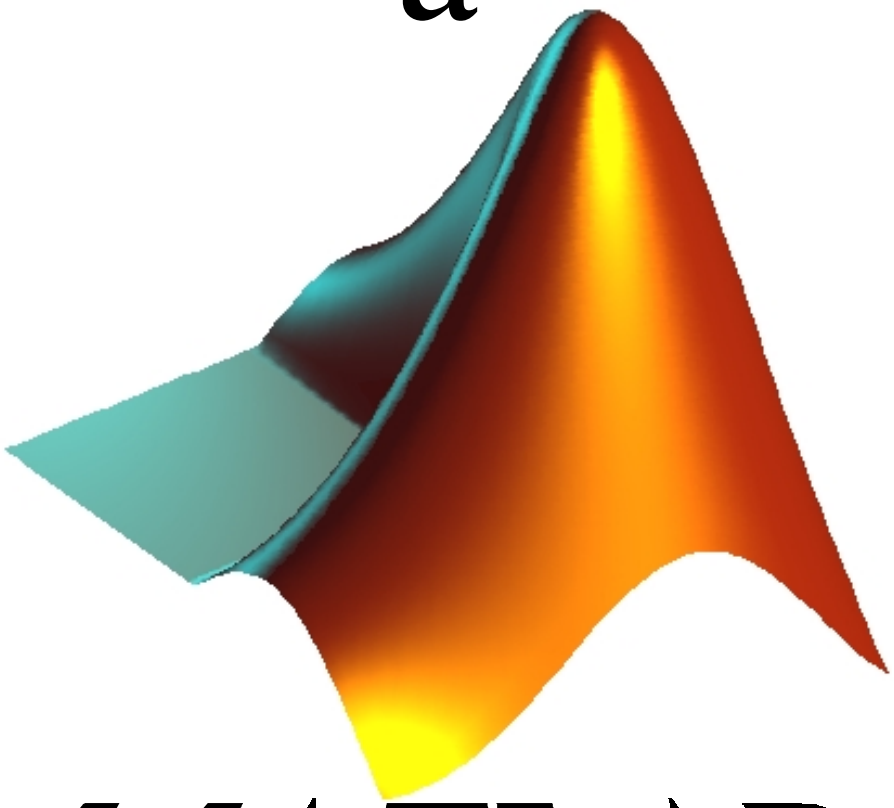
Introduction

à



MATLAB

Auteur : Serge Bedwani

ARL
Applied Research Laboratories

Table des matières

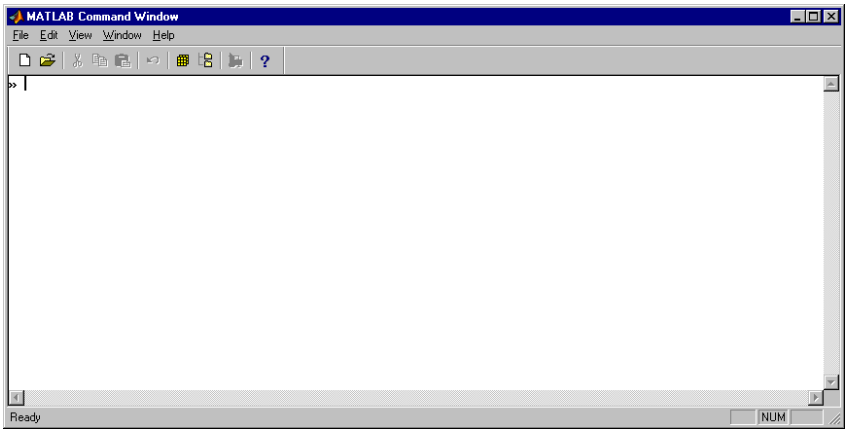
1.0 Introduction

MATLAB est une abréviation de MATrix LABoratory. Ecrit à l'origine, en Fortran, par Cleve Moler, MATLAB était destiné à faciliter l'accès au logiciel matriciel. La version actuelle, écrite en C par The MathWorks Inc., existe en version "professionnelle" et en version "étudiant". Sa disponibilité est assurée sur plusieurs plates-formes : Sun, Bull, HP, IBM, compatibles PC, Macintosh, et plusieurs machines parallèles.

MATLAB est conforté par une multitude de boîtes à outils (toolboxes) spécifiques à des domaines variés. Un autre atout de MATLAB, est sa portabilité; la même portion de code peut être utilisée sur différentes plates-formes sans la moindre modification.

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

L'environnement MATLAB se présente sous la forme d'un espace de travail (Workspace)



où un interpréteur de commande exécute des opérations et fonctions MATLAB. Les sources de celles-ci sont disponibles, écrites en "langage" MATLAB, voir 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 par la méthode de Runge-Kutta (*ode23 et ode45*), l'intégration numérique, la recherche des solutions d'équations algébriques ou transcendantes, la création et manipulation de polynômes (*poly, polyder, polyval, conv, deconv*), la transformée de Fourier rapide (*ffr, fft2, ifft*).

Des fonctions propres au traitement de données, 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.

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.

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 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**.

Les boîtes à outils (toolbox) dédiées à des domaines techniques spécifiques, sont :

- Le traitement du signal
- La régulation automatique
- L'identification
- Les réseaux de neurones
- La logique floue
- Le calcul symbolique

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 réponse 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.

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 maintenant une référence au niveau mondial, non seulement dans les universités et instituts de recherche, mais aussi dans le milieu industriel.

2.0 Notions de base de MATLAB

2.1 Aide dans MATLAB

help <fonction ou commande> ou *helpwin.*: fournit de l'aide sur l'utilisation de la fonction ou de la commande indiquée.

Lookfor<mot-clé> : fournit la liste des fonctions et commandes contenant le mot-clé spécifié dans la première ligne de leur texte d'aide.

2.2 Fichiers de commande

Un fichier de commande est un fichier ASCII d'extension *.M* contenant une suite de commandes MATLAB. Il peut être exécuté directement en tapant simplement son nom dans l'espace de travail MATLAB.

2.3 Fichiers de données

Il est possible dans MATLAB, de charger un fichier, contenant par exemple une série de mesure, par la commande suivante :

```
Load fichier.ext
```

Il en est de même avec la sauvegarde. Par exemple, on désire sauvegarder la variable t qui est un vecteur, par l'intermédiaire de la commande suivante :

```
Save fichier.ext t -ASCII
```

2.4 Fonctions

De nouvelles fonctions peuvent être ajoutées à MATLAB par l'utilisateur. Il suffit de créer un fichier de nom *nom_de_fichier.m* contenant les commandes à exécuter et dont l'en-tête 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 décibels et la retourne dans y

```
function[y]=lin2dB(x)
% function [y]=lin2dB(x)
% conversion de x en dB
   y=20*log10(x);
```

le fichier correspondant est sauvé sous le nom *lin2dB.m*

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 %.
Lorsque l'on tape dans l'espace de travail *help lin2dB* , cela nous fourni une aide en ligne de la fonction.

2.5 Représentation graphique 2D (coordonnées cartésiennes)

La commande `plot` dont la syntaxe est la suivante :

```
plot(x,y,s)
```

permet de tracer des graphiques (courbes ou nuages de points) de vecteurs de dimensions compatibles (y en fonction de x). Le choix du type et de la couleur du tracé peut se faire avec le paramètre facultatif qui est une chaîne composée de 1 à 3 caractères parmi ce qui suit :

couleurs		*tracés discontinus (symboles)*		*tracés continus*	
`y`	jaune	`.`	point	`-`	trait continu
`m`	magenta	`o`	cercles	`:`	pointillés
`c`	cyan	`x`	croix	`-.`	trait-point
`r`	rouge	`+`	plus	`--`	trait-trait
`g`	vert	`*`	étoiles		
`b`	bleu				
`w`	blanc				
`k`	noir				

Le type de tracé est par défaut le trait continu. De même, MATLAB fixe une couleur par défaut si elle n'est pas spécifiée.

2.5.1 Courbes de fonctions

Le tracé de la courbe d'une fonction commence par le choix de l'intervalle de définition (ou du tracé).

```
x=-10*pi:10*pi;
y=sin(x).*exp(-0.1*x);
plot(x,y,'g')
```

Un quadrillage, réalisé par la commande `grid` , donne plus de lisibilité au graphique. La commande `grid off` supprime le quadrillage du graphique courant.

La commande `title` ajoute un titre au graphique de la fenêtre courante.

```
title('Tracé de la courbe d''une fonction')
```

Les commandes `xlabel` et `ylabel` permettent de définir des titres pour les axes des abscisses et des ordonnées.

```
xlabel(' x : axe des abscisses')
ylabel(' y : axe des ordonnées')
```

Dans certains cas, un texte explicatif est nécessaire sur le graphique, ceci peut être réalisé par la commande `text` comme suit :

```
text(x,y,'texte explicatif')
```

x et y sont les coordonnées du début du texte.

Il es possible aussi de placer du texte à un endroit quelconque du graphique sélectionné par l'utilisateur à l'aide de la souris, grâce à la commande `gtext`.

```
gtext('texte explicatif')
```

Remarque : avec la version 5.3 de MATLAB, des boutons sont intégrés dans la fenêtre graphique, ce qui permet d'insérer entre autre du texte directement sur le graphique, sans passer par les commandes vues ci-dessus.

Il est possible de tracer plusieurs courbes en utilisant une seule fois la fonction `plot` comme suit :

```
plot(x1, y1, s1, x2, y2, s2, …)
```

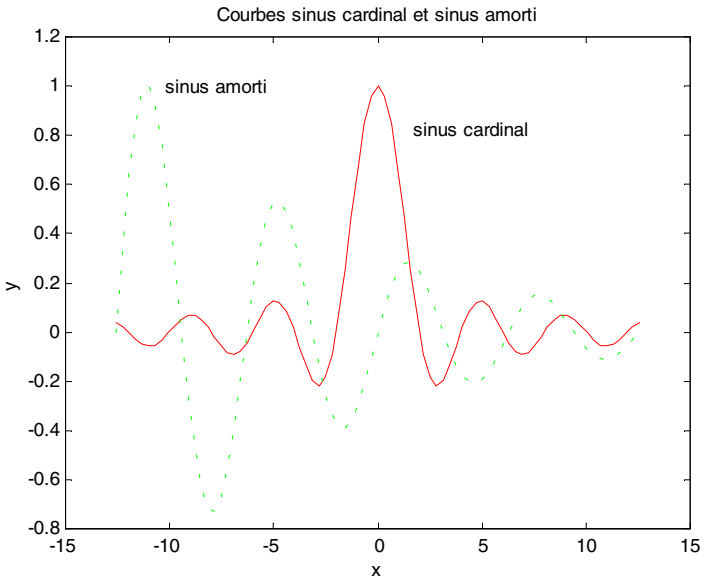
Si y représente un vecteur de nombres complexes, `plot(y)` trace les points images des éléments de y.

La commande `figure(gcf)` permet de passer de la ligne de commandes à la fenêtre graphique courante.

L'utilisation des commandes `hold on` ou `hold off` offre ou supprime la posibilité de tracer plusieurs courbes dans la même fenêtre graphique.

Exemple :
Dans l'exemple qui suit, nous allons tracer des courbes représentant les fonctions sinus cardinal et sinus amorti, dans une même fenêtre graphique.

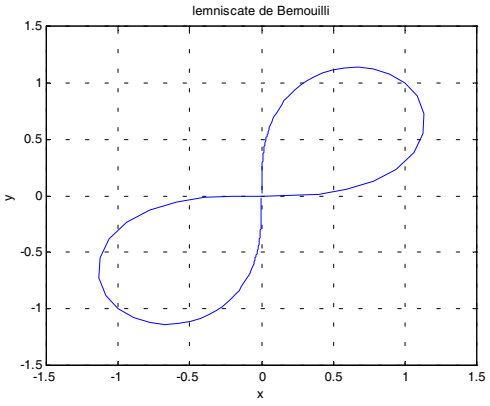
```
x=-4*pi:pi/10:4*pi;
% sinus cardinal en trait rouge continu
y1=sinc(x/2);
plot(x,y1,'r')
hold on
% sinus amorti en pointillés, couleur verte
y2=sin(x).*exp(-0.1*x)/3;
plot(x,y2,'g:');
% documentation du graphique
xlabel('x')
ylabel('y')
title('Courbes sinus cardinal et sinus amorti')
```



2.5.2 Courbes paramétriques

La procédure de tracé de courbes paramétriques est identique à celles de courbes de fonctions.

```
t=-100:0.1:100;
x=(2*t)./(1+t.^4);
y=(2*t.^3)./(1+t.^4);
plot(x,y)
title('lemniscate de Bernouilli')
xlabel('x')
ylabel('y')
grid on
```

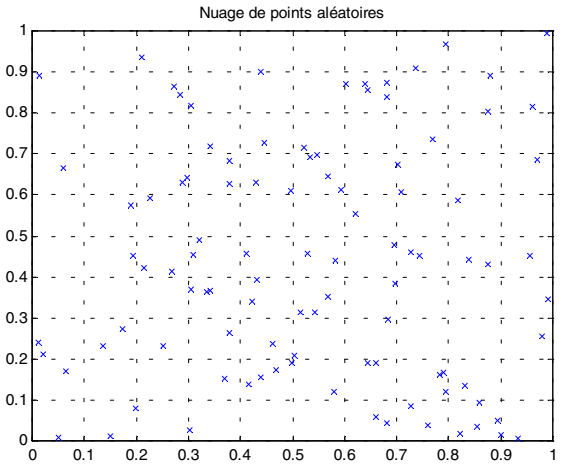


2.5.3 Nuage de points

Le tracé d'un nuage de points se fait de la même façon que celui d'une courbe. On précisera dans la commande `plot`, le symbole (`*, +, x, ., o`) à affecter aux différents points.

Exemple: représentation graphique d'un nuage de points aléatoires

```
N=100;
x=rand(1,N);
y=rand(1,N);
plot(x,y,'x');
grid on
title('Nuage de points aléatoires')
```



2.6 Représentation graphique 2D (coordonnées polaires)

Le tracé de courbes en coordonnées polaires sera réalisé par la fonction `polar`, qui obéit à la syntaxe suivante :

```
polar(theta, rho, 'type')
```

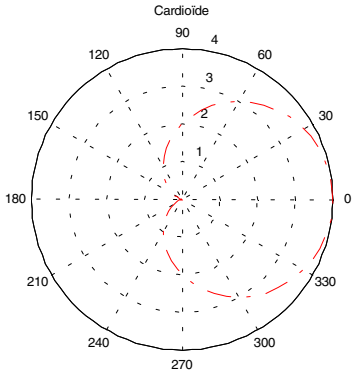
Le type de courbe correspond à la nature du tracé (continu, discontinu, etc.) et à sa couleur. Les différentes possibilités sont identiques à celles de la fonction `plot`.

Limaçons

Un graphique qui a pour équation polaire

$$r = a + b\cos\theta \quad \text{ou} \quad r = a + b\sin\theta$$

où `a` et `b` ne sont pas nuls, s'appelle un limaçon. La cardioïde est un cas particulier du limaçon pour lequel $|a| = |b|$.



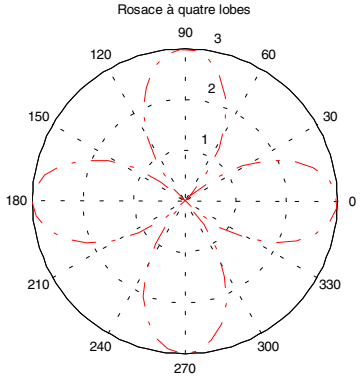

```
theta=-pi:0.1:pi;
r=2+2*cos(theta);
polar(theta,r,'r-.')
title('Cardioïde')
```

Rosaces

Un graphique qui a pour équation polaire

$$r = a\sin n\theta \quad \text{ou} \quad r = \cos n\theta$$

où `n` est un entier positif supérieur à 1 et `a` un nombre réel quelconque, est constitué de boucles nouées à l'origine. Lorsque `n` est impair on obtient `n` boucles, et `2n` boucles si `n` est pair.




```
theta=-pi:0.1:pi;
r=3*cos(2*theta);
polar(theta,r,'r-.')
title('Rosace à quatre lobes')
```

2.7 Représentation graphique 2D (diagrammes)

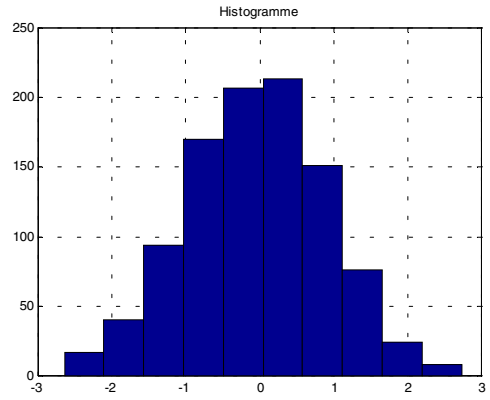
Le tracé d'histogramme se fera à l'aide de la commande `hist`.

`hist(x,N)` : trace l'histogramme des valeurs du vecteur x en N classes

`[n,x] = hist(y,N)` : retourne des valeurs à utiliser avec la fonction `bar`.

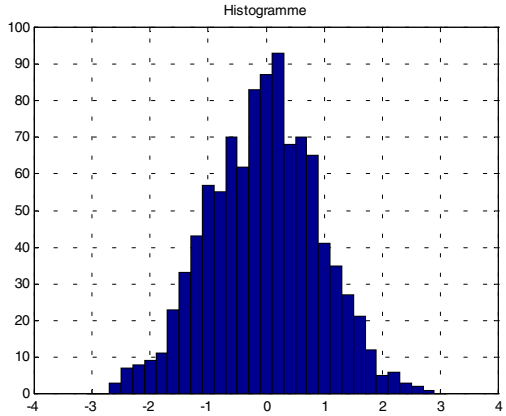
Comme exemple, nous générons un vecteur de taille 1000 dont les composantes sont les valeurs d'une variable aléatoire gaussienne, centrée et réduite. La représentation des valeurs de l'histogramme sont répartis en dix classes.

```
y=randn(1000,1);
hist(y,10);
grid on, title('Histogramme')
```



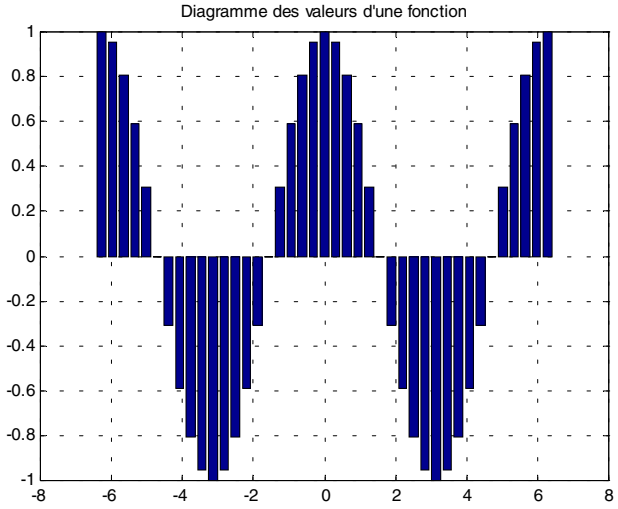
Au lieu de spécifier directement le nombre de classes pour la commande `hist`, nous pouvons indiquer un vecteur qui représente l'intervalle du tracé ainsi que la largeur des classes.

```
x=-3:0.2:3;
hist(y,x), grid on
title('Histogramme')
```



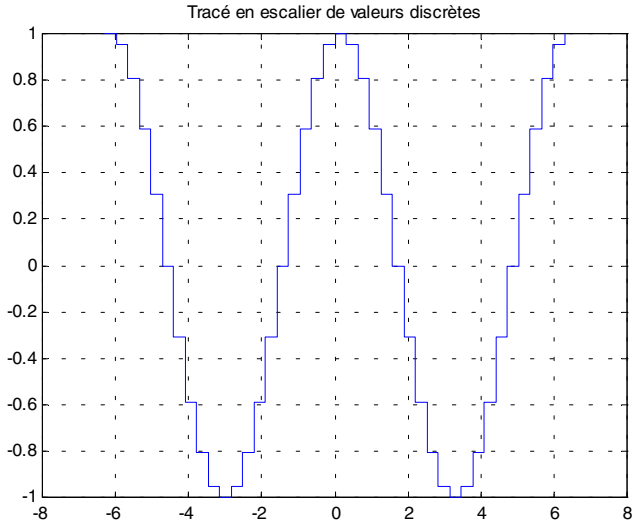
La commande `bar(x,y)` dessine un diagramme sous forme de barres des valeurs de `y` en fonction de celles de `x`.

```
x=-2*pi:pi/10:2*pi;
y=cos(x);
bar(x,y)
grid on
title('Diagramme des valeurs d''une fonction')
```



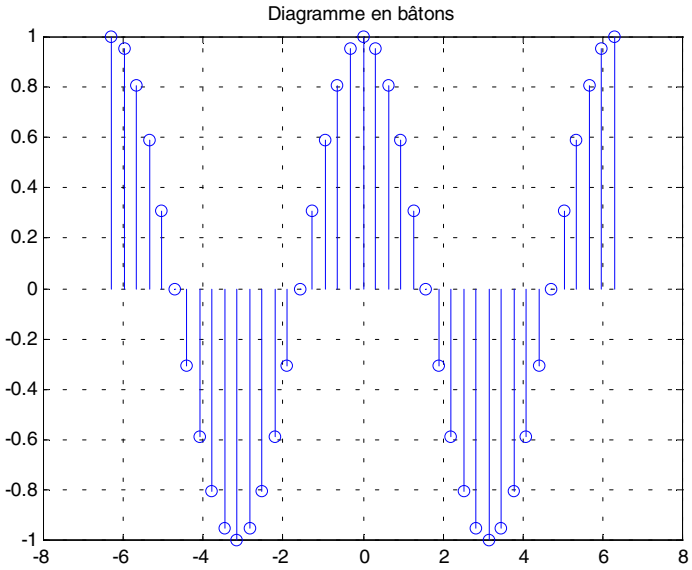
La commande `stairs(x,y)` trace les valeurs discrètes de `y` en fonction de celles de `x` sous forme bloquée ou en marches d'escaliers.

```
stairs(x,y)
grid on
title('Tracé en escalier de valeurs discrètes')
```



La fonction `stem` permet le tracé d'une séquence de valeurs discrètes.

```
stem(x,y)
grid on, title('Diagramme en bâtons')
```



2.8 Représentation des Axes logarithmiques

```
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 dooivent avoir le même nombre d'éléments)

2.9 Représentation graphique 3D

2.9.1 Courbes 3D

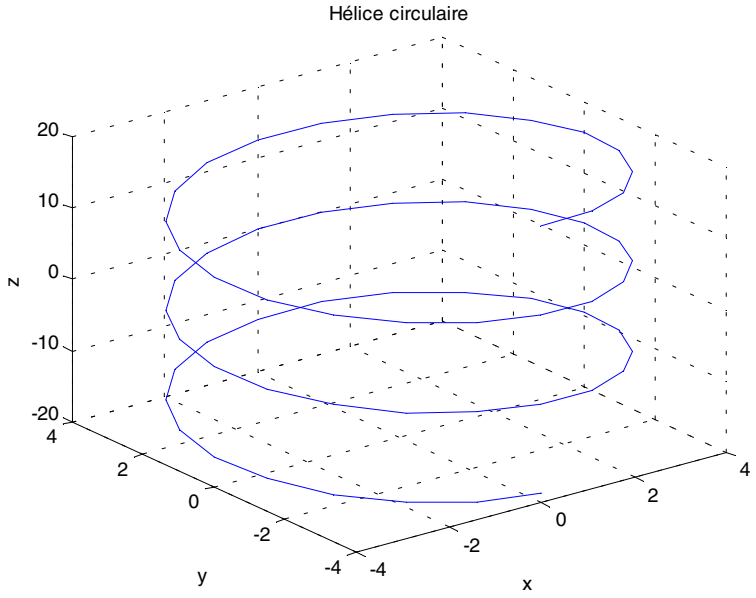
Le tracé de courbes dans l'espace se fera à l'aide de l'instruction `plot3` qui obéit à une syntaxe analogue à celle de `plot`.

```
plot3(x,y,z,'symb')
```

Le choix du type de courbe ainsi que la couleur du tracé se fait avec le paramètre facultatif `'symb'`.

Exemple : tracé d'une hélice circulaire définie par une équation paramétrique.

```
t=-3*pi:pi/10:3*pi;
x=4*sin(t);
y=4*cos(t);
z=2*t;
plot3(x,y,z)
title('Hélice circulaire')
grid
xlabel('x')
ylabel('y')
zlabel('z')
```



2.9.2 Surfaces

Soit l'exemple d'une fonction à 2 variables (sinus cardinal 3D) :

$$z = \frac{\sin(x^2 + y^2)}{x^2 + y^2}$$

pour x et y variant de -π à π avec un pas de π/10.

```
x=-pi/2:pi/30:pi/2;
y=x;
```

On génère deux matrices carrées X et Y qui définissent le domaine de calcul de z, on utilisera pour ceci la fonction `meshgrid`.

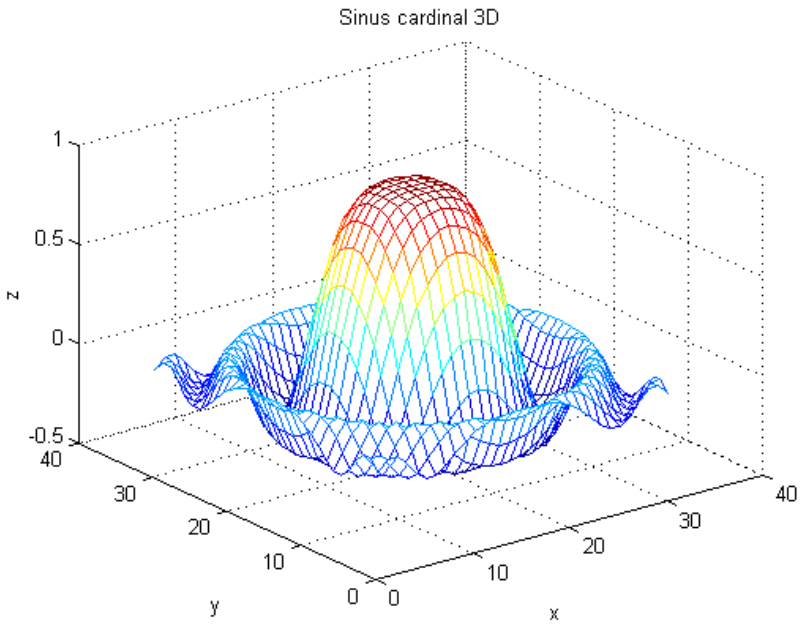
```
[X,Y]=meshgrid(x,y);
```

On évalue la fonction z et on stocke les données dans la variable Z.

```
Z=sinc(X.^2+Y.^2);
```

On dessine la surface représentative de la fonction.

```
mesh(Z)
```



Pour d'autres exemples, je vous invite à taper `graf3d` dans l'environnement de MATLAB, pour des exemples complémentaires.

3.0 Les bases de MATLAB

3.1 Vecteurs

Un vecteur ligne est introduit de la façon suivante :

```
V=[1,2,3,4]
```

Astuce : si l'introduction du vecteur est terminée par un point-virgule, on évite l'affichage de celui-ci, il en est de même avec les autres instructions.

L'accès aux composantes d'un vecteur s'effectue directement par des commandes du genre :

```
V(2)
```

Attention : 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.

3.2 Matrices

Une matrice peut être construite de différentes manières :

```
m=[5,2,8,1;10,20,30,40;22,24,26,28]
```

ce qui nous donne dans l'environnement MATLAB :

```
m=
     5    2    8    1
     10   20   30   40
     22   24   26   28
```

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

```
v1=[5,2,8,1];
v2=[10,20,30,40];
v3=[22,24,26,28];
m=[v1;v2;v3]
```

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

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

et l'on obtient : 40.

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

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

et l'on obtient le vecteur-colonne :

```
 1
40
28
```

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

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

et l'on obtient :

```
20    30    40
24    26    28
```

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

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

ce qui produit :

```
2     1
20    40
24    28
```

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

```
m'
```

ce qui produit dans notre cas :

```
5     10    22
2     20    24
8     30    26
1     40    28
```

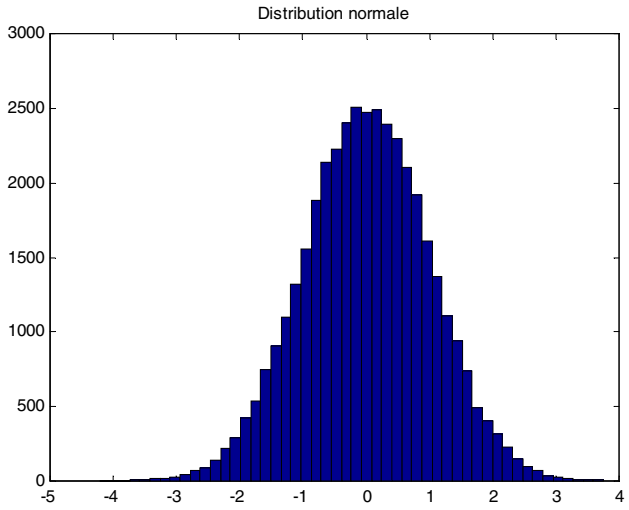
3.2.1 Construction des matrices particulières

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).

Matrices aléatoires :

On peut générer des matrices aléatoires dont les éléments sont distribués normalement avec une moyenne nulle et une variance unité à l'aide de la commande `randn(m,n)`. Pour une distribution uniforme, on utilisera la commande `rand(m,n)`. Les paramètres `m` et `n` désignent respectivement le nombre de lignes et de colonnes.

```
n=200;
A_norm=randn(n);
n_classes=50;
hist(A_norm(:),n_classes)
```



3.3 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)`).

3.4 Interpolation polynômiale

Dans MATLAB un polynôme p(x) est représenté par la liste de ses coefficients dans l'ordre des puissances décroissantes.

Exemple : $p(x) = x^3 - 6x^2 - 72x - 27$ est représenté par :

```
p=[1 -6 -72 -27]
```

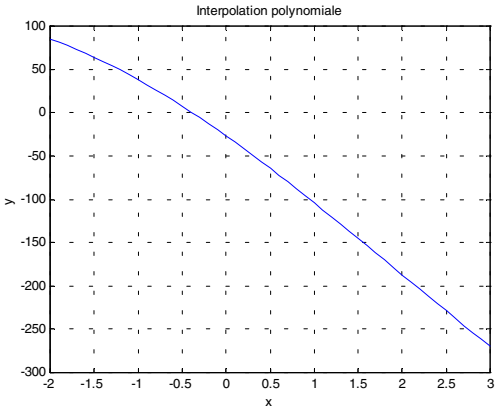
Pour évaluer le polynôme p(x) en $x = x_0$, on effectue :

```
polyval(p,x0)
```

Exemple : `polyval(p,-1.7)` , on obtient : `ans = 73.1470`

Pour obtenir le graphe de p(x) sur l'intervalle [-2,3] on effectue :

```
x=-2:0.1:3;
y=polyval(p,x)
plot(x,y);
```



3.5 Interpolation linéaires et non linéaires

La commande `interp1(x,y,z)` retourne un vecteur de mêmes dimensions que z dont les valeurs correspondent aux images des éléments de z déterminés par interpolation sur x et y.

```
f = interp1(x,y,z,'type')
```

La chaîne *'type'* spécifie un type d'interpolation parmi les suivants :

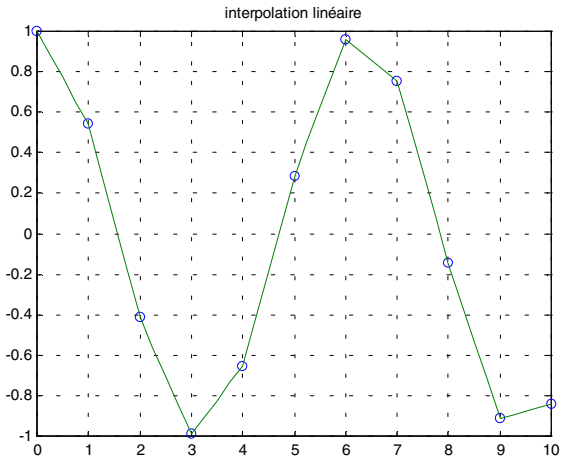
`'linear'` : interpolation linéaire
`'spline'` : interpolation par splines cubiques
`'cubic'` : interpolation cubique

Si l'on ne spécifie pas le type, l'interpolation linéaire est prise par défaut.

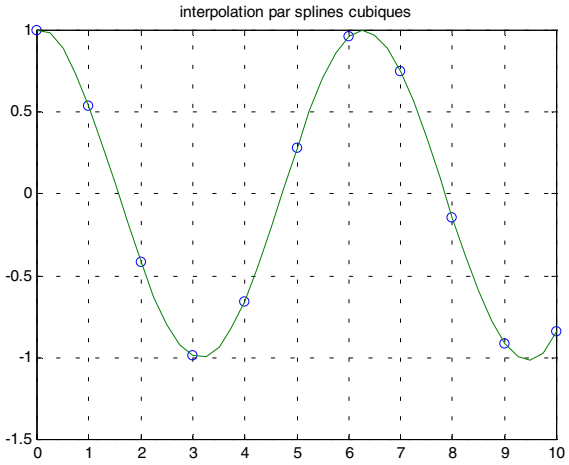
Exemple : visualisation des différents types d'interpolations sur les valeurs discrètes de la fonction cosinus.

```
% Code commun aux trois interpolations
x=0:10;
y=cos(x); %points à interpoler
z=0:0.25:10; %le pas du vecteur z inférieur à celui de x
```

```
%Interpolation linéaire
figure(1)
f=interp1(x,y,z);
plot(x,y,'o',z,f)
```

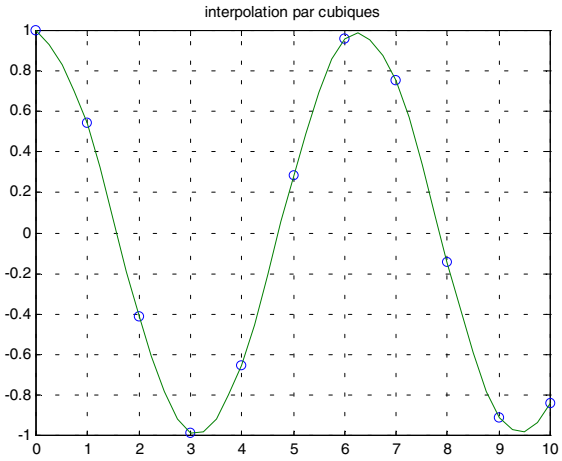


```
%Interpolation par
%splines cubiques
figure(2)
f=interp1(x,y,z,'spline');
plot(x,y,'o',z,f)
```



L'interpolation par splines cubiques peut être aussi obtenue par invocation de la commande `spline(x,y,z)`.

```
%Interpolation par cubique
figure(3)
f=interp1(x,y,z,'cubic');
plot(x,y,'o',z,f)
```



3.6 Interpolation au sens des moindres carrés

La commande `p=polyfit(x,y,n)` retourne le polynôme p de degré n permettant d'approcher la courbe `y=f(x)` au sens des moindres carrés.

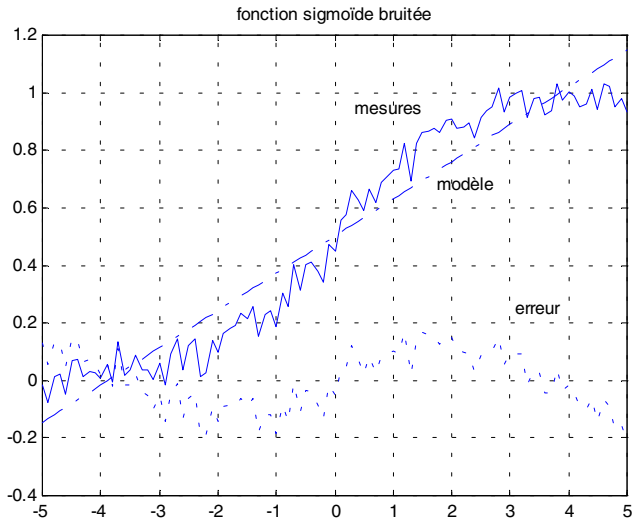
Afin d'en déduire l'erreur entre la courbe expériementale et le modèle obtenu, on dispose de la fonction `polyval(p,x)` qui retourne la valeur du polynôme p pour toutes les composantes du vecteur ou de la matrice x.

Pour expliquer l'application de ces fonctions, nous allons simuler une courbe expériementale par une sigmoïde à laquelle nous superposons un bruit gaussien.

```
%intervalle de définition et calcul de la sigmoïde
x=-5:0.1:5;
%fonction sigmoïde bruitée
y=1./(1+exp(-x))+0.05*randn(1,length(x));
%tracé de la sigmoïde bruitée
plot(x,y)
title('fonction sigmoïde bruitée')
%polynôme d'ordre 1 d'interpolation
p=polyfit(x,y,1);
%valeurs du polynôme d'interpolation
polyn=polyval(p,x);
%tracé du polynôme d'interpolation
hold on
plot(x,polyn,'-.')
%calcul de l'erreur d'interpolation
err=y-polyn;
%tracé de la courbe de l'erreur
plot(x,err,':')
grid
hold off
%affichage du polynôme d'interpolation
disp('polynôme d''interpolation')
p
var_err=num2str(std(err)^2);
disp(['variance de l''erreur d''interpolation : ' ,var_err])
```

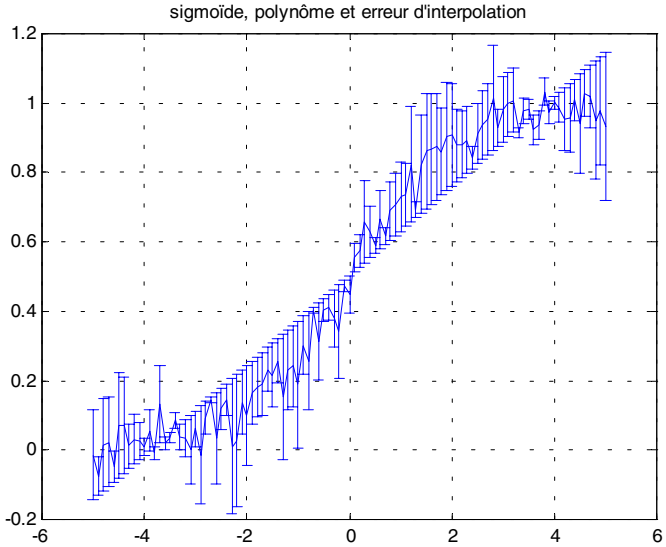
On obtient les résultats suivants :

```
polynôme d'interpolation

p =
    0.12926261594795   0.50178471754000

variance de l'erreur d'interpolation : 0.010239
```



La fonction `errorbar` permet de tracer simultanément, la courbe que l'on veut estimer par un polynôme et l'erreur d'estimation.

```
figure(2)
err=y-polyn;
errorbar(x,y,err)
title('sigmoïde, polynôme et erreur d''interpolation')
grid
```



3.7 Fonctions MATLAB pour les équations et systèmes différentiels

MATLAB dispose des fonctions `ode23` et `ode45` pour la résolution d'équations et de systèmes d'équations différentielles utilisant la méthode de Runge-Kutta.

La syntaxe des fonctions `ode23` et `ode45` est la suivante :

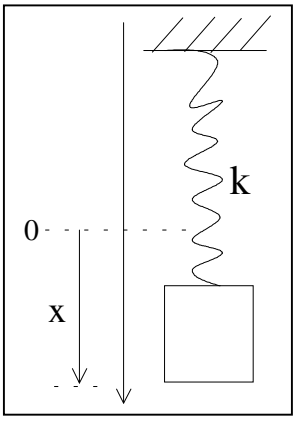
```
[x,y] = ode23(f, x0, xfinal, y0, tol, trace)

[x,y] = ode45(f, x0, xfinal, y0, tol, trace)
```

`f`	: chaîne de caractères représentant le nom d'un fichier M dans lequel est définie l'équation différentielle
`x0, xfinal`	: valeurs initiale et finale de la variable x
`y0`	: vecteur colonne de valeurs initiales de la fonction y=f(x)
`tol`	: précision désirée(facultative), la valeur par défaut est 10^{-3} pour ode23 et 10^{-6} pour ode45
`trace`	: affichage des résultats intermédiaires (1 : oui, 0 : non), ce paramètre est facultatif, sa valeur par défaut est 0.

Les fonctions `ode23` et `ode45` utilisent la méthode de Runge-Kutta-Fehlberg avec un calcul de pas automatique.

Le système d'équations différentielles doit être représenté dans un fichier fonction sous la forme d'un vecteur colonne où chaque ligne représente une équation du système.

Exemple : Pendule élastique amorti
Considérons un corps de masse m, accroché à un ressort de constante de raideur k. La résistance de l'air crée une force de frottement proportionnelle à la vitesse avec un coefficient f.



La position x de la masse est repérée par rapport à l'origine 0 correspondant à la longueur naturelle du ressort. Par application de la loi fondamentale de la dynamique, on obtient l'équation différentielle qui régit l'évolution de la position x.

$$\ddot{x} = -\frac{f}{m}\dot{x} - \frac{k}{m}x + g$$

Cette équation du second ordre peut être transformée en un système de 2 équations du premier ordre.

$$\begin{cases} u' = -\frac{f}{m}u - \frac{k}{m}v + g \\ v' = u \end{cases}$$

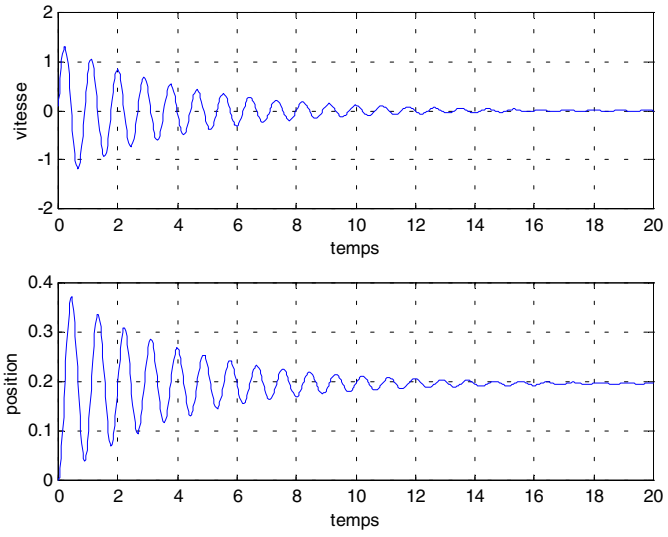
Pour résoudre ce système différentiel, on utilisera la fonction `ode45` prévue par MATLAB. Cette fonction utilise la méthode de Runge-Kutta avec un calcul de pas automatique.
Pour utiliser la fonction `ode45`, nous avons besoin de deux fichiers : un fichier fonction dans lequel on décrit le système différentiel et un fichier script qui fait appel à la fonction `ode45` et qui réalise le tracé des courbes.

fichier rsprime.m

```
function sd=rsprime(t,s)
global k m f g
%système différentiel
sd=[(-k*s(2)-f*s(1))/m+g;s(1)];
```

fichier rs_nl.m

```
global k m f g
k=10; m=0.2; f=0.1; g=9.81;
%conditions initiales
t0=0; tf=20; x0=0.1; v0=0;
[t,x]=ode45('rsprime',t0,tf,[x0,v0]);
%tracé de la vitesse
subplot(211)
plot(t,x(:,1))
xlabel('temps')
ylabel('vitesse')
grid
%tracé de la position
subplot(212)
plot(t,x(:,2))
xlabel('temps')
ylabel('position')
grid
```



4.0 La programmation avec MATLAB

4.1 Opérateurs et caractères spéciaux

Le tableau suivant résume les opérateurs que l'on peut rencontrer dans MATLAB ainsi qu'un certain nombre de caractères spéciaux.

Symbole	***fonction***
`+`	Addition de réels et de matrices
`-`	Soustraction de réels et de matrices
`*`	Produit de réels et de matrices
`.*`	Produit élément par élément de matrices
`^`	Élévation à une puissance de réels et de matrices
`.^`	Puissance élément par élément de matrices
`\`	Division à gauche de réels et de matrices
`/`	Division à droite de réels et de matrices (division classique)
`./`	Division élément par élément de matrices
`==`	Égalité
`~=`	Différent
`<`	Strictement inférieur
`<=`	Inférieur ou égal
`>`	Strictement supérieur
`>=`	Supérieur ou égal
`&`	ET logique (AND)
`|`	OU logique (OR)
`~`	NON logique (NOT)
`xor`	OU exclusif (XOR)
`%`	Commentaires
`=`	Affectation
`'`	Transposée de matrices et délimiteur de chaînes de caractères
`!`	Commandes système (échappement)
`,`	Séparation d'instructions ou de réels dans une matrice
`;`	Séparation d'instructions (pas d'affichage de valeurs intermédiaires) ou de lignes d'une matrice
`...`	Symbole de continuation (pour terminer une instruction ou une expression à la ligne suivante)
`.`	Point décimal
`..`	Répertoire parent du répertoire en cours
`[ ]`	Définition de vecteurs ou de matrices
`( )`	Utilisation dans des expressions ou des fonctions

4.2 Instructions et commandes structurées

MATLAB dispose des instructions structurées suivantes : `for, while` et `if`.

4.2.1 Instruction for

syntaxe

```
for variable = expression
      instructions
end
```

Exemple :

```
n=7;
x=[];
for i=1:n,
        x=[x, sqrt(i)];
end
```

Les boucles `for` peuvent être imbriquées entre elles et avec les autres instructions de contrôle.

4.2.2 Instruction while

syntaxe

```
while expression
      instructions
end
```

Exemple :

```
n=0;
while (x^n <= Max)
        n=n+1;
end
```

4.2.3 Instruction if

syntaxe

```
if expression
      instructions I1
else
      instructions I2
end
```

En cas d'instructions conditionnelles imbriquées, un `else` est toujours associé au `if` le plus proche. Dans le cas de sélections multiples, on utilisera l'instruction `elseif`.

Exemple :

```
if moy >= 5.5
        mention = 'très bien';
elseif moy >= 5
        mention = 'bien';
elseif moy >= 4.5
        mention = 'Assez bien';
elseif moy >= 4
        mention = 'Passable';
else
        mention = 'Ajourné(e)';
end
```

4.2.4 Instructions de rupture de séquence

`break` : Termine l'exécution d'une boucle. Si plusieurs boucles sont imbriquées, `break` permet de sortir de la boucle la plus proche.

`return` : Cette instruction permet de revenir au fichier M ayant appelé le programme courant ou à la ligne de commandes MATLAB.

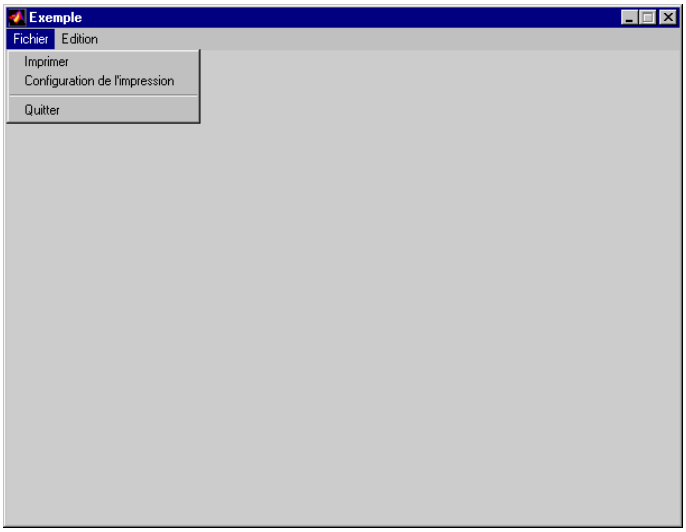
`error('message')` : Affiche le message spécifié, émet un "bip" et interrompt l'exécution du programme.

4.3 Réalisation d'interfaces graphiques

Visualisons directement un exemple, pour expliquer la création de menus, dans une fenêtre MATLAB.

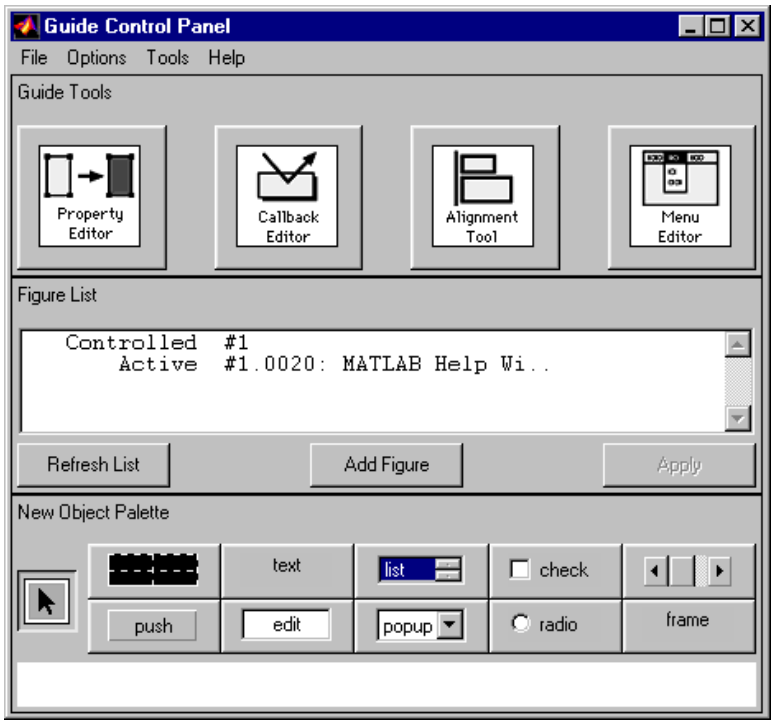
```
% Création de la fenêtre graphique
figure(gcf) ;
% Déclaration des propriétés de la fenêtre
set(gcf,'Resize','Off','Name','Exemple',...
        'Menubar','None','NumberTitle','off',...
        'Position',[120 120 600 420]);
% Définitions des menus
fichier=uimenu('Label','Fichier');
edition=uimenu('Label','Edition');

% Définitions des sous-menus
uimenu(fichier,...
       'label','Imprimer',...
       'callback','print -f');
uimenu(fichier,...
       'label','Configuration de l''impression',...
       'callback','print -dsetup');
uimenu(fichier,...
       'Label','Quitter',...
       'separator','on',...
       'Callback','close(gcf)');

uimenu(edition,...
       'label','Copier au format Metafile',...
       'callback','print -dmeta');
uimenu(edition,...
       'label','Copier au format Bitmap',...
       'callback','print -dbitmap');
```

Le résultat obtenu est le suivant :



Il existe une autre façon de réaliser une interface graphique depuis la version 5 de MATLAB. Cette outil est destiné à ceux qui aiment travailler de façon visuelle (même principe que Delphi, Visual C++, Visual Basic).
L'invocation de l'éditeur se fera à l'aide de la commande `guide`. Une fenêtre principale contenant les outils de création (panneau de contrôle) et une fenêtre vierge pour la création de l'interface sont alors présentes à l'écran.

```
guide
```



Pour plus d'explication, l'utilisateur peut consulter le livre intitulé "Apprendre et Maîtriser MATLAB" chapitre 5 'La programmation orientée Objets', page 416.

Et pour plus de détails sur la réalisation d'interfaces graphiques, au chapitre 10, page 301.

5.0 Complément : Analyse de systèmes dynamiques linéaires à l'aide de la boîte à outils "*control systems*"

5.1 Introduction de fonctions de transfert (transmittances)

5.1.1 Introduction sous forme polynômiale

L'introduction de fonctions de transfert s'effectue en deux temps, les numérateurs et dénominateurs doivent être donnés séparément. Le principe est simple : les numérateurs et dénominateurs 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

$$G(s) = \frac{Y(s)}{U(s)} = 50 \cdot \frac{1 + s \cdot 12}{1 + s \cdot 4 + s^2 \cdot 0.2}$$

Son introduction dans MATLAB se fait par les commandes :

```
numG=50*[12,1];
denG=[0.2,4,1];
```

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 :

$$G(s) = \frac{3.4567}{s}$$

est introduite sous la forme :

```
numG=3.4567;
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.

5.1.2 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]';
pG=[0,-11,-5+j*3,-5-j*3]';
kG=5.32;
```

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

5.2 Passage d'un modèle à l'autre

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

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

MATLAB place alors les zéros et les pôles dans les vecteurs-colonne zG et pG, respectivement, et le facteur d'Evans dans la variable kG. Les noms zG, pG et kG sont arbitraires.

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 :

Forme polynômiale ⇨ forme en zéros, pôles et gain

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

Forme polynômiale ⇨ modèle d'état

```
[AG,BG,CG,DG]=tf2ss(numG,denG);
```

Forme en zéros, pôles et gain ⇨ forme polynômiale

```
[numG,denG]=zp2tf(zG,pG,kG);
```

Forme en zéros, pôle et gain ⇨ modèle d'état

```
[AG,BG,CG,DG]=zp2ss(zG,pG,kG);
```

Modèle d'état ⇨ forme polynômiale

```
[numG,denG]=ss2tf(AG,BG,CG,DG);
```

Modèle d'état ⇨ forme en zéros, pôles et gain

```
[zG,pG,kG]=ss2zp(AG,BG,CG,DG);
```

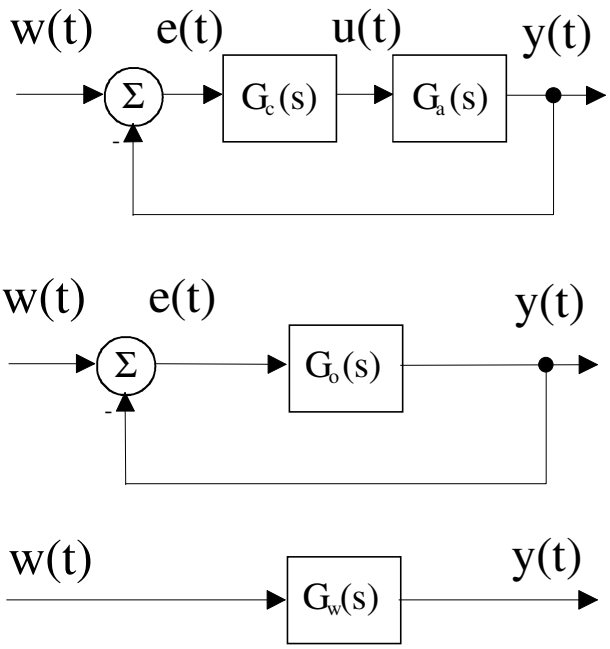
Modèle d'état ⇨ autre modèle d'état, avec matrice de transformation T

```
[AG2,BG2,CG2,DG2]=ss2ss(AG,BG,CG,DG,T);
```

5.3 Construction de schémas fonctionnels

5.3.1 Fonctions *series*, *cloop*

Prenons par 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, il faut procéder par étapes, comme indiqué par la figure ci-dessous :



1) Calcul de $G_a(s)$ par la mise en série de G_{a1} et G_{a2}.

On fait usage de la fonction `series` :

```
[numGa,denGa]=series(numGa1,denGa1,numGa2,denGa2);
```

2) Calcul de $G_o(s)$ par la mise en série de G_c et G_a

On procède de même qu'en 1) :

```
[numGo,denGo]=series(numGc,denGc,numGa,denGa);
```

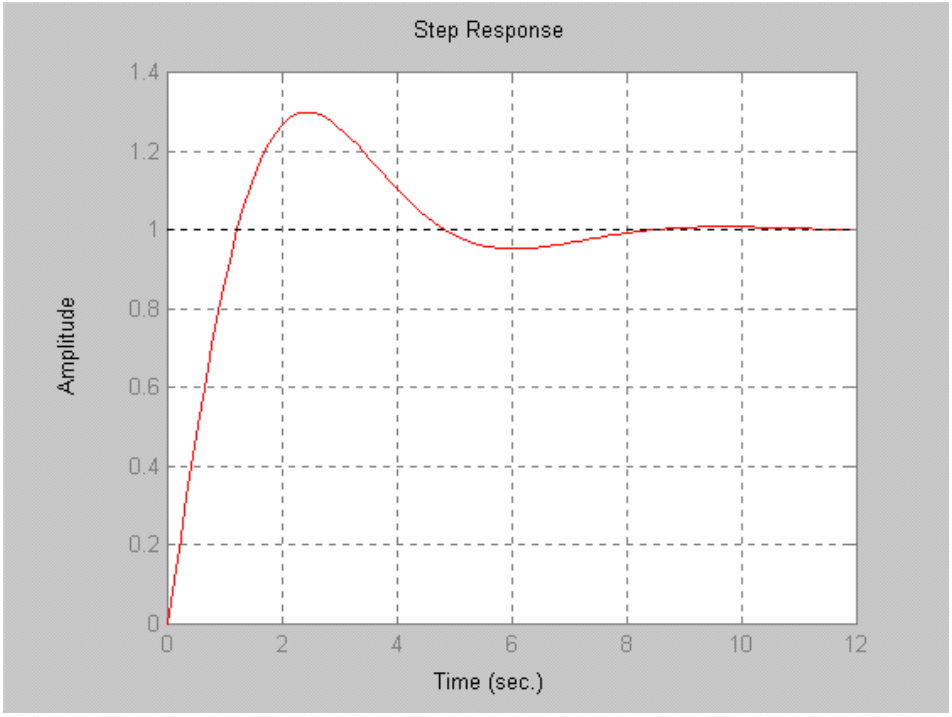
3) Calcul de $G_w(s)$ par fermeture de la boucle (retour unitaire)

On fait usage de la fonction `cloop` comme suit :

```
[numGw,denGw]=cloop(numGo,denGo);
```

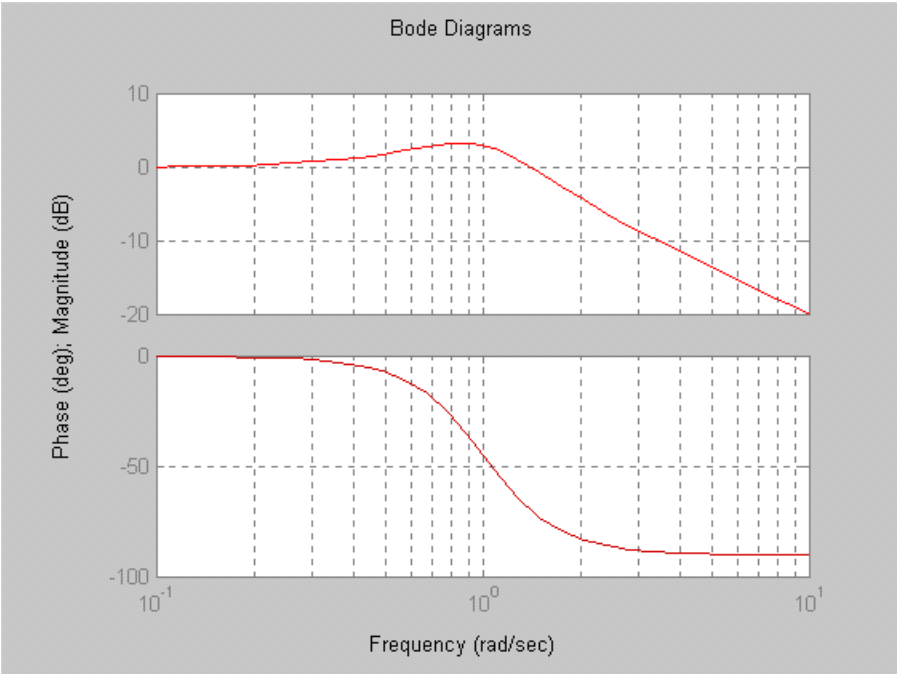
La réponse indicielle en boucle fermée peut alors être calculée et tracée par :

```
step(numGw,denGw);
```



de même que la réponse harmonique en boucle ouverte par :

```
bode(numGo,denGo);
```

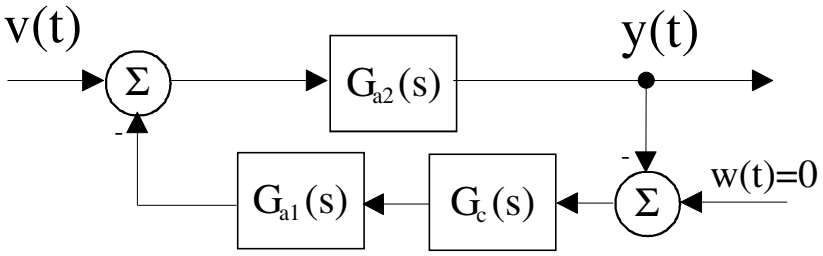


5.3.2 Fonction feedback

La fonction `feedback` est utile pour calculer la fonction de transfert équivalente du systèmes ayant pour schéma fonctionnel :

```
[numGv,denGv]=feedback(numGa,denGa,numGc,denGc,signe);
```

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



5.4 Calcul et tracé de réponses de systèmes dynamiques linéaires

5.4.1 Réponse temporelle

Les fonctions `impulse, step, lsim`, et pour les systèmes numériques, `dimpulse, dstep, dlsim` sont disponibles. Leur signification est la suivante :

SYSTEMES ANALOGIQUES :

Réponse impulsionnelle :

```
impulse(numG,denG,t);
```

Réponse indicielle :

```
step(numG,denG,t);
```

Réponse à une entrée u quelconque définie par l'utilisateur :

```
lsim(numG,denG,u,t);
```

Réponse du système lorsque ses conditions initiales sont x0 :

```
initial(A,B,C,D,x0,t);
```

SYSTEMES NUMERIQUES :

Réponse impulsionnelle :

```
dimpulse(numH,denH,n);
```

Réponse indicielle :

```
dstep(numH,denH,n);
```

Réponse à une entrée u quelconque définie par l'utilisateur :

```
dlsim(numH,denH,u);
```

Réponse du système lorsque ses conditions initiales sont x0 :

```
dinitial(A,B,C,D,x0,n);
```

Les paramètres t, n et w sont optionnels et MATLAB les détermine lui-même par défaut. Si l'on souhaite les fixer, on peut par exemple le faire selon les indications suivantes :

`t` : (facultatif) vecteur-ligne de temps, spécifiant les instants où la réponse doit être calculée (ce paramètre est obligatoire si l'on utilise `lsim`)

Exemple :

Crée un vecteur temps variant entre 0[s] et 0.2[s] par pas de 0.01[s].

```
t=[0:0.01:0.2];
ou
t=linspace(0,0.2,201);
```

`n` : (facultatif) nombre d'échantillons désirés

Exemple :

Seuls les 10 premiers échantillons de la réponse seront affichés.

```
n=10;
```

`w` : (facultatif) vecteur-ligne de pulsation, spécifiant les pulsations où la réponse doit être calculée.

Exemple :

Crée un vecteur pulsation de 100 points espacés logarithmiquement et variant entre 10^2[rad/s] et 10^5[rad/s].

```
w=logspace(2,5,100);
```

`u` : vecteur-colonne de l'entrée simulée du système (ce paramètre est obligatoire si l'on utilise `lsim` ou `dlsim`)

Exemple :

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.

```
u=1.5*sin(2*pi*10*t)';
```

Lors du calcul de la réponse harmonique de systèmes numériques, 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 sauvera les résultats de ses calculs :

SYSTEMES ANALOGIQUES	SYSTEMES NUMERIQUES
`[y,x,t]=impulse(numG,denG,t);`	`[y,x]=dimpulse(numH,denH,n);`
`[y,x,t]=step(numG,denG,t);`	`[y,x]=dstep(numH,denH,n);`
`[y,x]=lsim(numG,denG,u,t);`	`[y,x]=dlsim(numH,denH,u);`

y est un vecteur-colonne contenant la réponse cherchée et t un vecteur-ligne contenant les instants auxquels elle a été calculée. x est une matrice contenant l'évolution du vecteur d'état système dynamique.

5.4.2 Réponse fréquentielle

Les fonctions `bode, nyquist, nychols,` et pour les systèmes numériques, `dbode, dnyquist, dnychols` sont disponibles. Leur signification est la suivante :

SYSTEMES ANALOGIQUES	SYSTEMES NUMERIQUES
`bode(numG,denG,w)`	`Dbode(numH,denH,h,w)`

Si les résultats des calculs doivent être sauvegardés en vue d'un traitement ultérieur, on indique également des arguments de sortie :

SYSTEMES ANALOGIQUES	SYSTEMES NUMERIQUES
`[A,phi,w]=bode(numG,denG,w)`	`[A,phi,w]=dbode(numH,denH,h,w)`

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

5.5 Analyse des propriétés des systèmes

Gain statique d'un système analogique :

```
dcgain(num,den);
```

Gain statique d'un système numérique :

```
ddcgain(num,den);
```

Taux d'amortissement ζ, pulsation propre non-amortie ω_n et pôles d'un système analogique ayant `den` pour dénominateur :

```
damp(den);
```

Taux d'amortissement ζ, pulsation propre non-amortie ω_n et pôles d'un système discret ayant `den` pour dénominateur :

```
ddamp(den);
```

5.6 Calcul ,affichage des marges de gain et de phase

Marges de phase et de gain d'un système analogique.
Si les arguments de sortie ne sont pas spécifiés, trace le lieu de Bode et montre graphiquement la valeur des marges.

```
[Am,phi_m]=margin(numGo,denGo);
```

5.7 Tracé du lieu d'Evans

Trace le lieu d'Evans 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.

```
rlocus(num,den,k)
```

Affiche les courbes équi-amortissement des plans de s et de z respectivement. A exécuter immédiatement après `rlocus`.

```
sgrid
zgrid
```

Configuration pôles-zéros de la fonction de transfert.

```
pzmap(num,den)
```

Tracé interactif du lieu d'Evans. 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.

```
[k,p]=rlocfind(num,den)
```

5.8 Divers

Force la compensation pôle-zéro.

```
minreal(num,den)
```

Affiche les fonctions de transfert comme fractions rationnelles en s ou en z.

```
printsys(num,den,'s')
printsys(num,den,'z')
```

Réunis le numérateur et le dénominateur en une seul variable composée d'une matrice.

```
G0=tf(num,den)
```

L'inverse de la fonction précédente, partant d'une seul variable composée d'un vecteur en un numérateur et un dénominateur .

```
[num,den]=tfdata(G0,'v')
```

6.0 Applications (exemples)

6.1 Réalisation d'une fonction qui affiche le lieu de Bode

Pour tracer un lieu de Bode, il faut avoir au préalable la toolbox "system control". Mais nous allons réaliser une fonction traçant le lieu de Bode, sans toolbox!!!

De plus cette fonction est capable de déterminer la marge de phase φ_m et de gain A_m ainsi que les pulsations ω_{co} et ω_π correspondantes.

Tout d'abord, il s'agit de créer un fichier fonction, avec des paramètres d'entrées, tels que le numérateur et le dénominateur, donnés dans le domaine de s, de la fonction de transfert que l'on désire tracer.

```
function[]=bode_sbe(num,den,w_min,w_max,titre,nb_pts)
```

Nous traiterons ensuite les erreurs et les valeurs par défaut avec la commande `nargin`.

Il ne faut pas oublier que les variables définies dans une fonction, ne sont pas vues de l'extérieur! D'où l'emploi de `global`.

La suite consiste à créer la fenêtre graphique avec ses menus, tel que décrit au chapitre 4.3.

Algorithme du lieu de Bode :

Création d'un espacement logarithmique avec `logspace`.

```
logspace(d1,d2,N)
```

qui génère N valeurs espacées d'un pas logarithmique entre 10^{d1} et 10^{d2} , si N est omis sa valeur par défaut est 50.

Création de la réponse fréquentielle complexe dans un vecteur.

```
h=freqs(num,den,w)
```

L'amplitude du diagramme de Bode est déterminé par l'extraction du module du nombre complexe.

```
mag=abs(h)
```

Pour la phase, cela devient un petit peu plus compliqué, vu que l'arctangente, ne traite pas des angles au-delà de $\pm 180°$. Nous allons donc fabriquer un algorithme capable de gérer la phase à $\pm 360°$.

Il ne faut pas oublier de transformer les radians en degré à la fin, puisque MATLAB travail en radian.
La phase suivante est le tracé de l'amplitude et de la phase.
Pour cela nous utiliserons la commande `subplot`, qui divise la fenêtre graphique en plusieurs zones.

```
subplot(m,n,p)
```

avec `m`, qui désigne le nombre de lignes et `n` le nombre de colonnes et trace le graphique qui suit cette instruction dans la zone de numéro `p` (la numérotation se fait de gauche à droite et ligne par ligne).

Pour l'affichage de l'amplitude, le graphique aura ses échelles en logarithmique.

```
loglog(f,mag)
```

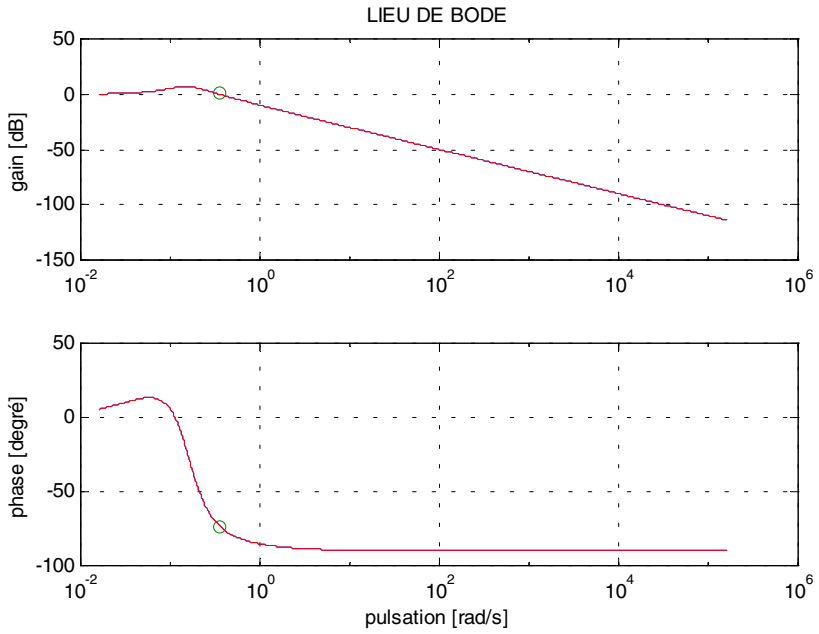
Pour l'affichage de la phase, l'échelle de la pulsation sera logarithmique, mais pas celle de la phase.

```
semilogx(f,phase)
```

Et pour finir, il est intéressant de voir, qu'il est possible de modifier les propriétés d'un objet graphique. Dans notre cas, nous allons modifier la couleur des courbes en rouges.

```
line(f,mag,'color','r')
line(f,phase,'color','r')
```

Exemple de tracé du lieu de Bode :



Fichier Source :

(bode_sbe.m)

```
%/*
%*****************************************************************************
%  bode_sbe.M
%*****************************************************************************
%
% Lieu de Bode sans TOOLBOX !!!!!!
%
%*****************************************************************************
%*****************************************************************************
%Version                Date            Auteur          Motif
% 0                   28/04/00           SBE          Creation
%
%*****************************************************************************
%*/
% Trace le lieu de Bode
% ATTENTION, la phase de la fonction de transfert n'est valable
% que dans l'intervalle de +-360°
%
% function [] = bode_sbe(num,den,w_min,w_max,titre,nb_pts)
%
% num      :  numérateur de la fonction G(s)
% den      :  dénominateur de la fonction G(s)
% w_min    :  borne inférieur du diagramme de bode 10^w_min
% w_max    :  borne supérieur du diagramme de bode 10^w_max
% titre    :  titre du diagramme, à donner entre guillemets simples
% nb_pts   :  nombre de points à calculer

function[]=bode_sbe(num,den,w_min,w_max,titre,nb_pts)

 nargs = 6;

 if nargin < nargs,    nb_pts=1000;        end
 if nargin < nargs-1,  titre='LIEU DE BODE'; end
 if nargin < nargs-2,  w_max=6;        end
 if nargin < nargs-3,  w_min=-1; end
 if nargin < nargs-4,
  error('Il manque le numérateur et le dénominateur');
 end

   % Variables externes
   global hZoomOn hZoomOff hZoomreset
   % Variables internes
   quadrant=1;
   previous_quadrant=1;
   quadrant_III_to_II=0;
   quadrant_II_to_III=0;

   close all;
   figure(gcf)
   set(gcf,'Resize','Off','Name','Lieu de Bode',...
         'Menubar','None','NumberTitle','off',...
         'Position',[120 120 600 420]);
   fichier=uimenu('Label','Fichier');
   sauver=uimenu(fichier,...
                    'label','Sauvegarde');
                    uimenu(sauver,...
                           'label','Sauvegarder au format BMP',...
                           'callback','print -dbitmap bode');
                    uimenu(sauver,...
                           'label','Sauvegarder au format Metafile',...
                           'callback','print -dmeta bode');
                    uimenu(sauver,...
                           'label','Sauvegarder au format postcript',...
                           'callback','print -deps bode');
```

```
   impression=uimenu(fichier,...
                     'label','Impression');
                      uimenu(impression,...
                             'label','Imprimer Lieu de Bode',...
                             'callback','print -f');
                      uimenu(impression,...
                             'label','Configuration de l''impression',...
                             'callback','print -dsetup');
   uimenu(fichier,...
          'Label','Quitter',...
          'separator','on',...
          'Callback','close(gcf)');
   edition=uimenu('Label','Edition');
                  uimenu(edition,...
                         'label','Copier au format Metafile',...
                         'callback','print -dmeta');
                  uimenu(edition,...
                         'label','Copier au format Bitmap',...
                         'callback','print -dbitmap');
   hZoom=uimenu('Label','Zoom');
                hZoomOn=uimenu(hZoom,...
                               'label','Enclenchement du zoom',...
                               'callback',['type_zoom=1;','gest_opt']);
                hZoomOff=uimenu(hZoom,...
                                'label','Déclenchement du zoom',...
                                'callback',['type_zoom=2;','gest_opt']);
                hZoomreset=uimenu(hZoom,...
                                  'label','Zoom Reset',...
                                  'separator','on',...
                                  'callback',['type_zoom=3;','gest_opt']);
   set(hZoomOff,'checked','on');
   uimenu('Label','Quitter','Callback','close(gcf)');
w=logspace(w_min,w_max,nb_pts);
h=freqs(num,den,w);
f=w/(2*pi);
mag=abs(h);
i=1;
while i~= length(h)+1
 reel=real(h(i));
 imaginaire=imag(h(i));
 if (imaginaire>=0)
     if ((reel>=0)&(quadrant_II_to_III==0))
        quadrant=1;
        phase(i)=atan2(imaginaire,reel)*(180/pi);
     else
       if (quadrant_II_to_III==0)
          quadrant=2;
          if (previous_quadrant==3)
             quadrant_II_to_III=1;
             phase(i)=(atan2(imaginaire,reel)*(180/pi))-360;
          else
             phase(i)=atan2(imaginaire,reel)*(180/pi);
          end
       else
          if (reel>=0)
             quadrant=1;
             phase(i)=(atan2(imaginaire,reel)*(180/pi))-360;
          else
             quadrant=2;
             phase(i)=(atan2(imaginaire,reel)*(180/pi))-360;
          end
       end
     end
 else
     if ((reel>=0)&(quadrant_III_to_II==0))
        quadrant=4;
        phase(i)=atan2(imaginaire,reel)*(180/pi);
     else
       if (quadrant_III_to_II==0)
          quadrant=3;
          if (previous_quadrant==2)
```

```
            quadrant_III_to_II=1;
            phase(i)=(atan2(imaginaire,reel)*(180/pi))+360;
         else
            phase(i)=atan2(imaginaire,reel)*(180/pi);
         end
      else
         if (reel>=0)
            quadrant=4;
            phase(i)=(atan2(imaginaire,reel)*(180/pi))+360;
         else
            quadrant=3;
            phase(i)=(atan2(imaginaire,reel)*(180/pi))+360;
         end
      end
    end
 end
 if (abs(phase(i))>=359)
  warning('ATTENTION, la phase de la fonction de transfert dépasse 360°');
 end
 previous_quadrant=quadrant;
 i=i+1;
end
subplot(2,1,1),loglog(f,mag);
line(f,mag,'color','r');
title(titre)
ylabel('gain [dB]')
grid on
subplot(2,1,2),semilogx(f,phase);
line(f,phase,'color','r');
xlabel('pulsation [rad/s]')
ylabel('phase [degré]')
grid on
clc;
% Recherche de la marge de phase et de gain
k=2;
while k~=length(mag)
 if ((20*log10(mag(k))<0)&(20*log10(mag(k-1))>0))|(20*log10(mag(k))==0)
    position_zero=k;
    break
 else
    k=k+1;
 end
end
if exist('position_zero')
  pulsation_wco=f(position_zero);
  marge=180-abs(phase(position_zero));
  marge_phase=phase(position_zero);
  magnitude=mag(position_zero);
  if marge<=0
    phi_m=0;
  else
    phi_m=marge;
  end
  fprintf('Marge de phase : %f[°]\n', phi_m)
  fprintf('Pulsation wco : %f[rad/s]\n', pulsation_wco)
else
 warning('Attention, pas de marge de phase, le gain ne passe pas par 0[dB]')
 pulsation_wco=0;
 marge_phase=0;
 magnitude=0;
end
fprintf('\n')
k=2;
while k~=length(phase)
 if ((phase(k)<-180)&(phase(k-1)>-180))|(phase(k)==-180)
    position_phase=k;
    break
 else
    k=k+1;
 end
end
```

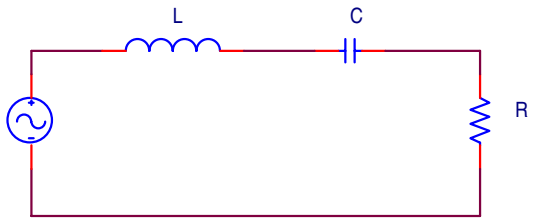
```
if exist('position_phase')
  pulsation_wpi=f(position_phase);
  am=mag(position_phase);
  if am<1
    am=-am;
  end
  fprintf('Marge de gain : %f[dB]\n',am)
  fprintf('Pulsation wpi : %f[rad/s]\n',pulsation_wpi)
else
 warning('Attention, pas de marge de gain, la phase ne passe pas par -180')
end
magdb = 20*log10(mag);
subplot(2,1,1),semilogx(f,magdb,pulsation_wco,magnitude,'o');
line(f,magdb,'color','r');
title(titre)
ylabel('gain [dB]')
grid on
subplot(2,1,2),semilogx(f,phase,pulsation_wco,marge_phase,'o');
line(f,phase,'color','r');
xlabel('pulsation [rad/s]')
ylabel('phase [degré]')
grid on
```

(gest_opt.m)

```
global hZoomOn hZoomOff hZoomreset
set(hZoomOn,'checked','off');
set(hZoomOff,'checked','off');
set(hZoomreset,'checked','off');

if type_zoom == 1
  set(hZoomOn,'checked','on');
  zoom on;
elseif type_zoom == 2
  set(hZoomOff,'checked','on');
  zoom off;
elseif type_zoom == 3
  set(hZoomOff,'checked','on');
  zoom out;
  zoom off;
end
```

6.2 Réponse en fréquence d'un filtre RLC

Nous allons maintenant réaliser un filtre RLC, et tracer son lieu de Bode avec la fonction que nous venons de créer dans l'exemple précédent.
Schéma électrique :



Equation correspondante :

$$V_{out}(s) = V_{in}(s) \cdot \frac{R}{sL + \frac{1}{sC} + R}$$

Ce qui nous donne comme fonction de transfert :

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{sRC}{s^2LC + sRC + 1} = \frac{sRC}{\left(s \cdot \frac{2LC}{RC - \sqrt{(RC)^2 - 4LC}} + 1\right) \cdot \left(s \cdot \frac{2LC}{RC + \sqrt{(RC)^2 - 4LC}} + 1\right)}$$

Rappel : le numérateur et le dénominateur doivent être déclarés avec les puissances de s décroisssantes.
Bien sûr qu'il est plus approprié de donner le dénominateur sous la forme de Bode, pour la compréhension, mais cela devient plus compliqué à exprimer.
Si cela avait été le cas, comme le dénominateur est du second ordre, il aurait été judicieux d'utiliser la convolution, comme suit :

```
den=conv([T1 1],[T2 1])
```

Fichier Source :
(rlc.m)

```
% Fichier : rlc.m
% Auteur  : Serge Bedwani
% Date    : 04.05.00
% But     : Réponse en fréquence d'un filtre RLC

L=5;       % Self [H]
C=1.25e-6; % Condensateur [F]
R1=10000;  % Résistance [Ohm]
R2=100;    % Résistance [Ohm]

numG1=[C*R1 0];
denG1=[L*C C*R1 1];

numG2=[C*R2 0];
denG2=[L*C C*R2 1];
```

```
% Diagramme de Bode
bode_sbe(numG1,denG1,0,4,'Réponse pour R=10k'); pause;
bode_sbe(numG2,denG2,0,4,'Réponse pour R=10k'); pause;
% Forme de Bode
racine1=sqrt((R1*C)^2-(4*L*C));
numB1=[C*R1 0];
denB1=conv([((2*L*C)/((R1*C)-racine1)) 1],[((2*L*C)/((R1*C)+racine1)) 1]);
bode_sbe(numB1,denB1,0,4,'Réponse pour R=10k sous forme de Bode');
pause;
racine2=sqrt((R2*C)^2-(4*L*C));
numB2=[C*R2 0];
denB2=conv([((2*L*C)/((R2*C)-racine2)) 1],[((2*L*C)/((R2*C)+racine2)) 1]);
bode_sbe(numB2,denB2,0,4,'Réponse pour R=0.1k sous forme de Bode');
```

Diagramme de Bode obtenu avec R = 10k :

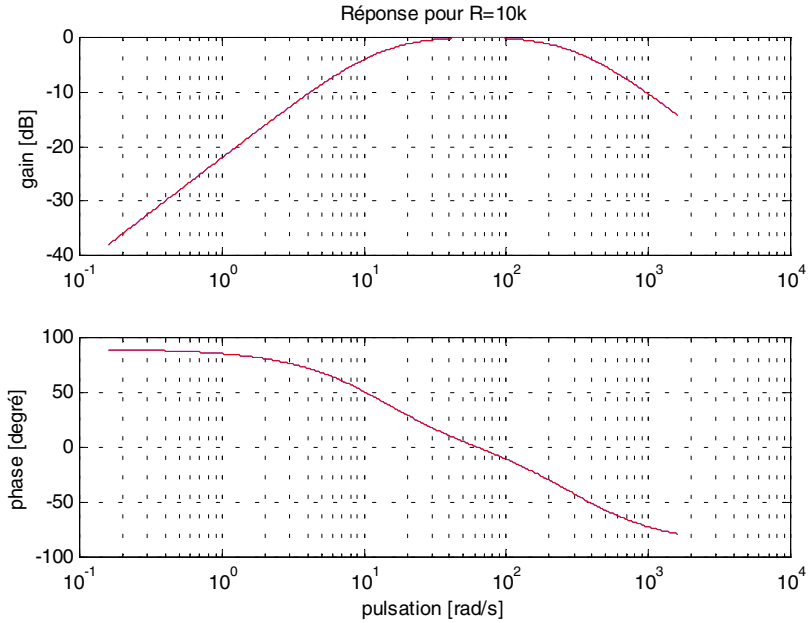
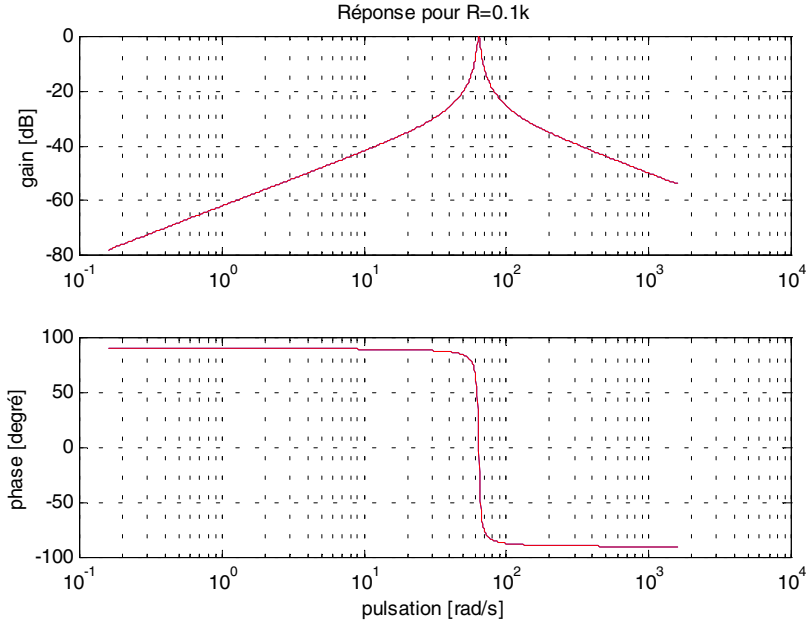


Diagramme de Bode obtenu avec R = 0.1k :



6.3 Sous-échantillonnage d'une sinusoïde

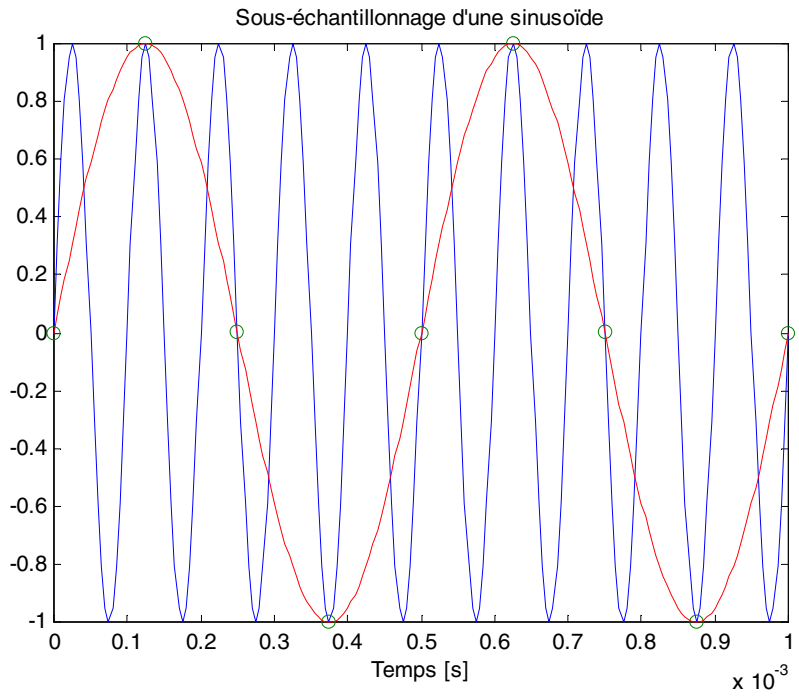
Considérons un signal sinusoïdal x(t) de fréquence f_o = 10kHz que l'on échantillonne avec une fréquence f_e = 8kHz.
Fréquence de repliement :

$$f_r = |f - nf_e| = f_o - f_e = 10\text{kHz} - 8\text{kHz} = 2\text{kHz}$$

Fichier Source :
(sousech.m)

```
% Fichier : sousech.m
% Date    : 9.05.00
% But     : Sous-échantillonnage d'une sinusoïde
% paramètres
fo=10e3;  % fréquence du signal sinusoïdal
fe=8e3;   % fréquence d'échantillonnage
To=1/fo;  % période du signal sinusoïdal
Te=1/fe;  % période d'échantillonnage
fa=fo-fe; % fréquence apparente
tmax=10*To;
% Calcul de x(t)
t=0:tmax/200:tmax;
xt=sin(2*pi*t/To);
% Calcul du signal apparent
xta=sin(2*pi*t*fa);
% Echantillonnage de x(t)
tn=0:Te:tmax;
xn=sin(2*pi*tn/To);
% Traçage des courbes
plot(t,xt,tn,xn,'o',t,xta,'-');
title('Sous-échantillonnage d''une sinusoïde');
xlabel('Temps [s]');
```

Signal sinusoïdal de fréquence f_o et signal sinusoïdal dû au sous-échantillonnage :



6.4 Suite d'impulsions rectangulaires

La suite d'impulsions rectangulaires (SIR) est un signal particulièrement important car elle apparaît dans de nombreuses applications telles que l'échantillonnage, la modulation d'impulsions, etc.

Evaluons donc la série de Fourier complexe de la SIR x(t). Par définition des coefficients complexes X(jk), on a:

$$X(jk) = \frac{1}{T} \cdot \int_{-T/2}^{+T/2} x(t) \cdot e^{(-jk \cdot 2\pi f_o \cdot t)} \cdot dt \quad \text{avec} \quad f_o = 1/T$$

En tenant compte de la définition de la SIR, il vient :

$$X(jk) = \frac{A}{T} \cdot \int_{-\Delta t/2}^{+\Delta t/2} e^{(-jk \cdot 2\pi f_o \cdot t)} \cdot dt$$

$$X(jk) = \frac{A}{T} \cdot \frac{-1}{jk2\pi f_o} \left[e^{(-jk \cdot 2\pi f_o \cdot \frac{\Delta t}{2})} - e^{(+jk \cdot 2\pi f_o \cdot \frac{\Delta t}{2})} \right]$$

Les relations d'Euler permettent de passer de la différence des exponentielles à un sinus et d'écrire ces coefficients sous une forme plus simple :

$$\boxed{X(jk) = A \cdot \frac{\Delta t}{T} \cdot \frac{\sin(k\pi f_o \Delta t)}{k\pi f_o \Delta t}}$$

Fichier Source :
(sir.m)

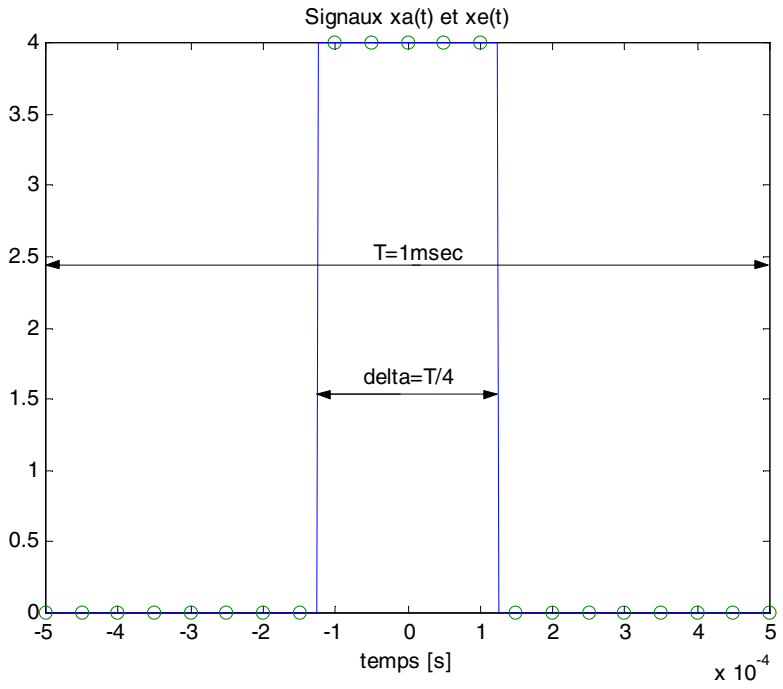
```
% Fichier : sir.m
% Date    : 9.05.00
% But     : Echantillonnage d'une SIR

clear all;close all;

% paramètres
T     = 1e-3;  % période
delta  = T/4;   % période/4
fo     = 1/T;   % fréquence SIR
fe     = 20*fo; % fréquence échantillonnage
Te     = 1/fe;  % période échantillonnage
A      = 4;     % amplitude en Volt

% signal temporel xt = x(t)
Nmax = 500;
t=-T/2:T/Nmax:T/2;
lt0=round((Nmax/T)*((T-delta)/2));
lt1=(Nmax/T)*delta;
x0=zeros(1,lt0);
x1=A*ones(1,lt1);
xt=[x0,x1,x0];
```

```
% echantillonnage xtn = x(nTe)
tn=-T/2:Te:+T/2;
Nmax=ceil(T/Te); % arrondi >= (T/Te)
lt0=round((Nmax/T)*((T-delta)/2));
lt1=(Nmax/T)*delta;
x0=zeros(1,lt0);
x1=A*ones(1,lt1);
xtn=[x0,x1,x0];

% traçage des signaux temporels
figure(gcf);
plot(t,xt,tn,xtn,'o');
title('Signaux xa(t) et xe(t)');
xlabel('temps [s]');

% spectre original
f=-fe:fo:fe;
xf=(A*(delta/T))*sinc(f*delta);

% spectre en +fe et -fe
xfp1=sinc((f-fe)*delta);
xfm1=sinc((f+fe)*delta);

% traçage des spectres
figure(gcf+1);
subplot(3,1,1);
plot(f,xf);stem(f,xf);
title('Spectre original');

subplot(3,1,2);
stem(f,xfp1);
title('Spectre dû à +fe');

subplot(3,1,3);
stem(f,xfm1);
title('Spectre dû à -fe');
xlabel('fréquence [Hz]');

% effet du repliement spectral
figure(gcf+1);
subplot(1,1,1);
plot(f,xf,'x');
hold on;
stem(f,xf+xfp1+xfm1);
title('Effet du repliement spectral: x = Xa(jf), o = Xe(jf)');
xlabel('fréquence [Hz]');
hold off;
```

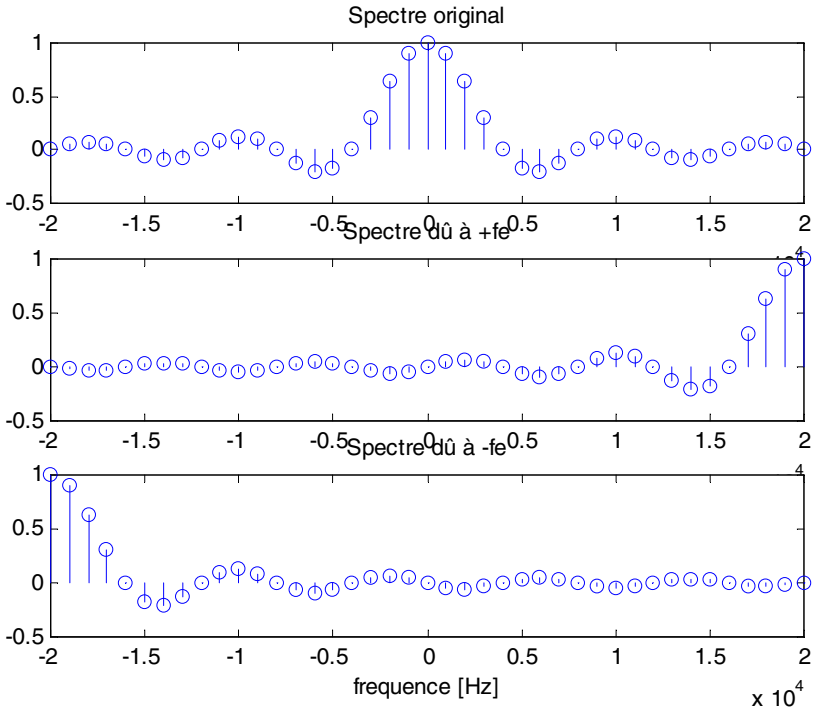
Prenons un exemple :

SIR d'amplitude 4V de période T = 1msec et de largeur $\Delta t = T/4$ que l'on échantillonne avec $T_e = T/20$.

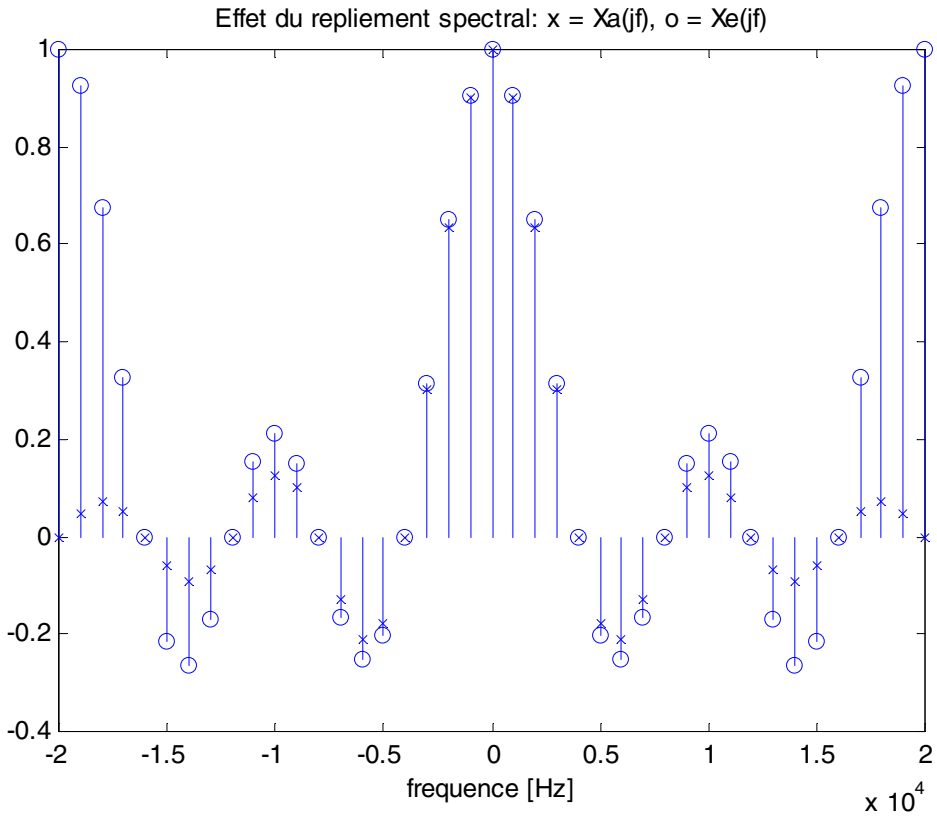
Signaux temporels, x(t) et x(nTe) :



Spectres :



Effet du repliement spectral :



7.0 Références bibliographiques

Introduction au logiciel MATLAB
Cours & exercices
Auteur : Michel Etique, EIVD

Laboratoire d'analyse numérique
Auteur : Louis Chassot, EIVD

Apprendre et Maîtriser MATLAB
Versions 4&5 et SIMULINK
Auteurs : M.Mokhtari & A.Mesbah
Edition Springer
ISBN 3-540-62773-1

Applications de MATLAB 5 et SIMULINK 2
Auteurs : M.Mokhtari & M.Marie
Edition Springer
ISBN 2-287-59651-8

Electronics and circuit analysis using MATLAB
John O. Attia
Editeur : CRC BIRKHAUESER
ISBN 0-8493-1176-4

8.0 Services Internet

Site "The MathWorks" : http://www.mathworks.com

Site "SCIENTIFIC SOFTWARE" : http://www.ssg.fr

Fournisseur Suisse :
Scientific Computers SC AG
Schürmattstrasse 6+8
3073 Gümligen
Homepage : www.scientific.ch