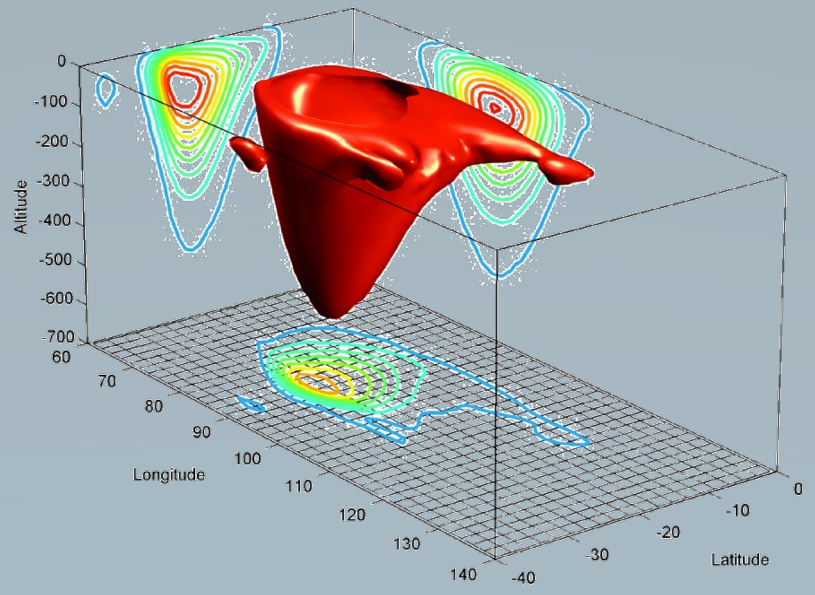
ENSET Mohammedia
Introduction à Matlab
Pr ECH-CHHIBAT



Introduction : Matlab, qu'est-ce que c'est?

- MATLAB = MATrix LABoratory
- Logiciel commercial de **calcul matriciel** à syntaxe simple
- **Interpréteur** : les instructions sont interprétées et exécutées ligne par ligne
- Fonctionne dans plusieurs **environnements** : *Windows*, *Unix*, *Mac*…



Matlab est **coûteux**

Il en existe un clone disponible en *freeware* mais qui n'a pas encore toutes les *toolboxes* équivalentes : **SCILAB**

Introduction : Matlab, quel usage?

- Langage simple et très efficace optimisé pour le **traitement des matrices**

  par exemple, pas besoin de programmer des boucles pour modifier un à un les éléments d'une matrice, la matrice est traitée comme une simple variable

- Environnement complet pour le **calcul** et la **visualisation** :

  - Contient une **interface graphique** puissante et une grande variété d'algorithmes scientifiques

  - Dispose de centaines de **fonctions** mathématiques, scientifiques, techniques

  - Comprend un ensemble d'outils spécifiques à un domaine : les ***toolboxes*** (« boîtes à outils ») : collections de fonctions qui étendent l'environnement de Matlab



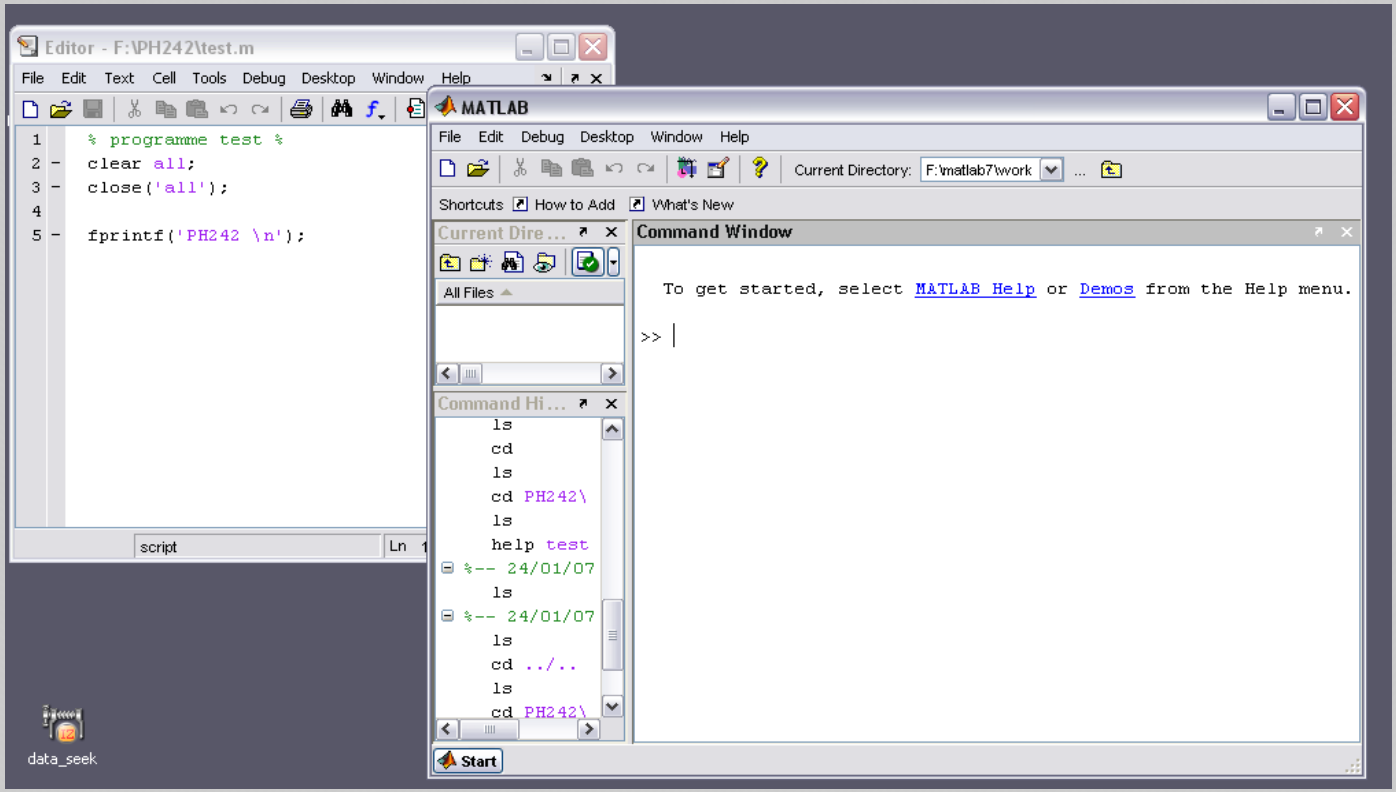
Matlab ne remplace pas les langages type **C** ou **Fortran**, importants pour les calculs de haute performance qui nécessitent une grande mémoire et un temps de calcul très long

Introduction : Matlab, comment faire?

Trois étapes très simples :

1. Ouvrir Matlab : cliquer sur **l'icône** (*windows*) ou taper la commande « **matlab** » (*unix*)
2. Écrire un script ou une « suite complexe d'instructions » : **éditeur de texte** de Matlab
3. Exécuter ce script : après le prompt « >> » dans l'**espace de travail**

Chaque instruction peut être directement tapée dans l'espace de travail



Éléments de base et aides

- Les instructions sont séparées les unes des autres par « ; ».
- Dans l'espace de travail, on revient à la commande précédente par la flèche du haut; on peut le faire plusieurs fois de suite.
- Quelques commandes équivalentes à Unix :

  - Pour connaître le répertoire courant : >> cd
  - Pour connaître ce qu'il y a dans le répertoire courant : >> ls
  - Pour changer de répertoire : >> cd nom_repertoire

- Les principales aides :

  >> help nom_fonction : donne de l'aide sur une fonction

  ex

  ```
  >> help sin
      SIN(X) is the sinus of the elements of X
  >>
  ```

  >> helpdesk : documentation en hypertexte (en ligne)

  >> lookfor mot_cle : recherche d'un mot clé (lent!)

  >> whos : liste des variables ouvertes dans l'espace de travail

Variables et opérations élémentaires

Variables élémentaires

```
>> varA=3  → met la valeur 3 dans la variable varA
   varA = 3
>> varB=2;
>> varC='bonjour';
>> whos
    Name          Size        Bytes    Class
    varA          1x1             8    double array  → type par défaut
    varB          1x1             8    double array
    varC          1x5            10    char array
>> clear  → efface toutes les variables de l'espace de travail
>> whos
>>
```

Opérations élémentaires : + - * / ^

ex

```
>> som=(varA+varB)^2
    som = 25
>>
```

Commentaires : %

ex

```
>> som=(varA+varB)^2;     % je fais une somme
>>
```

Variables spéciales et nombres complexes

Variables spéciales

```
>> pi
   ans = 3.1416
```

```
>> realmin
  ans = 2.2251e-308
```

```
>> eps
  ans = 2.2204e-016
```

```
>> realmax
  ans = 1.7977e+308
```

```
>> i
  ans = 0 + 1.0000i
```

```
>> nan
  ans = NaN
```

```
>> inf
  ans = Inf
```

Nombres complexes

```
>> c1=1-2i
  c1 = 1.0000 - 2.0000i
>> c2=3*(2-sqrt(-1)*3)
 c2 = 6.0000 - 9.0000i
```

Quelques fonctions associées : real(c1), imag(c1), abs(c1), angle(c1), conj(c1)

Fonctions mathématiques

De nombreuses **fonctions mathématiques** existent déjà dans matlab sans que l'on ait besoin de les programmer

ex sin, cos, tan, sinh, cosh, tanh…

asin, acos, atan, asinh, acosh…

exp, log, log10, sqrt…

fix, floor, ceil, round, mod, rem, sign…

factor, isprime…

besselj, besselh, gamma, legendre…

fprintf, fclose, fopen, fread…

min, max, mean…

```
>> cos(pi)
  ans = -1
```

Entrées - Sorties

Deux commandes utiles pour gérer l'espace de travail :

>> save → écrit toutes les variables de l'espace de travail dans le fichier matlab.mat

>> load → charge dans l'espace de travail toutes les variables du fichier matlab.mat

Fonctionnent également pour une seule variable :

```
>> varA=2;
>> save fichier.mat varA
>> clear
>> load fichier.mat
>> whos
   Name        Size                Bytes  Class
   varA        1x1                     8  double array
```

Création de vecteurs

Par défaut, le vecteur est une ligne à plusieurs colonnes

-Vecteur ligne par énumération des composants

```
>> vecA=[1  3.4  5  -6]
   vecA = 1.0000    3.4000    5.0000    -6.0000
```

-Vecteur ligne par description

```
>> vecB=[0 : 10 : 60]
  vecB = 0    10    20    30    40    50    60
```

→ [valeur initiale : incrément : valeur finale]

-Vecteur colonne

```
>> col_vecB=vecB'
  col_vecB =
           0
          10
          20
          30
          40
          50
          60
```

Adressages et indexages



Dans Matlab, l'indexage commence à 1 !

On reprend les mêmes exemples : vecA = [1 3.4 5 -6] et vecB = [0 : 10 : 60]

```
>> vecB(3)
  ans = 20
```
→ 3ème élément du vecteur vecB

```
>> vecB(2 : 4)
  ans = 10    20    30
```
→ un bloc de composantes du vecteur

```
>> vecB([6  4  7  1])
  ans = 50    40    60    0
```
→ une sélection de composantes du vecteur

```
>> vecC=[vecA  vecB]
  vecC = 1.0000    3.4000    5.0000    -6.0000    0    10    20    30    40    50    60
```
→ accolage de deux vecteurs

En combinant tout ça…

```
>> vecD=[vecA(2:3) vecB(1:2:6)]
  vecD = 3.4000    5.0000    0   20   40
```

Création de matrices

Une matrice est un ensemble de lignes ayant toutes le même nombre de colonnes

-Matrice définie par énumération des éléments :

```
>> M1=[1  2  3 ; 4  5  6 ; 7  8  9]
  M1 =
        1   2   3
        4   5   6
        7   8   9
```

→ les lignes sont séparées par des « ; »

-On peut étendre aux matrices les autres manières de définir les vecteurs :

```
>> M2=[1:1:3 ; 11:1:13]
  M2 =
        1    2    3
       11   12   13
```

→ !! respecter le même nombre de colonnes par ligne !!

-Transposition d'une matrice :

```
>> M3=M2'
  M3 =
        1   11
        2   12
        3   13
```

Opérations scalaires - matrices

Une opération scalaire-matrice agit sur chaque élément de la matrice

```
>> M2' * 10
  ans =
        10   110
        20   120
        30   130
```

De même : 4 * M2 M2-10 M2/4

Une exception :

```
>> M2^2
  ??? Error using ==> ^
  Matrix must be square
```

Car dans ce cas, matlab veut calculer M2 * M2 ; La solution est l 'usage du point qui force l'opération sur chaque élément :

```
>> M2.^2
  ans =
        1     4     9
        121   144   169
```

Opérations entre matrices

-Les **additions** ou **soustraction** entre matrices ne sont possibles que si les dimensions des matrices sont **les mêmes**

-Le **produit matriciel** n'est possible que lorsque les dimensions sont cohérentes :

```
>> M1 * M2'
  ans =
        14    74
        32   182
        50   290
```

-**Multiplication** ou **division** élément par élément :

`>> M2. * M3` → M2 et M3 ont les mêmes dimensions

`>> M2. / M3` → chaque élément de M2 est divisé par l'élément équivalent de M3

`>> M2. \ M3` → chaque élément de M3 est divisé par l'élément équivalent de M2

`>> M3. / M2` → chaque élément de M3 est divisé par l'élément équivalent de M2

Caractéristiques des matrices

Caractéristiques principales :

```
>> size (M2)
  ans =
        2    3
```
→ 2 lignes et 3 colonnes

```
>> length (M2)
  ans =
        3
```
→ équivaut à **max (size (M2))** : dimension maximale

```
>> min (M2)
  ans =
        1    2    3
```
→ usage équivalent pour **max (M2)** et pour **mean (M2)**

```
>> M2 (2 , :)
  ans =
        11  12  13
```
→ toutes les colonnes de la deuxième ligne

```
>> mean (M2 (2 , :))
  ans =
        12
```

Matrices particulières

Elles servent *notamment* à **initialiser** la dimension des matrices :

```
>> M4 = ones (3)
  M4 =
        1   1   1
        1   1   1
        1   1   1

>> M5 = zeros (2,5)
  M5 =
        0   0   0   0   0
        0   0   0   0   0
        0   0   0   0   0

>> a = 2;
>> b = 3;
>> M6 = NaN * ones (a,b)
  M6 =
      NaN   NaN   NaN
      NaN   NaN   NaN
```

Récapitulatif des notations

[ ]	Énumération d'éléments
:	Descripteur d'éléments de vecteurs / matrices
( )	Ensemble d'arguments
,	Séparateur d'arguments
;	Séparateur des lignes dans les matrices Suppression de l'affichage dans l'espace de travail du résultat de l'évaluation d'une instruction
'	Transposition d'une matrice / vecteur
.	Force l'opérateur à s'appliquer sur chaque élément d'une matrice / vecteur
%	Délimiteur de commentaires
…	Continuation de l'instruction sur la ligne suivante

5.1. Opérateurs logiques et de relation

<	plus petit	>	plus grand
<=	plus petit ou égal	>=	plus grand ou égal
==	égal	~=	différent
&	et	|	ou
~	not		

any (vecA) retourne 1 si l'un des éléments de vecA est non nul

all (vecA) retourne 1 si tous les éléments de vecA sont nuls

isequal (x,y) retourne 1 si x et y sont égaux

ischar (x) retourne 1 si x est un caractère

…

L'instruction conditionnée IF

Syntaxe :

```
if expression logique
    séquence d'instructions
end
```

```
if expression logique
    séquence d'instructions 1
else
    séquence d'instructions 2
end
```

expression logique est une expression dont le résultat peut être vrai ou faux

Il n'y a pas de mot clé « then

```
>> if (expression1)
            {commandes si expression1 « vrai »
elseif (expression2)
            {commandes si expression2 « vrai »}
 else
{commandes si pas d' expression « vrai »}
end
 >>
```

```
>> x = 123; y = exp(log(x));
>> tst = ( x==y );
>> if tst, disp('x est egal a y '),
      else disp('x est different de y '),
   End
```

5.2. Les structures répétitives

a) Les boucles « **for** » :
:

```
>> for i = 1:4
       for j = 1:3
          M (i,j) = i^2 + j;
       end
   end
>> M
```

b) Les boucles « **while** »

```
>> while (expression)
       {commandes}
   end
>>
```

Exemple:

```
>> n = 4;
>> k = 1; nfac = 1;
>> while k <= n
nfac = nfac*k;
k = k+1;
end
>> nfac
nfac =
24
>>
```

Pour stopper une boucle en cours d'exécution : « **Ctrl C** »

5.3. M-files ou scripts

Un script (ou M-file) est un fichier (question.m par ex) contenant des instruction matlab.

ex :

```
% ce programme affiche un message
clear all;
reponse = input ('fait-il beau?   (y/n)   ', 's');
if (reponse=='y')
        fprintf ('vous êtes de bonne humeur \n');
else
        fprintf ('vous êtes de mauvaise humeur \n');
end
```

Les scripts sont exécutés séquentiellement dans l'espace de travail : ils peuvent donc accéder aux variables qui s'y trouvent déjà, les modifier, en créer d'autres…

On exécute un script en utilisant le nom du script comme commande :

```
>> question
  fait-il beau?   (y/n)   y
  vous êtes de bonne humeur
>> help question
  ce programme affiche un message
>>
```

5.4. Fonctions

On peut écrire des fonctions Matlab que l'on peut ensuite appeler dans un script ou directement dans l'espace de travail.

ex :

```
function imax = index_max(vecA)
% fonction qui calcule l'indice de la valeur maximale d'un vecteur
% utilisation : i = index_max (vecteur)
 ia = max (vecA);
imax = ia;
```

Utilisation de cette fonction, dans un script ou dans l'espace de travail :

```
>> help index_max
  fonction qui calcule l'indice de la valeur maximale d'un vecteur
  utilisation : i = index_max (vecteur)
>> vec = [1 5 6 3 5 2];
>> i = index_max (vec)
  i =
            3
>>
```

5.4. Fonctions

Plus généralement:

function [vars_1, ..., vars_m] = fonc(vare_1, ..., vare_n)
sequence d'instructions

ou vars_1, ..., vars_m sont les variables de sortie de la fonction;

vare_1, ..., vare_n sont les variables d'entr´ee de la fonction;

```
function [r,q] = modulo(a,n)
% Calcule la valeur de a modulo n en prenant pour systeme de residus
% 1, ... , n au lieu de 0, ... , n-1.
%
% appel : [r,q] = modulo(a,n)
%
% Arguments de sortie :
% r : le residu
% q : le quotient
q = floor(a./n);
r = a - n*q;
% si le reste de la division entiere vaut 0, le residu vaut par
convention n
if r == 0, r = n; end
```

Graphe en deux dimensions

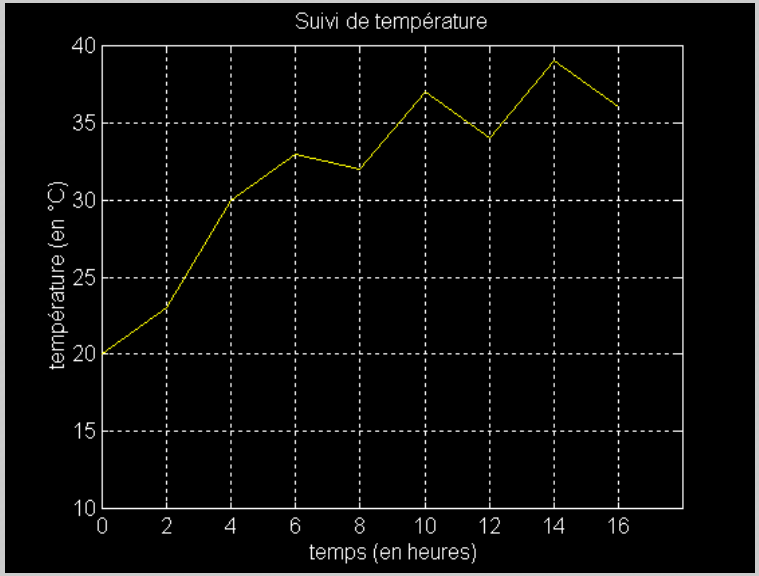
Fonctions :

plot	trace point par point un graphe 2D
semilogx	identique à *plot* mais avec échelle logarithmique pour l'axe des abscisses
semilogy	identique à *plot* mais avec échelle logarithmique pour l'axe des ordonnées
loglog	identique à *plot* mais avec échelles logarithmiques pour les deux axes
grid	ajoute une grille
xlabel	ajoute une légende pour l'axe des abscisses
ylabel	ajoute une légende pour l'axe des ordonnées
title	ajoute un titre
axis	modifie les échelles des axes
zoom	effectue un zoom
gtext	place une légende avec la souris
hold	ajoute un graphe dans la fenêtre courante
figure	crée une nouvelle fenêtre

Graphe en deux dimensions

Exemple 1

Temps (heures)	0	2	4	6	8	10	12	14	16
Température (°C)	20	23	30	33	32	37	34	39	36




```
>> temps = [0 2 4 6 8 10 12 14 16]
temps =
0 2 4 6 8 10 12 14 16
>> temperature = [20 23 30 33 32 37 34 39 36]
temperature =
20 23 30 33 32 37 34 39 36
>> plot (temps , temperature)
>> grid on
>> xlabel ( 'temps (en heures)' )
>> ylabel ( 'température (en °C)' )
>> title ( 'Suivi de température' );
>> axis ( [ 0 18 10 40 ] );
>> plot ( temps , temperature ,'+' )
>> plot ( temps , temperature , 'co' )
>> plot ( temps , temperature , ':' )
>> plot ( temps , temperature , '--' )
```

Graphe en deux dimensions

Autres options

y	Jaune
m	Magenta
c	Cyan
r	Rouge
g	Vert
b	Bleu
w	Blanc
k	Noir

.
o
x
+
-
*
:
-.
--

```
>> plot ( temps , temperature , '.' ) %graphique avec des points
>> plot ( temps , temperature , 'y' ) % graphique en jaune
```

Graphe en deux dimensions

Exemple 2 : graphe d'une fonction à une variable

$$y = \frac{t(1+t^2)}{10}$$

Création du vecteur t :

>> t = 0 : 0.01 : 1

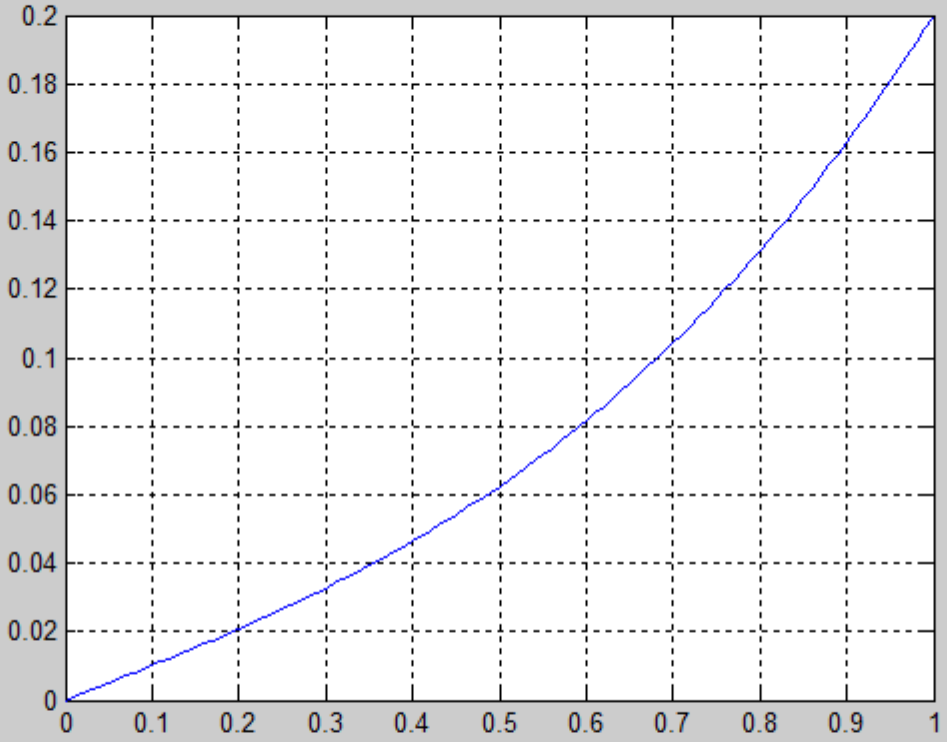
Création du vecteur y :

Attention : ne pas oublier le point devant les opérateurs multiplication, puissance et division : **.*, ./ et .^**

>> y = t.*(1 + t.^2)./10

>> plot (t , y)

>> grid on



Représentation graphique d'une fonction à une variable y = f(x)

Fonctions

fplot	trace point par point le graphe d'une fonction
grid	ajoute une grille
xlabel	ajoute une légende pour l'axe des abscisses
ylabel	ajoute une légende pour l'axe des ordonnées
title	ajoute un titre
axis	modifie les échelles des axes
zoom	effectue un zoom
gtext	place une légende avec la souris
hold	ajoute un graphe dans la fenêtre courante
figure	crée une nouvelle fenêtre

Représentation graphique d'une fonction à une variable y = f(x)

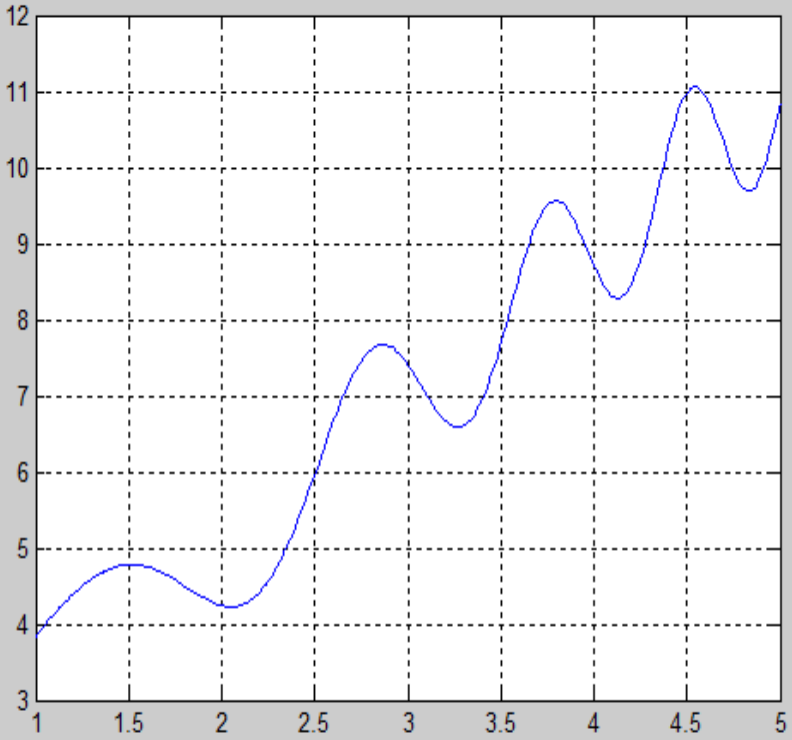
Première méthode :

On veut tracer le graphe de la fonction :
$y=f(x)=1+2x+\sin(x^2)$

>> fplot('1+ 2*x + sin(x*x)', [1 5])

Remarque : la variable doit nécessairement s'appeler x

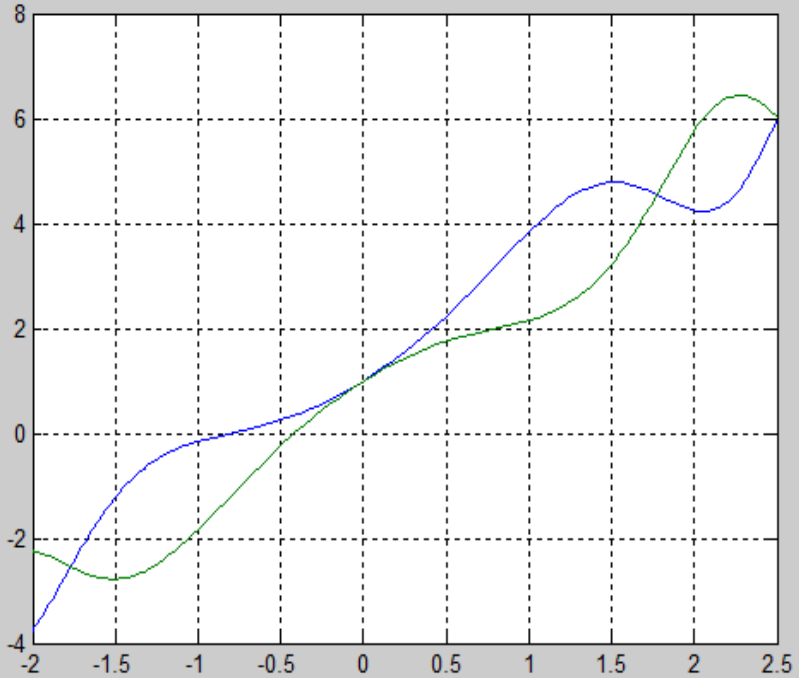
>> fplot('1+ 2*x + sin(x*x)', [2 3 2 10])
>> grid on
>> grid off
>> xlabel('axe des abscisses')
>> ylabel('axe des ordonnées')
>> title('y=f(x)')
>> zoom on
click droit : zoom arrière
click gauche : zoom avant
click gauche et glissé : zoom d'une zone
>> zoom off



Représentation graphique d'une fonction à une variable y = f(x)

Pour tracer plusieurs graphes dans la même fenêtre :

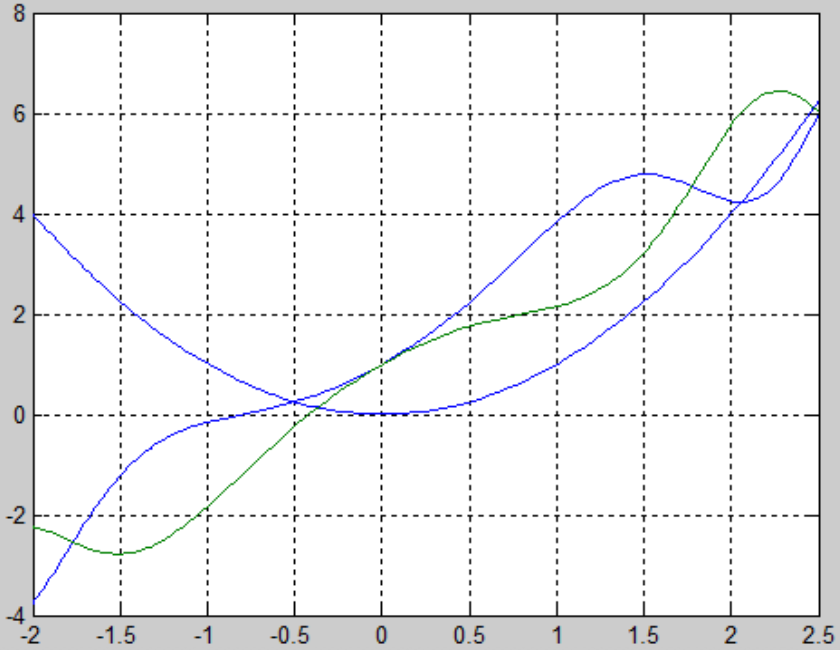
>> fplot('[1+ 2*x + sin(x*x) , 1+ 2*x - sin(x*x)]', [-2 2.5])



On peut ajouter des graphes à un existant:

>>hold on

>> fplot('x^2', [-2 2.5])

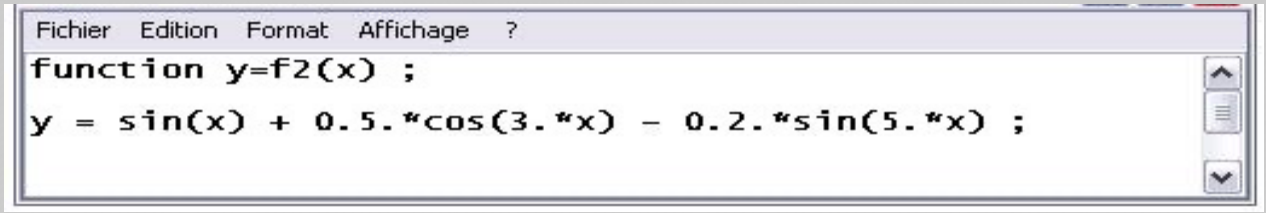


Représentation graphique d'une fonction à une variable y = f(x)

Deuxième méthode :

$$y = f(x) = \sin(x) + \frac{\cos(3x)}{2} - \frac{\sin(5x)}{5}$$

Nous allons créer le fichier .m de la fonction :



>> fplot('f2', [0 10])

Plusieurs graphique sur la même fenêtre

La fonction matlab utilisée est:

subplot(*Ligne,Colonne,Position*)

- ***Ligne*** : nombre de lignes que contiendra la matrice qui découpe la fenêtre.
- ***Colonne*** : nombre de colonnes que contiendra la matrice qui découpe la fenêtre.
- ***Position*** : entier qui indique dans quelle case de la matrice le graphique sera inséré. Les cases se comptent de gauche à droite et de haut en bas.

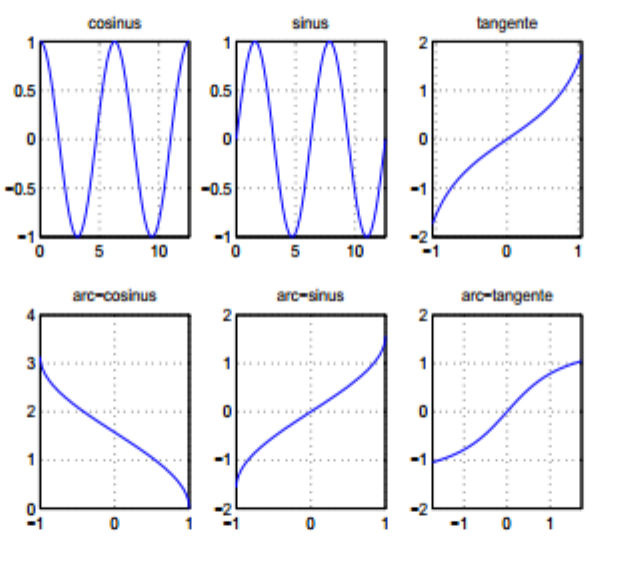
cette fonction permet d'insérer plusieurs graphiques à l'intérieur d'une même fenêtre.

la fenêtre est décomposée en une matrice de dimension voulue et le graphique est tracé dans la case demandée.

Plusieurs graphique sur la même fenêtre

Exemple

```
>> subplot(2,3,1), fplot('cos',[0 4*pi]), title('cosinus'), grid
>> subplot(2,3,2), fplot('sin',[0 4*pi]), title('sinus'), grid
>> subplot(2,3,3), fplot('tan',[-pi/3 pi/3]), title('tangente'), grid
>> subplot(2,3,4), fplot('acos',[-1 1]), title('arc-cosinus'), grid
>> subplot(2,3,5), fplot('asin',[-1 1]), title('arc-sinus'), grid
>> subplot(2,3,6), fplot('atan',[-sqrt(3) sqrt(3)]), title('arc-tangente'), grid
```



MATLAB et asservissement
fonctions de transfert

Introduction sous forme polynômiale:

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

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

Son introduction dans MATLAB se fait par les commandes :

Les noms donnés aux numérateurs et dénominateurs sont libres.

```
>> numG=36 *[ 1 0 , 1 ] ;
>> denG=[ 0 . 1 , 2 , 1 ] ;
>> G=tf(numG,denG)
```

MATLAB et asservissement
fonctions de transfert

Introduction sous forme de zéros, pôles et gain:

```
>> z=[1,3]
>> p=[2,6,3]
>> k=5
>> G2=zpk(z,p,k)
```

```
Zero/pole/gain:
  5 (s-1) (s-3)
-----------------
(s-2) (s-3) (s-6)
```

MATLAB et asservissement

fonctions de transfert

Les pôles et les zéros d'une fonction de transfert:
Les pôles et zéros des fonctions de transfert sont obtenus à l'aide de la fonction:

tf2zp (transfer function to zero pole):
[zG, sG,kG]=tf2zp (numG, denG) ;

MATLAB place alors les zéros et les pôles dans les vecteurs colonne zG et sG, respectivement, et le gain dans la variable kG. Les noms zG, sG et kG sont arbitraires. On accède à ces valeurs par exemple en tapant leur nom sous MATLAB :
>>**sG**
Fonction de transfert à partir des pôles, zéros et gain:
Zp2tf (zero pole to transfer function):

>>[numG,denG] = zp2tf(z,p,k)
>>G=tf(numG,denG)

MATLAB et asservissement
fonctions de transfert

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.

```
>> p=[3  6  2  9  6  2]

p =

     3     6     2     9     6     2

>> roots(p)

ans =

  -2.1653
   0.4293 + 1.1025i
   0.4293 - 1.1025i
  -0.3466 + 0.3160i
  -0.3466 - 0.3160i
```

MATLAB et asservissement
fonctions de transfert

Soit la fonction de transfert suivante:

$$H(s) = \frac{4}{3s^3 + 4s^2 + 5s + 6}$$

Calculer le gain, les pôles et les zéros de H(s)

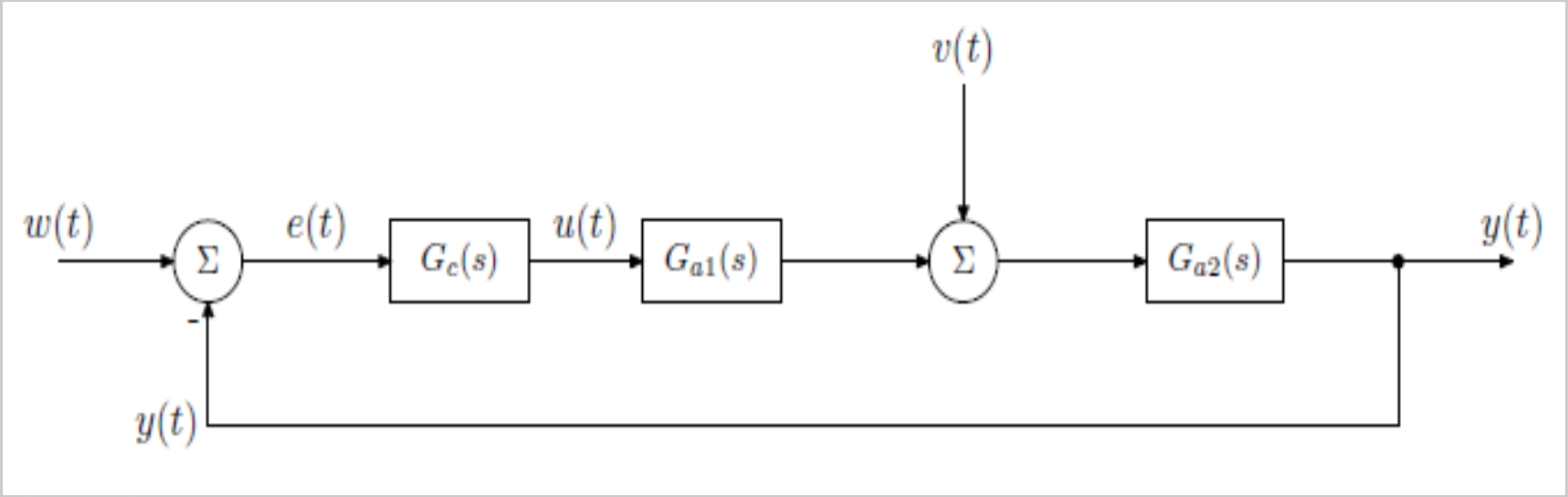
```
>>clc
>>clear
>>num=[4];
>>den=[3 4 5 6];
>>%Pour trouver les pôles et zéros
>>[z,p,k] = tf2zp(num,den);
>>disp('poles:')
>>disp(p)
>>disp('zeros:')
>>disp(z)
>>%Maintenant à partir des pôles et zéros, trouvons la fonction de transfert:
>>[num2,den2] = zp2tf(z,p,k);
>>disp(num2)
>>disp(den2)
```

MATLAB et asservissement

fonctions de transfert

Fonction multiplication et feedback

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

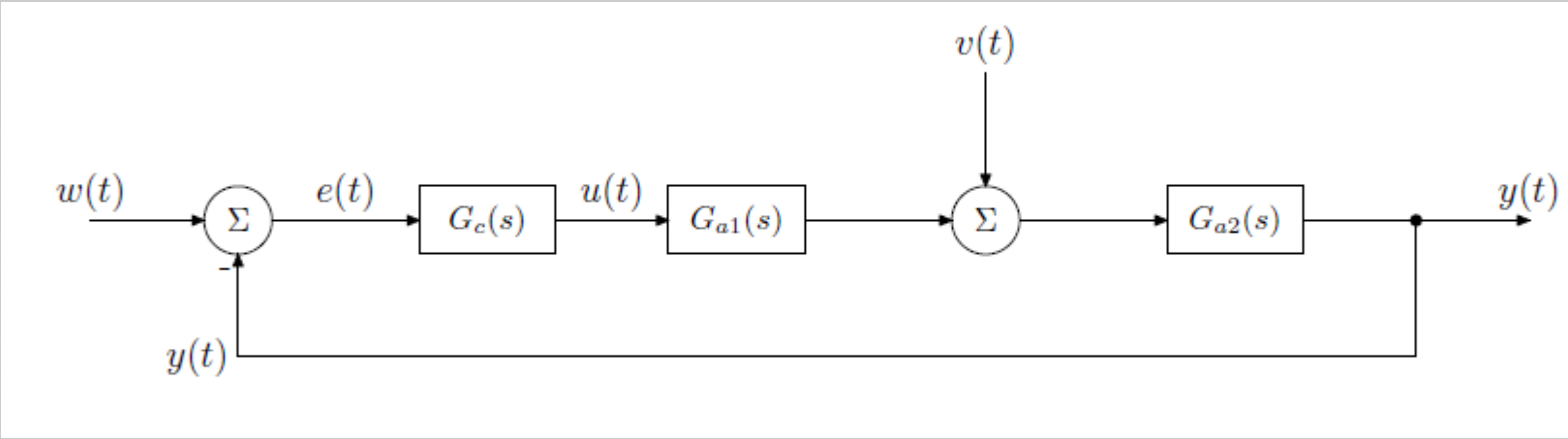


Construction de schémas fonctionnels

1. Calcul de Ga(s) par la mise en série de Ga1(s) et Ga2(s).
On fait usage de la fonction multiplication :
Ga = Ga1*Ga2 ;
2. Calcul de Go(s) par la mise en série de Gc(s) et Ga(s). On procède de même :
Go = Gc*Ga ;
3. Calcul de Gw(s) par fermeture de la boucle (retour unitaire). On fait usage de la fonction **feedback** comme suit :
Gw = feedback(Go, 1)
4. La réponse indicielle en boucle fermée peut alors être calculée et tracée par :
step (Gw)
On peut au passage calculer le gain statique (gain DC) de Gw(s).
dcgain (Gw)
5. La réponse harmonique en boucle ouverte peut par exemple aussi être calculée:
bode (Go)
6. Finalement, on peut afficher la configuration pôle-zéro de Gw(s) :
pzmap(Gw)

Construction de schémas fonctionnels

Application



On prenant v(t)=0, pour les fonctions de transfert

$$G_c(s) = 2 \cdot \frac{s+20}{s}$$

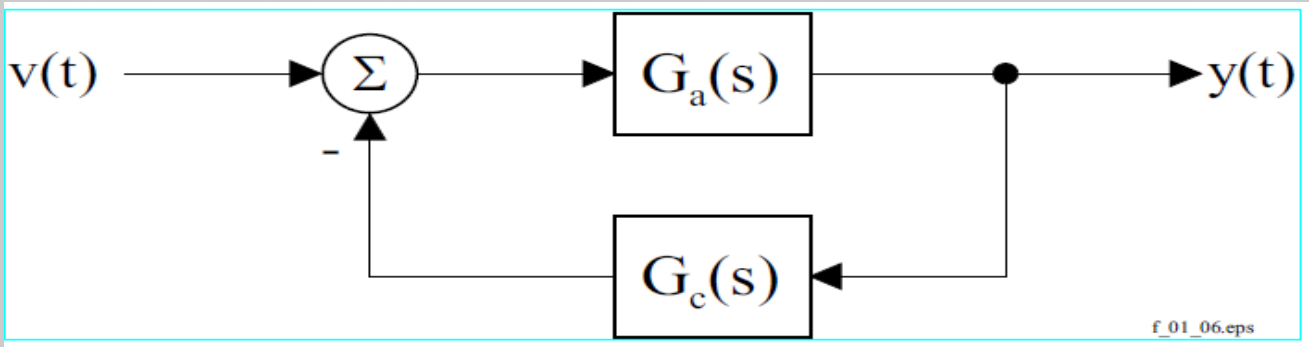
$$G_{a1}(s) = 0.01 \cdot \frac{s+10}{s^2+2 \cdot s+0.11}$$

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

Sous Matlab, reprendre les étapes de 1 à 6 et faire un rapport résumant votre travail. Calculer les pôles, les zeros et le gain de la fonction Gw

Construction de schémas fonctionnels

fonction de transfert en boucle fermée: fonction **feedback**



G(s) =Y (s)/V (s)=Ga(s)/(1 + Ga(s) *Gc(s))

G=feedback (Ga,Gc , signe) ;

Si signe est omit alors la valeur par défaut est -1: **G=feedback (Ga,Gc) ;**
la fonction de transfert Gc(s) est en contre-réaction, si signe est différent de -1 alors il faut le mettre.

Calcul et tracé de réponses de systèmes dynamiques linéaires

>>impulse (G) %réponse impulsionnelle
>>Step (G) %réponse indicielle
>> **lsim(G,u,t);**

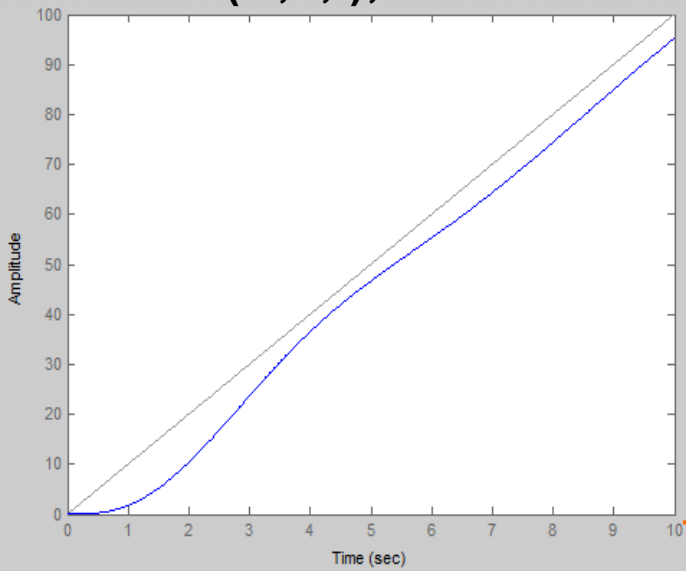
Les coordonnées et les abscisses peuvent être mémoriser pour une utilisation ultérieur:

>>t=[0 :0.01 :0.2] ; %Définir l'intervalle de **t** avant,
>>[y1 , t]=**impulse** (G, t) ; %réponse impulsionnelle
>>[y2 , t]=**step(G**, t) ; %réponse indicielle
>>[y3]=**lsim(G**, u , t) ; %réponse à une entrée u qque

>>Plot(t,y1); %une courbe
>>Plot(t,[y1 y2]) %deux courbes

Exemple: **lsim**

```
>>clc
>>clear
>>t=0:0.01:10;
>>%FT: H(s)=4/(3s2+2s+4)
>>num=[4];
>>den=[3 2 4];
>>G=tf(num,den);
>>%Entrée du système
>>u=3*t;
>>lsim(G,u,t);
```



Réponse fréquentielle

Les fonctions **bode, nyquist** et **nichols** sont disponibles. Leur utilisation est la suivante :
bode (G)
bode (G,w)
nyquist (G)
nyquis t (G,w)
Si les résultats des calculs doivent être sauvegardés en vue d'un traitement ultérieur, on indique également des arguments de sortie ; dans le cas de la fonction **bode**, on aura par exemple :
[A, phi ,w] = **bode** (G) ;
[A, phi] = **bode** (G,w) ;
A et phi sont des vecteurs-colonne contenant respectivement le gain et la phase et w un vecteur-ligne contenant les pulsations correspondantes.

Réponse fréquentielle

Analyse des propriétés des systèmes

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

Calcul affichage des marges de gain et de phase

[Am,phim]=**margin**(G) : Marges de phase et de gain d'un système.

Si les arguments de sortie ne sont pas spécifiés, trace le diagramme de Bode et montre graphiquement la valeur des marges

https://books.google.co.ma/books?id=F55-q-0vGfYC&pg=PA206&lpg=PA206&dq=asservissement&source=bl&ots=xfAmG5-BlY&sig=1DKOp0Ms4k8RjwzpBYdertow4as&hl=fr&sa=X&ei=f2DoVOOCD4mE7Qao5IHwDw&redir_esc=y#v=onepage&q=simulink%20asservissement&f=true