République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de 8 mai 1945 Guelma

Faculté des Sciences et de la Technologie
Département de Génie Mécanique
Tronc Commun Sciences et Technologies

Brochure de la matière

TP Langage de Programmation
Deuxième année sciences et Technologies

Enseignant responsable de la matière
Bensouilah Hamza

Grade : M.C.B.

2014

Sommaire

TP 5 : Initiation à la Programmation

TP 6 : Applications et Exercices

Introduction générale

L'objectif de cette brochure est d'initier les étudiants au logiciel de calcul scientifique Matlab de la compagnie *Mathworks* et à la programmation dans cet environnement.

L'idée est d'exposer les bases de cet outil de travail et par la suite habiliter les étudiants à résoudre les problèmes de l'engineering avec celui-ci.

Matlab est beaucoup plus qu'un langage de programmation, il s'agit d'une console d'exécution au même titre que les consoles DOS ou UNIX.

Comme toutes les consoles, Matlab permet d'exécuter des fonctions, d'attribuer des valeurs à des variables, etc.

Plus spécifiquement, Matlab permet d'exécuter des opérations mathématiques, de manipuler des matrices, de tracer facilement des graphiques.

La facilité de développement des applications dans son langage fait qu'il est pratiquement devenu le standard dans son domaine, actuellement, on trouve des nombreuses boîtes à outils (Toolboxes) qui contiennent des fonctions spécialisées permettant d'utiliser l'environnement Matlab pour résoudre des classes spécifiques de problèmes.

Matlab est un interpréteur : les instructions sont interprétées et exécutées ligne par ligne il fonctionne dans plusieurs environnements tels que Windows, Unix, Macintosh.

Il existe deux modes de fonctionnement :

- Mode interactif: Matlab exécute les instructions au fur et à mesure qu'elles sont données par l'usager.
- Mode exécutif: Matlab exécute ligne par ligne un "fichier M" (programme en langage MATLAB).

La présente brochure se présente en six TP et une annexe permettant aux étudiants de la deuxième année Tronc Commun Sciences et Technologies l'initiation au logiciel Matlab.

Dans le premier TP on trouve une introduction générale sur Matlab, dans cette partie on donne un aperçu rapide et général sur la prise en main et les premiers pas de lancement du logiciel.

Le deuxième TP est réservé au vecteurs et matrices, dans cette section l'étudiant doit être en mesure d'introduire les différents espaces de données qui sont les vecteurs (1D) ligne et colonne d'une part et les matrices (2D) d'autre part, une partie de ce chapitre donne les opérations élémentaires entre (vectorielle et scalaire) les matrices et vecteurs.

Les principales instructions de contrôles du logicielle Matlab sont regroupées dans le TP trois, dans cette partie l'étudiant va apprendre la syntaxe utilisées sous Matlab les testes de comparaison logique (booléen), et les boucles de répétition conditionnel et inconditionnel, à la fin de ce chapitre on trouve un ensemble d'exemples d'applications.

On trouve dans le TP quatre les différentes possibilités de tracer une courbe sous Matlab, dans ce TP les courbes en 2D et 3D sont réalisés par les étudiants et cela en trois types de coordonnées : cartésiennes, cylindriques et sphérique.

Le TP cinq représente une initiation à la programmation, les étudiants découvriront les notions de base des algorithmes, les différentes étapes pour construire un programme, ce TP se termine par quelques exemples d'applications.

Le TP six est réservé uniquement aux exercices et applications permettant à l'étudiant la bonne compréhension des différentes commandes étudiées dans les TP précédents par exemple la résolution d'un système d'équations non linéaire et des problèmes de programmation réels.

A la fin on trouve une annexe rassemblant les différents mots clés avec leurs significations, cette annexe est considérée comme une aide aux étudiants et elle doit être distribuée avec le premier TP.

TP1 :

Introduction à MATLAB

TP 1 : Introduction à Matlab

1. Généralités sur MATLAB

Le logiciel MATLAB (*MAT*rix *LAB*oratory) consiste en un langage interprété qui s'exécute dans une fenêtre dite d'*exécution*. L'intérêt de MATLAB tient, d'une part, à sa simplicité d'utilisation : pas de complication, déclaration implicite des variables utilisées et, d'autre part, à sa richesse fonctionnelle : arithmétique matriciel et nombreuses fonctions de haut niveau dans de nombreux domaines (analyse numérique, graphique,…). La programmation sous MATLAB consiste à écrire des *scripts* de commandes MATLAB, exécutables dan la fenêtre d'exécution. En outre, grâce aux diverses *Toolboxes* spécialisés (ensemble de scripts MATLAB), MATLAB s'enrichit au fur et à mesure [1].

2. Lancement de MATLAB

Lors de son lancement (à partir du bureau) la fenêtre d'exécution s'ouvre. Il est alors possible d'exécuter différents types de commandes dans cette fenêtre (chaque commande doit être suivie par « Enter » pour qu'elle soit exécutée ; ce qui apparaît après est le résultat de cette commande), par exemple la commande **help** :

```
>> help
HELP topics

matlab\general      - General purpose commands.
matlab\ops          - Operators and special characters.
matlab\lang         - Programming language constructs.
matlab\elmat        - Elementary matrices and matrix manipulation.
matlab\elfun        - Elementary math functions.
matlab\specfun      - Specialized math functions.
matlab\matfun       - Matrix functions - numerical linear algebra.
matlab\datafun      - Data analysis and Fourier transforms.
matlab\polyfun      - Interpolation and polynomials.
matlab\funfun       - Function functions and ODE solvers.
matlab\sparfun      - Sparse matrices.
matlab\scribe       - Annotation and Plot Editing.
          ……..
```

```
>> help elfun
 Elementary math functions.

 Trigonometric.
  sin       - Sine.
  sind      - Sine of argument in degrees.
  sinh      - Hyperbolic sine.
  asin      - Inverse sine.
  asind     - Inverse sine, result in degrees.
  asinh     - Inverse hyperbolic sine.
  cos       - Cosine.
  cosd      - Cosine of argument in degrees.
  cosh      - Hyperbolic cosine.
  acos      - Inverse cosine.
  acosd     - Inverse cosine, result in degrees.
  acosh     - Inverse hyperbolic cosine.
```

```
Exponential.
   exp         - Exponential.
   expm1       - Compute exp(x)-1 accurately.
   log         - Natural logarithm.
   sqrt        - Square root.
   log10       - Common (base 10) logarithm.          ……
Complex.
   abs          - Absolute value.
   angle        - Phase angle.
   complex      - Construct complex data from real and imaginary parts.
   conj         - Complex conjugate.
   imag         - Complex imaginary part.
   real         - Complex real part.                   ……
```

Permettant ainsi de voir toutes les fonctions mathématiques élémentaires dont dispose MATLAB. On peut maintenant préciser la recherche si l'on veut avoir une idée plus précise de la fonction **log** par exemple :

```
>> help log
 LOG        Natural logarithm.
      LOG(X) is the natural logarithm of the elements of X.
     Complex results are produced if X is not positive.

         See also log1p, log2, log10, exp, logm.
```

Remarque: les commandes MATLAB doivent toujours être tapées en minuscules même si dans l'aide en ligne elles apparaissent en *majuscules*.

3. Variables

Une caractéristique de MATLAB est que les variables n'ont pas à être déclarées.

Exemple :

```
>> a=1
a =
   1
```

Pour afficher la valeur d'une variable ; écrire le nom de la variable suivie par « Enter »

```
>> a
a =
   1
```

4. Identificateurs

Les règles de dénomination des variables sont très classiques :

- Un identificateur débute par une lettre, suivie de lettres, de chiffres ou du caractère souligné(_), par exemple : a, b, x, z1, z2, alpha, nu125, var_2, var_5, ….etc

- Sa longueur est inférieure ou égale à 31 caractères,
- Les majuscules sont distinctes des minuscules.

Exemple :

```
>> A
??? Undefined function or variable 'A'.
```

Voici quelques identificateurs prédéfinis :

- ans : Résultat de la dernière évaluation.
- pi : 3.146….
- inf : Infini (1/0)
- NaN : « Not a Number » (0/0)
- i, j : i et j représentent tous deux le nombre imaginaire unité ($\sqrt{-1}$)
- realmin : Plus petit nombre réel positif
- realmax : Plus grand nombre réel positif

Exemple :

```
>> a+5
>> b=a+pi
>> c=1/0
>> x=0/0
>> y=sqrt(-1)
>> realmax
```

5. Opérations Arithmétiques :

+	:	Addition
-	:	Soustraction
*	:	Multiplication
/	:	Division à droite
\	:	Division à gauche
^	:	Puissance

Exemple :

```
>> 5+0.01
>> 3-10
>> 9*19
>> 1/2
>> 1\2
>> 2^4
```

6. Quelques fonctions sur les variables :

who % afficher la liste des variables utilisées.
whos % même résultat que « who » avec plus de détails.
size % la taille d'une variable.
save % sauvegarder tout les variables.
clear % supprimer tout les variables.
dir % afficher le contenu du répertoire actuel.
load % récupérer les variables sauvegarder.
clc % effacer l'écran sans supprimer les variables.

7. Commentaires :

Le symbole « % » introduit un commentaire, celui-ci n'est pas évalué par l'interpréteur,

```
>> log(1)     % logarithme
```

Lorsqu'une ligne de commande contient au moins une erreur, un message d'erreur sera affiché avec spécification de son type,

```
>> 5-h
>> 2^^6
>> a(2)
```

8. Format d'affichage :

Pour choisir le format d'affichage pour les nombres, on utilise l'instruction **format**,

format short
format long
format short e
format long e
format hex

Exemple:

```
>> a=sin(3*pi/5)
a =
   0.9511
>> format  long
>> a
a =
  0.95105651629515
>> format  long  e
>> a
a =
   9.510565162951536e-001
```

Remarque:

Lorsqu'on ajoute un " ; " à la fin d'une instruction, elle est exécutée mais le résultat n'est pas affiché.

Exemple :

```
>> x=exp(1)
x =
   2.718281828459046e+000
>> y
??? Undefined function or variable 'y'.
>> y=x*2;
>> y
y =
   5.436563656918091e+000
```

On peut aussi définir des variables de type chaine de caractères comme suivant :

```
>> X='TP  MATLAB'
X =
TP  MATLAB
```

```
>> Y='Langage De Programation'
Y =
Langage De Programation
```

```
>> A='Votre Nom Est:'
A =
Votre Nom Est:
```

TP2 :

Les Vecteurs et les Matrices

TP 2 : Les Vecteurs et les Matrice

1. Vecteurs et matrices

On déclare un vecteur colonne de la façon suivante :

```
>> u=[1 ; 3 ; -1]
u =
   1
   3
  -1
```

Un vecteur ligne de la façon suivante :

```
>> v=[1 , 3 , -1]
v =
   1   3   -1
```

Ou bien :

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

Une matrice d'ordre 3x2 :

```
>> A=[1 , 2 ; -1 , 3 ; 4 , 0]
A =
   1   2
  -1   3
   4   0
>> B=[1   2 ; 3   4]
B =
   1   2
   3   4
```

La ',' ou l'espace sert à séparer les éléments d'une ligne et ';' les éléments colonnes.

Pour spécifier un élément d'un vecteur, d'une matrice, on utilise la syntaxe suivante :

```
>> u(2)
ans =
   3
>> v(3)
ans =
  -1
>> A(3,2)
ans =
   0
>> B(1,2)=0
B =
   1   0
   3   4
>> B(2)=5
B =
   1   0
   5   4
```

L'utilisation d'indice hors limite provoque une erreur, comme le montre cet exemple :

```
>> A(3,3)
??? Index exceeds matrix dimensions.
```

```
>> length(v)          (Permet de donner la longueur d'un vecteur)
ans =
   3
```

On peut se servir de raccourcis bien utiles et plus efficaces pour remplir des vecteurs ou des tableaux. En voici quelques-uns :

```
>> u1=1:10          (Incrémentation automatique de 1 à 10 avec un pas de 1)
u1 =
   1   2   3   4   5   6   7   8   9   10
```

```
>> v1=1:2:10        (Incrémentation automatique de 1 à 10 avec un pas de 2)
v1 =
   1   3   5   7   9
```

```
>> Id3=eye(3)                (Matrice identité d'ordre 3)
Id3 =
   1   0   0
   0   1   0
   0   0   1
```

```
>> Un=ones(2)          (Matrice constituée de 1 d'ordre 2)
Un =
   1   1
   1   1
```

```
>> Z=zeros(2,3)        (Matrice nulle d'ordre 2x3)
Z =
   0   0   0
   0   0   0
```

De même, il existe des syntaxes particulières permettant d'extraire des lignes ou des colonnes de matrices :

```
>> A1=[11  12  13 ; 21  22  23 ; 31  32  33]
A1 =
  11   12   13
  21   22   23
```

```
   31    32    33

>> A1(:,1)                    (colonne 1 de la matrice A1)
ans =
   11
   21
   31

>> A1(2,:)                    (Ligne 2 de la matrice A1)
ans =
   21    22    23
```

2. Opérations Matricielles

Les opérations matricielles usuelles sont définies par +, -, *, /, ^

C = A +B est la somme matricielle, $C_{ij} = A_{ij} + B_{ij}$

C = A * B est le produit matricielle, $C_{ij} = \sum_k A_{ik} B_{kj}$

C = A / B est la division matricielle, $C = AB^{-1}$

C = A ^ 3 est la troisième puissance matricielle (C = A*A*A)

Il faut remarquer que ces opérations sont bien définies seulement si les matrices ont des dimensions cohérentes. Pour l'addition A + B, A et B doivent avoir la même taille, pour le produit A*B, le nombre des colonnes de A et le nombre des lignes de B doivent être égaux, les opérations A/B et B^3 demande une matrice B carrée [3].

3. Opérations élémentaires

```
>> u=[1  2  3]
u =
    1    2    3
>> v=[-1  1  1]
v =
   -1    1    1
>> w=u+v                      (Addition)
w =
    0    3    4

>> ut=u'                      (Transposition d'un vecteur ligne ou colonne)
ut =
    1
    2
    3
```

```
>> ut2=[ut   ut]
ut2 =
   1    1
   2    2
   3    3
```

(Concaténation en ligne de deux vecteurs colonnes qui donne une matrice 3x2)

```
>> ut3=[ut ;  ut]
ut3 =
   1
   2
   3
   1
   2
   3
```

(Concaténation en colonne de deux vecteurs colonnes qui donne une matrice 6x1)

```
>> ps=v * ut
ps =
   4
>> u*v
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

(Produit qui conduit au produit scalaire)

```
>> M=ut*v
M =
   -1    1    1
   -2    2    2
   -3    3    3

>> L=M+2*eye(3)
L =
    1    1    1
   -2    4    2
   -3    3    5

>> y=L\ut
y =
   0.1667
   0.3333
   0.5000
```

(Produit qui conduit à une matrice)

(*Résolution du système linéaire* $L.y = ut$)

4. Opérations élément par élément

Pour exécuter des opérations entre matrices élément par élément il faut faire précéder l'operateur d'un point. Les opérateurs élément par élément sont donc [5] : .* ./ .^

C = A .* B est le produit élément par élément, $C_{ij} = A_{ij}B_{ij}$

C = A ./ B est la division élément par élément, $C_{ij} = \frac{A_{ij}}{B_{ij}}$

C = A .^3 est la troisième puissance élément par élément, $C_{ij} = A_{ij}^3$

Exemples :

Exécuter les instructions suivantes :

```
>> A = [1 2 3 ; 4 5 6]
>> B = [7 8 9 ; 10 11 12]
>> C = [13 14 ; 15 16 ; 17 18]
>> A + B
>> A. + B
>> A + C
>> A. + C
```

```
>> A * C
>> A. * C
>> A * B
>> A. * B
>> A ^2
>> A. ^2
>> A. ^(1/2)
```

TP3 :

Les Structures de Contrôle

TP 3 : Les Structures de contrôle

Dans cette section, on indique les principales syntaxes permettant de contrôler le flux d'exécution d'une application MATLAB. On les utilise essentiellement en mode Programmation [2].

1. Opérateurs booléens

Avant de décrire la syntaxe du test sous MATLAB, indiquons les principaux opérateurs de relation ainsi que les opérateurs booléens qu'utilise MATLAB.

Strictement inférieur à	<
Inférieur ou égale à	<=
Strictement supérieur à	>
Supérieur ou égale à	>=
Egale à	= =
Différent de	~ =
Et logique (and)	**&**
Ou logique (or)	|
Non logique (not)	~

Le résultat d'un test est un booléen qui, sous MATLAB, prend la valeur 1 pour vrai et 0 pour faux. Par exemple, on a les résultats suivants :

```
>> r=1<2
r =
   1
>> r=~ ((1>2) | (0~=0))          % (traduction MATLAB de l'expression logique : non (1 > 2 ou 0 ≠ 0))
r =
   1
```

Il existe d'autres fonctions booléennes, par exemple ***xor, isfinitr, isnan, isinf,*** …. dont on trouvera la description en faisant ***help ops***.

2. Syntaxe du test if

Ce type de structure de programmation est très utile pour vérifier des conditions.

```
if   expression booléenne
        Instructions
end
```

```
if   expression booléenne
        Instructions
else
        Instructions
end
```

```
if   expression booléenne 1
        instructions
     else if   expression booléenne 2
        instructions
     else
        Instructions
     end
end
```

3. Syntaxe du branchement (switch)

Les boucles *switch* permettent parfois de remplacer les boucles *if-elseif-else*,

```
switch (expression : est un scalaire ou une chaîne de caractères)
case value 1
instructions (instructions effectuées si expression=value1)
case value 2
instructions
…….
otherwise
instructions
end
```

4. Syntaxe de boucle (while et for)

Une boucle *while* permet de répéter une opération tant qu'une condition n'est pas remplie.

```
while expression
instructions
end
```

Les boucles *for* sont très utiles dans la plupart des applications mathématiques (par exemple, pour effectuer un calcul sur tous les éléments d'un vecteur) [2].

```
for indice=début : pas : fin (si le pas n'est pas précisé, par défaut il vaut 1)
instructions
end
```

Pour sortir d'un test ou d'une boucle, on utilise la commande break (voir le help).

Exemple 1 : On reçoit un entier ***a***, s'il est impaire négatif, on le rend positif et s'il est impaire positif, on lui ajoute 1. S'il est pair, on ajoute 2 à sa valeur absolue.

```
a = input('Donner un entire de votre choix  a =')
if  a < 0  &  mod(a,2)  ~= 0  % mod permet de trouver le reste d'une division
        b = -a ;
elseif  a >= 0  &  mod(a,2) ~= 0
        b = a + 1;
else
        b = abs(a)+2;
end
```

Exemple 2 : Si l'on veut calculer les carrés des nombres pairs entre 0 et 10 :

```
for   i= 0  : 2  : 10
        carre = i^2
end
```

Exemple 3 : On veut trouver le nombre d'entiers positifs nécessaires pour avoir une somme plus grande que 100. On pourrait réaliser cette tache de la manière suivante :

```
n = 0 ;   % initialisation, des valeurs
      somme = 0 ;
while  somme < 100
      n=n+1 ;  % itération de n
      somme = somme + n ;  % nouvelle somme
end
```

5. Scripts et m-files

MATLAB permet d'écrire des programmes par l'enregistrement des scripts (programmes) sous forme de fichier de texte appelées *m-files*, en raison de leur extension.
Dans les versions récentes de MATLAB il existe un petit éditeur intégré que l'on peut appeler à partir du menu [3] :

File ⟶ New ⟶ M-file

6. Exécution d'un m-file

Pour exécuter le script contenu dans un *m-file*, il suffit de taper le nom de ce *m-file* dans la fenêtre de commande suivi de <Enter>.

Exemple 4 :

Refaire les mêmes exemples 1, 2 et 3 en utilisant des scripts de commandes (*m-files*).

Exemple 5 :

Ecrivez une fonction calculant le signe d'un nombre ***(a),*** (-1, 0 ou 1 suivant que le nombre est négatif, nul ou positif). La structure de contrôle ***if….else …end*** peut être utile ici.

```
function   s=sgn(a)
if  a>0
   s=1;
elseif   a<0
   s=-1;
else
   s=0;
end;
```

Exemple 6 :

A l'aide de l'opérateur « **:** » écrivez une fonction de trois paramètres ***a, b, n*** qui calcule un vecteur à n composantes également réparties entre ***a*** et ***b***.

```
function   v=linearspace(a,b,n)
if  n<2
  error('n doit être entier > 1');
end;
h=(b-a)/(n-a);
v=a:h:b
```

Exemple 7 :

Résolvez les équations du premier et second degré avec deux fonctions d'en-têtes respectives : *function x = equ1(a, b)* et *function x = equ2 (a, b, c)* en réfléchissant aux problèmes suivants :

- Quel est le domaine de validité de vos fonctions ?

```
function   x = equ1(a,b)
if   a == 0
  if   b ~=0
     x = [ ];
  else
     x = NaN;
  end;
else
  x = -b/a;
end;
```

- Que doit être *x* ? Un paramètre de sortie unique est il suffisant ?
- Peut-on faire en sorte que l'appel *equ2(a, b)* soit valide et équivalent à *equ1(a, b)* ?

```
function    x = equ2(a,b,c)
if    nargin == 2
   x = equ1(a,b);
elseif    a == 0;
   x = equ1(b,c);
else
   delta = b*b - 4*a*c;
   if    delta >= 0
      if b > 0
         x = (-b-sqrt(delta))/(2*a);
      else
         x = (-b+sqrt(delta))/(2*a);
      end;
      if    x == 0
         x = [x    0];
      else
         x = [x     (c/a)/x];
      end;
   end;
end;
```

Exemple 8 :

Ecrivez une fonction recherchant dans un vecteur ***v*** la composante maximale. Quel est le nombre minimum de comparaisons nécessaires ?

```
function     i = maxi(t);
ind = 0;
maxim = -Inf;
for     i = 1:length(t)
   if      t(i) > maxim
      maxim = t(i);
      ind = i;
   end;
end;
```

Exemple 9 :

Reproduisez le tableau suivant :

x	*exp(x)*	*exp(-x)*	*dériv*
1.0000 e-004	*1.0001 e+0000*	*9.9990 e-001*	*1.0000 e+000*
1.0000 e-008	*1.0000 e+000*	*1.0000 e-000*	*1.0000 e+000*
1.0000 e-012	*1.0000 e+000*	*1.0000 e-000*	*1.0000 e+000*
1.0000 e-016	*1.0000 e+000*	*1.0000 e-000*	*5.5511 e-001*
1.0000 e-020	*1.0000 e+000*	*1.0000 e-000*	*0.0000 e+000*

La quatrième colonne est la différence des 2 précédentes divisés par 2 fois la première.

```
x = cumprod(1.0e-4*ones(5,1));
a = [x, exp(x), exp(-x), (exp(x) - exp(-x))./(2*x)];
fprintf('%15s %15s %15s %15s\n','x', 'exp(x)', 'exp(-x)', 'dériv');
fprintf('%15.4e %15.4e %15.4e %15.4e\n', a');
```

Exemple 10 :

Ecrivez une fonction calculant les ***n*** premières lignes du triangle de Pascal. On pourra utiliser les relations : $C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$ et $C_n^1 = C_n^n = 1$.

```
function     c = triang(n)
c = zeros(n,n);
c(:,1) = 1;
for    i = 2:n
   for    j = 2:i
      c(i,j) = c(i-1,j-1) + c(i-1,j);
   end;
end;
```

TP4 :

Les Graphes

TP 4 : Les Graphes

1. Introduction :

MATLAB peut produire des graphiques couleurs 2D et 3D impressionnants. Il fournit aussi les outils et moyens de personnaliser et de modifier pratiquement tous leurs aspects, facilement et de manière parfaitement contrôlée [1].

2. Graphiques 2D :

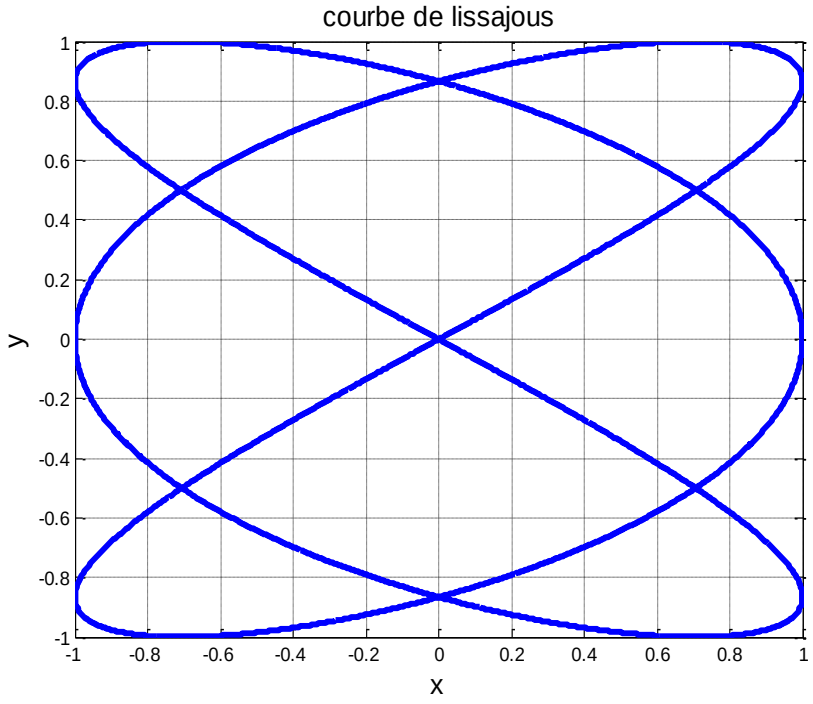
La commande *plot* permet de tracer des graphiques *xy*. Avec ***plot(x, y)*** on trace *y* en fonction de *x* ; *x* et *y* sont des vecteurs de données de même dimensions.

Définition de l'intervalle de x et calcul des valeurs de la fonction y=f(x)	*Tracé de la fonction*	*Documentation du graphique*
>> **x = -pi : 0.1 : pi;** >> **y = sin(x) ;**	>> **plot(x, y)**	>> **grid** >> **xlabel ('x') , ylabel ('y')** >> **title ('y=f(x)')**

Au graphique courant vous pouvez ajouter un titre, une grille, une légende pour les axes ou du texte sur le graphe grâce aux commandes : *title*, *grid*, *xlabel*, *ylabel* et *text*.

On peut aussi tracer des courbes paramétriques :

```
>> t = 0:0.001:2*pi;
>> x = cos(3*t);
>> y = sin(2*t);
>> plot(x, y)
>> grid
>> xlabel ('x')
>> ylabel ('y')
>> title ('courbe de lissajous')
```



2.1. Graphiques (2D) en coordonnées cartésiennes :

La commande *plot(x, y, s)* permet de tracer des graphiques (courbes ou nuages de point) de vecteurs de dimensions compatibles (*y* en fonction de *x*) [1]. Le choix du type et de la couleur du tracé peut se faire avec le paramètre facultatif *s* qui est une chaine composée de 1 à 3 caractères parmi ce qui suit :

couleurs		tracés discontinus (symbole)		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	s	carrés		
w	blanc	d	diamants		
k	noir	p	étoiles à cinq têtes		

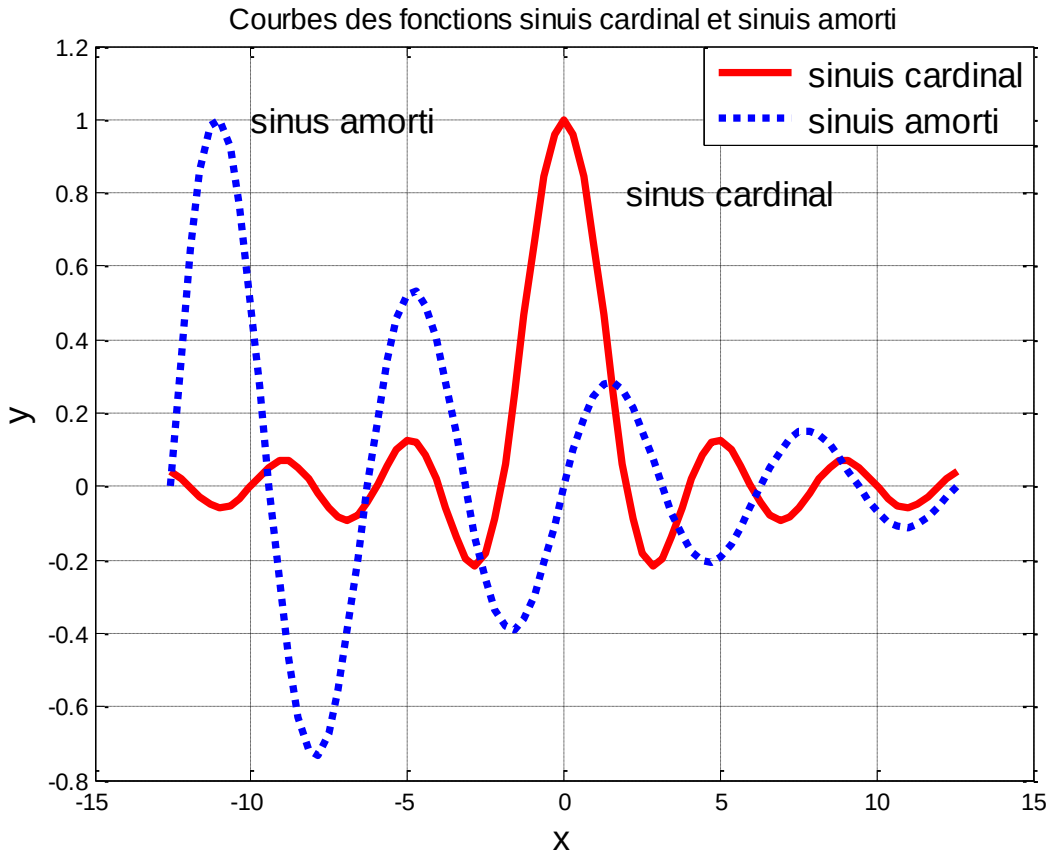
a. Courbes de fonctions

```
>> x = -10*pi : 10*pi;                    %choix de l'intervalle de définition
>> y = sin(x).*exp(-0.1*x);                %calcul des valeurs de la f(x) = sin(x) * e^(-0.1x)
>> plot (x, y, 'g')          %tracé de la fonction f(x) = sin(x) * e^(-0.1x) (en couleur verte)
>> grid                                    %Un quadrillage donne plus de lisibilité au graphe
>> title (' Tracé de la courbe d ' 'une fonction')          %ajouter un titre au graphe
>> xlabel ('  x : axe des abscisses')
>> ylabel ('  y : axe des ordonnées')
>> text (10,-3,'Courbe f(x)')      %texte explicatif, (10,-3) coordonnées du début du texte
>> gtext ('Sinusoïde amortie')   %texte à un endroit quelconque sélectionné par l'utilisateur
```

Exemple de courbes (Nom du fichier courbes.m)

Nous proposons le tracé des courbes des fonctions *sinus cardinal* et *sinus amorti* dans une même fenêtre graphique. Un texte explicatif sera inséré à coté de chacune d'elles [1].

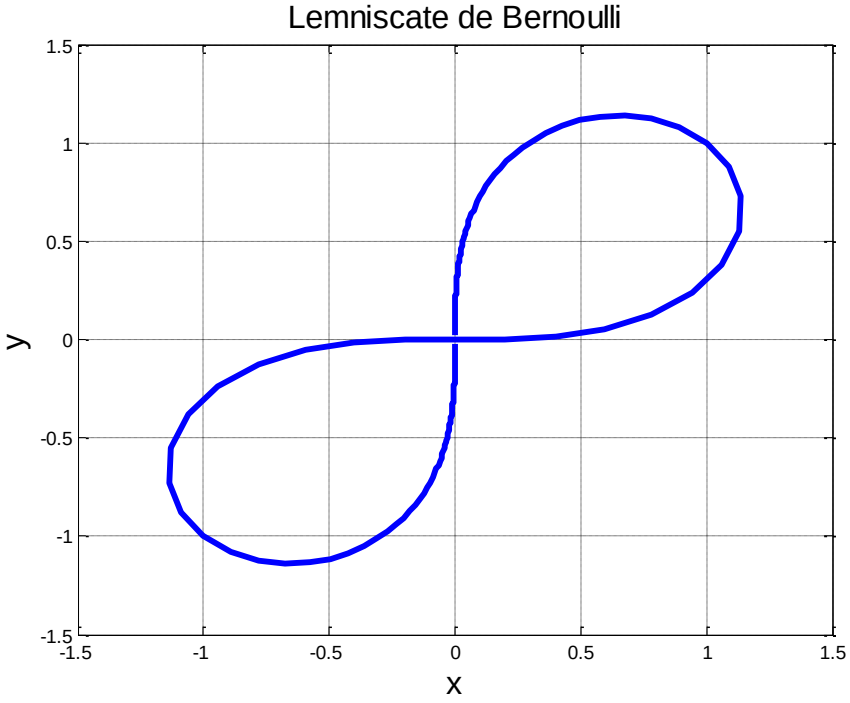
```
>> x = -4*pi : pi/10 : 4*pi;
% sinus cardinal en trait rouge continu
>> y1 = sinc (x/2);
>> plot (x, y1,'r')
>> hold on                    %garder la même fenêtre graphique
% sinus amorti en pointillées, couleur verte
>> y2 = sin(x).*exp (-0.1*x)/3;
>> plot (x, y2, 'g:')
% documentation du graphique
>> xlabel ('x')
>> ylabel ('y')
>> title ('Courbes des fonctions sinuis cardinal et sinuis amorti')
% insertion du texte explicatif
>> txt_x = [-10 ; 2];
>> txt_y = [1 ; 0.8];
>> txt_val = ['sinus amorti  ' ; 'sinus cardinal'];
>> text (txt_x, txt_y, txt_val);
% insertion d'une légende
>> h = legend(' sinuis cardinal ',' sinuis amorti',2);
```



b. **Courbes paramétriques** (Nom du fichier cb_param.m)

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

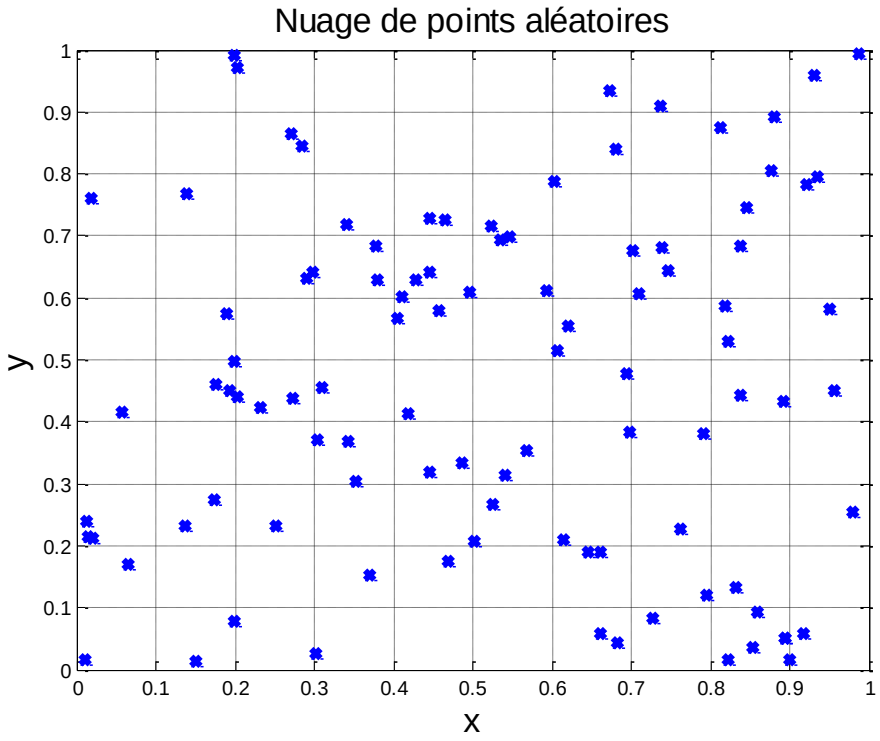
```
>> t = -100 : 0.1 : 100;
>> x = (2*t)./(1+t.^4);
>> y = (2*t.^3)./(1+t.^4);
>> plot (x, y)
>> title('Lemniscate de Bernoulli')
>> xlabel ('x')
>> ylabel ('y')
>> grid
```



c. **Nuage de points** (Nom du fichier nuage.m)

On précisera dans la commande *plot*, le symbole (*, +, x, ., o) à affecter aux différents points.

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



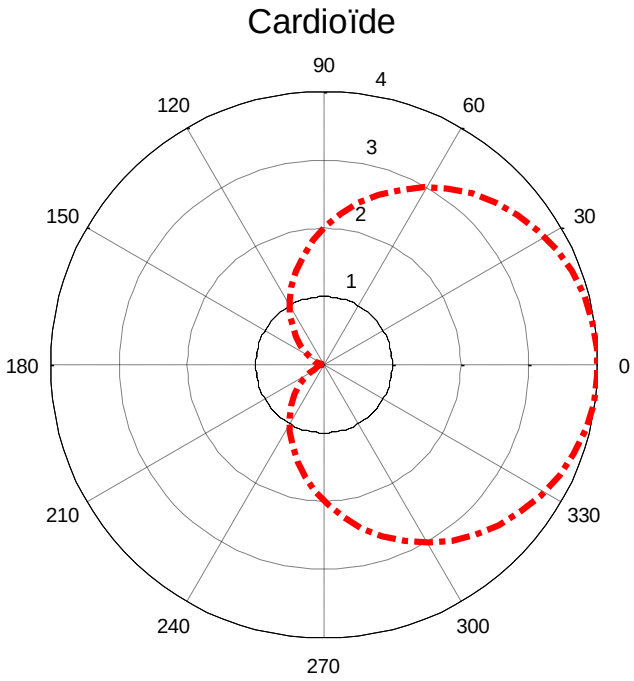
2.2. Graphiques (2D) en coordonnées polaires :

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, ect.) et à sa couleur. Les différentes possibilités sont identiques à celles de la fonction *plot* [1].

a) **Limaçons**

Un graphe qui a pour une équation polaire $r = a + b\,cos\theta$ ou $r = a + b\,sin\theta$, Ou a et b ne sont pas nuls, s'appelle un Limaçon. La cardioïde est un cas particulier du Limaçon pour lequel $|a| = |b|$. (Nom du fichier *cb_pol1.m*)

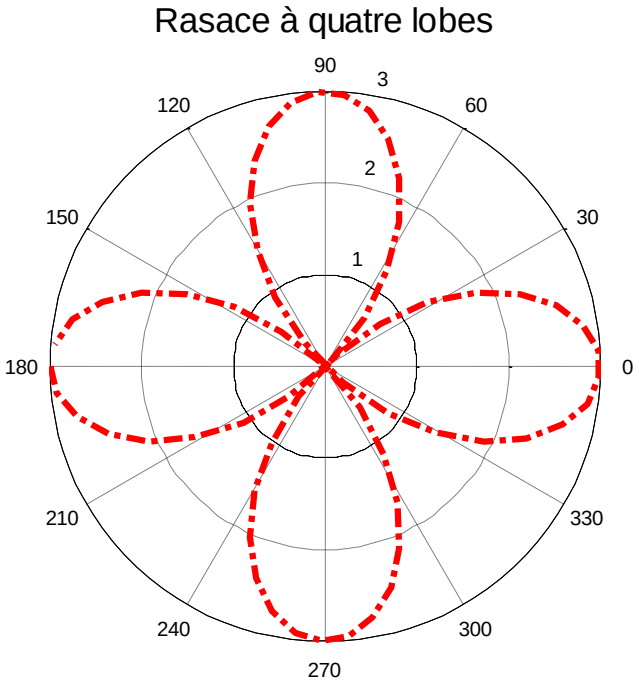
```
>> % courbes en coordonnées polaires : 'Cardioïde'
>> theta = -pi : 0.1 : pi;
>> r = 2+2*cos(theta);
>> polar (theta, r, 'r-.')
>> title ('Cardioïde')
```



b) Rosaces

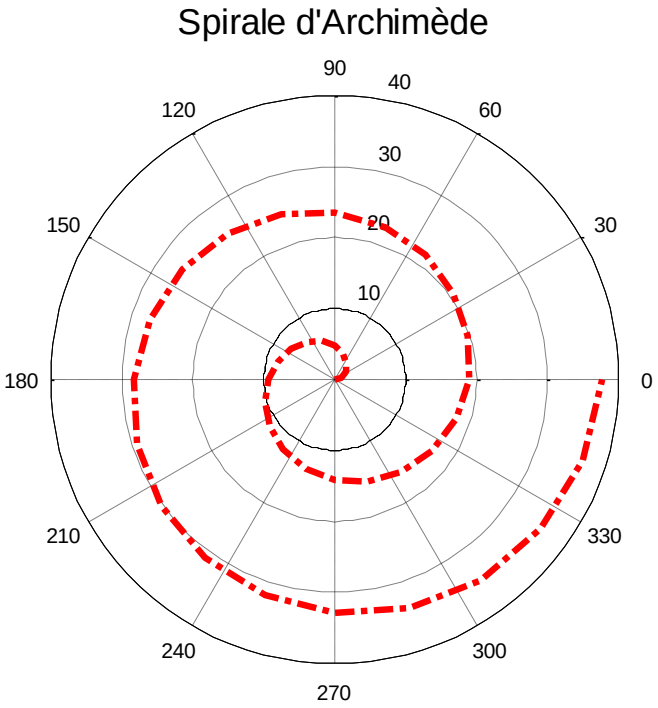
Un graphique qui a pour équation polaire $r = a\,sin(n\theta)$ ou $r = cos(n\theta)$ où *n* est un entier positif supérieur à 1 et *a* un nombre réel quelconque, est constitué de boucles nouées à l'origine. Lorsque *n* est impair on obtient *n* boucles, et *2n* boucles si *n* est pair. (Nom du fichier *cb_pol2.m*)

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



c) Spirale d'Archimède

La spirale d'Archimède a pour équation en coordonnées polaires $r = a\theta$ (a est un réel). (Nom du fichier *cb_pol3.m*)

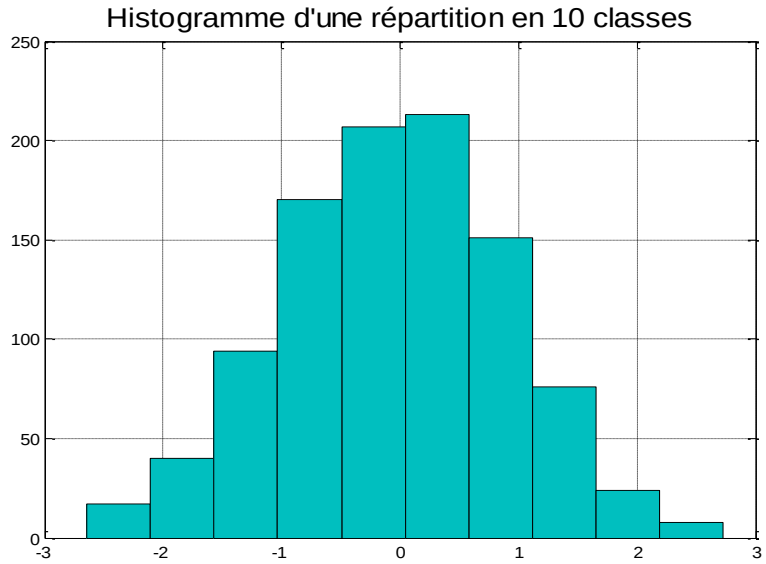



```
>> theta = 0 : pi/10 : 4*pi;
>> r = 3*theta;
>> polar (theta, r, 'r-.')
>> title ('Spirale d''Archimède')
```

d) Les diagrammes

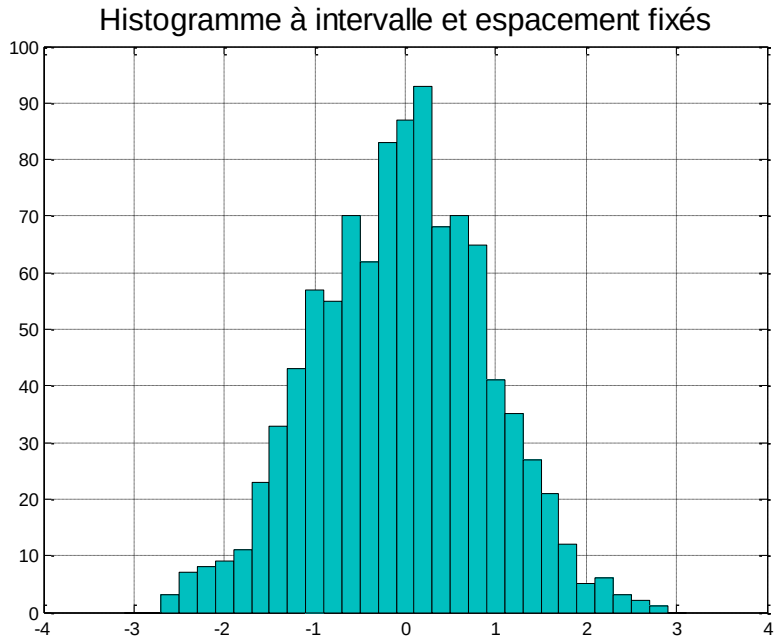
Le tracé d'histogrammes se fera à l'aide de la commande *hist*.

```
>> y = randn(1000,1);
>> hist (y, 10);
>> grid
>> title ('Histogramme d''une répartition en 10 classes')
```

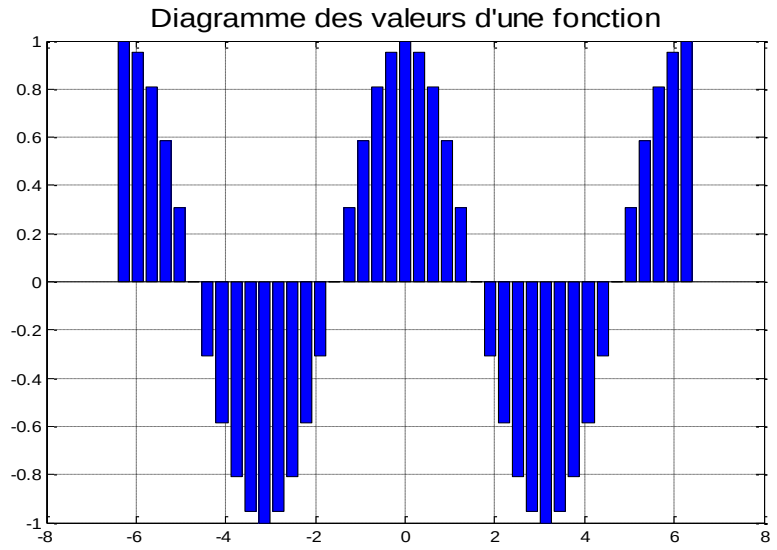


Au lieu de spécifier directement le nombre de classes pour la commande *hist*, nous pouvons indiquer un vecteur qui représente l'intervalle du tracé ainsi que la largeur des classes [1].

```
>> x = -3 : 0.2 : 3;
>> hist (y, x), grid
>> title ('Histogramme à intervalle et espacement fixés')
```



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



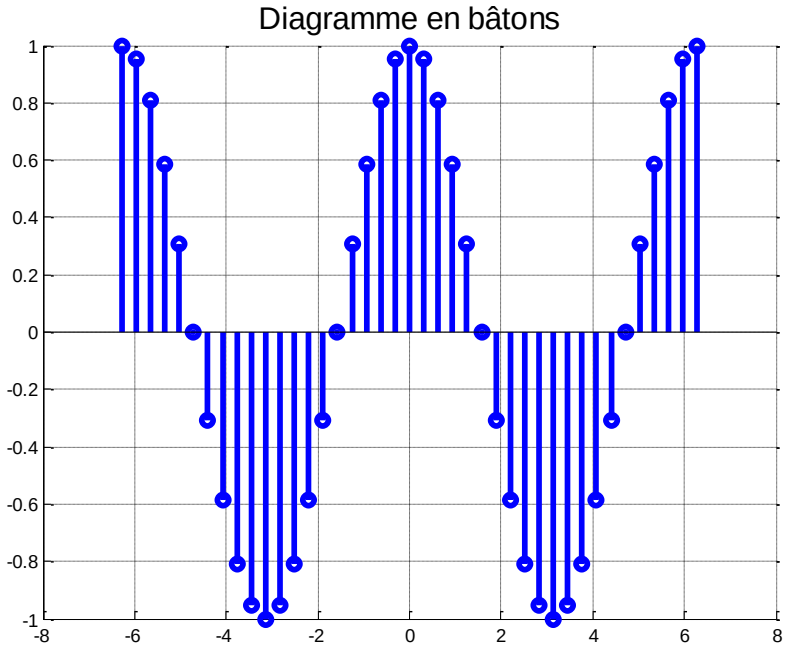

```
>> x = -2*pi : pi/10 : 2*pi;
>> y = cos(x);
>> bar (x, y), grid
>> title('Diagramme des valeurs d''une fonction')
```

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

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

La fonction *stem(x, y)* permet le tracé d'une séquence de valeur discrètes.

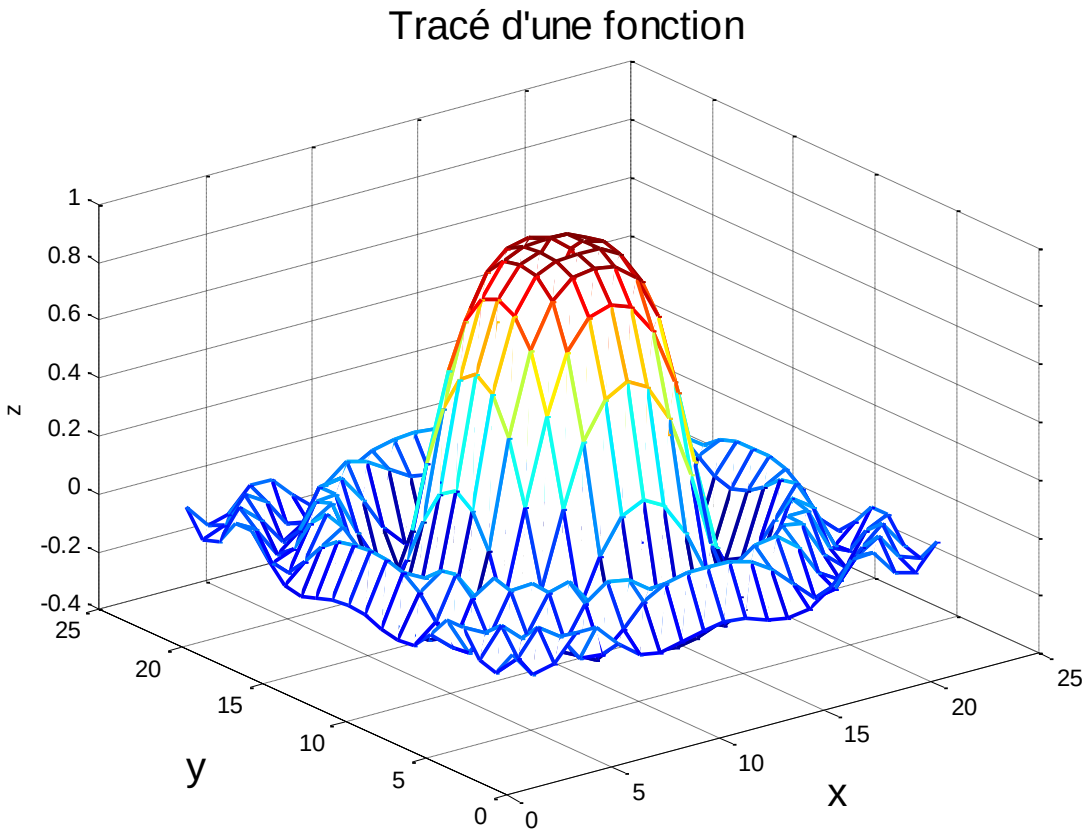
```
>> stem(x, y)
>> grid, title ('Diagramme en bâtons')
```



3. Les graphiques 3D :

Soit l'exemple suivant d'une fonction à 2 variables *x*, *y* : $z = sin(x^2 + y^2)/(x^2 + y^2)$

```
>> x = -pi : pi/10 : pi;          %Pour x variant de −π à +π avec un pas de π/10
>> y = x;                         %Affectation des valeurs de x dans y
>> [X, Y] = meshgrid(x , y);      %On génère deux matrices carrées X et Y qui définissent le domaine de
                                  %calcul de z, on utilisera pour ceci la fonction meshgrid :
>> Z = sin (X.^2+Y.^2)./ (X.^2+Y.^2);   %On évalue la fonction z et on stocke les données dans la variable Z
>> mesh(Z), grid                  %On dessine la surface représentative de la fonction
>> xlabel('x'), ylabel('y'), zlabel('z'), grid   %Nous pouvons rajouter un titre pour le tracé (title), des légendes
>> title ('Tracé d''une fonction')   %pour les axes (xlabel, ylabel et zlabel) ainsi qu'un quadrillage (grid)
```

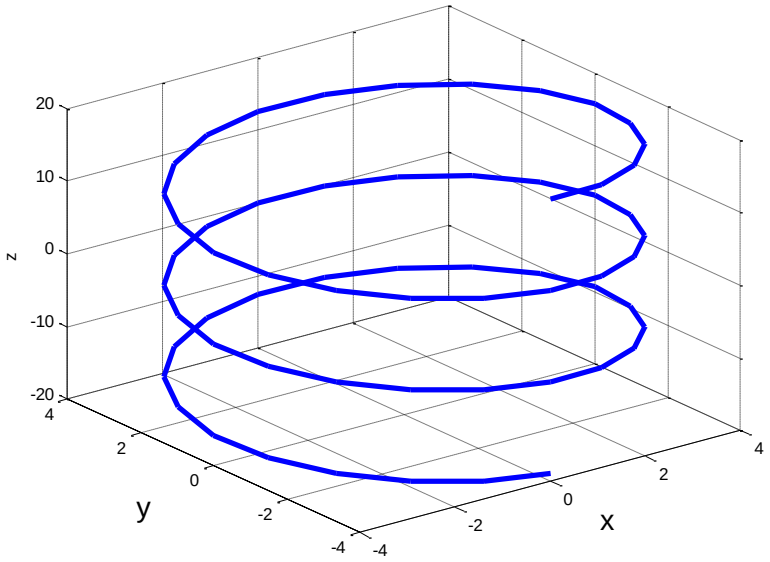


3.1. Courbes 3D :

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'*.

Nous proposons comme exemple, le tracé d'une hélice circulaire définie par une équation paramétrique, (Nom du fichier *crb_3d1.m*)

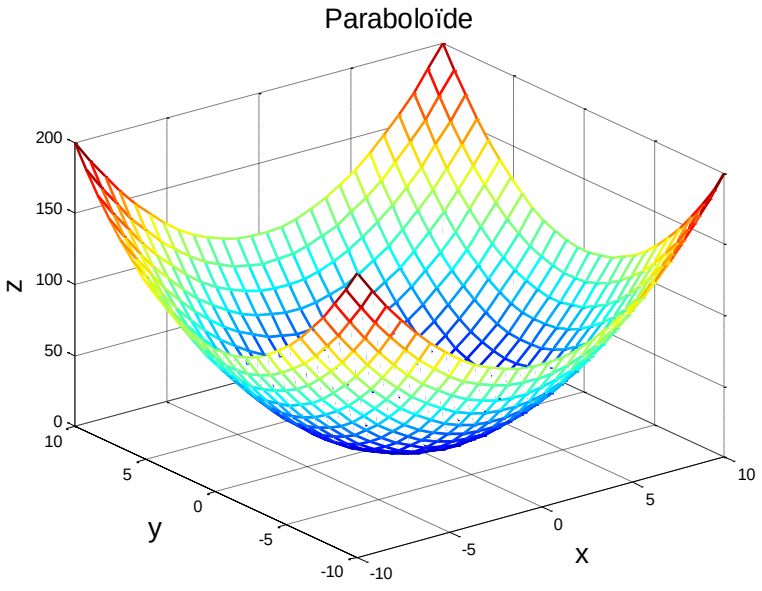
```
%Hélice circulaire : Courbe gauche
>> t = -3*pi : pi/10 : 3*pi;
>> x = 4*sin(t);
>> y = 4*cos(t);
>> z = 2*t;
>> plot3 (x, y, z)
%Documentation du graphique
>> title ('Hélice circulaire')
>> grid, xlabel('x'), ylabel('y'), zlabel('z')
```



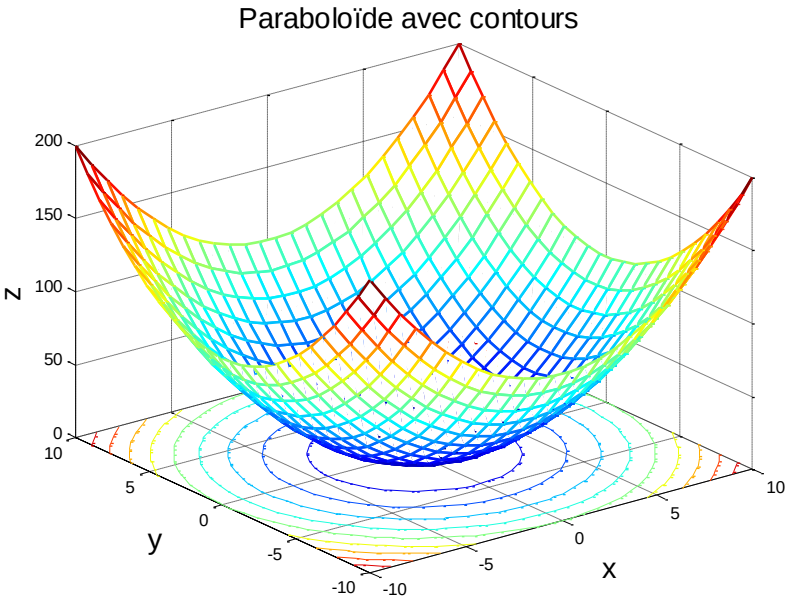
3.2. Surfaces 3D :

Soit l'exemple d'une fonction à 2 variables : $z = x^2 + y^2$, pour x et y variant de -10 à 10 avec un pas de 0.8

```
>> x = -10 : 0.8 : 10;
>> y = x;
>> [X Y] = meshgrid(x, y);
>> Z = X.^2 + Y.^2;
>> mesh(X, Y, Z)
>> grid
>> title('Paraboloïde')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

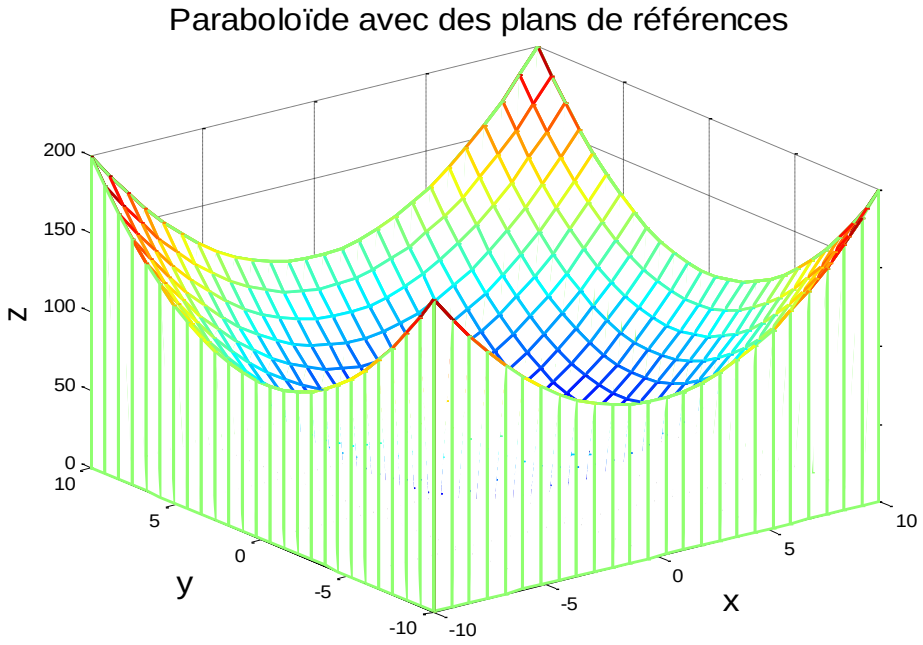


La commande *meshc*, de même syntaxe que *mesh*, trace une surface avec une projection des contours sur le plan *(Ox, Oy)*. Elle est la combinaison des commandes *mesh* et *contour*.

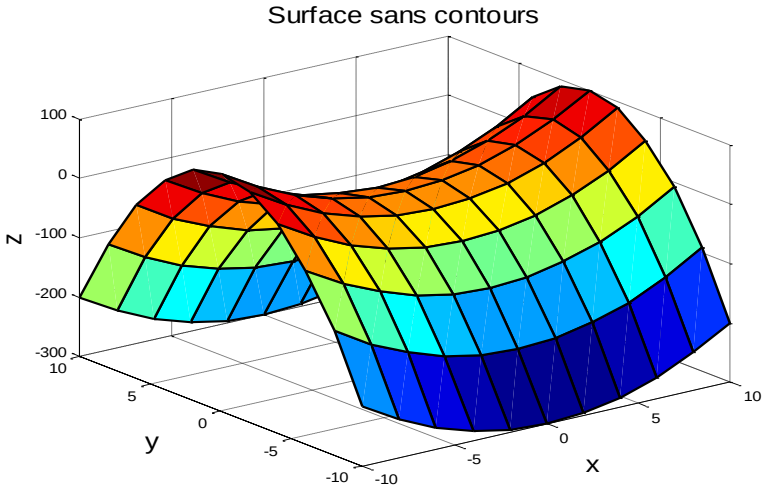



```
>> meshc (X, Y, Z)
>> grid
>> title ('Paraboloïde avec
contours ')
>> xlabel ('x')
>> ylabel ('y')
>> zlabel ('z')
```

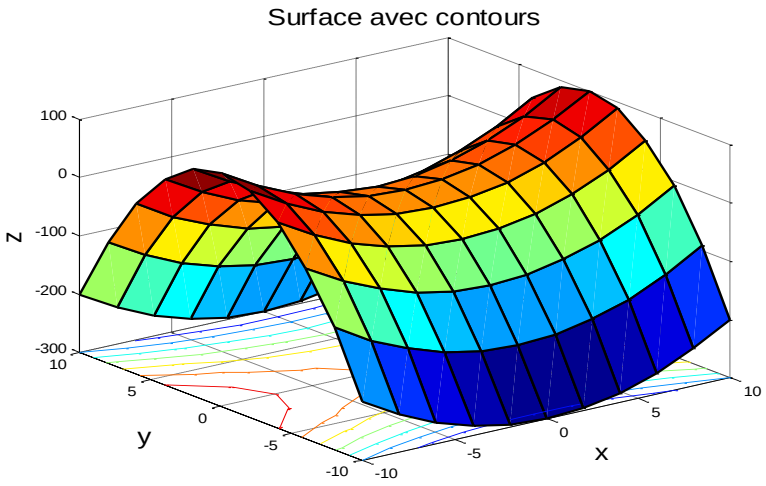
```
>> meshz (X, Y, Z)%surface avec des plans de références
                % aux limites de chaque axe
>> grid
>> title ('Paraboloïde avec des plans de références ')
>> xlabel ('x')
>> ylabel ('y')
>> zlabel ('z')
```



Les commandes *surf*, *surfc* et *surf1* ont une syntaxe analogue à celle de *mesh*, cependant, on obtient une surface colorée au lieu d'une représentation en fil de fer.

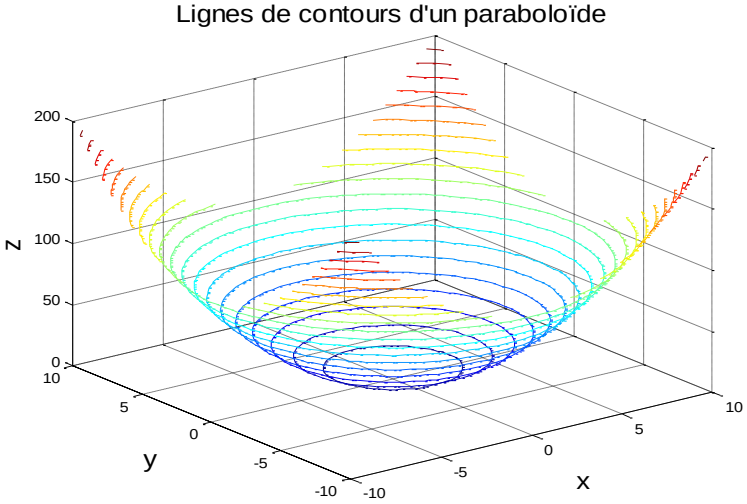



```
%Tracé de la surface z = x^2 − 3 y^2
>> x = -10 : 2 :10;
>> y = x;
>> [X, Y] = meshgrid(x, y);
>> Z = X.^2 - 3* Y.^2;
%Sans contours : surf
>> figure (1)
>> surf (X, Y, Z)
>> grid
>> title ('Surface sans contours')
>> xlabel('x'), ylabel('y'), zlabel('z')
```




```
%Avec contours : surf
>> figure (2)
>> surfc (X, Y, Z)
>> grid
>> title ('Surface avec contours')
>> xlabel('x'), ylabel('y'), zlabel('z')
```

La commande *contour3* a pour syntaxe générale *[C, H] = contour3(x, y, z, N)*




```
>> x = -10 : 0.5 :10; y = x;
>> [X, Y] = meshgrid(x, y);
>> Z = X.^2 + Y.^2;
>> contour3(X, Y, Z, 20)
>> contour3(X, Y, Z, 20)
>> title ('Lignes de contours d''un
paraboloïde')
>> grid, xlabel('x'), ylabel('y'),
zlabel('z')
```

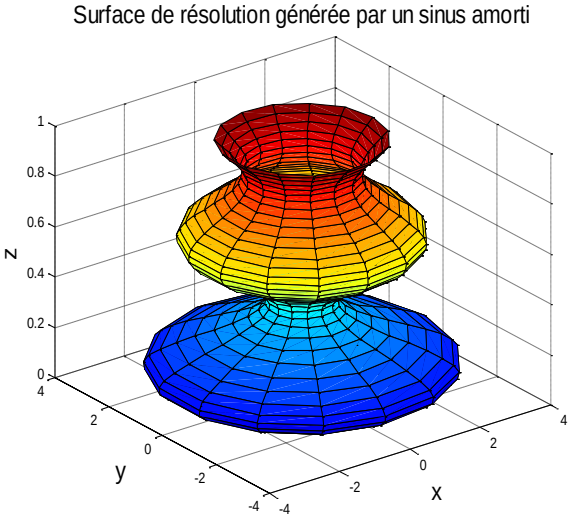
3.3. Volumes et surfaces de révolution

a) Cylindres

La génération d'une surface de révolution d'axe Oz et ayant comme génératrice une courbe $R = f(x)$ se fera à l'aide de la fonction *cylinder* selon la syntaxe suivante

[x, y, z] = cylindre(R, N)

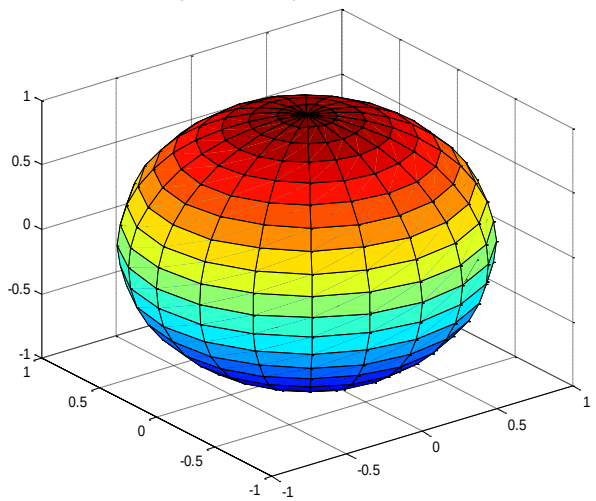
R : vecteur représentant les variations du rayon (courbe génératrice), il doit être régulièrement espacé. *N* : entier représentant le nombre de points sur la circonférence pour un rayon donné. La résolution du tracé est d'autant meilleure que N est grand.




```
%courbe génératrice de la surface de révolution
>> t = -2*pi : pi/10 : 2*pi;
>> r = sin(t).*exp (-0.1*t)+2;
%coordonnées des points de la surface
>> N =15;
>> [x, y, z] = cylinder(r,N);
%tracé de la surface
>> surf (x, y, z)
>> title ('Surface de résolution générée par un sinus amorti')
>> grid, xlabel ('x'), ylabel ('y'), zlabel ('z')
```

b) Sphères

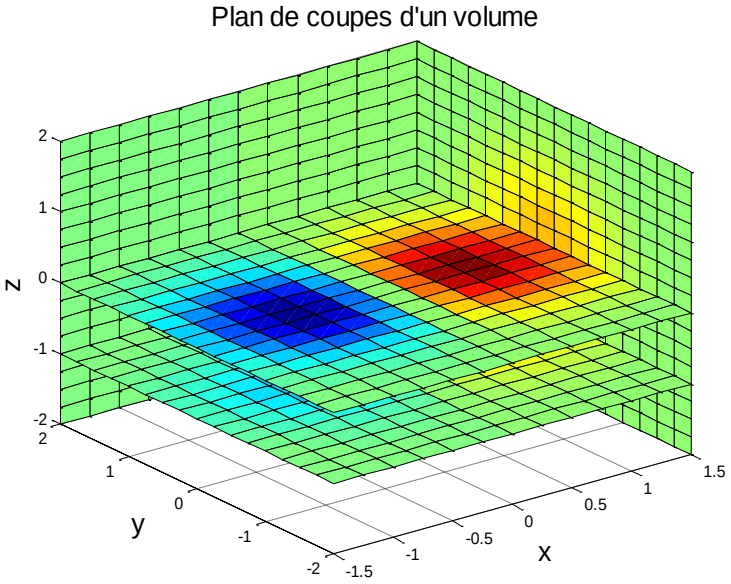
La commande *[x, y, z] = sphere(N)* génère trois matrices carrées *x*, *y*, *z* d'ordre (N+1) qui représentent la sphère de rayon unité. L'instruction *surf(x, y, z)* produit le graphique de la sphère. Si *N* n'est pas précisé, la valeur 20 est prise par défaut [1].




```
>> N = 20;
>> [x1, y1, z1] = sphere(N);
>> figure (gcf)
>> surf (x1, y1, z1)
>> title('Sphère de rayon unité, N=20')
```

c) Volumes

La commande slice offre la possibilité de tracer des plans de coupes d'un volume. Sa syntaxe est la suivante *slice(x, y, z, XI, YI, ZI, N)*, *x*, *y*, *z* : coordonnées des points générés par *meshgrid(x, y, z)*, où *x*, *y* et *z* représentent les domaines de variation des coordonnées, *XI*, *YI*, *ZI* : plans des coupes suivant les différents axes, *N* : longueur du vecteur x.




```
%domaines de variation des coordonnées
>> x = -1.5 : 0.25 : 1.5;
>> y = -2 : 0.25 : 2;
>> z = -2 : 0.25 : 2;
%génération du volume
>> [X, Y, Z] = meshgrid(x, y, z);
>> V = X .* exp(-X .^ 2 -Y .^ 2 -Z .^ 2);
%plans de coupes
>> XI = [1.5];
>> YI = [2];
>> ZI = [0 -1];
%tracé des plans de coupes du volume
>> slice(X,Y,Z,V,XI,YI,ZI)
>> title ('Plan de coupes d''un volume')
>> xlabel ('x'), ylabel ('y'), zlabel ('z')
```

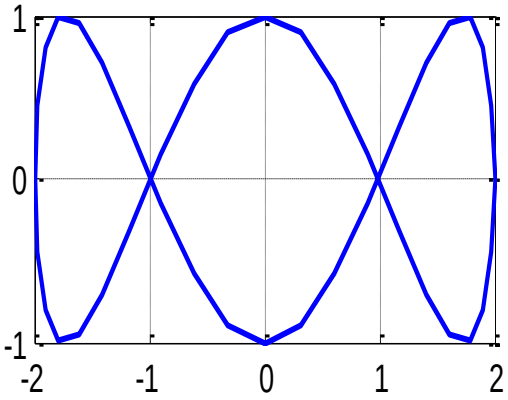
4. Subdivision de la fenêtre graphique

L'instruction *subplot(m, n, p)* divise la fenêtre graphique courante en *m*n* zones graphiques (*m* : lignes et *n* : colonnes) et trace le graphe qui suit cette instruction dans la zone de numéro *p* [1].

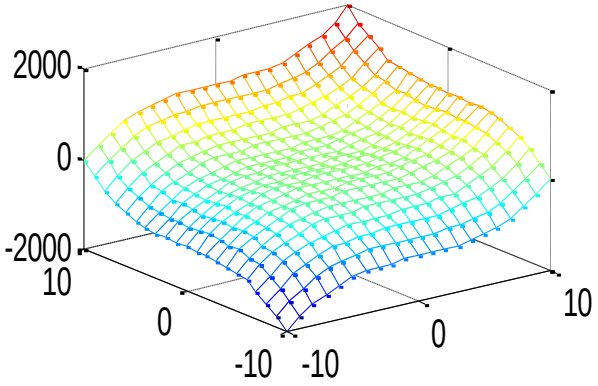
```
t = -pi : pi/20 : pi;
x = 2*cos(t);
y = sin(3*t);
subplot(3, 2, 1)
plot(x,y)
grid
title('Courbe paramétrique 2D')
subplot(3, 2, 2)
x = -10 : 10;
y = x;
[X, Y] = meshgrid(x, y);
Z = X.^3 + Y.^3;
mesh(X, Y, Z)
grid
title('Surface en fil de fer')
subplot(3, 2, 3)
t = -10 : 0.1 : 10;
x = 10*sin(t).*cos(t);
y = 10*sin(t).^2;
z = 10*cos(t);
plot3(x, y, z);
```

```
title('Courbe paramétrique 3D')
grid
subplot(3, 2, 4)
x = -5 : 5;
y = x;
[X, Y] = meshgrid(x, y);
Z = sqrt(X.^2 + Y.^2);
surfc(X, Y, Z)
grid
title('Surface avec contours')
t = -pi : pi/10 : pi;
r = sin(t) + 2;
N = 40;
subplot(3, 2, 5)
cylinder(r, N);
grid
title('Surface de révolution')
subplot(3, 2, 6)
X = randn(1000, 1);
hist(X, 30);
grid
title('Histogramme')
```

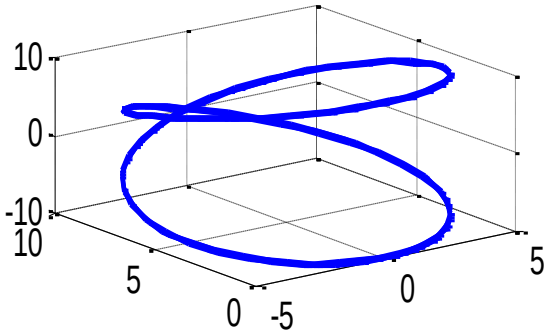
Courbe paramétrique 2D



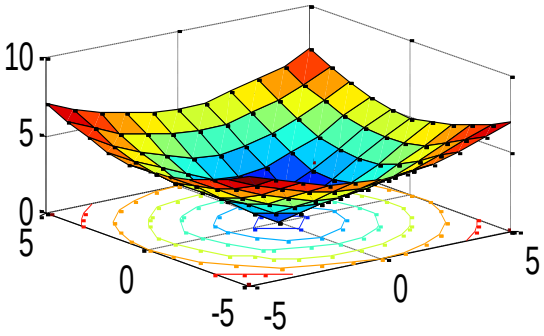
Surface en fil de fer



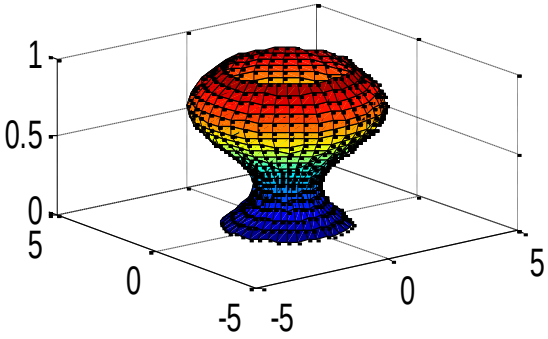
Courbe paramétrique 3D



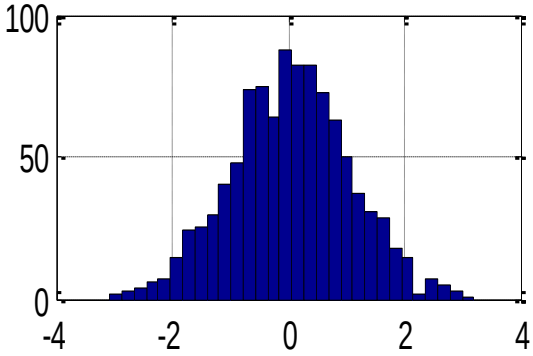
Surface avec contours



Surface de révolution



Histogramme



TP5 :

Initiation à la Programmation

TP 5 : Initiation à la Programmation

1. Introduction :

Quand un problème physique, mathématique, chimique, économique …etc, devient complexe, on a recours à la notion des Algorithmes [5].

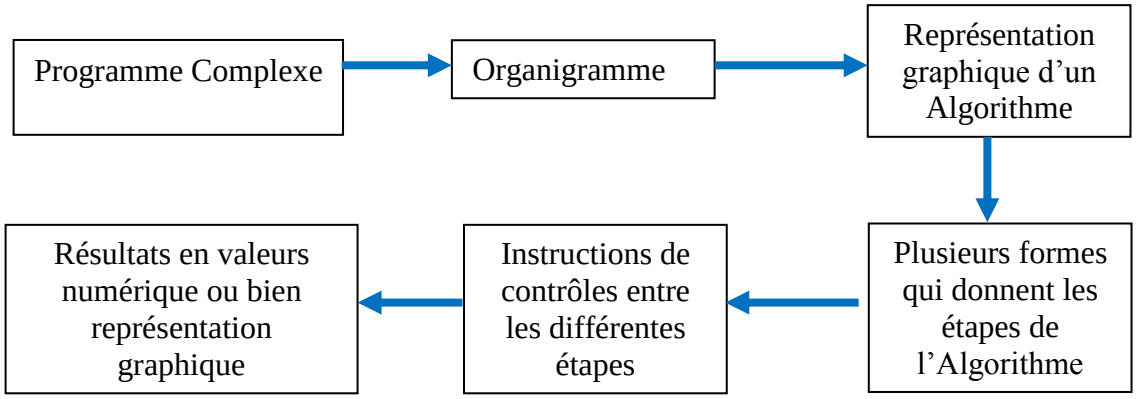
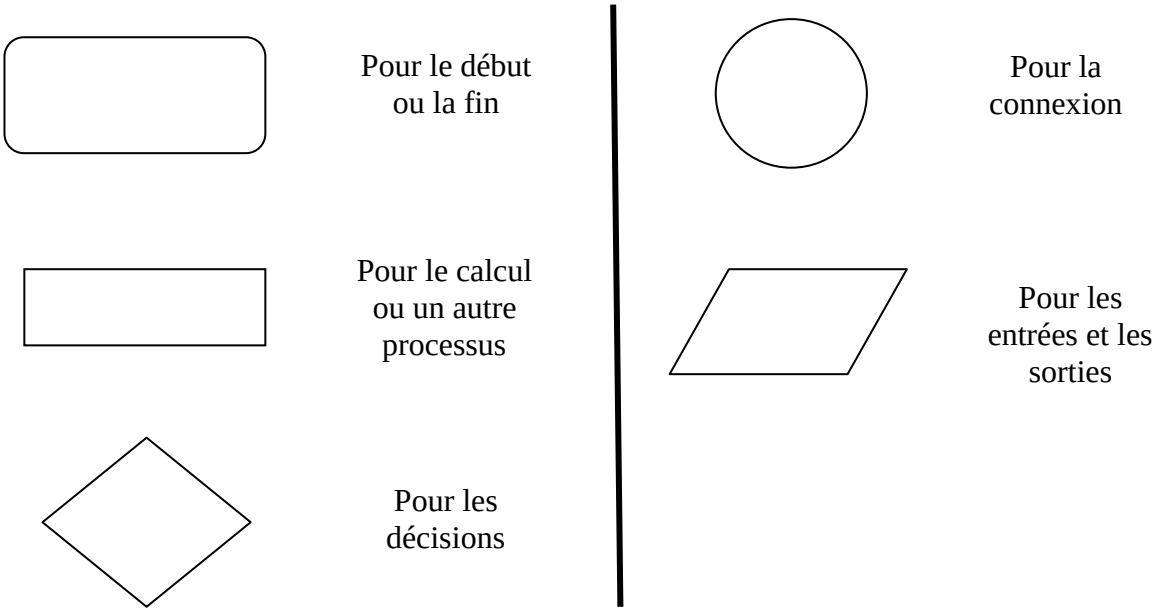


Figure 1 : Différentes étapes pour la résolution d'une problématique.

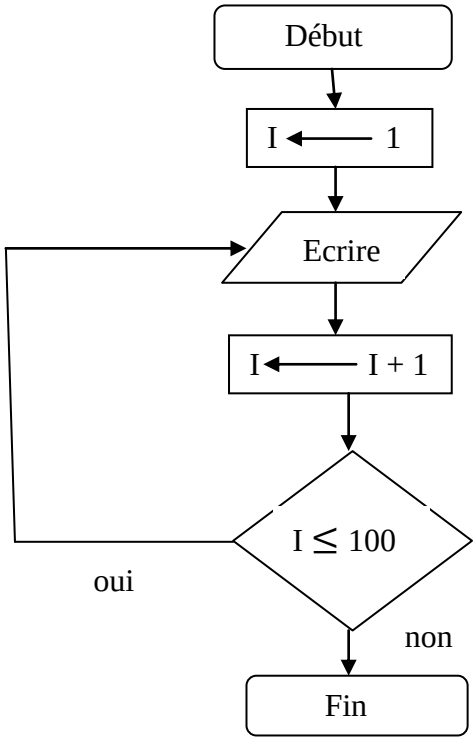
Remarque : Algorithme, c'est la résolution d'un problème étape par étape.

En particulier, dans les organigrammes on encadre chaque opération, instruction ou série d'instruction dans des cadres de différentes formes et on indique la direction du flux de contrôle par des lignes (direction en flèches) qui relient les cadres [5].

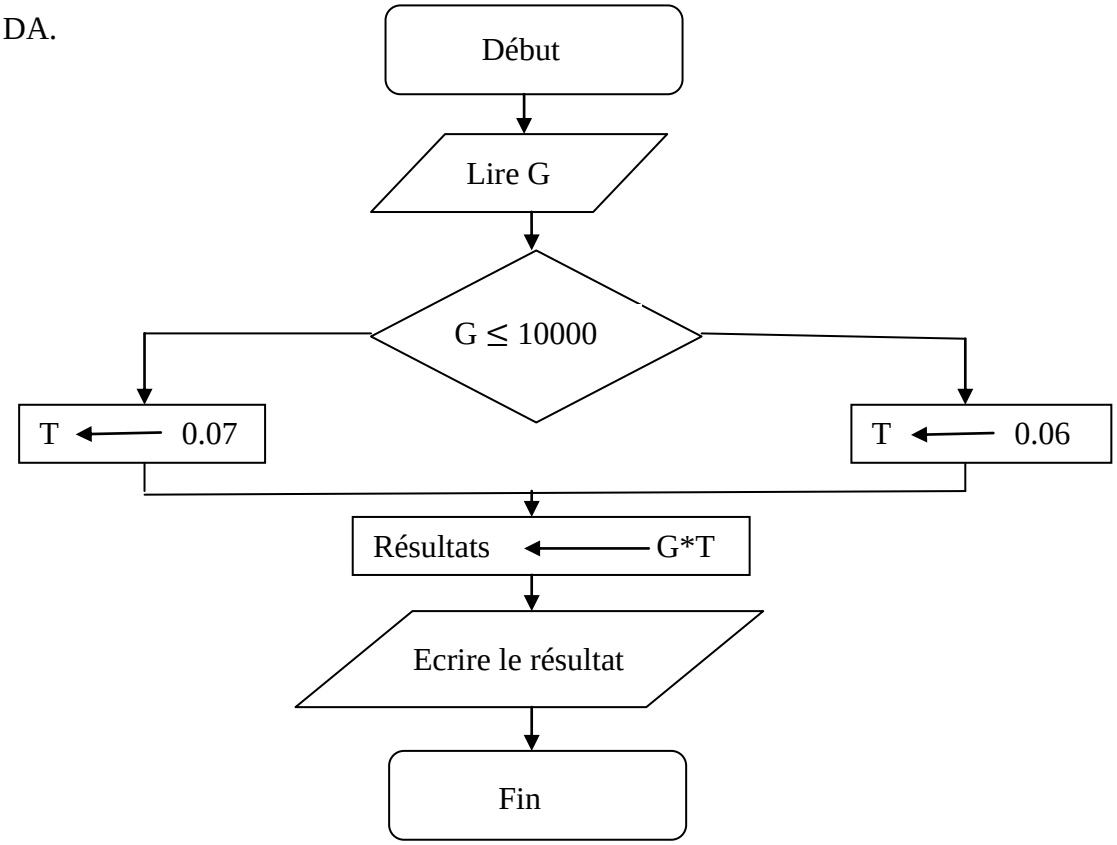
2. Les différentes formes des opérations algorithmiques :



Exemple : écrire un organigramme qui calcule et affiche la somme des entiers positifs inférieures à 100 ?

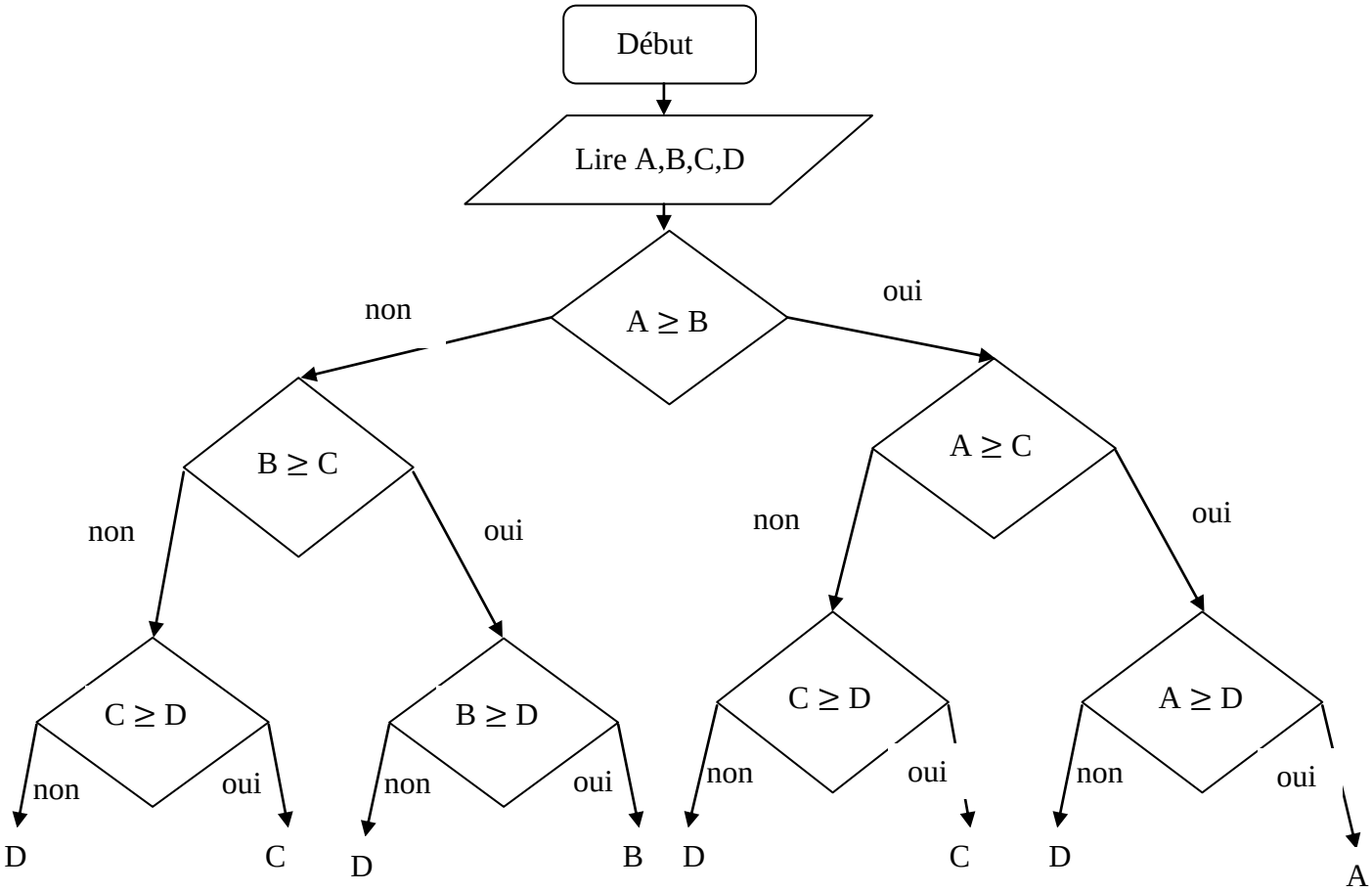


Exemple : écrire un organigramme qui donne le pourcentage du taux de bénéfice, (T) égale à 7% si le gain (G) inférieur ou égale à 10000 DA et 6% si le gain (G) supérieure ou égale à 10000 DA.



Exemple : donner le plus grand nombre entre quatre entiers A, B, C et D.

Remarque : on a $2^0 + 2^1 + 2^2 = 7$ boites de décision, en effet, le nombre de boites augmente avec le nombre des entiers à comparer.



Si on a 5 chiffres à comparer, $2^0 + 2^1 + 2^2 + 2^3 = 15$ boites de décisions, pour 6 chiffres : $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31$, le coup de calcul est très élevé pour un nombre considérable de chiffres.

Nombre de chiffre augmente ⟶ Le temps de calcul augmente

Exemple : donner le programme en MATLAB qui calcul la somme : $\frac{1}{1} + \frac{1}{3} + \frac{1}{5} + \cdots . . \frac{1}{21}$

Et le produit : $\frac{2}{1} * \frac{4}{3} * \frac{6}{5} * \ldots . . \frac{22}{21}$?

La Somme	Le Produit
Sum =0 , K = 1 if (K ≤ 21) Sum = Sum +1/K	Prod = 1, K= 1 if (K ≤ 21) Prod = Prod * (K+1)/K

K = K+2 end Sum	K = K +2 end Prod

Exemple: utiliser la boucle for pour calculer n !

```
n = 4
nfac = 1
for    k = 1 :n
      nfac = nfac*k ;
end
nfac
```

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

Exemple : afficher la fonction sin(x) sur l'intervalle [−π, π].

```
>> x = [−pi : 0 . 1 : pi ] ;
>> y = sin ( x ) ;
>> plot ( x , y ) ;
```

3. Systèmes d'équations non linéaires :

On se propose de travailler pour la recherche des racines de la fonction non linéaire suivante [4] :

$$f(x) = e^x - 2 * \cos(x)$$

Dans MATLAB, on peut obtenir la même solution avec la fonction '*fzero*'. Pour cela, il faut créer le *fichier.m* MATLAB ('*f.m*', par exemple) dans lequel sera programmé **f(x)**.

```
% ******************************
function f=f(x)
f=exp(x)-2*cos(x)
% ******************************
```

Pour obtenir la solution de **f(x) = 0** au voisinage de **x = 0,5**, on exécute la commande '*fzéro('f',0.5)*'. Cette fonction affiche les valeurs obtenues de **f(x)** à chaque itération pour donner à la fin la solution recherchée **x***.

>>d=fzero('f',0.5)

f =

-8.8818e-016

d =

0.5398

La boite à outil '*Optimisation Toolbox*' propose les fonctions '***fsolve***' et '***fsolve2***' qui permettent respectivement, la recherche des racines d'une fonction dans un intervalle donné et les solutions d'équations non linéaires et de systèmes d'équations non linéaires [4].

Nous allons appliquer ces fonctions à la résolution d'un système de deux équations non linéaires à 2 inconnues, par exemple :

$$2 * x_1 - x_2 - e^{-x_1} = 0$$
$$-x_1 + 2 * x_2 - e^{-x_2} = 0$$

à partir de l'itération x0 = [-5 -5].

Il faut, d'abord, créer le fichier *myfun.m* MATLAB

```
% ******************************
function F = myfun(x)
F = [2*x(1) - x(2) - exp(-x(1));
    -x(1) + 2*x(2) - exp(-x(2))];
% ******************************
```

Après, on fait appel à la routine suivante.

```
>> x0 = [-5; -5];
>> [x,fval] = fsolve(@myfun, x0)
```

Après 33 évaluations de la fonction *myfun*, un zéro est trouvé.

```
x =
0.5671
0.5671
```

TP6 :

Applications et Exercices

TP6 : Applications et Exercices

Question 1 :

a. Affecter à $[A]$, $[B]$, $[C]$, et $[D]$ les matrices suivantes :

$$[A] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad [B] = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \\ 7 & 8 & 9 \end{bmatrix}$$

$$[C] = \begin{bmatrix} -9 \\ 2 \\ -1 \end{bmatrix} \qquad [D] = [7 \quad -2 \quad 11]$$

b. A partir des matrices $[A]$, $[B]$, $[C]$, et $[D]$ calculer les expressions suivantes :

$$[E] = [A] * [B] * [C]$$
$$[F] = [D] * [C]$$
$$[L] = [C] * [D]$$

Question 2 :

Calculez la transposé de $[A]$ et de $[L]$, et mettez à zéro l'élément (2,2) de la matrice $[H] = [B] + [L]$?

Question 3 :

1. Calculer la matrice produit $[K] = [B]' * [L]$
2. Calculer la matrice somme $[M] = [A] + [B]$, pourquoi le résultat donne une erreur ?
3. Créer une matrice nulle d'ordre (4×4) et une matrice d'ordre (3×3) constituée de 1 ?

Question 4 :

Soient deux fonctions, $y = 1.5 * (2 - \cos(x))$ et $z = \sin(\mathrm{x}) + 2\cos(\mathrm{x}/3)$, avec x est compris entre -20 et 20.
Sur le même graphe, tracer y et z en fonction de x en distinguant les deux courbes par des styles différents et attribuer au graphe un titre et une légende ?

Question 5 :

Sur un M-file, taper puis exécuter le programme suivant, donner une explication pour chaque ligne de ce programme ?

```
vect=[];
i=1;
while (length(vect)<10)
   i=i+1;
   if  (rem(i,5)==0)
      vect = [vect i];
   end
end
```

Exemple 1 :(solutions du système d'équations non linéaires**)** [4]

Ci-dessous, nous éditons un fichier (m) dans lequel on cherchera graphiquement les solutions du système d'équations non linéaires suivantes :

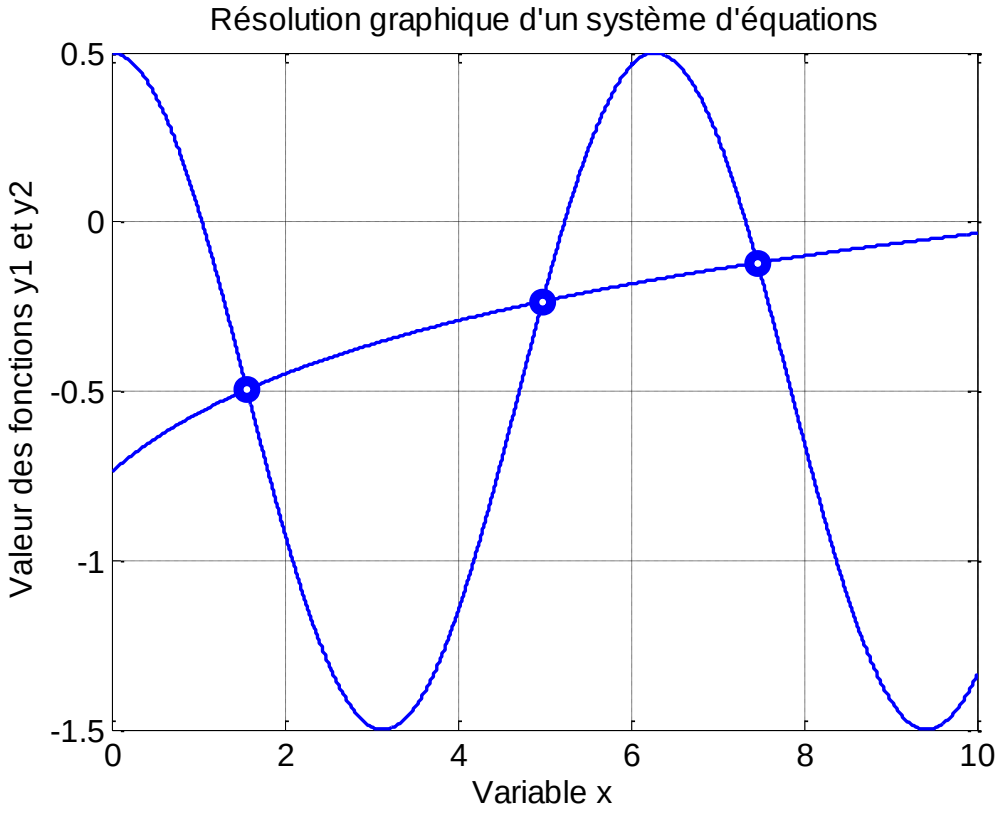
$$\begin{cases} -\cos(x) + y = -0.5 \\ \ln(x+1) - 3\arcsin(y) = 2.5 \end{cases}$$

Pour cela on tracera 2 courbes y_1 et y_2 en fonction de la même variable x, extraites des 2 équations du système.

$$\begin{cases} y1 = \cos(x) - 0.5 \\ y2 = \sin((\ln(x+1) - 2.5)/3) \end{cases}$$

Le fichier *sol_system.m* utilise un domaine allant de 0 à 10 par pas de 0.001 pour la variable x. Le pas doit être assez petit pour avoir une plus grande probabilité de trouver les solutions.

```
% Définition du domaine de valeurs de la variable x
x=0:0.001:10;
% Expressions des 2 fonctions y1 et y2
y1=cos(x)-0.5;
y2=sin((-2.5+log(x+1))/3);
% Affichage des 2 courbes des 2 fonctions y1 et y2
plot(x,y1)
% On garde la même fenêtre pour tracer y2
hold on
% Tracé de y2
plot(x,y2)
grid % insertion d'une grille
% On cherche les indices des valeurs x, y1 ou y2
% qui satisfont l'égalité de y1 et de y2 avec une
% précision de 0.0005
i  = find(abs(y1-y2)<0.0005);
% affichage des indices des solutions
disp('Indices des solutions :')
i
% Affichage des solutions
disp('Valeurs des solutions :')
sol=x(i)
% Tracé des points de rencontres des 2 courbes
% aux valeurs de ces indices
h=plot(x(i),y1(i),'*');
set(h,'Linewidth',5)
% Légendes des axes
xlabel('Variable x')
ylabel('Valeur des fonctions y1 et y2')
title('Résolution graphique d''un système d''équations')
```




```
Indices des solutions :
i =       1568       4982       7466
Valeurs des solutions :
sol =   1.5670    4.9810    7.4650
```

Exemple 2 : (calcule la factorielle) [5]

Une fonction est un fichier qui possède des paramètres d'appel et des paramètres de retour, l'utilisateur peut étendre les possibilités du langage à son domaine particulier en créant ses propres fonctions. Comme exemple de fonction, nous allons créer une fonction qui calcule la factorielle d'un nombre en utilisant la fonction prod que nous avons étudiée dans le chapitre sue les vecteurs. (factorielle.m)

```
function    fact = factorielle (n)
% Cette fonction calcule la factorielle
% d'un nombre entier n
% Test de la nature entière et positive
% de l'argument d'entrée
if   ~ ((fix(n)==n) & (n >= 0))
   error('Le nombre doit être entier positif')
end
% Calcul de la factorielle par la commande prod
fact  =  prod(1:n);
```

```
>> factorielle(5)
ans =
   120
```

```
>> factorielle(7)
ans =
   5040
```

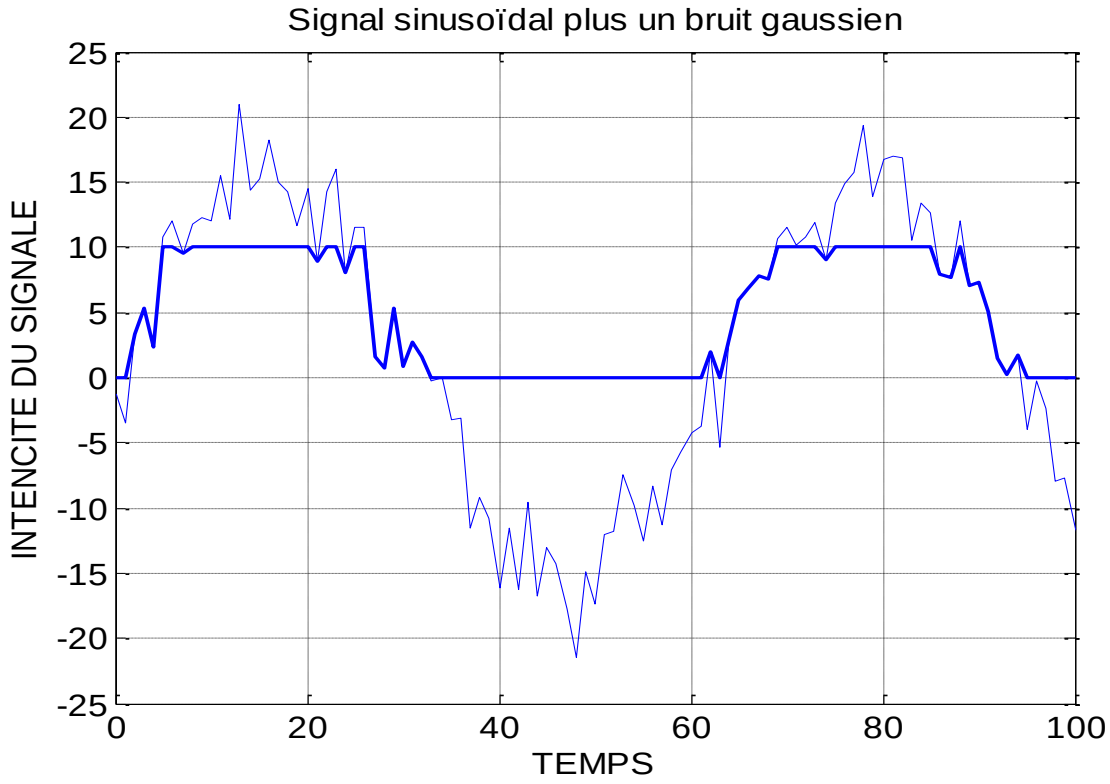
Exemple 3 : (analyse d'un signal sinusoïdal) [5]

L'exemple suivant permettra de limiter un signal u(t) variant dans le temps, entre les valeurs u_min et u_max en utilisant les opérateurs relationnels et logiques. (sature.m)

Nous allons appliquer la fonction sature à un signal sinusoïdal auquel on a ajouté un bruit gaussien.

```
function     u_limite = sature(u, u_min, u_max)
% Limitation d'un signal
% Si signal < u_min alors signal = u_min,
% Si signal > u_max alors signal = u_max,
% Si u_min <= signal <= u_max alors signal non modifié
% Expressions logiques retournant 2 ou 1
expr1 = (u >= u_max);
expr2 = (u <= u_max);
expr3 = ((u < u_max) & (u > u_min));
u_limite = expr1.* u_max + expr2.* u_min + expr3.*u;
```

```
close   all
clear   all
u_min=0;     % Les limites du signal
u_max=10;
% Signal sinusoïdal bruité
t=0:100;
bruit = 3 * randn(size(t));
u = 15 *sin(0.1*t)+bruit;
% La saturation
sgn=sature(u,u_min,u_max);
% Signal non saturé
figure(1)
plot(t,u);
% Signal saturé
hold on
h=plot(t,sgn);
set(h,'LineWidth',2)
```



Annexe

Annexe

1. Liste de quelques fonctions Matlab

Cette liste regroupe les commandes et fonctions les plus usuelles. Elle illustre également la richesse des possibilités offertes. Pour plus de détails, il faut évidemment consulter l'aide en linge et profiter des nombreux exemples qui sont proposés (*help Matlab*):

1.1. Information sur l'espace de travail

Nom de la commande	*Fonction de la commande*
who	Affiche les variables courantes
whos	Affiche les variables courantes avec leurs dimensions
save	Sauve l'espace de travail sur disque
load	Restaure l'espace de travail à partir du disque
clear	Efface les variables et fonctions de la mémoire
close	Ferme la fenêtre courante
pack	Réorganise la mémoire
size	Renvoie la taille d'une matrice
length	Renvoie la longueur d'un vecteur
disp	Affiche une matrice de texte
clc	Efface le contenu de la fenêtre de commandes
format	Définit le format d'affichage

1.2. Caractères spéciaux

Nom de la commande	*Fonction de la commande*
[ ]	Définition de matrices ou vecteurs ; enserre les arguments de sortie des fonctions
( )	Gère la priorité des opérations ; enserre les arguments d'entrée des fonctions
.	Point décimal
..	répertoire parent
…	Indique une ligne suite
,	Séparateur d'arguments ou d'instructions
;	Fin de lignes (matrices) ou suppression de l'affichage
%	Commentaires
:	Manipulation de sous matrices ou génération de vecteurs

1.3. Opérateurs logiques

Nom de la commande	*Fonction de la commande*
<	Inférieur à
&	Et
>	Supérieur `à
|	Ou
<=	Inférieur ou égal à
~	Non
>=	Supérieur ou égal à
xor	Ou exclusif
==	Egal à
~=	Différent de
=	Assignation

1.4. Variables prédéfinies, durée et date

Nom de la commande	*Fonction de la commande*
ans	Réponse à une expression sans assignation
eps	Précision de la virgule flottante
realmax	Plus grand nombre flottant
realmin	Plus petit nombre flottant positif
pi	π
i, j	$\sqrt{-1}$
Inf	∞
NaN	Not a Number
flops	Nombre d'opérations flottantes par seconde
nargin	Nombre d'arguments d'entrée d'une fonction
nargout	Nombre d'arguments de sortie d'une fonction
date	Date courante
clock	Horloge
etime	Durée d'exécution
cputime	Temps CPU de calcul écoulé

1.5. Instructions de contrôle

Nom de la commande	*Fonction de la commande*
if, else, elseif	Teste conditionnel
for	Instruction de répétition avec compteur
while	Instruction de répétition avec test
end	Terminaison de if, for et while
break	Interrompt une boucle
return	Retour
error	Affiche un message et interrompt l'exécution

1.6. Fonctions mathématiques

1.6.1. Fonctions élémentaires

Nom de la commande	*Fonction de la commande*
abs	Valeur absolue ou module d'une grandeur complexe
angle	Argument d'une grandeur complexe
sqrt	Racine carrée
real	Partie réelle
imag	Partie imaginaire
conj	Complexe conjugué
rem	Reste de la division
exp	Exponentielle
log	Log népérien
log10	Log décimal
log2	Log base 2
gcd	PGCD
lcm	PPCM

1.6.2. Fonctions trigonométriques

Nom de la commande	*Fonction de la commande*
sin, asin, sinh, asinh	sinus, arc-sinus, sinus hyperbolique, arc-sinus hyperbolique
cos, acos, cosh, acosh	cosinus, arc-cosinus, cosinus hyperbolique, arc-cosinus hyperbolique
tan, atan, tanh, atanh	tangent, arc-tangent, tangent hyperbolique, arc-tangent hyperbolique
cot, acot, coth, acoth	cotangent, arc-cotangent, cotangent hyperbolique, arc-cotangent hyperbolique

1.7. Matrices et algèbre linéaire

1.7.1. Opérateurs sur les matrices

Nom de la commande	*Fonction de la commande*
+, -, *, ^	Addition, Soustraction, Multiplication, Puissance
**/, **	Division à droite, Division à gauche
'	Transposition conjuguée

1.7.2. Opérateurs sur les composantes matricielles

Nom de la commande	*Fonction de la commande*
+, -, .*, .^	Addition, Soustraction, Multiplication, Puissance
./, .	Division à droite, Division à gauche
.'	Transposition

1.7.3. Manipulation des matrices

Nom de la commande	*Fonction de la commande*
diag	Création ou extraction de la diagonale
rot90	Rotation de 90 degrés
fliplr	Retournement gauche-droit
flipud	Retournement haut-bas
reshape	Redimensionnement
tril	Partie triangulaire inférieure
triu	Partie triangulaire supérieure
.'	Transposition
:	Conversion matrice > vecteur

1.7.4. Matrices prédéfinies

Nom de la commande	*Fonction de la commande*
zeros	Matrice de 0
ones	Matrice de 1
eye	Matrice identité
diag	Matrice diagonale
magic	Carré magique
linspace	Vecteurs à composantes linéairement espacées
logspace	Vecteurs à composantes logarithmiquement espacées
meshgrid	Grille pour les graphiques 3D
rand	Nombres aléatoires à répartition uniforme
randn	Nombres aléatoires à répartition normale
pascal	Pascal

1.7.5. Décomposition et factorisation de matrices

Nom de la commande	*Fonction de la commande*
inv	Inversion d'une matrice carrée
lu	Décomposition *LU*
chol	Factorisation de *Cholesky*
qr	Décomposition *QR*
nuls	Moindres carrés non-négatif
orth	Orthogonalisation
eig	Valeurs et vecteurs propres
svd	Décomposition en valeurs singulières
pinv	Pseudo-inverse

1.7.6. Opérations sur les matrices

Nom de la commande	*Fonction de la commande*
poly	Polynôme caractéristique
det	Déterminant d'une matrice carrée
eig	Valeurs propres d'une matrice carrée
trace	Trace

1.8. Textes et chaînes de caractères

Nom de la commande	*Fonction de la commande*
num2str	Convertit un nombre en chaîne
int2str	Convertit un nombre entier en chaîne
str2num	Convertit une chaîne en nombre
eval	Convertit un texte en code Matlab
strcmp	Comparaison de chaînes
upper	Conversion en majuscule
lower	Conversion en minuscule
hex2num	Convertit une chaîne hexadécimale en flottant
hex2dec	Convertit une chaîne hexadécimale en entier
dec2hex	Convertit un entier en une chaîne hexadécimale

1.9. Fonctions graphiques

1.9.1. Graphiques 2D

Nom de la commande	*Fonction de la commande*
plot	Dessine le graphe d'une fonction
loglog	Graphe en échelle log-log
semilogx	Graphe en échelle semi-log (abscisse)
semilogy	Graphe en échelle semi-log (ordonnée)
polar	Graphe en coordonnées polaires
bar	Graphe en barres
stairs	Graphe en marches d'escalier
stem	Graphe de raies
hist	Histogramme
rose	Histogramme en coordonnées polaires
compass	Représentation de vecteurs à partir de l'origine
feather	Représentation de vecteurs sur un axe linéaire

1.9.2. Annotation de graphiques

Nom de la commande	*Fonction de la commande*
title	Titre du graphique
xlabel	Légende pour l'abscisse
ylabel	Légende pour l'ordonnée
zlabel	Légende pour la cote
grid	Dessin d'une grille
text	Texte
gtext	Placement de texte avec la souris
ginput	Entrée graphique par la souris

1.9.3. Objets 3D

Nom de la commande	*Fonction de la commande*
sphere	Génération de sphères
cylinder	Génération de cylindres

1.9.4. Graphiques tridimensionnels

Nom de la commande	*Fonction de la commande*
mesh	Surface maillée
meshc	Combinaison mesh + dessin des équi-niveaux
meshz	Surface maillée avec plan de référence
surf	Surface 3D à facettes
surfc	Combinaison surf + dessin des équi-niveaux
plot3	Dessin de lignes et points en 3D
contour	Dessin 2D des équi-niveaux
contourc	Calcul des valeurs utilisées par contour
contour3	Dessin 3D des équi-niveaux
pcolor	Dessine en pseudo couleur
image	Affiche une image

1.10. Analyse de données et statistiques

Nom de la commande	*Fonction de la commande*
max	Valeur max d'un vecteur ou d'une matrice
min	Valeur min d'un vecteur ou d'une matrice
mean	Valeur moyenne d'un vecteur ou d'une matrice
median	Valeur médiane d'un vecteur ou d'une matrice
std	Ecart type
sort	Tri en ordre croissant
sum	Somme des éléments d'un vecteur ou d'une matrice
prod	Produit des éléments d'un vecteur ou d'une matrice
cumsum	Vecteur des sommes partielles cumulées
cumprod	Vecteur des produits partiels cumulés
hist	Histogramme

Références bibliographiques

Références bibliographiques

1- Apprendre à maitriser MTLAB. Sous-titre, version 4 et 5 et SIMULINK. Auteurs, Mohand Mokhtari, Abdelhalim Mesbah.

2- Introduction à MATLAB. Technique de l'ingénieur, 1998.

3- MATLAB pour les ingénieurs : Version 6 et 7. Biran Adrian, Techno-Sciences.

4 Initiation sur MatLab, Par Rafic YOUNES Enseignant – Chercheur à la Faculté de Génie – Université Libanaise

5 Numerical Methods, Auteurs J. Douglas Faires et Richard L. Burden, 3 Edition 2002.