, en modifiant deux callbacks de la figure de façon à appeler animator('move') quand on déplace la souris bouton enfoncé et animator('stop') quand on relâche le bouton.
'stop' remet la réactivité de l'interface dans son état initial
'move' mesure la position du curseur sur l'axe, propriété 'CurrentPoint' dans le vecteur currpt, et l'affiche sur l'objet texte 'Text1'. On symbolise également cette position par une « bulle » objet 'toto'.

On peut voir avec l'éditeur d'interfaces guide le contenu des autres callbacks et les propriétés des objets de l'interface :

Effacer
```
cla % efface l'axe actif (zone blanche de tracé)
```

Terminer
```
close(gcf) % ferme la figure courante
```

Zones Editables Edit1, Edit2, Edit3 :
les callbacks qui souhaitent utiliser leur contenu vont les lire, comme par exemple pour lire la liste des zéros de la fonction de transfert:
```
h =findobj(gcbf,'Tag','Edit1');
a = str2num(get(h,'String'));
```

Axes pour recevoir les tracés : pas de callback ici

labels StaticText3, StaticText2 …
pour y afficher des textes permanents ou temporaires, on utilise la propriété String

Pour afficher des coordonnées de la souris dans la zone texte Text1, on utilise de même la propriété String, voir la fonction animator **ci-contre.**