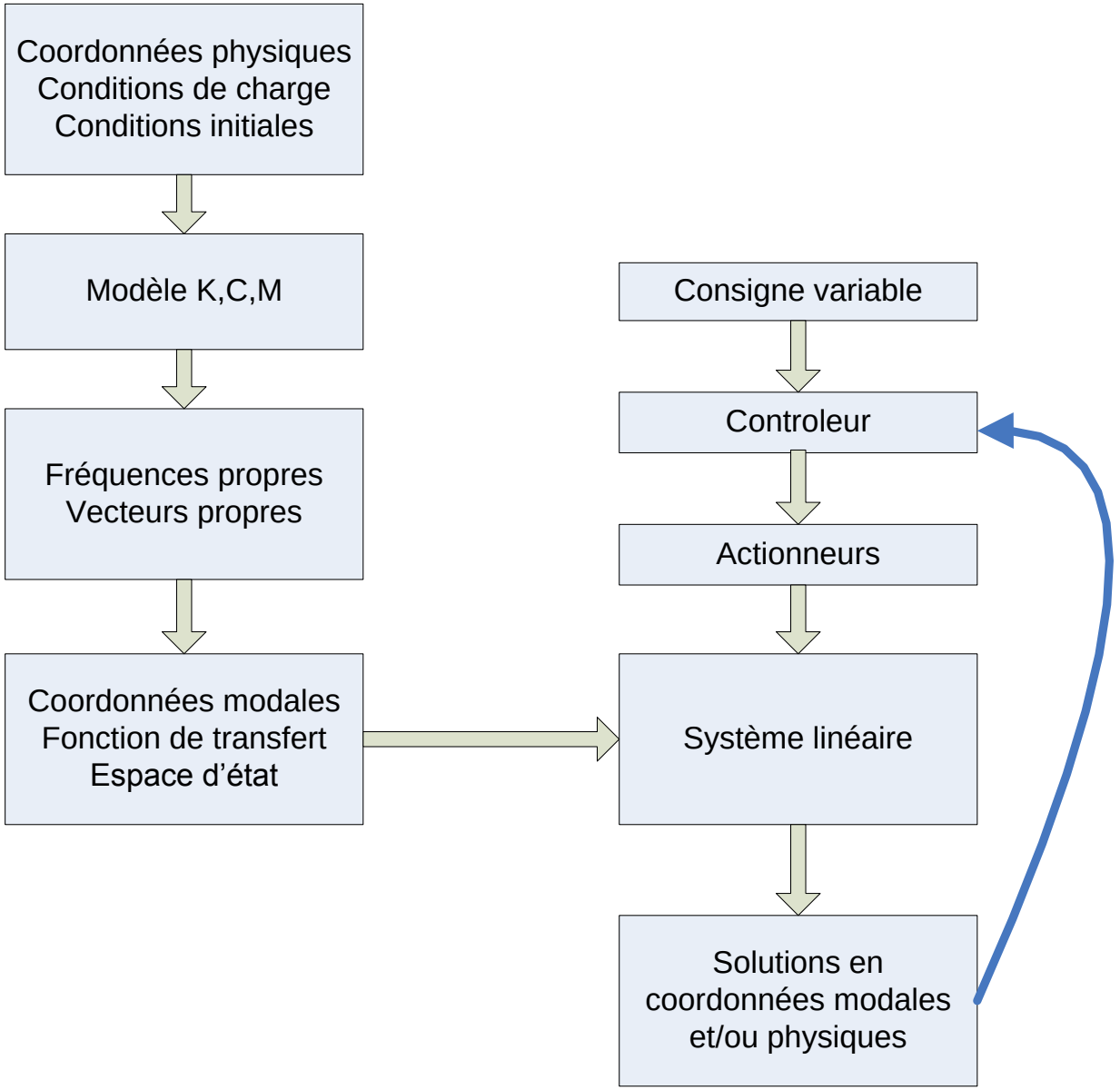
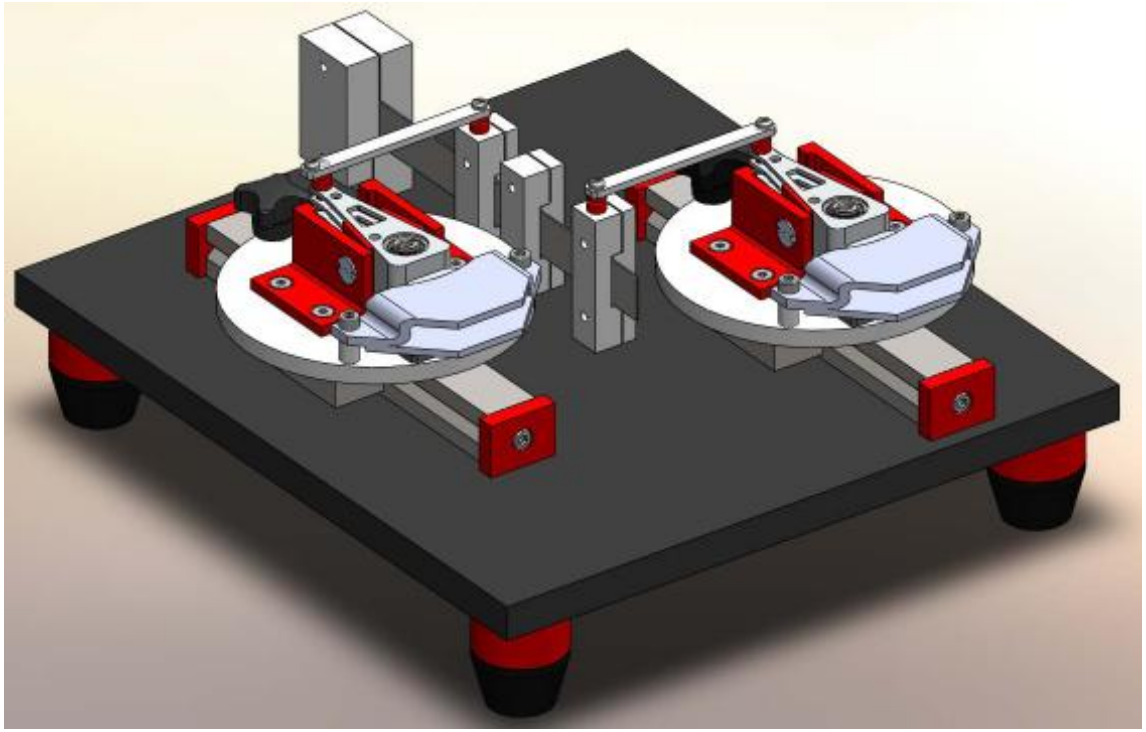
Introduction à l'analyse modale

Analyse modale d'un système mécatronique



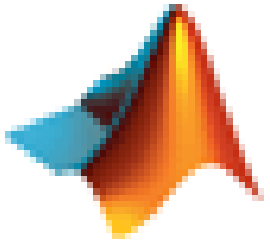


Plan du cours

18.02.2010	Présentations, introduction, initiation à Matlab
25.02.2010	Systèmes SDOF
04.03.2010	Initiation à Simulink
11.03.2010	Modélisation de structures (EF) en Matlab
18.03.2010	Modélisation d'un système actif
25.03.2010	Exercices, TE-1
01.04.2010	Modélisation intégrée d'un système mécatronique
15.04.2010	
22.04.2010	
01.05.2010	Examen

Introduction à Matlab

MATLAB



MATLAB est un logiciel scientifique de calcul numérique créé en 1984 qui possède aujourd'hui une position dominante en :

- Recherche
- Enseignement (universités, écoles d'ingénieurs)
- Industrie (automobile, avionique, espace, chimie, finance, …)

Outre le logiciel de base, MATLAB se décline en une quantité de toolbox «boîte à outils» supplémentaires

SIMULINK

- SIMULINK est une plateforme de modélisation et de simulation de systèmes dynamiques.
- Il offre un environnement de développement graphique et une bibliothèque de blocs qui permettent de simuler divers systèmes de contrôle, communication, traitement de signaux.
- SIMULINK est entièrement intégré à MATLAB, ce qui procure une grande souplesse d'utilisation.
- Il permet de créer des modèles de « haut niveau » avec une décomposition hiérarchique en blocs.

Support pour applications

- Communauté d'utilisateurs très active, regroupée autour du fournisseur Mathworks: forum, partage de codes, concours, …

  http://www.mathworks.com/matlabcentral/

- Un immense répertoire gratuit de toolbox spécialisés et d'applications Matlab et Simulink:

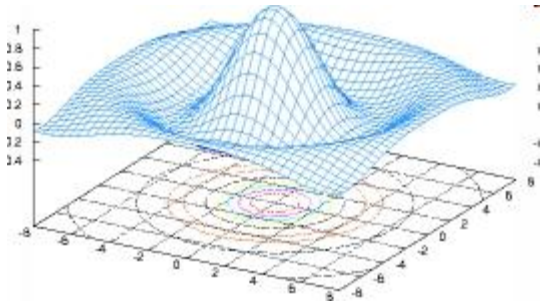
  http://www.mathworks.com/matlabcentral/fileexchange/categories

- Pas de communauté francophone (?).

Clones Matlab open source

- Développement d'une solution Open Source initiée par l'INRIA : **Scilab** et **Scicos** en remplacement de MATLAB et SIMULINK.

- **Octave**: syntaxe presque identique à Matlab



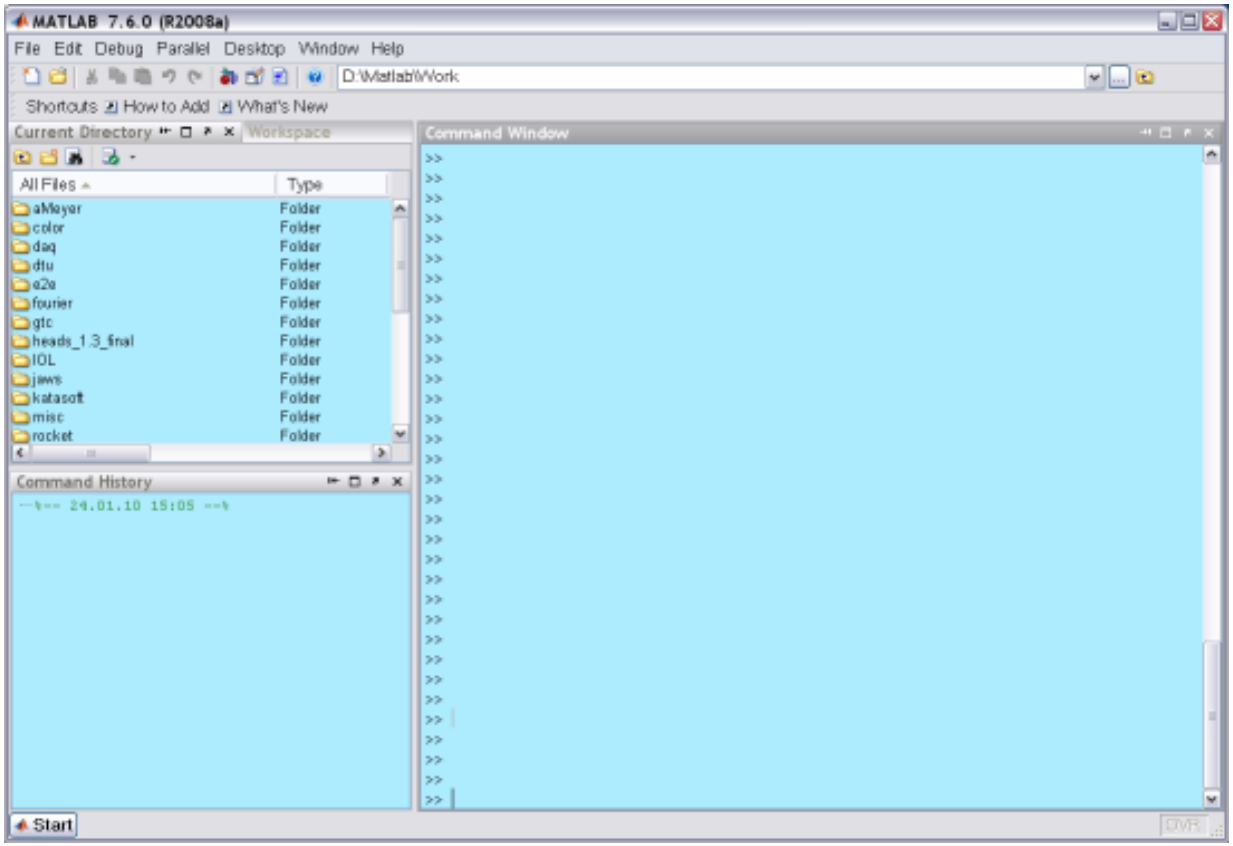
Matlab: introduction

- Logiciel interactif (interpréteur) de calcul **basé sur le calcul matriciel**
  Les réels ou entiers sont des matrices 1x1 !
- Environnement de calcul scientifique
  - Résolution de systèmes linéaires
  - Calcul valeurs propres, vecteurs propres
  - Transformée de Fourier rapide
  - Algorithmes d'optimisation
  - Résolution d'équation différentielles
  - etc. …
- Affichages graphiques
  - Visualiser des courbes, des surfaces, …
- Programmation : édition de scripts

  ➔ Mise en œuvre très rapide

Syntaxe du langage

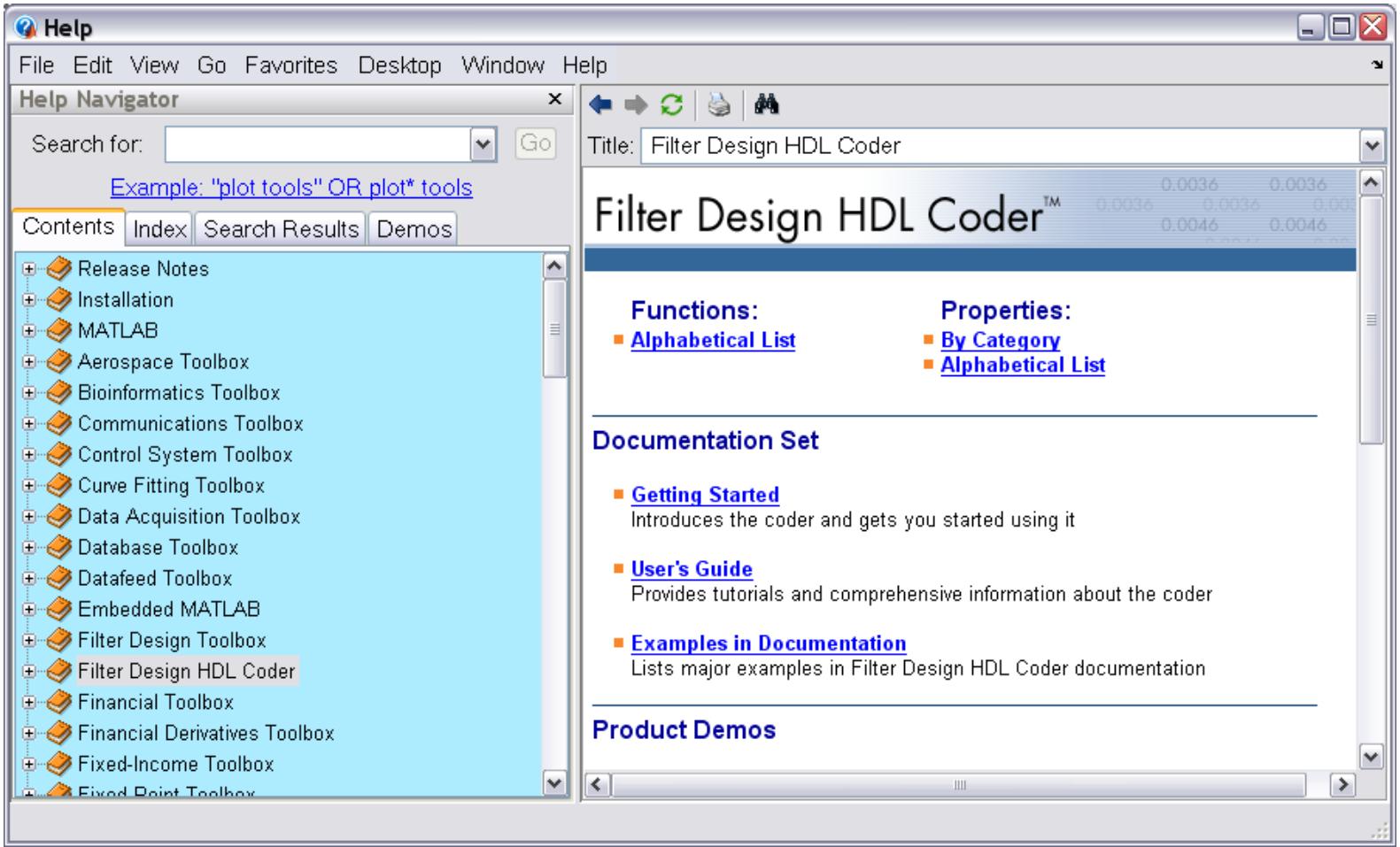
- Suite d'instructions séparées ou non par des points virgules.
  Sans point virgule ➔ affichage du résultat



- Pour les enchaînements de commandes un peu compliqué
  ➔ Écrire un **script** dans un fichier .m (M-Files)

Utilisation de l'aide en ligne

1. Dans la ligne de commande :
   help [function]
2. Menu Help >> Product Help



Premiers pas…

- Définir un scalaire :

```
>> a=21
a =
     21

>> c=2+i
c =
   2. + i
```

- Fonctions courantes:

Trigonométrie : sin, cos, tan, asin, acos, atan
Complexes : conj, real, imag
Fonctions classiques : exp, log, log2, log10, norm

- Quelques constantes

pi
Complexes: i (i*i=-1)

Vecteurs

- **Vecteur ligne**
  Valeurs séparées par des virgules ou des espaces

```
>> v=[1 2 3]
v =
     1     2     3

>> v=[1, 2, 3]
v =
     1     2     3

>> v=[1+3*i, 2+pi, 4*i]
v =
   1. + 3.i   5.1416    4.i
```

- **Vecteur colonne**
  Valeurs séparées par des points virgules

```
>> v=[1;2;3]

v =

     1
     2
     3
>> v=[pi;2+i;8]
v =
   3.1416
   2. + i
   8.
```

- **Transposition :**

```
>> v=[1 2 3]
 v  =
    1.    2.    3.
```

```
>> v'
 ans  =
    1.
    2.
    3.
```

Génération de vecteurs

Syntaxe : [BorneInf : BorneSup]
[BorneInf : pas : BorneSup]

```
>> v=[0:4]
v =
     0     1     2     3     4

>> v2=[0:0.25:1]
v2 =
         0    0.25    0.5    0.75    1.
```

Fonction linspace(a,b,N) :

Crée un vecteur de N composantes uniformément réparties entre a et b.

```
>> linspace(0,1,5)
ans =
         0    0.25    0.5    0.75    1.
```

Matrices

- Matrices
  Lignes séparées par des points virgules

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

M =

     1     2     3
     4     5     6
     7     8     9
```

```
>> P=[1 0 0 0 0;...
-->   1 1 0 0 0;...
-->   1 2 1 0 0;...
-->   1 3 3 1 0;...
-->   1 4 6 4 1]
 P  =

    1.    0.    0.    0.    0.
    1.    1.    0.    0.    0.
    1.    2.    1.    0.    0.
    1.    3.    3.    1.    0.
    1.    4.    6.    4.    1.
```

Vision Matricielle de Matlab :

- Vecteur ligne de dimension N
  = Matrice 1xN
- Vecteur colonne de dimension N
  = Matrice Nx1
- Scalaire = Matrice 1x1

Accès aux éléments des matrices

- **Vecteurs**
  Avec u(i). Attention : premier indice = 1

```
>> u=[0:5]
u =
     0     1     2     3     4     5
>> u(2)
ans =
     1
```

- **Matrices**
  Avec A (ligne,colonne). Attention: premier indice = 1,1

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

>> A(3,1)
ans =
     7
```

Opérations matricielles

+	Addition	C = A + B
−	Soustraction	C = A - B
*	Multiplication	C = A * B
^	Puissance	C = A^2 ou bien C = A * A
‘	Transposée	C = A' ou bien C = transpose(A)
\	division gauche	x = A\b
/	division droite	x = b/A

```
>> A = [1 2 ; 3 4]
A =
     1     2
     3     4

>> A*A
ans =
     7    10
    15    22
```

```
>> u=[1 2 3];
>> v=[1;2;3];
>> u*v
ans =
    14

>> u*u
 !--error 10
inconsistent multiplication
```

Attention aux correspondances de tailles pour *, /

Opérations matricielles éléments par éléments

.*	Multiplication élément à élément
.^	Puissance élément par élément
.\	division gauche élément par élément
./	division droite élément par élément

```
>> A = [1 2 ; 3 4]
A =
     1     2
     3     4

>> C=A.*A
C =
     1     4
     9    16
```

```
>> A.^3
 ans  =

    1.     8.
   27.    64.

>> A^3
 ans  =

   37.    54.
   81.   118
```

Fonctions sur les matrices

- Les fonctions scalaire courantes (sin, exp…) peuvent aussi s'appliquer sur des matrices
  (composante par composante)

```
>> A=[pi pi/2; pi/2 pi]
A =
    3.1416    1.5708
    1.5708    3.1416

>> cos(A)
ans =
   -1.0000    0.0000
    0.0000   -1.0000
```

```
>>  u=linspace(0,1,4)
u =
         0    0.3333    0.6667    1.0000

>> log(u)
 !--error 32
singularity of log or tan function
```

Extraction de sous-matrice

- On peut extraire facilement une sous-matrice ou un sous-vecteur avec l'opérateur ':'

```
>> A=[1 2 3 ; 4 5 6; 7 8 9];
>> A(:,1)
ans =
     1
     4
     7

>> A(2,:)
ans =
     4     5     6
>> A(1:2,:)
ans =

     1     2     3
     4     5     6
```

```
>> u=[1:5]
u =
     1     2     3     4     5

>> u(1:3)
ans =

     1     2     3
```

Quelques fonctions matricielles…

eye :

```
>> I4 = eye(4,4)
 I4  =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.
```

rand :

```
>> rand(3,2)
 ans  =

    0.7263507    0.2320748
    0.1985144    0.2312237
    0.5442573    0.2164633
```

diag :

```
>> d = [1 3 9 27];
>> diag(d)
 ans  =
    1.    0.    0.    0.
    0.    3.    0.    0.
    0.    0.    9.    0.
    0.    0.    0.   27.
```

```
>> A = [1 2 3 ; 4 5 6 ; 7 8 9];
>> diag(A)
 ans  =

    1.
    5.
    9.
```

ones :

```
>> ones(2,4)
 ans  =

    1.    1.    1.    1.
    1.    1.    1.    1.
```

zeros :

```
>> zeros(2,2)
 ans  =

    0.    0.
    0.    0.
```

Principales Opérations sur les Matrices

Fonction	Description
ones(i,j)	crée un tableau de i lignes j colonnes contenant des 1
zeros(i,j)	crée un tableau de i lignes j colonnes contenant des 0
eye(i,j)	crée un tableau de i lignes j colonnes avec des 1 sur la diagonale principale et 0 ailleurs
toeplitz(u)	crée une matrice de Toeplitz symétrique dont la première ligne est le vecteur u
diag(u)	crée une matrice carrée avec le vecteur u sur la diagonale et 0 ailleurs
diag(U)	extrait la diagonale de la matrice U
triu(A)	renvoie la partie supérieure de A
tril(A)	renvoie la partie inférieure de A
linspace(a,b,n)	crée un vecteur de n composantes uniformément réparties de a à b
A\b	résolution du système linéaire `Ax=b`
cond(A)	conditionnement d'une matrice (norme euclidienne)
det(A)	déterminant d'une matrice
rank(A)	rang d'une matrice
inv(A)	inverse d'une matrice
pinv(A)	pseudo inverse d'une matrice
svd(A)	valeurs singulières d'une matrice
norm(A)	norme matricielle ou vectorielle
u'	prend le transposé de u
u*v	multiplication matricielle
u+v	addition matricielle
u-v	soustraction matricielle
u.* v	multiplication des tableaux u et v terme à terme
u./v	division du tableau u par le tableau v terme à terme
find(C(A))	indices des composantes du tableau A vérifiant la condition C(A)

Exercices

(1) Soient les deux vecteurs suivant : $u = \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}$ $v = \begin{pmatrix} -3 \\ 2 \\ 1 \end{pmatrix}$

Calculer $3u, \quad \sqrt[3]{uv}, \quad \|u\|_2, \quad \|u+v\|_2$

(2) Soit la matrice A : $A = \begin{bmatrix} 2 & 1 & 5 \\ 3 & 1 & 7 \\ 4 & 3 & 2 \end{bmatrix}$

Calculer $Au, \; AA^T, \; \|Au\|_2, \; I_3 - A, \; \left(AA^T\right)^{-1}$

(3) Soit les complexes : $a = 3+4i \quad et \quad b = 2+i$

Calculer le module de a et b, la partie réelle et la partie imaginaire de $a^2 + b$

(4) Générer un vecteur *w* de 10 valeurs réparties uniformément entre 0 et pi
Créer le vecteur *z* contenant les 5 premières valeurs de *w*

Fonctions utiles : eye, sqrt, norm, linspace, conj, imag, real, …

Scripts et Fonctions

- Scripts (fichiers '.m') :
  - Enchaînement de commandes Matlab
    regroupées sous un nom de fichier

    ➔ Typiquement comme programme principal

    ou simplement pour conserver une trace de son travail

- Fonctions (fichiers '.m') :
  - Enchaînement de commandes qui renvoie une ou des valeurs

    ➔ Permet de découper un programme long et compliqué en plusieurs fonctions pour un meilleur lisibilité

    ➔ Permet d'exécuter le même code avec des paramètres différents sans copier-coller

    ➔ Calcul avec un certain algorithme, qui pourra être éventuellement remplacé par un autre algo juste en changeant le nom de la fonction appelée.

Fonctions

- Un fonction est en générale écrite dans un fichier .m

Syntaxe :

```
function [o_1, ..., o_M] = toto(x_1, ..., x_N)
...
...
End       % facultatif
```

- toto: nom de la fonction
- o_1, …, o_M: arguments de sortie
  Pour récupérer les valeurs calculées par la fonction
- x_1, …, x_N: arguments d'entrées
  Passage par valeur: valeurs non modifiées dans le programme appelant.

- Un fichier .m peut regrouper plusieurs fonctions

Fonctions: exemple

Fichier polaire.m

```
function [r,theta]=polaire(x,y)
  r = sqrt(x.^2+y.^2);
  theta = atan(y./x);
end   % pas obligatoire
```

Appel de la fonction :

```
>> fact1(4)
Ans =
    24.

>> polaire(2,3)
ans =
    3.6056

>> r=polaire(2,3)
r =
    3.6056

>> [r,t]=polaire(2,3)
r =
    3.6056
t =
    0.9828
```

ATTENTION aux arguments de sortie !

Par défaut : un seul

Syntaxe pour récupérer les deux arguments de sortie

Fonctions anonymes

Ces fonctions (généralement assez courtes) sont définies dans le programme (ou fonction) qui les appelle.

```
func = @(x) x./cos(x) + cos(x).^2;

figure
plot (x,func(x))
grid; title ('Fonction')
```

Exercices

(1) Définir la fonction suivante : $f(x) = \frac{x^5 - 3}{\sqrt{x^2 + 1}}$

Tester f pour quelques valeurs. Par exemple $f(1)$=-1.4142 ou $f(0)$=-3

Votre implémentation de f fonctionne-t-elle avec un vecteur x ?
➔ Générez un vecteur u de 10 valeurs réparties uniformément entre 0 et 1 et testez f(u)

(2) Définir une fonction *resoud_eq_2nd* qui prend en paramètre a, b, c et résout l'équation du second degré $ax^2 + bx + c = 0$
(renvoyer les deux solutions possibles)

Tester avec $x^2 - 2x + 1 = 0$, $x^2 + 1 = 0$

Boucle *for*

- Parcourt un vecteur d'indices et effectue à chaque pas toutes les instructions délimitées par l'instruction end.

```
x=0;
for k=0:2
    x = x + k*2;
end
x
```

Renvoie x=6

- ATTENTION !
  Pour les calculs éviter les boucles *for*, et tirer au maximum partie du calcul matriciel

Boucle précédente équivalente à:

```
sum(2*(0:2))
```

Boucle *while*

- Effectue une suite de commandes tant qu'une condition est satisfaite.

```
epsilon = 0;
while ( epsilon<1 )
    epsilon = epsilon + 0.1
end
```

- Opérateurs logiques dans les tests :

français	test Matlab
et	&
ou	|
non	~
égal	==
différent	~=
plus petit que	<
plus grand que	>
plus petit ou égal à	<=
plus grand ou égal à	>=

Test conditionnel *if*

- Syntaxe :

```
if expression
          …
elseif expression
          …
(elseif expression
          …)
else
          …
end
```

Exemple :

my_abs.m

```
function [y]=my_abs(x)
  if ( x<0 )
       y = -x;
  else
       y = x;
  end
```

Appel :

```
>> my_abs(-7)
ans =
     7
```

Test *switch*

Syntaxe :

```
switch expression
  case case_expr
    commande,... ,commande
  case case_expr1, case_expr2
    commande,... ,commande
...
  otherwise
    commande,... ,commande
end
```

Exemple :

```
n=round(4*rand(1,1))
switch n
  case 0
        disp('cas numero 0')
  case 1
        disp('cas numero 1')
  case 2
        disp('cas numero 2')
  otherwise
        disp('autre cas')
end
```

Exercices

1) Ecrire une fonction test (x) qui renvoie 1 si x > 0.5 et 0 sinon.
   Que se passe-t-il avec un x un vecteur ?

2) Écrire une fonction test2 (x) qui renvoie sin(x) si x > 0.5 et x sinon.
   Que se passe-t-il avec un x un vecteur ?

Sorties Graphiques

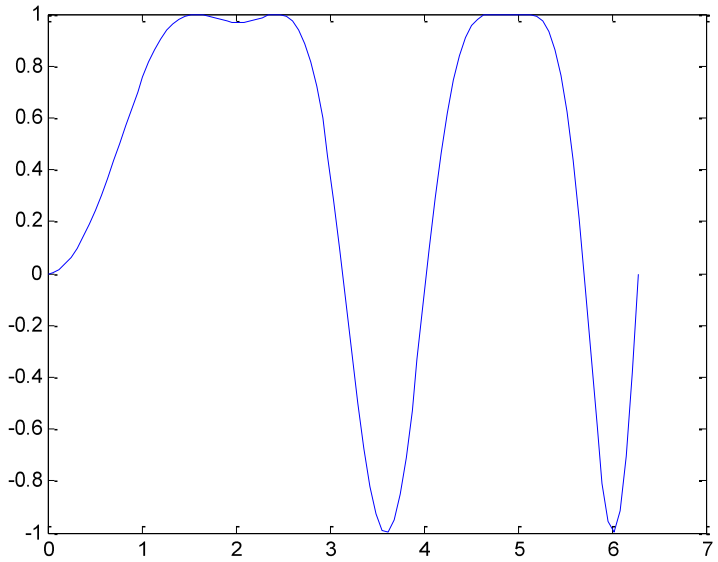
- Toutes les sorties graphiques se basent sur des données discrètes
  (Rangées dans des matrices ou des vecteurs colonne)

- Exemple : pour tracer y=f(x)
  x et y doivent être des vecteurs colonnes

- Visualisation dans une fenêtre graphique (avec zoom, …)

Exemple :

```
>> x=linspace(0,2*pi,100);
>> y=sin(x.*sin(x));
>> plot(x,y);
```



Courbes Planes: plot

Syntaxe:

plot(x,y) ou **plot**(x,y,style)

Avec x et y deux vecteurs de même dimension et s une chaîne de caractère d'option

➔ Affiche la courbe passant par chaque coordonnées (x(i),y(i))

Quelques Options Disponibles:

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dotted line

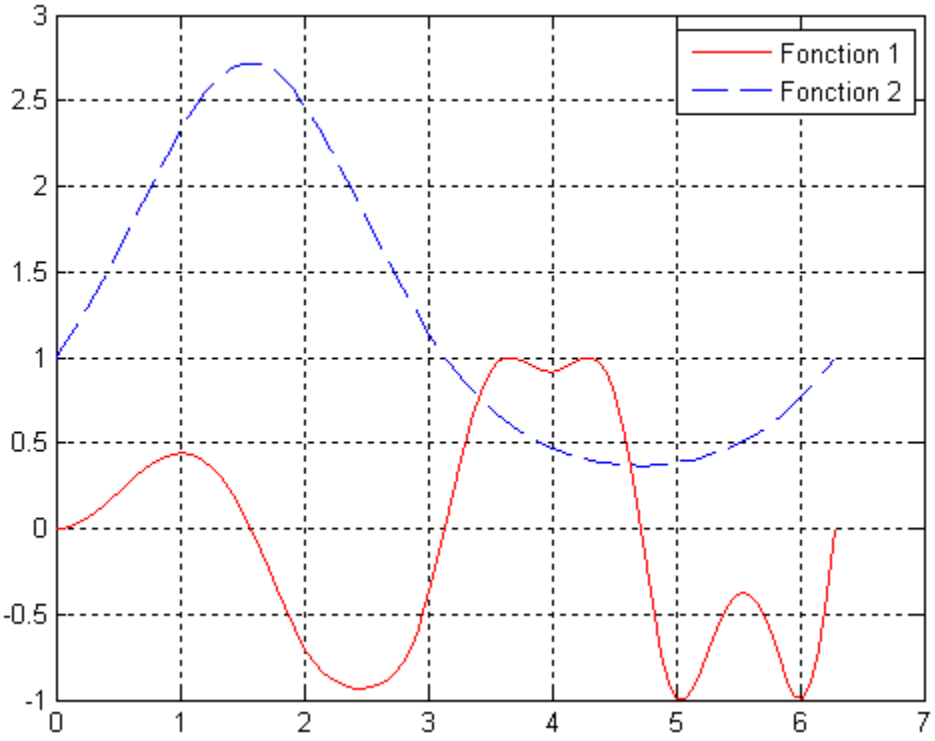
Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Fonctions utiles

- **figure**
- **title**(’titre’) : titre du graphique
- **Xlabel(’x’), ylabel(’y’)**
  - légende des axes Ox, Oy
- **legend**(’lab1’,’lab2’,…)
  - Légende des courbes
- **clf([num]**: Clear figure
- **grid on** : grillage
- **hold on** : permet de superposer des graphiques
- **hold off** :
- **close all** : ferme toutes les fenêtres graphiques

```
x = linspace(0,2*pi,100);
y1 = sin(x.*cos(x).*sin(x));
y2 = exp(sin(x));

clf(0)
title ('Test d''affichage')
xlabel ( 'Axe X')
ylabel ( 'Axe Y')
plot (x,y1,'r');
hold on
plot (x,y2,'b--');
hold off
grid on
legend ('Fonction 1', 'Fonction 2');
```



Courbes et Surfaces 3D

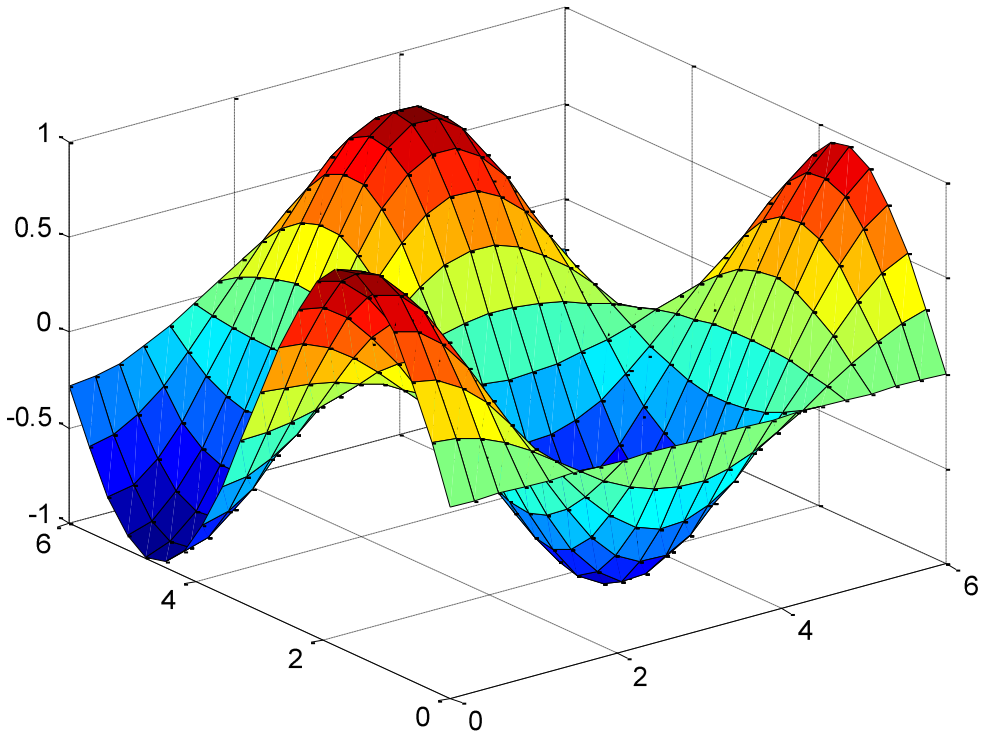
courbes :

plot3(x,y,z)

surfaces :

surf(x,y,z)

```
>> t=[0:0.3:2*pi]';
>> z=sin(t)*cos(t');
>> surf(t,t,z)
```



Exercices

(1) Générez un vecteur `t` de 60 valeurs réparties uniformément entre 0 et 10*pi.
Affichez avec plot3 la courbe passant par les points : `x=sin(t) y=cos(t) z=t.`

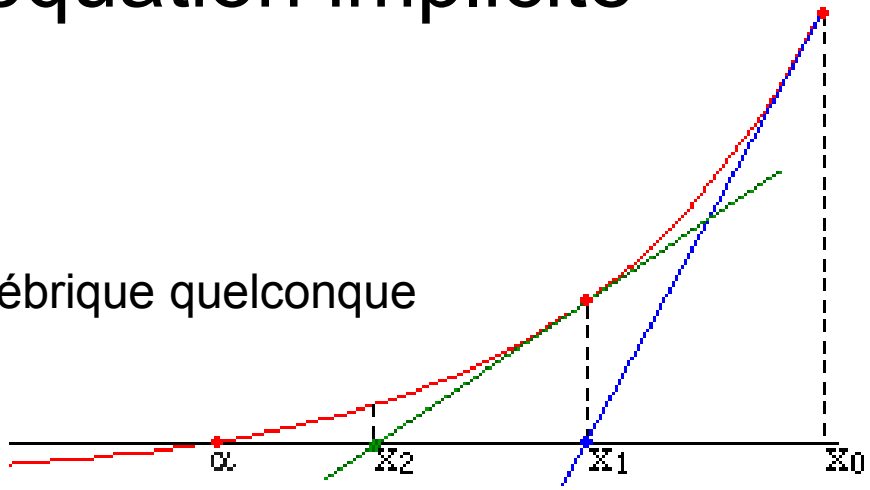
Redessiner la courbe avec 200 valeurs entre 0 et 10*pi et l'afficher à coté de la courbe précédente (avec **subplot**).

(2) Afficher la surface correspondant à la fonction hauteur
z = f(x,y) = exp(-x)*cos(y)
(fonction plot3d)

Exercice: solution d'un équation implicite

Méthode de Newton

pour trouver le zéro d'une fonction



- On cherche la solution d'une équation algébrique quelconque

$$f(x) = 0$$

- Rappel de l'algorithme

  On part d'une abscisse x_0 et on approxime la courbe par sa tangente en x_0.

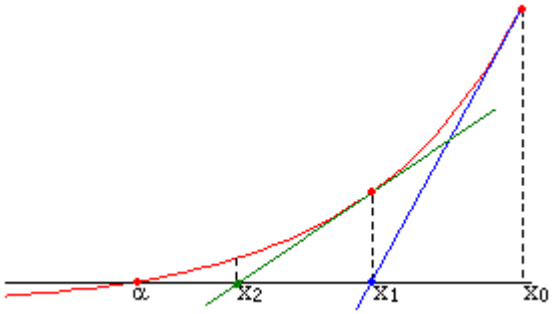
  On calcule le zéro de cette approximation linéaire qui donne x_1 avec:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

  On réitère la méthode (calcul de la dérivée en x_1, calcul de x_2…) tant que

$$f(x_n) > \text{PRECISION}.$$

- Remarques :
  - Cette méthode requiert qu'on puisse calculer la dérivée f '
  - Inclure un contrôle du nombre d'itérations pour éviter une boucle infinie



- Ecrire dans le fichier newton.m une fonction qui:
  - prend en paramètre la fonction f, l'estimation initiale x_0, et le paramètre PREC
  - renvoie la racine de f.

```
function [racine] = newton(f, x0, PREC)
(…)
```

- Tester avec:

$$f(x) = x^2 - 4 = 0$$

$$f(x) = \frac{x}{\cos x} + \cos^2 x = 0$$

  - Tracer la fonction et la dérivée
  - Calculer la racine