R é p u b l i q u e A l g é r i e n n e D é m o c r a t i q u e e t P o p u l a i r e
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf

Faculté d'Architecture et de Génie Civil
Département de Génie Civil

Polycopié

Le calcul scientifique appliqué au Génie Civil sous MATLAB

Préparé et présenté par

BABA HAMED Fatima Zohra

Préface

Durant les trois dernières décennies, des progrès remarquables ont eu lieu dans la vitesse de traitement des ordinateurs, leur capacité de stockage, de manipulation et de présentation de grandes quantités de données et d'informations, ainsi que dans la capacité des ordinateurs à communiquer avec d'autres ordinateurs sur les réseaux.
La preuve de ces progrès peut être trouvée dans les bureaux actuels d'ingénierie où les puissances des ordinateurs utilisés sont de plus en plus importantes.
Le nombre de codes de calcul utilisés par les ingénieurs est en constante augmentation. Cette tendance est due en partie à la série croissante des tâches pour lesquelles les ingénieurs utilisent maintenant des ordinateurs.
Étant donné que les langages informatiques sont généralement conçus pour résoudre un certain nombre de problèmes de génie, le choix du bon langage pour accomplir la tâche est d'une importance capitale.

L'utilisation de **MATLAB** est avantageuse pour résoudre un problème qui peut être idéalement représenté par des matrices, en se servant des opérations de l'algèbre matricielle linéaire et de la représentation relativement simple des graphiques en deux et trois dimensions.
Le calcul de la solution d'équations linéaires, la présentation, la manipulation et l'affichage des données d'ingénierie sont peut-être les meilleurs exemples de problèmes pour lesquels **MATLAB** est idéalement adapté.

Ce polycopié s'adresse aux étudiants de première année Master en Génie Civil, option voie et ouvrage d'art. Il est rédigé de manière à attirer l'attention du lecteur sur les applications pratiques du sujet traité.
Non seulement le langage de programmation **MATLAB** est exceptionnellement simple à utiliser (tous les objets de données sont supposés être des tableaux), le code de programme MATLAB sera beaucoup plus court et plus simple dans la mise en œuvre d'un programme équivalent en langage C, Fortran ou Java. **MATLAB** est donc un langage idéal pour la création de prototypes de solutions logicielles à des problèmes d'ingénierie.

Le polycopié est divisé en cinq chapitres. Le chapitre 1 : Introduction à l'environnement **MATLAB** présente le mode de programmation sous **MATLAB**. Il inclut tous les concepts de base que l'étudiant doit savoir. Les sujets traités comprennent les variables, les variables arithmétiques, les matrices, les matrices de calcul, les structures de contrôle, les fonctions intégrées de la matrice, M-Files, et bien d'autres notions.

Le chapitre 2 : Introduction à la programmation avec **MATLAB** traite l'aspect de la programmation avec **MATLAB**. Il présente les différents mécanismes d'écriture et d'exécution des programmes.

Le chapitre 3 : Applications des méthodes numériques avec **MATLAB**, s'intéresse à l'application des méthodes numériques sous **MATLAB**. Ce chapitre illustre la puissance de travail du **MATLAB** étape par étape à travers la formulation et la solution d'un grand nombre d'applications d'ingénierie impliquant la solution des équations linéaires et non linéaires ainsi que le calcul intégral.

Le chapitre 4 : Calcul des structures (barres et poutres) selon la méthode des Eléments Finis par MATLAB est une introduction au calcul des structures selon la méthode des Eléments Finis (MEF) par **MATLAB**. Les calculs des éléments barres, des systèmes treillis et des éléments poutres sont effectués.

Le 5ème et dernier chapitre : Calcul des structures sous effets dynamique et sismique par MATLAB explique le calcul des structures sous effets dynamiques et sismiques par **MATLAB**. Le lecteur pourra ainsi réaliser le calcul des systèmes à 1 degré de liberté (libre et amortis) ainsi que le calcul des systèmes à plusieurs degrés de liberté.

Tables des matières

.

Chapitre I : Introduction à l'environnement MATLAB

1. Introduction - Historique

MATLAB est une abréviation de Matrix LABoratory, écrit à l'origine, en Fortran, par C. Moler.

MATLAB était destiné à faciliter l'accès au logiciel matriciel développé dans les projets LINPACK et EISPACK. La version actuelle, écrite en C par the MathWorks Inc., existe en version professionnelle et en version étudiant. Sa disponibilité est assurée sur plusieurs plateformes : Sun, Bull, HP, IBM, compatibles PC (DOS, Unix ou Windows), Macintoch, iMac et plusieurs machines parallèles.

MATLAB est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant calcul numérique et visualisation. C'est un environnement performant, ouvert et programmable qui permet de remarquables gains de productivité et de créativité.

MATLAB est un environnement complet, ouvert et extensible pour le calcul et la visualisation. Il dispose de plusieurs centaines (voire milliers, selon les versions et les modules optionnels autour du noyau Matlab) de fonctions mathématiques, scientifiques et techniques. L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numérique et symbolique de façon fiable et rapide. Grâce aux fonctions graphiques de MATLAB, il devient très facile de modifier interactivement les différents paramètres des graphiques pour les adapter selon nos souhaits.

L'approche ouverte de MATLAB permet de construire un outil sur mesure. On peut inspecter le code source et les algorithmes des bibliothèques de fonctions (Toolboxes), modifier des fonctions existantes et ajouter d'autres.

MATLAB possède son propre langage, intuitif et naturel qui permet des gains de temps de CPU spectaculaires par rapport à des langages comme le C, le TurboPascal et le Fortran. Avec MATLAB, on peut faire des liaisons de façon dynamique, à des programmes C ou Fortran, échanger des données avec d'autres applications (via la DDE Dynamic Data Exchange : MATLAB serveur ou client) ou utiliser MATLAB comme moteur d'analyse et de visualisation.

MATLAB comprend aussi un ensemble d'outils spécifiques à des domaines, appelés Toolboxes (ou Boîtes à Outils). Indispensables à la plupart des utilisateurs, les Boîtes à Outils sont des collections de fonctions qui étendent l'environnement MATLAB pour résoudre des catégories spécifiques de problèmes.

MATLAB permet le travail interactif soit en mode commande, soit en mode programmation ; tout en ayant toujours la possibilité de faire des visualisations graphiques. Considéré comme l'un des meilleurs langages de programmation, MATLAB possède les particularités suivantes par rapport à ces langages :

• la programmation facile,
• la continuité parmi les valeurs entières, réelles et complexes,
• la gamme étendue des nombres et leur précision,
• la bibliothèque mathématique très compréhensive,
• l'outil graphique qui inclut les fonctions d'interface graphique et les utilitaires,
• la possibilité de liaison avec les autres langages classiques de programmation (C ou Fortran).

La bibliothèque des fonctions mathématiques dans MATLAB donne des analyses mathématiques très simples. En effet, l'utilisateur peut exécuter dans le mode commande n'importe quelle fonction mathématique se trouvant dans la bibliothèque sans avoir à recourir à la programmation.

Pour l'interface graphique, des représentations scientifiques et même artistiques des objets peuvent être créées sur l'écran en utilisant les expressions mathématiques. Les graphiques sur MATLAB sont simples et attirent l'attention des utilisateurs, vu les possibilités importantes offertes par ce logiciel.

MATLAB n'est pas le seul environnement de calcul scientifique, il existe d'autres concurrents dont les plus importants sont Maple et Mathematica. Il existe même des logiciels libres qui sont des clones de MATLAB comme Scilab et Octave.

2. L'environnement MATLAB

MATLAB affiche au démarrage plusieurs fenêtres. Selon la version on peut trouver les fenêtres suivantes :

- **Current Folder:** indique le répertoire courant ainsi que les fichiers existants.
- **Workspace:** indique toutes les variables existantes avec leurs types et valeurs.
- **Command History**: garde la trace de toutes les commandes entrées par l'utilisateur.
- **Command Window**: utilisée pour formuler nos expressions et interagir avec MATLAB. C'est la fenêtre que nous utilisons tout au long de ce chapitre.

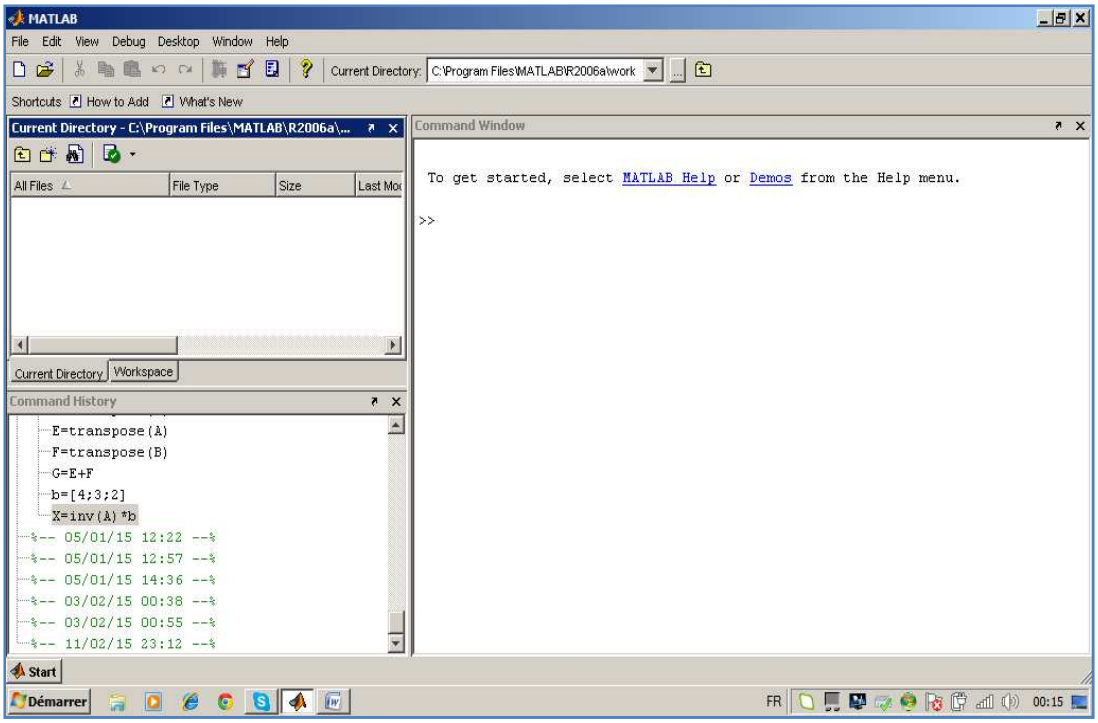


Figure I.1 : L'environnement **MATLAB**

MATLAB est beaucoup plus qu'un langage de programmation. Il s'agit d'une console d'exécution (**shell**) permettant d'exécuter des fonctions, d'attribuer des valeurs à des variables, etc. La console MATLAB permet d'effectuer des opérations mathématiques, de manipuler des matrices et de tracer facilement des graphiques.

Le langage MATLAB n'est pas un langage compilé, à chaque appel d'un SCRIPT (ou d'une FUNCTION), le logiciel lit et exécute les programmes ligne par ligne.

L'utilisateur peut grâce à l'invite MATLAB affecter des valeurs à des variables et effectuer des opérations sur ces variables. Par exemple :

```
>> x = 4
x =
4
>> y = 2
y =
2
>> x + y
ans =
6
>> x * y
ans =
8
```

Ici, il faut noter que lorsque l'utilisateur ne fixe pas de variable de sortie, MATLAB place le résultat d'une opération dans **ans**. Il est toujours possible de connaître les variables utilisées et leur type à l'aide de la fonction **whos**. Par exemple, pour les manipulations précédentes :

```
>>whos
Name Size Bytes Class
ans 1x1 8 double array
x 1x1 8 double array
y 1x1 8 double array
Grand total is 3 elements using
24 bytes
```

La solution de x+y est donc perdue. Il est donc préférable de toujours donner des noms aux variables de sortie :

```
>> x = 4;
>> y = 2;
>> a = x + y
a =
        6
>> b = x * y
b =
        8
```

La fonction **clear** permet d'effacer des variables. Par exemple :

```
>> clear x % on efface x de la
mémoire
>> whos
Name Size Bytes Class
a 1x1 8 double array
b 1x1 8 double array
y 1x1 8 double array
Grand total is 3 elements using 24
bytes
```

Le signe de pourcentage **%** permet de mettre ce qui suit sur une ligne en commentaire (MATLAB n'en tiendra pas compte à l'exécution).

Pour les variables de type caractère : **char**, la déclaration se fait entre apostrophes. Il est possible de concaténer (lier) des mots à l'aide des crochets.

```
>> mot1 = 'Génie'
>> mot2 = 'Civil';
>> mot1_2 = [mot1 ' ' mot2] % l'emploi de ' '
permet d'introduire un espace
mot1_2 =
Génie Civil
```

3. Les fichiers SCRIPT et FUNCTION.

Pour des tâches répétitives, il est pratique et judicieux d'écrire de courts programmes, qu'on sauvegarde, pour effectuer les calculs désirés. Il existe deux types de fichiers qui peuvent être programmés avec MATLAB : les fichiers **SCRIPT** et **FUNCTION**. Dans les deux cas, il faut lancer l'éditeur de fichier et sauvegarder le fichier avec l'extension **.m**.

Le fichier **SCRIPT** permet de lancer les mêmes opérations que celles écrites directement à l'invite MATLAB. Toutes les variables utilisées dans un SCRIPT sont disponibles à l'invite MATLAB.

Cette approche est définie en Matlab par les M-Files, qui sont des fichiers pouvant contenir les données, les programmes (scripts) ou les fonctions que nous développons.

Pour créer un M-Files il suffit de taper la commande **edit**, ou tout simplement aller dans le menu : File → New → M-Files (ou cliquer sur l'icône).

Une fenêtre d'édition comme celle- ci va apparaître :

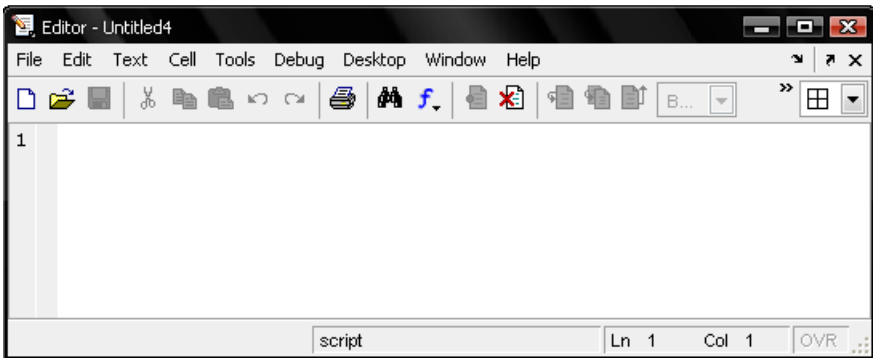


Figure I.2 : La fenêtre d'édition **MATLAB**

Par exemple, le fichier test.m qui reprend l'exemple précédent:

```
% test.m
clear all
x = 4;
y = 2;
a = x + y
b = x * y
whos
```

MATLAB contient un grand nombre de fonctions prédéfinies comme **sin**, **cos**, **sqrt**, **sum**, etc..
Il est possible de créer nos propres fonctions en écrivant leurs codes « source » dans des fichiers M-Files (portant le même nom de fonction) en respectant la syntaxe suivante :

```
function [r1, r2, …, rn] = nom_fonction (arg1, arg2, …, argn)

      % le corps de la fonction
      . . .
      r1 = . . .  % la valeur retournée pour r1
      r2 = . . .  % la valeur retournée pour r2
      . . .
      rn = . . .  % la valeur retournée pour rn
end               % le end est facultatif
```

Où : $\mathbf{r_1...r_n}$ sont les valeurs retournées, et $\mathbf{arg_1...arg_n}$ sont les arguments.
Le rôle d'une **fonction** est d'effectuer des opérations sur une ou plusieurs entrées pour obtenir un résultat qui sera appelé sortie.

Par exemple :

```
function [a,b] = ma_function(x,y)
a = x + y;
b = x * y;
```

On obtient

```
>> [a,b] = ma_fonction(4,2)
a =
        6
b =
        8

>> whos

Name Size Bytes Class
a 1x1 8 double array
b 1x1 8 double array
Grand total is 2 elements using 16 bytes
```

4. Les principales constantes, fonctions et commandes

MATLAB définit les constantes suivantes :

La constante	Sa valeur
pi	π=3.1415...
exp(1)	e=2.7183...
i	$= \sqrt{-1}$
j	$= \sqrt{-1}$
Inf	∞
NaN	Not a Number (Pas un nombre)
eps	$\varepsilon \approx 2 \times 10^{-16}$.

Parmi les fonctions fréquemment utilisées, on peut noter les suivantes :

La fonction	Sa signification
sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian)
sqrt(x)	la racine carrée de x ➔ $\sqrt{x}$

abs(x)	la valeur absolue de x ➔ |x|
exp(x)	= e^x
log(x)	logarithme naturel de x ➔ ln(x)=$\log_e(x)$
log10(x)	logarithme à base 10 de x ➔ $\log_{10}(x)$
imag(x)	la partie imaginaire du nombre complexe x
real(x)	la partie réelle du nombre complexe x
round(x)	arrondi un nombre vers l'entier le plus proche
floor(x)	arrondi un nombre vers l'entier le plus petit ➔ max{n|n≤x, n entier}
ceil(x)	arrondi un nombre vers l'entier le plus grand ➔ min{n|n≥x, n entier}

MATLAB offre beaucoup de commandes pour l'interaction avec l'utilisateur. Nous nous contentons pour l'instant d'un petit ensemble, et nous exposons les autres au fur et à mesure de l'avancement du cours.

La commande	**Sa signification**
who	Affiche le nom des variables utilisées
whos	Affiche des informations sur les variables utilisées
clear x y	Supprime les variables x et y
clear, clear all	Supprime toutes les variables
clc	Efface l'écran
exit, quit	Fermer l'environnement MATLAB
format	Définit le format de sortie pour les valeurs numériques {format long : affiche les nombres avec 14 chiffres après la virgule format short: affiche les nombres avec 04 chiffres après la virgule format bank : affiche les nombres avec 02 chiffres après la virgule format rat : affiche les nombres sous forme d'un ratio (a/b)}

5. Les vecteurs et les matrices

5.1 Les nombres en MATLAB

MATLAB utilise une notation décimale conventionnelle, avec un point décimal facultatif '.' et le signe '+' ou '–' pour les nombres signés. La notation scientifique utilise la lettre 'e' pour spécifier le facteur d'échelle en puissance de 10. Les nombres complexes utilisent les caractères 'i' et 'j' (indifféremment) pour designer la partie imaginaire. Le tableau suivant donne un résumé :

Le type	Exemples	
Entier	5	-83
Réel en notation décimale	0.0205	3.1415926
Réel en notation scientifique	1.60210e-20	6.02252e23 (1.60210×10^{-20} et 6.02252×10^{23})
Complexe	5+3i	-3.14159j

MATLAB utilise toujours les nombres réels (double précision) pour faire les calculs, ce qui permet d'obtenir une précision de calcul allant jusqu'à 16 chiffres significatifs.

Mais il faut noter les points suivants :

- Le résultat d'une opération de calcul est par défaut affiché avec quatre chiffres après la virgule.
- Pour afficher davantage de chiffres utiliser la commande **format long** (14 chiffres après la virgule)**.**
- Pour retourner à l'affichage par défaut, utiliser la commande **format short.**
- Pour afficher uniquement 02 chiffres après la virgule, utiliser la commande **format bank.**
- Pour afficher les nombres sous forme d'un ratio, utiliser la commande **format rat**.

La commande	Signification
format short	affiche les nombres avec 04 chiffres après la virgule
format long	affiche les nombres avec 14 chiffres après la virgule
format bank	affiche les nombres avec 02 chiffres après la virgule
format rat	affiche les nombres sous forme d'un ratio (a/b)

Par exemple :

```
» 5/3
ans =
1.6667
» format long
» 5/3
ans =
 1.66666666666667
» format bank
» 5/3
ans =
        1.67
» format short
» 5/3
ans =
   1.6667
» 5.2*4.3
ans =
22.3600
» format rat
» 5.2*4.3
ans =
 559/25
» 1.6667
ans =
 16667/10000
```

La fonction **vpa** peut être utilisée afin de forcer le calcul et présenter plus de décimaux significatifs en spécifiant le nombre de décimaux désirés.

Par exemple :

```
>> sqrt(2)
ans =
                    1.4142
>> vpa(sqrt(2),50)
ans =
                1.4142135623730950488016887242096980785696718753769
```

5.2 Les opérations mathématiques.

L'élément de base de MATLAB est la matrice. C'est-à-dire qu'un scalaire est une matrice de dimension 1x1, un vecteur colonne de dimension n est une matrice n x 1, un vecteur ligne de dimension n, une matrice 1 x n. Contrairement aux langages de programmation usuels, il n'est

pas obligatoire de déclarer les variables avant de les utiliser et, de ce fait, il faut prendre toutes les précautions dans la manipulation de ces objets.

Les scalaires se déclarent directement, par exemple :

```
>> x = 0;
>> a = x;
```

Les vecteurs lignes se déclarent en séparant les éléments par des espaces ou des virgules de:

```
>> V_ligne = [0 1 2]
V_ligne =
        0 1 2
```

Les vecteurs colonnes se déclarent en séparant les éléments par des points-virgules :

```
>> V_colonne = [0;1;2]
V_colonne =
0
1
2
```

Il est possible de transposer un vecteur à l'aide de la fonction **transpose**:

```
>> V_colonne = transpose(V_ligne)
V_colonne =
0
1
2
```

Le double point (:) est l'opérateur d'incrémentation dans MATLAB. Ainsi, pour créer un vecteur ligne des valeurs de 0 à 1 par incrément de 0.2, il suffit d'utiliser :

```
>> V = [0:0.2:1]
V =
Columns 1 through 6
0 0.2000 0.4000 0.6000 0.8000
1.0000
```

Par défaut, l'incrément est de 1. Ainsi, pour créer un vecteur ligne des valeurs de 0 à 5 par incrément de 1, il suffit d'utiliser :

```
>> V = [0:5]

V =
0 1 2 3 4 5
```

On peut accéder à un élément d'un vecteur et même modifier celui-ci directement :

```
>> a = V(2);
>> V(3) = 3*a
V =
        0 1 3 3 4 5
```

La fonction linspace :

La création d'un vecteur dont les composants sont ordonnés par intervalle régulier et avec un nombre d'éléments bien déterminé peut se réaliser avec la fonction :

linspace (début, fin, nombre d'éléments).

Le pas d'incrémentation est calculé automatiquement par Matlab selon la formule :

$$\text{le pas} = \frac{\text{fin} - \text{debut}}{\text{nombre d'éléments} - 1}$$

Par exemple :

```
>> X = linspace(1,10,4)            % un vecteur de quatre éléments de 1 à 10
        X =
                1    4    7    10

>> Y = linspace(13,40,4)           % un vecteur de quatre éléments de 13 à 40
        Y =
                13   22   31   40
```

La taille d'un vecteur (le nombre de ses composants) peut être obtenue avec la fonction `length` comme suit :

```
>> length(X)              % la taille du vecteur X
        ans =
                4
```

Les opérations usuelles d'addition, de soustraction et de multiplication par scalaire sur les vecteurs sont définies dans MATLAB :

```
>> V1 = [1 2];
>> V2 = [3 4];
>> V = V1 + V2 % addition de vecteurs
V =
        4 6
>> V = V2 - V1 % soustraction de vecteurs
V =
        2 2
>> V = 2*V1 % multiplication par un scalaire
V =
        2 4
```

On peut aussi créer des matrices, par exemple :

```
>> V1 = [1 2];
>> V2 = [3 4];
>> V = [V1;V2]
V =
1 2
3 4
```

Les matrices peuvent aussi être construites directement :

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

On peut évidemment avoir accès aux éléments de la matrice par :

```
>> m21 = M(2,1) % 2e ligne, 1ere colonne
m21 =
        3
```

Il est possible d'inverser **inv()** et de transposer **transpose()**les matrices :

```
>> invM = inv(M)
invM =
-2.0000 1.0000
1.5000 -0.5000
>> transpM = transpose(M)
transpM =
1 3
2 4
```

Un des intérêts de MATLAB est la possibilité d'utiliser directement les opérations mathématiques prédéfinies pour les matrices. L'addition et la soustraction sont directes (attention aux dimensions) ainsi que la multiplication par un scalaire :

```
>> A = [1 2;3 4];
>> B = [4 3;2 1];
>> C = A+B % addition
C =
    5    5
    5    5
>> D = A-B % soustraction
D =
   -3   -1
    1    3
>> C = 3*A % multiplication par un scalaire
C =
    3    6
    9   12
>> C = A*B % multiplication de matrices
C =
    8    5
   20   13
>> D = A/B % division de matrices
D =
   1.5000   -2.5000
   2.5000   -3.5000
```

Il est utile de noter les possibilités suivantes :

- L'accès à un élément de la ligne **i** et la colonne **j** se fait par : **A(i,j)**
- L'accès à toute la ligne numéro **i** se fait par : **A(i,:)**
- L'accès à toute la colonne numéro **j** se fait par : **A(:,j)**

Par exemple :

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]     % création de la matrice A
        A =
                1    2    3    4
                5    6    7    8
                9   10   11   12

>> A(2,3)                                   % l'élément sur la 2ème ligne à la 3ème colonne
        ans =
                7

>> A(1,:)                                   % tous les éléments de la 1ère ligne
        ans =
                1    2    3    4


>> A(:,2)                                   % tous les éléments de la 2ème colonne
        ans =
                2
                6
                10

>> A(2:3,:)                                 % tous les éléments de la 2ème et la 3ème ligne
        ans =
                5    6    7    8
                9   10   11   12

>> A(1:2,3:4)                               % La sous matrice supérieure droite de taille 2x2
        ans =
                3    4
                7    8

>> A([1,3],[2,4])                           % la sous matrice : lignes(1,3) et colonnes (2,4)
        ans =
                2    4
                10   12

>> A(:,3) = []                              % Supprimer la troisième colonne
        A =
                1    2    4
                5    6    8
                9   10   12
>> A(2,:) = []                              % Supprimer la deuxième ligne
        A =
                1    2    4
                9   10   12

>> A = [A , [0;0]]                          % Ajouter une nouvelle colonne {ou A(:,4)=[0;0]}
        A =
                1    2    4     0
                9   10   12     0
```

```
>> A = [A ; [1,1,1,1]]              % Ajouter une nouvelle ligne {ou A(3,:)=[1,1,1,1]}
        A =
                1    2    4    0
                9   10   12   0
                1    1    1    1
```

Les dimensions d'une matrice peuvent être acquises en utilisant la fonction **size**. Cependant, avec une matrice A de dimension **m × n** le résultat de cette fonction est un vecteur de deux composants, une pour **m** et l'autre pour **n**.

```
>> d = size(A)
        d  =
                3        4
```

Ici, la variable **d** contient les dimensions de la matrice A sous forme d'un vecteur. Pour obtenir les dimensions séparément on peut utiliser la syntaxe :

```
>> d1 = size (A, 1)                 % d1 contient le nombre de lignes (m)
        d1 =
                3

>> d2 = size (A, 2)                 % d2 contient le nombre de colonnes (n)
        d2 =
                4
```

5.3 Génération automatique des matrices

Dans MATLAB, il existe des fonctions qui permettent de générer automatiquement des matrices particulières. Dans le tableau suivant nous présentons les plus utilisées :

La fonction	Signification
zeros(n)	Génère une matrice n × n avec tous les éléments = 0
zeros(m,n)	Génère une matrice m × n avec tous les éléments = 0
ones(n)	Génère une matrice n × n avec tous les éléments = 1
ones(m,n)	Génère une matrice m × n avec tous les éléments = 1
eye(n)	Génère une matrice identité de dimension n × n
magic(n)	Génère une matrice magique de dimension n × n
rand(m,n)	Génère une matrice de dimension m × n de valeurs aléatoires

6. Application

1. On veut vérifier que la multiplication de matrices n'est pas commutative. Soient deux matrices :

$$A = \begin{pmatrix} 3 & 4 & 4 \\ 6 & 5 & 3 \\ 10 & 8 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 4 & 5 & 8 \\ 3 & 11 & 12 \\ 2 & 1 & 7 \end{pmatrix}$$

Réalisez un code MATLAB qui permet, pour les matrices **A** et **B** données, de vérifier que :

$$A * B - B * A \neq 0$$

2. En utilisant les matrices définies au premier exercice, vérifiez l'identité suivante :

$$(A + B)^T = A^T + B^T$$

Utilisez la fonction **transpose** de MATLAB pour transposer les matrices.

3. On désire résoudre un système d'équations algébriques linéaires, c'est-à-dire, un système représenté par :

$$A * x = b$$

Avec **A**, une matrice de dimension n x n, **x** et **b**, des vecteurs colonnes de dimension n. La solution de ce système est donnée par :

$$x = A^{-1} * b$$

En utilisant la fonction **inv** de **MATLAB** pour inverser la matrice, résoudre le système décrit par la matrice **A**, définie au premier exercice et le vecteur **b**, la première colonne de la matrice **B**, de ce même exercice.

Chapitre II : Introduction à la programmation avec MATLAB

1. Introduction

Nous avons vu jusqu'à présent comment utiliser MATLAB pour effectuer des commandes ou pour évaluer des expressions en les écrivant dans la ligne de commande, par conséquent les commandes utilisées s'écrivent généralement sous forme d'une seule instruction (éventuellement sur une seule ligne).

Cependant, il existe des problèmes dont la description de leurs solutions nécessite plusieurs instructions, ce qui réclame l'utilisation de plusieurs lignes. Comme par exemple la recherche des racines d'une équation de second degré (avec prise en compte de tous les cas possibles).

Une collection d'instructions bien structurées visant à résoudre un problème donné s'appelle un programme. Dans cette partie, on va présenter les mécanismes d'écriture et d'exécution des programmes en MATLAB.

2. Opérateurs de comparaison et opérateurs logiques

2.1 Les opérateurs de comparaison sont:

= = : égale à (X= = Y)

> : Strictement supérieur à (X > Y)

< : Strictement inférieur à (X< Y)

>= : supérieur ou égale à (X>= Y)

<= : inférieur ou égale à (X<= Y)

~= : différent de (X~=Y)

2.2 Les opérateurs logiques sont:

& : et (and) (X &Y)

| : ou (or) (X | Y)

- : Non (not) X (-X)

3. Les entrées/sorties

3.1 Entrée au clavier

L'utilisateur peut saisir des informations au clavier grâce à la commande x = input(...).

```
x=input ('Saisir une valeur de x : ');
Saisir une valeur de x : 5
x
x =
     5
```

3.2 Sortie à l'écran

Pour afficher quelque chose à l'écran, l'utilisateur peut utiliser la commande disp, qui affiche le contenu d'une variable (chaîne de caractères, vecteur, matrice...).

```
» A=[1 2 3];
» disp(A);
   1   2   3
```

4. Instructions de contrôle

Les instructions de contrôle sous Matlab sont tr`es proches de celles existant dans d'autres langages de programmation.

4.1 L'instruction while :

L'instruction **while** répète l'exécution d'un groupe d'instructions un nombre indéterminé de fois selon la valeur d'une condition logique. Elle a la forme générale suivante :

```
while (condition)
        Ensemble d'instructions
end
```

Tant que l'expression de **while** est évaluée à vrai (true), l'ensemble d'instructions s'exécutera en boucle.

Exemple :

Faire un programme sous **Matlab** qui calcule la somme suivante:

S=1+2/2! +3/3!+… on arrête le calcul quand S>2.5

```
» s=1;i=1;f=1;
» while s<=2.5
i=i+1
f=f*i;
s=s+i/f
end
i =
    2
s =
    2
i =
    3
s =
   2.5000
i =
    4
s =
   2.6667
```

4.2 L'instruction if :

L'instruction **if** est la plus simple et la plus utilisée des structures de contrôle de flux. Elle permet d'orienter l'exécution du programme en fonction de la valeur logique d'une condition. Sa syntaxe générale est la suivante :

```
if (condition)
      instruction_1
      instruction_2
                . . .
      Instruction_N
end
```

ou bien

```
if (condition)
      ensemble d'instructions 1
else
      ensemble d'instructions 2
end
```

Si la condition est évaluée à vrai (true), les instructions entre le **if** et le **end** seront exécutées, sinon elles ne seront pas (ou si un **else** existe les instructions entre le **else** et le **end** seront exécutées). S'il est nécessaire de vérifier plusieurs conditions au lieu d'une seule, on peut utiliser des clauses **elseif** pour chaque nouvelle condition, et à la fin on peut mettre un **else** dans le cas ou aucune condition n'a été évaluée à vrai.

Voici donc la syntaxe générale :

```
if (expression_1)
      Ensemble d'instructions 1
elseif (expression_2)
      Ensemble d'instructions 2
            ....
elseif (expression_n)
      Ensemble d'instructions n
else
      Ensemble d'instructions si toutes les expressions étaient fausses
end
```

Exemple :

Faire un programme sous MATLAB qui résout le problème suivant:

1. y = x si x<0
2. $y=x^2$ si x>0
3. y=10 si x=0

```
x=input('introduire la valeur de x ');
Introduire la valeur de x 6
if x<0
y=x;
end
if x>0
y=x^2;
end
if x==0
y=10;
end
disp('la valeur de y est: '),y
```

La valeur de y est:

y =

36

Exemple :

Faire un programme sous MATLAB qui résout le problème suivant:

1. y = x si x<0
2. y=x² si x≥0

```
x=input('introduire la valeur de x ');
introduire la valeur de x 6
if x<0
y=x;
else
y=x^2;
end
```

La valeur de y est:

y =

36

Exemple :

Créons un programme qui trouve les racines d'une équation de second degré désigné par : $ax^2+bx+c=0$. Voici le M-File qui contient le programme (il est enregistré avec le nom 'Equation2deg.m')

```
% Programme de résolution de l'équation a*x^2+b*x+c=0
a = input ('Entrez la valeur de a : ');      % lire a
b = input ('Entrez la valeur de b : ');      % lire b
c = input ('Entrez la valeur de c : ');      % lire c
delta = b^2-4*a*c ;                          % Calculer delta
if delta<0
   disp('Pas de solution')                   % Pas de solution
elseif delta==0
   disp('Solution double : ')                % Solution double
   x=-b/(2*a)
else
   disp('Deux solutions distinctes: ')                    %
Deux solutions
   x1=(-b+sqrt(delta))/(2*a)
   x2=(-b-sqrt(delta))/(2*a)
end
```

Si nous voulons exécuter le programme, il suffit de taper le nom du programme :

```
>> Equation2deg
        Entrez la valeur de a : -2

        Entrez la valeur de b : 1

        Entrez la valeur de c : 3

        Deux solutions :
        x1 =
                -1
        x2 =
                1.5000
```

Ainsi, le programme va être exécuté en suivant les instructions écrites dans son M-File. Si une instruction est terminée par un point virgule, alors la valeur de la variable concernée ne sera pas affichée, par contre si elle se termine par une virgule ou un saut à la ligne, alors les résultats seront affichés.

Remarque :

Il existe la fonction **solve** prédéfinie en MATLAB pour trouver les racines d'une équation (et beaucoup plus). Si nous voulons l'appliquer sur notre exemple, il suffit d'écrire :

```
>> solve('-2*x^2+x+3=0','x')
 ans =
        -1
        3/2
```

4.3 L'instruction switch :

L'instruction **switch** exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé à une clause **case** qui définit si ce groupe doit être exécuté ou pas selon l'égalité de la valeur de ce **case** avec le résultat d'évaluation de l'expression de **switch**. Si tous les **cases** n'ont pas été acceptés, il est possible d'ajouter une clause **otherwise** qui sera exécutée seulement si aucun **case** n'est exécuté.

Donc, la forme générale de cette instruction est :

```
switch (expression)
      case valeur_1
            Groupe d'instructions 1
      case valeur_2
            Groupe d'instructions 2
            . . .
      case valeur_n
            Groupe d'instructions n
      otherwise
            Groupe d'instructions si tous les cases ont échoué
end
```

Exemple :

```
x = input ('Entrez un
nombre : ') ;
switch(x)
case 0
disp('x = 0 ')
case 10
disp('x = 10 ')
case 100
disp('x = 100 ')
otherwise
disp('x n''est pas 0 ou 10
ou 100 ')
end
```

L'exécution va donner :

Entrez un nombre : 50

x n'est pas 0 ou 10 ou 100

4.4 L'instruction for :

L'instruction **for** répète l'exécution d'un groupe d'instructions un nombre déterminé de fois. Elle a la forme générale suivante :

```
for variable = expression_vecteur
      Groupe d'instructions
end
```

L'expression_vecteur correspond à la définition d'un vecteur : début : pas : fin ou début : fin

La variable va parcourir tous les éléments du vecteur défini par l'expression, et pour chacune il va exécuter le groupe d'instructions.

Exemple :

Dans le tableau suivant, nous avons trois formes de l'instruction **for** avec le résultat MATLAB :

L'instruction for	`for i = 1 : 4`
	`  j=i*2 ;`
	`  disp(j)`
	`end`
		`for i = 1 : 2 : 4`
		`  j=i*2 ;`
		`  disp(j)`
		`end`
			`for i = [1,4,7]`
			`  j=i*2 ;`
			`  disp(j)`
			`end`

Le résultat de l'exécution	2 4 6 8	2 6	2 8 14

5. Les graphiques et la visualisation des données en MATLAB

Partant du principe qu'une image vaut mieux qu'un long discours, MATLAB offre un puissant système de visualisation qui permet la présentation et l'affichage graphique des données d'une manière à la fois efficace et facile.

Dans cette partie, nous allons présenter les principes de base indispensables pour dessiner des courbes en MATLAB.

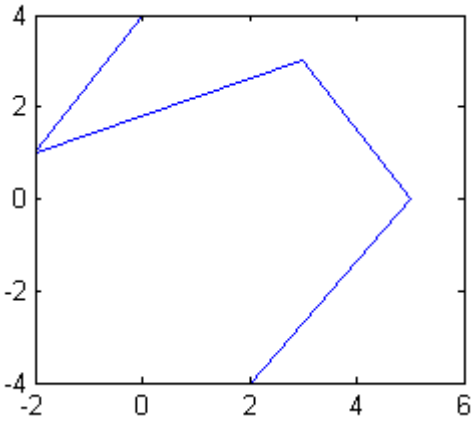
5. 1 La fonction plot :

La fonction **plot** est utilisable avec des vecteurs ou des matrices. Elle trace des lignes en reliant des points de coordonnées définis dans ses arguments, et elle a plusieurs formes :

Si elle contient deux vecteurs de la même taille comme arguments : elle considère les valeurs du premier vecteur comme les éléments de l'axe X (les abscisses), et les valeurs du deuxième vecteur comme les éléments de l'axe Y (les ordonnées).

Exemple:

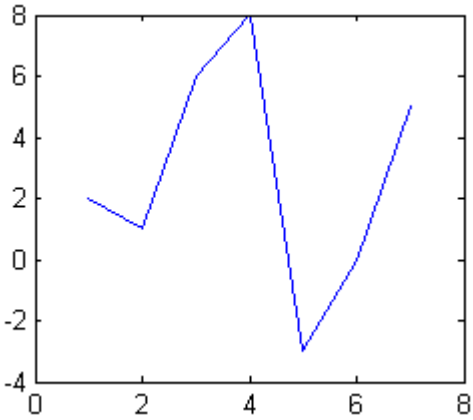
```
>> A = [2, 5, 3,-2,0]
A =
        2    5    3   -2    0
>> B = [-4, 0, 3,1,4]
B =
       -4    0    3    1    4
>> plot(A,B)
```



Si elle contient un seul vecteur comme argument : elle considère les valeurs du vecteur comme les éléments de l'axe Y (les ordonnées), et leurs positions relatives définiront l'axe X (les abscisses).

Exemple :

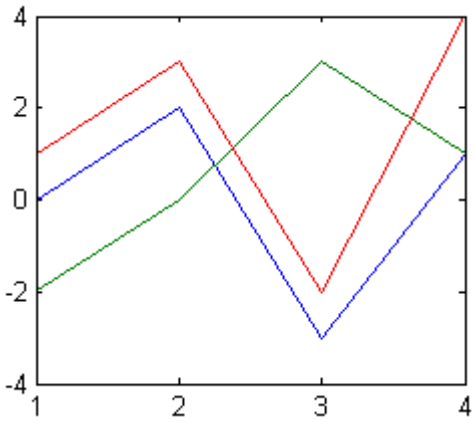
```
>> V = [2, 1, 6, 8,-3, 0,5]
V =
2   1   6   8  -3   0   5
>> plot(V)
```



Si elle contient une seule matrice comme argument : elle considère les valeurs de chaque colonne comme les éléments de l'axe Y, et leurs positions relatives (le numéro de ligne) comme les valeurs de l'axe X. Donc, elle donnera plusieurs courbes (une pour chaque colonne).

Exemple :

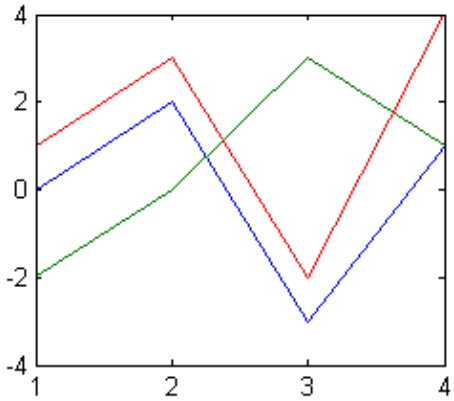
```
>> M = [0 -2 1;2 0 3;-3 3 -2;1 1 4]
        M =
           0   -2    1
           2    0    3
          -3    3   -2
           1    1    4
>> plot(M)
```



Si elle contient deux matrices comme arguments : elle considère les valeurs de chaque colonne de la première matrice comme les éléments de l'axe X, et les valeurs de chaque colonne de la deuxième matrice comme les valeurs de l'axe Y.

Exemple :

```
>> K = [1 1 1;2 2 2; 3 3 3;4 4 4]
K =
   1   1   1
   2   2   2
   3   3   3
   4   4   4
>> M = [0 -2 1; 2 0 3;-3 3 -2; 1 1 4]
M =
    0   -2    1
    2    0    3
   -3    3   -2
    1    1    4
 >> plot(K,M)
```



5. 2 Modification de l'apparence d'une courbe :

Il est possible de manipuler l'apparence d'une courbe en modifiant la couleur de la courbe, la forme des points de coordonnées et le type de ligne reliant les points.

Pour cela, on ajoute un nouvel argument (qu'on peut appeler un marqueur) de type chaîne de caractères à la fonction **plot** comme ceci :

plot (x, y, 'marqueur')

Le contenu du marqueur est une combinaison d'un ensemble de caractères spéciaux rassemblés dans le tableau suivant :

Couleur de la courbe		**Représentation des points**	
le caractère	**son effet**	**le caractère**	**son effet**
b ou blue	**courbe en bleu**	**.**	un point **.**
g ou green	**courbe en vert**	**o**	un cercle ●
r ou red	**courbe en rouge**	**x**	le symbole **x**
c ou cyan	**entre le vert et le bleu**	**+**	le symbole **+**
m ou magenta	**en rouge violacé vif**	*****	une étoile *****
y ou yellow	**courbe en jaune**	**s**	un carré ■
k ou black	**courbe en noir**	**d**	un losange ♦
Style de la courbe		**v**	triangle inferieur ▼
le caractère	**son effet**	**^**	triangle supérieur ▲
-	**en ligne plein** ———	**<**	triangle gauche ◀
:	**en pointillé** ·····	**>**	triangle droit ▶
-.	**en point tiret** – · – ·	**p**	pentagramme ★
--	**en tiret** – – –	**h**	hexagramme ✶

5. 3 Annotation d'une figure :

Dans une figure, il est préférable de mettre une description textuelle aidant l'utilisateur à comprendre la signification des axes et de connaître le but ou l'intérêt de la visualisation concernée.

Il est très intéressant également de pouvoir signaler des emplacements ou des points significatifs dans une figure par un commentaire signalant leurs importances.

- ✓ Pour donner un titre à une figure contenant une courbe nous utilisons la fonction **title** comme ceci :
  >> **title('titre de la figure')**

- ✓ Pour donner un titre pour l'axe vertical des ordonnées y, nous utilisons la fonction **ylabel** comme ceci :
  >> **ylabel('Ceci est l''axe des ordonnées Y')**

- ✓ Pour donner un titre pour l'axe horizontal des abscisses x, nous utilisons la fonction **xlabel** comme ceci :
  >> **xlabel('Ceci est l'axe des abscisses X')**

- ✓ Pour écrire un texte (un message) sur la fenêtre graphique à une position indiquée par les coordonnées **x** et **y**, nous utilisons la fonction **text** comme ceci :
  >> **text(x, y, 'Ce point est important')**

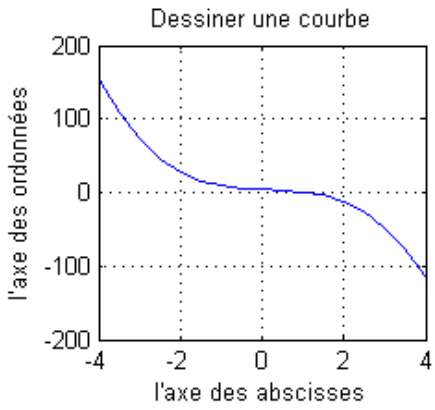
- ✓ Pour mettre un texte sur une position choisie manuellement par la souris, nous utilisons la fonction **gtext**, qui a la syntaxe suivante :
  >> **gtext('Ce point est choisi manuellement')**

- ✓ Pour mettre un quadrillage (une grille), nous utilisons la commande **grid** (ou **grid on**). Pour l'enlever nous réutilisons la même commande **grid** (ou **grid off**).

Exemple :

Dessinons la fonction : $y = -2x^3+x^2-2x+4$ pour x variant de -4 jusqu'à 4, avec un pas = 0.5.

```
>> x = -4:0.5:4;
>> y = -2*x^3+x^2-2*x+4;
>> plot(x,y)
>> grid
>> title('Dessiner une courbe')
>> xlabel('l'axe des abscisses')
>> ylabel('l'axe des ordonnées')
```



5. 4 Utiliser plot avec plusieurs arguments

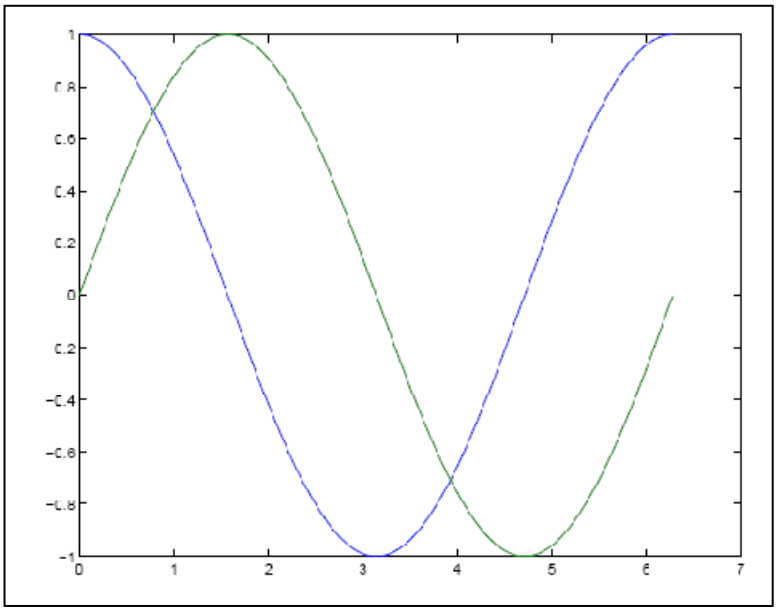
On peut utiliser plot avec plusieurs couples (x,y) ou triplets (x ,y, 'marqueur') comme arguments.

Exemple :

L'exemple qui suit:

```
>> x = [0:0.01:2*pi];
>> plot(x,cos(x),x ,sin(x))
```

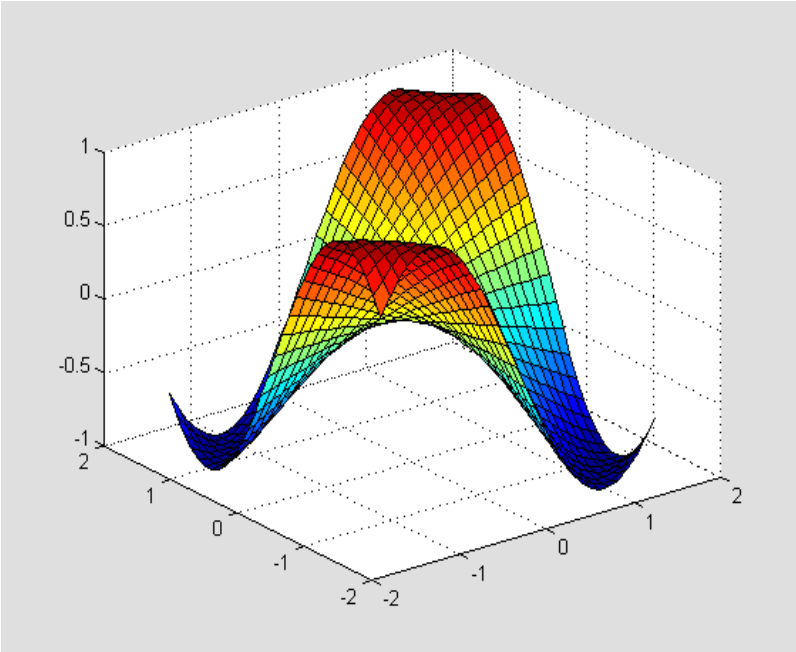
produit la sortie graphique suivante :



5. 5 Représentation graphique 3D

Supposons, par exemple que l'on veuille représenter la surface définie par la fonction z=sin(xy) sur le carré suivant [-p/2,p/2]x[-p/2,p/2] à l'aide d'une grille de points 31x31. On utilise la séquence de commandes :

```
» [xi,yi]=meshgrid(-pi/2:pi/30:pi/2); %génération d'une grille régulière
» zi=sin(xi.*yi);
» surf(xi,yi,zi,zi); % affichage de la surface
```



On peut modifier l'angle de vue via la commande **view** en spécifiant soit un point d'observation ou deux angles d'élévation. Noter que la commande **view(2)** déclenche directement une vue de dessus. On peut également choisir l'angle de vue à la souris en activant l'option **rotate3d on** et on annule ce mode par la commande **rotate3d off.** De même la commande **zoom in** permet d'effectuer des zooms à la souris, seulement en vue plane!

Il existe par ailleurs de nombreuses commandes permettant de contrôler les palettes de couleurs utilisées pour la représentation graphique, en particulier la commande **colormap**. Signalons une option intéressante permettant de lisser les couleurs : **shading interp**. Pour une description exhaustive de toutes ces possibilités faire **help graph3d**.

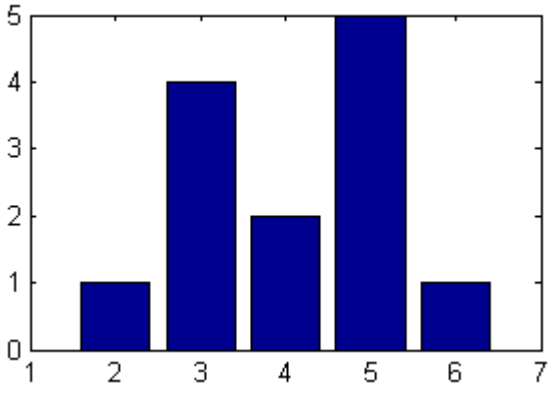
5. 6 D'autres types de graphiques:

Le langage MATLAB ne permet pas uniquement l'affichage des points pour tracer des courbes, mais il offre aussi la possibilité de tracer des graphes à bâtons et des histogrammes.

Pour tracer un graphe à bâtons nous utilisons la fonction **bar** qui a le même principe de fonctionnement que la fonction **plot**.

Exemple :

```
>> X=[2,3,5,4,6];
>> Y=[1,4,5,2,1];
>> bar(X,Y)
```



Il est possible de modifier l'apparence des bâtons, et il existe la fonction **barh** qui dessine les bâtons horizontalement, et la fonction **bar3** qui ajoute un effet 3D.

Parmi les fonctions de dessin très intéressantes non présentées ici, on peut trouver : **hist**, **stairs**, **stem**, **pie**, **pie3**, ...etc..

Nous signalons aussi que **Matlab** permet l'utilisation d'un système de coordonnées autre que le système cartésien comme le système de coordonnées polaires (pour plus de détail chercher les fonctions **compass**, **polar** et **rose**).

6 .Applications

1. Ecrire le code en MATLAB qui calcule la somme de la suite géométrique suivante :

$$\sum_{n=1}^{6} 2^n$$

2. Ecrire le code en MATLAB qui calcule les sommes

$$\sum_{j=1}^{N} j^p$$

Pour **N = 6** et **p** égal à **un**, **deux** et **trois**.

Chapitre III : Applications des méthodes numériques avec MATLAB

1. Introduction

L'analyse numérique est utilisée pour trouver des approximations à des problèmes difficiles tels que la résolution des équations non linéaires, l'intégration impliquant des expressions complexes. Elle est appliquée à une grande variété de disciplines telles que tous les domaines de l'ingénierie, de l'informatique, l'éducation, la géologie, la météorologie, et bien d'autres. Il y a des années, les ordinateurs à haute vitesse n'existaient pas, par conséquent, le calcul manuel exigeait beaucoup de temps et de travail laborieux. Mais maintenant que les ordinateurs sont devenus indispensables pour les travaux de recherche dans la science, l'ingénierie et d'autres domaines, l'analyse numérique est devenue une tâche beaucoup plus facile et plus agréable.

2. Résolution de systèmes linéaires par le logiciel MATLAB.

2. 1. Méthode du pivot de Gauss (méthode directe)

La méthode du pivot de Gauss est une méthode directe de résolution de système linéaire qui permet de transformer un système en un autre système équivalent échelonné. On résout le système ainsi obtenu à l'aide d'un algorithme de remontée.

Principe

On cherche à résoudre le système suivant de **n** équations à **n** inconnues $\mathbf{x_1,x_2,\ldots,x_n}$:

$$\begin{cases} a_{11}\,x_1 + a_{12}\,x_{2+} \ldots + a_{1n}\,x_n = b_1 \\ a_{21}\,x_1 + a_{22}\,x_{2+} \ldots + a_{2n}\,x_n = b_2 \\ \quad\quad . \\ \quad\quad . \\ \quad\quad . \\ a_{n1}\,x_1 + a_{n2}\,x_{1+} \ldots + a_{nn}\,x_n = b_n \end{cases}$$

Du point de vue matriciel, on a **Ax=b** Avec

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

Les données du système linéaire sont :

- les coefficients réels ou complexes a_{ij} pour i=1,…,n et j=1,…,p (n et p sont deux entiers connus),
- le second membre du système constitué par les nombres réels ou complexes b_i (i=1,…,n),

et L_i (i=1,…,n) désigne la $i^{ième}$ ligne du système (S).

Les inconnues à déterminer sont les x_j (j=1,…,p).

Le système est dit carré lorsque n=p. C'est le cas où il y a autant d'équations que d'inconnues.

On dira que le système est homogène lorsque le second membre est nul (b_i=0, i=1,…,n). On peut remarquer qu'un système linéaire homogène admet au moins la solution nulle x_i=0, ∀i=1,…,n (qui n'est pas nécessairement la seule).

Lorsque tous les coefficients « sous la diagonale » d'un système linéaire sont nuls, i.e. :

$$i > j \Rightarrow a_{ij} = 0$$

On dit que le système est échelonné.

a- la méthode du pivot de Gauss : Triangularisation

$$k=1,\dots,n-1\begin{cases} a_{ij}^{(k+1)}=a_{ij}^{(k)} & \left| \begin{array}{ll} i=1,\dots,k & j=1,\dots,n \end{array} \right. \\ a_{ij}^{(k+1)}=0 & \left| \begin{array}{ll} i=k+1,\dots,n & j=1,\dots,k \end{array} \right. \\ a_{ij}^{(k+1)}=a_{ij}^{(k)}-\dfrac{a_{ik}^{(k)}a_{kj}^{(k)}}{a_{kk}^{(k)}} & \left| \begin{array}{ll} i=k+1,\dots,n & j=k+1,\dots,n \end{array} \right. \\ b_i^{(k+1)}=b_i^{(k)} & \left| \begin{array}{l} i=1,\dots,k \end{array} \right. \\ b_i^{(k+1)}=b_i^{(k)}-\dfrac{a_{ik}^{(k)}b_k^{(k)}}{a_{kk}^{(k)}} & \left| \begin{array}{l} i=k+1,\dots,n \end{array} \right. \end{cases}$$

Soit **U** la matrice échelonnée du système, on a alors

$$U=(u_{ij})_{1\leq i,j\leq n}=(a_{ij}^{(n)})_{1\leq i,j\leq n}$$

b- Remontée et résolution

À présent la matrice **A** du système linéaire est échelonnée, on doit alors résoudre le système triangulaire :

Ux = b(n)

Puisque **b(n)** rappelons le, est le second membre échelonné, il a subi les mêmes opérations que la matrice échelonnée **U**.

On utilise alors un algorithme de remontée pour le système **Ux = b(n)** :

$$\begin{cases} x_n=\dfrac{y_n}{u_{nn}}=\dfrac{y_n}{a_{nn}^{(n)}}; \\ x_i=\dfrac{1}{u_{ii}}(y_i-\sum\limits_{j=i+1}^{n}u_{ij}x_j)=\dfrac{1}{a_{ii}^{(n)}}(y_i-\sum\limits_{j=i+1}^{n}a_{ij}^{(n)}x_j) \quad \forall i=n-1,n-2,\dots,1. \end{cases}$$

c- Exemple de résolution

Considérons le système suivant :

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_1 + 3x_2 - 2x_3 = -1 & L_2 \\ 3x_1 + 5x_2 + 8x_3 = 8 & L_3 \end{cases}$$

Avec

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & -2 \\ 3 & 5 & 8 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, b = \begin{pmatrix} 2 \\ -1 \\ 8 \end{pmatrix}$$

Première étape du pivot de Gauss pour éliminer les variables $\mathbf{x_1}$ dans les lignes $\mathbf{L_2}$et $\mathbf{L_3}$:

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_2 - 4x_3 = -3 & L_2 \leftarrow L_2 - L_1 \\ -x_2 + 2x_3 = 2 & L_3 \leftarrow L_3 - 3L_1 \end{cases}$$

Seconde étape du pivot de Gauss pour éliminer les variables $\mathbf{x_2}$ dans la ligne $\mathbf{L_3}$:

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_2 - 4x_3 = -3 & L_2 \\ -2x_3 = -1 & L_3 \leftarrow L_3 + L_2 \end{cases}$$

En remontant le système, on obtient aisément la solution **x** du système :

$$x = \begin{pmatrix} 3 \\ -1 \\ 1/2 \end{pmatrix}$$

```
clear all
close all
taille=input('Taille de la matrice : ');
for t = 1:taille
    for p = 1:taille
        str=sprintf('Entrer l''élément (%d,%d) de la matrice A : ',t,p);
        a(t,p) = input(str);
    end
end

    for t = 1:taille
    for p = 1:taille
        str=sprintf('Entrer l''élément (%d,%d) de la matrice B : ',t,p);
        b(t,p) = input(str);
    end
end
disp(a)
disp(b)
A=[a,b]
n=size(A,1)
    for k=1:n-1
    for i=k+1:n
w=A(i,k)/A(k,k)
    for j=k:n+1
A(i,j)=A(i,j)-w*A(k,j)
end
end
end
A
    for i=n:-1:1
s=0;
    for j=i+1:n
s=s+A(i,j)*x(j);
end
x(i)=(A(i,n+1)-s)/A(i,i)
end
x
```

Code MATLAB implémentant la méthode de Gauss

2. 2. Méthodes itératives

Principe

On cherche à obtenir une suite de vecteur $(X_n)_n$ convergeante vers $\overline{X}$ telleque $A\overline{X} = b$.

En posant $A = M - N$ où M est une matrice inversible, on propose la formule de récurrence $MX_{n+1} = NX_n + b$ ou encore $X_{n+1} = M^{-1}NX_n + M^{-1}b$. Si $\overline{X}$ est solution du problème, il vient que $(\overline{X} - X_n) = (M^{-1}N)^n \times (\overline{X} - X_0)$. $B = (M^{-1}N)$ est appelée la matrice d'itération.

Ainsi pour que $X_n \to \overline{X}$, il est nécessaire que $(M^{-1}N)^n \to 0$.

Remarque : Le cadre normal de convergence des méthodes itératives classiques est : « A matrice symétrique définie positive ».

Vitesse de convergence :

Théorème : $(B^n)_n \to 0$ ssi $\rho(B) < 1$. Où $\rho(B)$ est le rayon spectral de la matrice B.

Vitesse de convergence : Considérons la matrice d'itération B symétrique, alors $\|B\|_2 = \rho(B)$ et l'on obtient le résultat suivant : $\|x_k - x\|_2 \le \rho(B)^k . \|x_0 - x\|_2$.

En conclusion, le rayon spectral de la matrice d'itération mesure la vitesse de convergence. La convergence est d'autant plus rapide que le rayon spectral est plus petit. On admettra que ce résultat se généralise aux matrices quelconques.

2.3. Méthode Jacobi

Notation : $A = \begin{pmatrix} D & -F \\ -E & \end{pmatrix}$

On pose $A = M - N$ avec M = D et N = E + F selon le schéma ci-dessus. Ce qui donne la formule de récurrence suivante : $X_{n+1} = D^{-1}(E + F)X_n + D^{-1}b$.

Soit $x_k = (x_k^i)_{i=1,n}$ le vecteur itéré, on donne l'algorithme donnant x_{k+1} en fonction de x_k.

Pour i de 1 à n faire

$$x_{k+1}^i = \frac{-1}{a_{i,i}} \left[\sum_{j=1}^{i-1} a_{i,j} x_k^j + \sum_{j=i+1}^{n} a_{i,j} x_k^j - b_i \right]$$

Théorème : Soit A une matrice symétrique, à diagonale strictement dominante, c'est à dire $a_{i,i} > \sum_{i \neq j} |a_{i,j}|$ alors la méthode est convergente car $\rho\left(D^{-1}(E+F)\right) < 1$.

A priori, cette méthode converge pour une matrice A SDP (matrice symétrique définie positive).

2.4. Méthode Gauss-Seidel

Boucle directe

On reprend le schéma de matrice précédent et on pose $M = D - E$ et $N = F$. La formule de récurrence s'écrit : $(D-E)X_{n+1} = FX_n + b$.

On calcule x_{k+1}^i, la i^{ème} composante du vecteur X_{k+1} : $a_{i,i}x_{k+1}^i = -\sum_{j=1}^{i-1} a_{i,j}x_{k+1}^j - \sum_{j=i+1}^{n} a_{i,j}x_k^j + b_i$.

On propose l'algorithme suivant :

> Pour i de 1 à n faire
>
> $$x_{k+1}^i = \frac{1}{a_{i,i}}\left[-\sum_{j=1}^{i-1} a_{i,j}x_{k+1}^j - \sum_{j=i+1}^{n} a_{i,j}x_k^j + b_i\right]$$

On procède par écrasement du vecteur X_k : $\left(\underbrace{x_{k+1}^j, 1 \le j \le i-1}_{\text{déjà calculés}} \;\middle|\; \underbrace{x_{k+1}^i}_{\text{calcul en cours}} \;\middle|\; x_k^j, i+1 \le j \le n \right)^t$

.

Boucle rétrograde

Cette fois, on pose $M = D - F$ et $N = E$.

On propose l'algorithme suivant :

> Pour i de n à 1 faire
>
> $$x_{k+1}^i = \frac{1}{a_{i,i}}\left[-\sum_{j=1}^{i-1} a_{i,j}x_k^j - \sum_{j=i+1}^{n} a_{i,j}x_{k+1}^j + b_i\right]$$

```
clear all
close all

a11=input('A11 : ');
a12=input('A12 : ');
a13=input('A13 : ');
a21=input('A21 : ');
a22=input('A22 : ');
a23=input('A23 : ');
a31=input('A31 : ');
a32=input('A32 : ');
a33=input('A33 : ');

b1=input('B1 : ');
b2=input('B2 : ');
b3=input('B3 : ');

x1=input('X1 : ');
x2=input('X2 : ');
x3=input('X3 : ');

A=[a11 a12 a13;a21 a22 a23;a31 a32 a33]
B=[b1;b2;b3]
X=[x1;x2;x3]

for k=1:3
for i=1:3
x=0;
for j=1:3
if j~=i
x=x+((A(i,j)./A(i,i)).*X(j,1));
end
end
Xn(i,1)=-x+(B(i,1)./A(i,i));
end
A=[a11 a12 a13;a21 a22 a23;a31 a32 a33]
B=[b1;b2;b3]
X=[x1;x2;x3]

for k=1:3
for i=1:3
x=0;
for j=1:3
if j~=i
x=x+((A(i,j)./A(i,i)).*X(j,1));
end
end
Xn(i,1)=-x+(B(i,1)./A(i,i));
end
X=Xn;
disp(X)
end
```

Code MATLAB implémentant la méthode Jacobi

3. Les Polynômes dans MATLAB

3 .1. Opérations sur les polynômes dans MATLAB

Dans MATLAB, les polynômes sont représentés sous forme de vecteurs lignes dont les composantes sont données par ordre des puissances décroissantes. Un polynôme de degré n est représenté par un vecteur de taille (n+1).

Exemple :

Le polynôme :$f(x)= 8x^5+2x^3+3x^2+4x-2$ est représenté par :

```
>>f=[8 0 2 3 4 -2]
f =8 0 2 3 4 -2
```

3.1.1. Multiplication des polynômes

La fonction '**conv**' donne le produit de convolution de deux polynômes. L'exemple suivant montre l'utilisation de cette fonction.

Soient :

$f(x)= 3x^3+2x^2-x+4$

$g(x)= 2x^4-3x^2+5x-1$

Le produit de convolution : $h(x)= f(x) . g(x)$ est donné par :

```
>>f=[3 2 -1 4];
>>g=[2 0 -3 5 -1];
>>h=conv(f,g)
h =
6 4 -11 17 10 -19 21 -4
```

Ainsi, le polynôme h(x) obtenu est :

$h(x)= 6x^7+4x^6-11x^5+17x^4+10x^3-19x^2+21x-4$

3. 1.2. Division des polynômes

La fonction '**deconv**' donne le rapport de convolution de deux polynômes (déconvolution des coefficients du polynôme). L'exemple suivant montre l'utilisation de cette fonction.

Soient les mêmes fonctions précédentes f(x) et g(x) :

$f(x)= 3x^3+2x^2-x+4$

$g(x)= 2x^4-3x^2+5x-1$

La division de g(x) par f(x) :

$h(x) = \frac{g(x)}{f(x)}$ est donnée par la fonction 'deconv' :

```
>>f=[3 2 -1 4];
>>g=[2 0 -3 5 -1];
>>h=deconv(g,f)
h =
0.6667 -0.4444
```

et le polynôme h(x) obtenu est :

$h(x)= 0.6667\ x-0.4444$

3. 2.Manipulation de fonctions polynomiales dans MATLAB

Soit le polynôme suivant :

$P(x)= a_n\ x^n + a_{n-1}\ x^{n-1} +\ldots.+ a_2\ x^2 + a_1\ x + a_0$

Où n est le degré du polynôme et a_i (i=0 ,1 ,2, ….,n) sont les coefficients du polynôme.

Ce polynôme peut être écrit sous la forme :

$P(x)= ((\ldots(\ldots(a_n\ x + a_{n-1}\ x^{n-1})\ x + a_{n-2}\ )\ x\ \ldots. + a_1\ )x + a_0)$

Après factorisation, on a:

$$P(x)= a_n(x- r_1)(x- r_2)\ (x- r_3)\ \ldots(x- r_n)$$

où $r_0, r_1, r_2, \ldots, r_n$ sont les racines du polynôme $P(x)$.

Exemple :

$P(x)= x^4+2x^3-7x^2+8x+12$

Ce polynôme est équivalent à :

$$P(x)= (((\ (x+2)x-7)x-8)x+12)$$

Un polynôme d'ordre n possède n racines qui peuvent être réelles ou complexes.

Dans MATLAB, un polynôme est représenté par un vecteur contenant les coefficients dans un ordre décroissant.

Exemple :

Le polynôme : $2x^3+x^2+4x+5$ qui est représenté dans MATLAB par :

```
>>P=[2 1 4 5];
```

a pour racines r_i .

Pour trouver ces racines, on doit exécuter la fonction **'roots'**.

D'où :

```
>>r=roots(P);
>> r
r =
0.2500 + 1.5612i
0.2500 - 1.5612i
-1.0000
```

Les trois racines de ce polynôme (dont 2 sont complexes) sont données sous forme d'un vecteur colonne.

4. Résolution d'équations non linéaires (Méthode de Newton-Raphson)

4.1. La méthode de Newton-Raphson

La méthode de Newton-Raphson consiste à trouver la valeur x qui annulera la fonction f(x).

La méthode de Newton-Raphson permet d'approcher par itérations la valeur x au moyen de la relation suivante :

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

Si $|X_n - X_{n-1}| < \varepsilon$ alors X_n est le résultat de l'estimation de la racine.

Où ε représentent des erreurs d'approximations caractérisant la qualité de la solution numérique.

Ce critère d'arrêt a l'avantage d'éviter une possible division par 0.

.

Dans toutes les méthodes itératives, il est nécessaire pour éviter une divergence de la solution, de bien choisir la valeur initiale x_0. Celle-ci peut être obtenue graphiquement.

Exemple :

On se propose d'appliquer cette méthode pour la recherche des racines de la fonction non linéaire suivante :

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

Dans un premier temps, on se propose de tracer la courbe représentative de cette fonction en utilisant le programme ci-dessous 'NewtonRaphson.m':

```
% Etude de la fonction :
% f(x)=exp(x)-2.cos(x)
x=-1:0.1:1;
f=exp(x)-2*cos(x);
plot(x,f); grid on;
title('Fonction : f(x)=exp(x)-2.cos(x)');
```

Après exécution du programme, on obtient la courbe sur la figure ci-après. D'après cette courbe, il est judicieux de choisir un x_0= 0 ,5 ; car f (0 ,5) est proche de zéro pour avoir une convergence rapide. La fonction dérivée f'(x) a pour expression : $f'(x) = e^x + 2\sin(x)$.

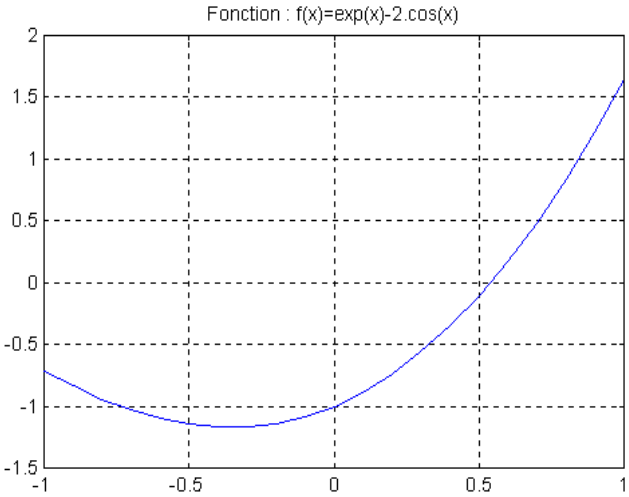


Figure III.1 : Localisation de la valeur x_0 où $f(x_0) \cong 0$

Pour chercher la solution de f(x), on peut rajouter au programme précédent 'NewtonRaphson.m' quelques lignes :

```
% Etude de la fonction :
% f(x)=exp(x)-2.cos(x)
clf;
x=-1:0.1:1;
f=exp(x)-2*cos(x);
figure(1);
plot(x,f); grid on;
title('Fonction : f(x)=exp(x)-2.cos(x)');
clear all;
clc;
x(1)=input('Donner la valeur initiale x(1): \n');
e=1e-10;
n=5000;
for i=2:n
    f=exp(x(i-1))-2*cos(x(i-1));
    diff=exp(x(i-1))+2*sin(x(i-1));
    x(i)=x(i-1)-f/diff;
    if abs(x(i)-x(i-1))<=e
        xp=x(i);
        fprintf('xp=%f\n',x(i));
        break;
    end
end
j=1:i;
figure(2);
plot(j,x(j),'*r',j,x(j));
xlabel('Nombre d''itérations');
title('Convergence de la solution : Méth. de Newt.-Raph.');
disp('Les valeurs successives de x(i) sont :');
x'
```

Code MATLAB implémentant la méthode NewtonRaphson

4.2. La méthode de la sécante (méthode multi-point)

Cette méthode est du type

$x_{n+1} = F(x_n, x_{n-1}, \cdots, x_{n-N})$

La méthode de Newton est rapide, mais nécessite le calcul de la dérivée de f en tout point x_n, ce qu'on n'a pas toujours. La plus simple et plus ancienne est la méthode de la sécante. Elle consiste à se donner deux points x_0 et x_1, tracer la droite qui passe par les points $(x_0, f(x_0))$ et $(x_1, f(x_1))$, elle coupe l'axe des x en x_2, et on recommence avec les points x_1 et x_2.

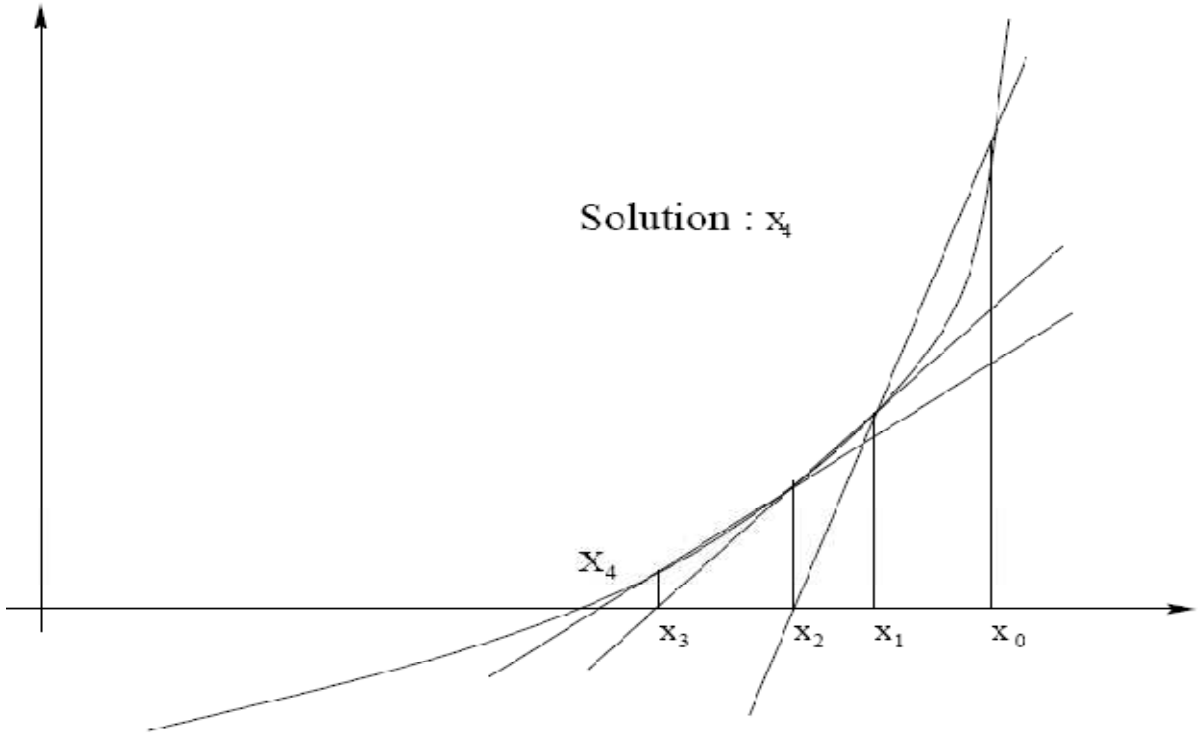


Figure III.2 : Méthode de la sécante

La méthode de la sécante permet d'approcher par itérations la valeur x au moyen de la relation suivante :

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

La solution dépend du critère d'arrêt

5. Intégration numérique des fonctions

Nous développons ci-après quelques méthodes qui permettent de calculer, sur un intervalle fini [a,b], l'intégrale définie $\int_a^b f(x)dx$ d'une fonction f continue donnée.

Ces méthodes sont particulièrement utiles dans le cas où les primitives de f ne sont pas des fonctions élémentaires ou sont trop difficiles à calculer.

Nous distinguerons deux optiques :

* la fonction à intégrer est remplacée par une fonction interpolante ou par une fonction d'approximation ;

* l'intégrale est approchée par une somme pondérée de valeurs prises par la fonction en des points situés dans un voisinage de [a,b].

5. 1 Méthode des trapèzes

On subdivise l'intervalle [a,b] en sous- intervalles $\{[x_{i-1},x_i]$, i = 1,2,…, n; $x_0 = a$; $x_n = b\}$ sur lesquels la fonction f est remplacée par le segment de droite qui joint les points $(x_{i-1}$, $f(x_{i-1}))$ et $(x_i$, $f(x_i))$.

Cette procédure revient à remplacer, sur [a,b], f par une fonction d'interpolation linéaire par morceaux. D'un point de vue géométrique, on assimile l'aire comprise entre le graphe de f et l'axe des x à la somme des aires de n trapèzes.

Considérons que la division en sous-intervalles est uniforme et posons :

$x_i = a + ih$ où $h = \frac{b-a}{n}$ et $f(x_i) = f_i$; i = 0, 1, 2, …, n.

Sur l'intervalle $[x_i , x_{i+1}]$ l'aire $\int_{x_i}^{x_{i+1}} f(x)dx$ est remplacée par $h(f_i + f_{i+1})/2$, aire du trapèze correspondant.

En additionnant les aires des n trapèzes, on obtient la formule des trapèzes :

$$\int_a^b f(x)dx \approx \frac{h}{2}(f_0 + 2 f_1 + 2 f_2 + \cdots + 2 f_{n-1} + f_n)$$

On peut montrer que l'erreur commise est proportionnelle à h^2 (si la fonction f est deux fois continûment dérivable).

On dit que la méthode des trapèzes est d'ordre 2. La formule est « exacte » pour les fonctions f de degré ≤ 1. La formule des trapèzes peut être améliorée en procédant comme suit, on développe f(x) en série de Taylor au voisinage de x_0 :

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{1}{2!}(x - x_0)^2 f''(x_0) + \ldots$$

En intégrant de x_0 à x_1, on obtient :

$$\int_{x_0}^{x_1} f(x)dx = hf(x_0) + \frac{h^2}{2!} f'(x_0) + \frac{h^3}{3!} f''(x_0) + \ldots$$

Comme : $f_1 = f(x_1) = f_0 + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + \ldots$

en multipliant par h/2 , on a : $\frac{h^2}{2!} f'(x_0) = \frac{h}{2}(f_1 - f_0) - \frac{h^3}{4} f''(x_0) + \ldots$

et l'intégrale devient :

$$\int_{x_0}^{x_1} f(x)dx = hf(x_0) + [\frac{h}{2}(f_1 - f_0) - \frac{h^3}{4} f''(x_0)] + [\frac{h^3}{3!} f''(x_0)] + \ldots = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12} f''(x_0) + \ldots$$

L'intégrale totale vaut :

$$\int_{x_0}^{x_n} f(x)dx \approx \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \ldots + 2f_{n-1} + f_n] - \frac{h^2}{12}\sum_{k=0}^{n-1} hf''(x_k)$$

Comme le dernier terme est une approximation de $\int_{x_0=a}^{x_n=b} f''(x)dx$ en le remplaçant par $f'(b) - f'(a)$, on obtient la formule des trapèzes améliorée :

$$\int_a^b f(x)dx \approx \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \ldots + 2f_{n-1} + f_n] - \frac{h^2}{12}[f'(b) - f'(a)]$$

L'erreur commise est proportionnelle à h^4 , et cette méthode est donc d'ordre 4.

Cette formule montre également que la formule des trapèzes est bien d'ordre 2 et que son erreur est donnée par :

$$E(h) = -\frac{h^2}{12}(b-a) f''(\xi) \quad , \quad \xi \in [a,b].$$

On peut également faire un programme pour calculer l'intégrale I.

Celui-ci appelé 'trapez.m' par exemple, est listé ci-dessous :

```
function I=trapez_v(f,h)
I=(sum(f)-(f(1)+f(length(f)))/2)*h;
```

Exemple :

Considérons par exemple la fonction à intégrer : $f(x) = x^2 + 2x - 1$ sur un intervalle [-10,8] où le pas est "h" égal à 1. En mode interactif dans MATLAB (mode commande), avant de lancer le programme 'trapez.m', on donne la fonction f ainsi que ses bornes :

```
>>x=-10:1:8;
>>f=x.^2+2*x-1;
>>h=1;
```

On exécute ensuite le programme 'trapez.m', et on obtient :

```
>>I=trapez_v(f,h)
I =
453
```

5. 2 Méthode de Simpson

C'est une méthode qui équivaut à remplacer la fonction à intégrer par des paraboles définies sur des sous-intervalles comprenant trois abscisses d'intégration successives.

On suppose que l'intervalle [a,b] est partagé en n sous-intervalles égaux :

$[x_{i-1}, x_i]$,tels que $x_i = a + ih$, avec $h = (b-a)/n$.

On groupe les points par trois, n doit donc être pair

$a = x_0, x_1, x_2 \mid x_2, x_3, x_4 \mid \ldots \mid x_{n-2}, x_{n-1}, x_n \mid = b.$

Et on remplace, sur chaque intervalle $[x_{i-1}, x_{i+1}]$, la fonction f par une parabole.

Pour l'intervalle $[x_0, x_2]$, la courbe représentée par f(x) est approchée par la parabole d'équation :

$$p(x) = f_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

$$= f_0 \frac{(x-x_1)(x-x_2)}{2h^2} + f_1 \frac{(x-x_0)(x-x_2)}{-h^2} + f_2 \frac{(x-x_0)(x-x_1)}{2h^2}$$

et l'intégrale est alors approchée par : $\int_{x_0}^{x_2} f(x)dx \approx \int_{x_0}^{x_2} p(x)dx = \frac{h}{3}[f_0 + 4f_1 + f_2]$

En répétant ce procédé pour les n (pair !) sous-intervalles, on a finalement la

Formule de Simpson : $$\int_a^b f(x)dx \approx \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 2f_{n-2} + 4f_{n-1} + f_n]$$

Cette formule est exacte pour les polynômes f(x) de degré ≤ 3.

Tout comme pour la méthode des trapèzes, on peut obtenir une formule de Simpson améliorée :

$$\int_a^b f(x)dx \approx \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 2f_{n-2} + 4f_{n-1} + f_n] - \frac{h^4}{180}[f'''(b) - f'''(a)]$$

L'erreur de cette formule est de l'ordre de h^6.

La méthode de Simpson est donc d'ordre 4 et son erreur est donnée par :

$$E(h) = -\frac{h^4}{180}(b-a)f^{(4)}(\xi) \quad , \quad \xi \in [a,b].$$

6. Applications

1. Ecrire le code MATLAB qui permet de résoudre le système d'équations linéaire par la méthode de Gauss-Seidel.

2. Ecrire le code MATLAB qui permet de résoudre l'équation $xe^x = 1$par la méthode de la sécante.

3. Ecrire le code MATLAB qui permet d'évaluer l'intégrale $f(x) = \sqrt{1 + e^x}$ sur l'intervalle [0 ,2] avec la méthode de Simpson pour n=2 , n=4 , n=8 et n=16 .

Chapitre IV : Calcul des structures (barres et poutres) selon la Méthode des Éléments Finis (MEF) par MATLAB

1. Introduction

La méthode des Eléments Finis consiste à découper la structure en éléments de forme simple et à choisir une approximation du déplacement sur chaque subdivision.

Pour analyser un phénomène naturel en général ou un problème d'ingénierie en particulier, on est souvent amené à développer un modèle mathématique pouvant décrire d'une manière aussi fiable que possible le problème en question.

Avec les progrès enregistrés dans le domaine de l'informatique et les performances des ordinateurs de plus en plus grandes ; plusieurs techniques de résolution numérique ont été ainsi développées et appliquées avec succès pour avoir des solutions satisfaisantes à des problèmes d'ingénierie très variés.

La méthode des éléments finis est l'une des techniques numériques les plus puissantes.

L'un des avantages majeurs de cette méthode est le fait qu'elle offre la possibilité de développer un programme permettant de résoudre, avec peu de modifications, plusieurs types de problèmes. En particulier, toute forme complexe d'un domaine géométrique où un problème est bien posé avec toutes les conditions aux limites, peut être facilement traité par la méthode des éléments finis.

2. Elément Barre

L'élément barre linéaire est un élément fini unidimensionnel où les coordonnées globale et locale coïncident. Il schématise un composant [IJ] d'une structure qui travaille uniquement en traction ou compression. L'élément linéaire possède un module d'élasticité E, une aire de section transversale A, et la longueur L.

Chaque élément de barre linéaire a deux nœuds comme le montre la Figure 5.1. Dans ce cas, la matrice de rigidité de l'élément est donnée par :

$$K=\begin{bmatrix} \frac{EA}{L} & -\frac{EA}{L} \\ -\frac{EA}{L} & \frac{EA}{L} \end{bmatrix}$$

Figure 5.1 : Elément Barre

La matrice de rigidité pour l'élément de barre linéaire est similaire à celle de l'élément de ressort avec la rigidité remplacée par EA / L. L'élément barre linéaire a seulement deux degrés de liberté, une à chaque nœud. En conséquence, pour une structure à n nœuds, la matrice de rigidité globale K sera de taille n × n (puisque nous avons un degré de liberté à chaque nœud).

Une fois la matrice de rigidité globale K est obtenue, nous avons l'équation de la structure suivante:

$$[K]\{U\} = \{F\}$$

Où U est le vecteur de déplacement nodal global et F est le vecteur de force nodale globale. À cette étape, les conditions aux limites sont appliquées manuellement pour les vecteurs U et F ; ce système d'équation est résolu par la séparation et l'élimination de Gauss. Enfin, une fois les déplacements et les réactions inconnues sont détectés, les forces sont obtenues pour chaque élément comme suit:

$$\{f\} = [k]\{u\}$$

Où f est le vecteur- force de dimension (2 x 1) de l'élément et u est le vecteur- déplacement de dimension (2 × 1) de l'élément. Les tensions dans les éléments sont obtenues en divisant les forces par les sections transversales correspondantes A.

Les fonctions MATLAB utilisées pour le calcul des éléments barres linéaires sont:

Calcul de la matrice de rigidité de chaque élément barre

```
function y = LinearBarStiffness(E,A,L)
% E module d'élasticité,  L longueur,  A  section de l'élément
% la dimension de la matrice de rigidité est 2x2
y = [E*A/L -E*A/L ; -E*A/L E*A/L];
```

Assemblage des matrices de rigidité

```
function y = LinearBarAssemble(K,k,i,j)
K(i,i) = K(i,i) + k(1,1) ;
K(i,j) = K(i,j) + k(1,2) ;
K(j,i) = K(j,i) + k(2,1) ;
K(j,j) = K(j,j) + k(2,2) ;
y = K;
```

Calcul du vecteur de force pour chaque élément barre

```
function y = LinearBarForces(k,u)
y = k * u;
```

Calcul de la contrainte

```
function y = LinearBarStresses(k, u, A)
% vecteur de déplacement u, A  section de l'élément
y = k * u/A;
```

Par Exemple :

Considérons l'ensemble, constitué de deux barres linéaires comme le montre la Figure ci-dessous

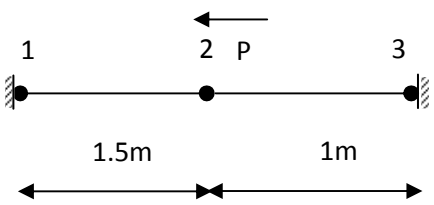


Figure 5.2 : Elément de deux-Barres pour l'exemple 1

Données :

E = 210 GPa, A = 0.003m2, P = 10 kN, déplacement du nœud 3 est de 0,002 m vers la droite.

Déterminer:

1. la matrice de rigidité globale de la structure.
2. le déplacement au noeud 2.
3. les réactions aux nœuds 1 et 3.
4. La contrainte dans chaque barre.

Solution:

Étape 1 - La discrétisation du domaine:

Ce problème est déjà discrétisé. Le domaine est divisé en deux éléments et trois noeuds. Les unités utilisées dans les calculs MATLAB sont kN et le mètre. Le Tableau ci-dessous montre la connexion de l'élément pour cet exemple.

Element	**Noeud 1**	**Noeud 2**
1	**1**	**2**
2	**2**	**3**

Étape 2 – calcul des matrices de rigidité

Les deux matrices de rigidité k1 et k2 sont obtenues en faisant appel à la fonction Linear Bar Stiffness de MATLAB. Chaque matrice a la taille 2 × 2.

```
» E=210e6;
» A=0.003;
» L1=1.5;
» L2=1;
» k1= LinearBarStiffness (E,A,L1)

k1 =

    420000   -420000
   -420000    420000

» k2= LinearBarStiffness (E,A,L2)

k2 =

    630000   -630000
   -630000    630000
```

Étape 3 – Assemblage de la matrice de rigidité globale:

Étant donné que la structure comporte trois nœuds, la taille de la matrice de rigidité globale est de 3 x 3.

Par conséquent, pour obtenir K nous avons d'abord mis en place une matrice nulle de taille 3 × 3, puis faire deux appels à la fonction LinearBarAssemble MATLAB puisque nous avons deux éléments de barre linéaires dans la structure. Chaque appel à la fonction assemble un élément. Ce qui suit sont les commandes MATLAB:

```
» K=zeros(3,3)

K =

   0    0    0
   0    0    0
   0    0    0

»  K= LinearBarAssemble (K,k1,1,2)

K =

    420000   -420000         0
   -420000    420000         0
         0         0         0

» K= LinearBarAssemble (K,k2,2,3)

K =

    420000   -420000         0
   -420000   1050000   -630000
         0   -630000    630000
```

Étape 4 - Application des conditions aux limites:

En utilisant la matrice de rigidité globale obtenue à l'étape précédente, on obtient le système d'équation suivant :

$$\begin{bmatrix} 420000 & -420000 & 0 \\ -420000 & 1050000 & -630000 \\ 0 & -630000 & 630000 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix}$$

Les conditions aux limites pour ce problème sont données à titre:

U1 = 0, F2 = -10, U3 = 0,002

nous obtenons:

$$\begin{bmatrix} 420000 & -420000 & 0 \\ -420000 & 1050000 & -630000 \\ 0 & -630000 & 630000 \end{bmatrix} \begin{Bmatrix} 0 \\ U_2 \\ 0.002 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ -10 \\ F_3 \end{Bmatrix}$$

Étape 5 - la résolution des équations:

La résolution du système d'équations obtenu sera effectuée par partitionnement (manuellement) et l'élimination de Gauss (avec MATLAB).

Nous obtenons:

[1050000] U2 + [-630 000] (0,002) = {-10}

La solution du système ci-dessus est obtenue en utilisant MATLAB comme suit. Notez que l'opérateur barre oblique inverse "\" est utilisé pour l'élimination de Gauss.

```
» k=K(2,2);
» k0=K(2,3);
» u0=0.002;
» f=[-10];
» f0=f-k0*u0

f0 =

      1250

» u=k\f0

u =

   0.0012
```

Le déplacement au nœud 2 est 0.0012 m.

Étape 6 - Post-traitement:

Dans cette étape, nous obtenons les réactions au niveau des nœuds 1 et 3, et la contrainte dans chaque barre en utilisant MATLAB comme suit. Nous avons d'abord mis en place le vecteur de déplacement nodal U global, puis nous calculons le vecteur de force nodale global F.

```
»  U=[0 ; u ; u0]

U =

        0
   0.0012
   0.0020

» F=K*U

F =

 -500.0000
  -10.0000
  510.0000
```

Ainsi, les réactions au niveau des nœuds 1 et 3 sont des forces de 500 kN (dirigés vers la gauche) et 510 kN (dirigés vers la droite), respectivement. Il est clair que la condition d'équilibre statique est satisfaite.
Ensuite, nous mettons en place l'élément nodal vecteurs déplacement U1 et U2, puis nous calculons les vecteurs de force f1 et f2 en faisant appel à la fonction « LinearBarForces » de MATLAB. Enfin, on divise chaque élément de la force par l'aire de section transversale de l'élément pour obtenir les contraintes dans l'élément.

```
u1=[0 ; U(2)]

u1 =

        0
   0.0012

» f1=LinearBarForces(k1,u1)

f1 =

 -500.0000
  500.0000

» sigma1=f1/A

sigma1 =

  1.0e+005 *

   -1.6667
    1.6667
```

```
» u2=[U(2) ; U(3)]

u2 =

   0.0012
   0.0020

» f2= LinearBarForces (k2,u2)

f2 =

 -510.0000
  510.0000

» sigma2=f2/A

sigma2 =

  1.0e+005 *

   -1.7000
    1.7000
```

La contrainte dans l'élément 1 est 1.667×10^5 kN / m^2 (166.7 MPa, traction) et la contrainte dans l'élément 2 est de 1.7×10^5 kN / m^2 (170MPa, traction). Alternativement, nous pouvons obtenir les contraintes élémentaires directement en faisant appel aux fonctions MATLAB « LinearBarStresses ». On obtient les mêmes résultats que ci-dessus.

```
» s1= LinearBarStresses (k1,u1,A)

s1 =

  1.0e+005 *

   -1.6667
    1.6667

» s2= LinearBarStresses (k2,u2,A)

s2 =

  1.0e+005 *

   -1.7000
    1.7000
```

3. Structures planes à treillis

Les structures à treillis sont constituées par des assemblages de barres liées par des joints de telle sorte que le chargement extérieur soit repris uniquement par des forces axiales dans les barres. La Figure 5.3 montre un exemple de système à treillis composé d'un assemblage de 13 barres et soumis à un chargement de deux forces.

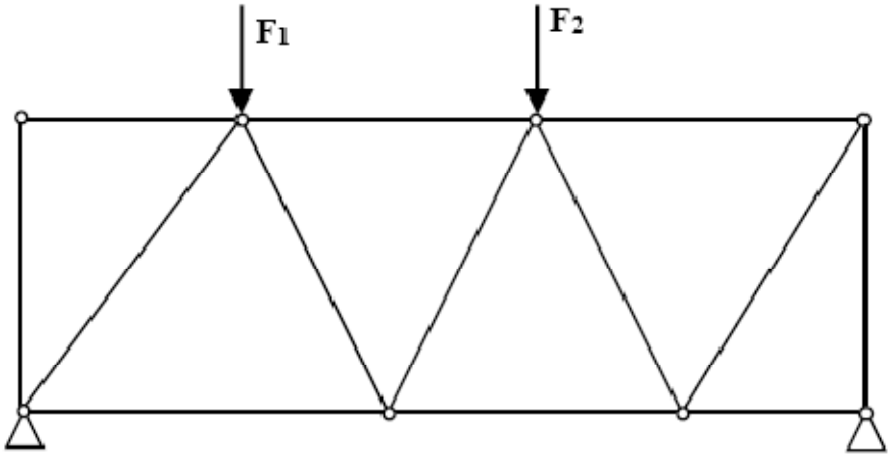


Figure 5.3 : Système à treillis

L'élément plan de treillis est un élément fini à deux dimensions avec les deux coordonnées locales et globales. Il est caractérisé par des fonctions de forme linéaire. L'élément plan en treillis a un module d'élasticité E, aire de section transversale A, et la longueur L. Chaque élément de treillis plan a deux nœuds est incliné avec un angle θ mesuré dans le sens antihoraire à partir de l'axe X positif global comme le montre la Figure 5.4 .Soit C = cos θ et S = sin θ. Dans ce cas, la matrice de rigidité de l'élément est donnée par :

$$k=\frac{EA}{L}\begin{bmatrix} C^2 & CS & -C^2 & -CS \\ CS & S^2 & -CS & -S^2 \\ -C^2 & -CS & C^2 & CS \\ -CS & -S^2 & CS & S^2 \end{bmatrix}$$

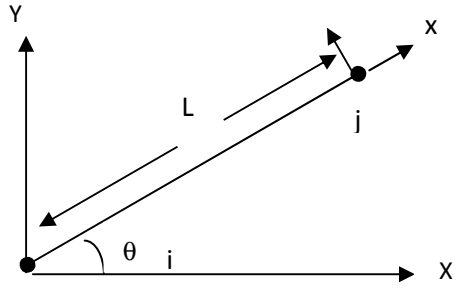


Figure 5.4 : Elément d'un treillis

L'élément plan de treillis a quatre degrés de liberté - deux à chaque noeud. En conséquence, pour une structure à n nœuds, la matrice de rigidité globale K sera de taille 2n × 2n (puisque nous avons deux degrés de liberté à chaque nœud). La matrice de rigidité K globale est assemblée en faisant des appels à la fonction MATLAB.

Une fois la matrice de rigidité globale K est obtenue, nous avons l'équation de la structure suivante:

$$[K]\{U\} = \{F\}$$

Où U est le vecteur de déplacement global nodal et F est vecteur de force global nodale. A cette étape, les conditions aux limites sont appliquées manuellement pour les vecteurs U et F. Puis le système d'équations linéaires obtenue est résolu par la séparation et l'élimination de Gauss. Enfin, une fois les déplacements et les réactions inconnues sont calculés, la force est obtenue pour chaque élément de la façon suivante:

$$f = \frac{EA}{L}[-C - S \;\; C \;\; S]\{u\}$$

Où f est la force dans l'élément (un scalaire) et u est le vecteur- déplacement de dimension (4 × 1) élément de déplacement vecteur. La contrainte de l'élément est obtenue en divisant la force de l'élément par sa section transversale A.

Les fonctions MATLAB utilisées pour le calcul des éléments treillis sont les suivantes :

Calcul de la longueur de l'élément en fonction des coordonnées du premier noeud (x1, y1) et du second nœud (x2, y2).

```
function y = Truss2dLength(x1,y1,x2,y2)
y = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
```

Calcul de la matrice de rigidité de chaque élément plan du treillis

```
function y = Truss2dStiffness (E,A,L, theta)
% E module d'élasticité,  L longueur,  angle theta (en degré), A
section de l'élément
% la dimension de la matrice de rigidité est 4x4
x = theta*pi/180;
C = cos(x);
S = sin(x);
y = E*A/L*[C*C C*S -C*C -C*S ; C*S S*S -C*S -S*S ; -C*C -C*S C*C C*S
; -C*S -S*S C*S S*S] ;
```

Assemblage des matrices de rigidité des éléments plan du treillis

```
function y = Truss2dAssemble(K,k,i,j)
K(2*i-1,2*i-1) = K(2*i-1,2*i-1) + k(1,1) ;
K(2*i-1,2*i) = K(2*i-1,2*i) + k(1,2) ;
K(2*i-1,2*j-1) = K(2*i-1,2*j-1) + k(1,3) ;
K(2*i-1,2*j) = K(2*i-1,2*j) + k(1,4) ;
K(2*i,2*i-1) = K(2*i,2*i-1) + k(2,1) ;
K(2*i,2*i) = K(2*i,2*i) + k(2,2) ;
K(2*i,2*j-1) = K(2*i,2*j-1) + k(2,3) ;
K(2*i,2*j) = K(2*i,2*j) + k(2,4) ;
K(2*j-1,2*i-1) = K(2*j-1,2*i-1) + k(3,1) ;
K(2*j-1,2*i) = K(2*j-1,2*i) + k(3,2) ;
K(2*j-1,2*j-1) = K(2*j-1,2*j-1) + k(3,3) ;
K(2*j-1,2*j) = K(2*j-1,2*j) + k(3,4) ;
K(2*j,2*i-1) = K(2*j,2*i-1) + k(4,1) ;
K(2*j,2*i) = K(2*j,2*i) + k(4,2) ;
K(2*j,2*j-1) = K(2*j,2*j-1) + k(4,3) ;
K(2*j,2*j) = K(2*j,2*j) + k(4,4) ;
y = K;
```

Calcul des forces

```
function y = Truss2dforce(E,L,theta,u)
% E module d'élasticité,  L longueur,  angle theta ( en degré),
vecteur de déplacement u
x = theta* pi/180;
C = cos(x);
S = sin(x);
y = E*A/L*[-C -S C S]* u;
```

Calcul des contraintes

```
function y = Truss2dStress(E,L,theta,u)
% E module d'élasticité,  L longueur,  angle theta ( en degré),
vecteur de déplacement u
x = theta * pi/180;
C = cos(x);
S = sin(x);
y = E/L*[-C -S C S]* u;
```

4. Elément Poutre

C'est un élément unidimensionnel [IJ] qui reprend toutes les hypothèses des poutres longues. Il intègre les énergies d'effort normal, d'effort tranchant, de flexion et de torsion.

L'élément poutre est un élément fini à deux dimensions où les coordonnées locale et globale coïncident. Il est caractérisé par des fonctions de forme linéaire. L'élément de poutre a un module d'élasticité E, moment d'inertie I, et la longueur L. Chaque élément de poutre comporte deux nœuds et est supposé être horizontal comme représenté sur la figure 5.5. Dans ce cas, la matrice de rigidité de l'élément est donnée par la matrice suivante, en supposant que la déformation axiale est négligée :

$$k=\frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & -6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}$$

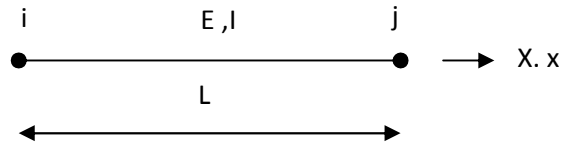


Figure 5.5 : Elément poutre

L'élément poutre a quatre degrés de liberté - deux à chaque nœud (un déplacement transversal et une rotation). La convention de signe utilisée est que le déplacement est positif si il pointe vers le haut et la rotation est positive si elle est dans le sens antihoraire. En conséquence, pour une structure avec n nœuds, la matrice de rigidité globale K sera de taille (2n × 2n) (puisque nous avons deux degrés de liberté à chaque nœud). La matrice de rigidité K globale est assemblée en faisant appels à la fonction « BeamAssemble » de MATLAB qui est écrite spécifiquement à cette fin. Ce procédé va être illustré en détail dans les exemples.

Les fonctions MATLAB utilisées pour le calcul des éléments poutres sont les suivantes :

Calcul de la matrice de rigidité de chaque élément poutre

```
function y = BeamStiffness(E,I,L)
% E module d'élasticité,  L longueur,  A  section de l'élément
% la dimension de la matrice de rigidité est 4x4
y = E*I/(L*L*L) * [12 6*L -12 6*L ; 6*L 4*L*L -6*L 2*L*L ; -12 -6*L
12 -6*L ; 6*L 2*L*L -6*L 4*L*L]
```

Assemblage des matrices de rigidité

```
function y = BeamAssemble(K,k,i,j)
K(2*i-1,2*i-1) = K(2*i-1,2*i-1) + k(1,1);
K(2*i-1,2*i) = K(2*i-1,2*i) + k(1,2);
K(2*i-1,2*j-1) = K(2*i-1,2*j-1) + k(1,3);
K(2*i-1,2*j) = K(2*i-1,2*j) + k(1,4);
K(2*i,2*i-1) = K(2*i,2*i-1) + k(2,1);
K(2*i,2*i) = K(2*i,2*i) + k(2,2);
K(2*i,2*j-1) = K(2*i,2*j-1) + k(2,3);
K(2*i,2*j) = K(2*i,2*j) + k(2,4);
K(2*j-1,2*i-1) = K(2*j-1,2*i-1) + k(3,1);
K(2*j-1,2*i) = K(2*j-1,2*i) + k(3,2);
K(2*j-1,2*j-1) = K(2*j-1,2*j-1) + k(3,3);
K(2*j-1,2*j) = K(2*j-1,2*j) + k(3,4);
K(2*j,2*i-1) = K(2*j,2*i-1) + k(4,1);
K(2*j,2*i) = K(2*j,2*i) + k(4,2);
K(2*j,2*j-1) = K(2*j,2*j-1) + k(4,3);
K(2*j,2*j) = K(2*j,2*j) + k(4,4);
y = K;
```

Calcul du vecteur de force pour chaque élément poutre

```
function y = BeamForces(k,u)
y = k * u;
```

Calcul du diagramme des moments fléchissants

```
function y = BeamShearDiagram(f, L)
x = [0 ; L];
z = [f(1) ; -f(3)];
hold on;
title(' Diagramme des moments fléchissants ');
plot(x,z);
y1 = [0 ; 0];
plot(x,y1,'k')
```

Calcul du diagramme des efforts tranchants

```
function y = BeamMomentDiagram(f, L)
x = [0 ; L];
z = [-f(2) ; f(4)];
hold on;
title(' Diagramme des efforts tranchants');
plot(x,z);
y1 = [0 ; 0];
plot(x,y1,'k')
```

Par exemple :

Considérons la poutre représentée sur la figure 5.7 ci-dessous.

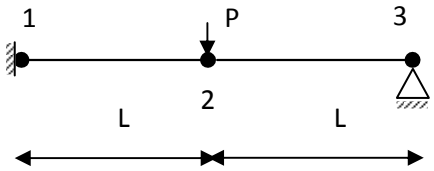


Figure 5.7: Poutre avec deux éléments

Données :

E = 210GPa, I = 60 × 10-^{6} m^{4}, P = 20 kN, et L = 2m,

Déterminer:

1. la matrice de rigidité globale de la structure.
2. le déplacement vertical au nœud 2.
3. les rotations aux nœuds 2 et 3.
4. les réactions aux nœuds 1 et 3.
5. les forces (efforts tranchants et moments) dans chaque élément.
6. le diagramme des efforts tranchants pour chaque élément.
7. le diagramme des moments de flexion pour chaque élément.

Solution:

Étape 1 - La discrétisation du domaine:

Nous allons mettre un nœud (nœud 2) à l'emplacement de la force concentrée afin que nous puissions déterminer les déplacements, les rotations, les efforts tranchants et les moments) en ce point. Alternativement, nous pouvons considérer la structure comme composée d'un élément de poutre et seulement deux nœuds et utiliser les forces nodales équivalentes pour prendre en compte la charge concentrée.

Cependant, il est nécessaire de trouver le déplacement vertical sous la charge concentrée. Par conséquent, le domaine est divisé en deux éléments, et trois nœuds. Les unités utilisées dans les calculs **MATLAB** sont kN et le mètre. Le Tableau 7.1 montre la connexion de l'élément pour cet exemple.

Element	**Noeud 1**	**Noeud 2**
1	**1**	**2**
2	**2**	**3**

Étape 2 -Calcul des matrices de rigidité:

Les deux matrices de rigidité élément (k1 et k2) sont obtenues en faisant appel à la fonction « BeamStiffness » de MATLAB. Chaque matrice a une taille 4 × 4.

```
» E=210e6;
» I=60e-6;
» L=2;
» k1= BeamStiffness (E,I,L)
k1 =

    18900    18900   -18900    18900
    18900    25200   -18900    12600
   -18900   -18900    18900   -18900
    18900    12600   -18900    25200
» k2= BeamStiffness (E,I,L)
k2 =

    18900    18900   -18900    18900
    18900    25200   -18900    12600
   -18900   -18900    18900   -18900
    18900    12600   -18900    25200
```

Étape 3 - assemblage de la matrice de rigidité globale:

Étant donné que la structure comporte trois nœuds, la taille de la matrice de rigidité globale est de 6 x 6.

Par conséquent, pour obtenir K nous avons d'abord mis en place une matrice nulle de taille 6 × 6, puis faire deux appels à la fonction Beam Assemble MATLAB puisque nous avons deux éléments de poutre de la structure. Chaque appel à la fonction assemble un élément. Ce qui suit sont les commandes MATLAB :

```
» K=zeros(6,6)

K =

     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

» K= BeamAssemble (K,k1,1,2)

K =

    18900    18900   -18900    18900        0        0
    18900    25200   -18900    12600        0        0
   -18900   -18900    18900   -18900        0        0
    18900    12600   -18900    25200        0        0
        0        0        0        0        0        0
        0        0        0        0        0        0

» K= BeamAssemble (K,k2,2,3)

K =

    18900    18900   -18900    18900        0        0
    18900    25200   -18900    12600        0        0
   -18900   -18900    37800        0   -18900    18900
    18900    12600        0    50400   -18900    12600
        0        0   -18900   -18900    18900   -18900
        0        0    18900    12600   -18900    25200
```

Étape 4 - Application des conditions aux limites:

La matrice pour cette structure est obtenue de la manière suivante en utilisant la matrice de rigidité globale obtenue à l'étape précédente:

$$10^3 \begin{bmatrix} 18.9 & 18.9 & -18.9 & 18.9 & 0 & 0 \\ 18.9 & 25.2 & -18.9 & 12.6 & 0 & 0 \\ -18.9 & -18.9 & 37.8 & 0 & -18.9 & 18.9 \\ 18.9 & 12.6 & 0 & 50.4 & -18.9 & 12.6 \\ 0 & 0 & -18.9 & -18.9 & 18.9 & -18.9 \\ 0 & 0 & 18.9 & 12.6 & -18.9 & 25.2 \end{bmatrix} \begin{Bmatrix} U_{1y} \\ \Phi_1 \\ U_{2y} \\ \Phi_2 \\ U_{3y} \\ \Phi_3 \end{Bmatrix} = \begin{Bmatrix} F_{1y} \\ M_1 \\ F_{2y} \\ M_2 \\ F_{3y} \\ M_3 \end{Bmatrix}$$

Insertion des conditions ci-dessous nous obtenons:

U1y = φ1 = 0, F2y = −20, M2 = 0, U3y = 0, M3 = 0

$$10^3 \begin{bmatrix} 18.9 & 18.9 & -18.9 & 18.9 & 0 & 0 \\ 18.9 & 25.2 & -18.9 & 12.6 & 0 & 0 \\ -18.9 & -18.9 & 37.8 & 0 & -18.9 & 18.9 \\ 18.9 & 12.6 & 0 & 50.4 & -18.9 & 12.6 \\ 0 & 0 & -18.9 & -18.9 & 18.9 & -18.9 \\ 0 & 0 & 18.9 & 12.6 & -18.9 & 25.2 \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ U_{2y} \\ \Phi_2 \\ 0 \\ \Phi_3 \end{Bmatrix} = \begin{Bmatrix} F_{1y} \\ M_1 \\ -20 \\ 0 \\ F_{3y} \\ 0 \end{Bmatrix}$$

Étape 5 - la résolution des équations:

La résolution du système d'équations sera effectuée par partitionnement (manuellement) et l'élimination de Gauss (avec MATLAB). Nous avons d'abord partager le système d'équations par l'extraction des sous-matrices en lignes 3-4 et 3-4 colonnes, lignes 3-4 et de colonne 6, ligne 6 et colonnes 3-4, et de la ligne et de la colonne 6. On obtient donc:

$$\begin{bmatrix} 37.8 & 0 & 18.9 \\ 0 & 50.4 & 12.6 \\ 18.9 & 12.6 & 25.2 \end{bmatrix} \begin{Bmatrix} U_{2y} \\ \Phi_2 \\ \Phi_3 \end{Bmatrix} = \begin{Bmatrix} -20 \\ 0 \\ 0 \end{Bmatrix}$$

La solution du système ci-dessus est obtenue en utilisant MATLAB comme suit. Notez que l'opérateur barre oblique inverse "\" est utilisé pour l'élimination de Gauss.

```
» k=[K(3:4,3:4) K(3:4,6) ; K(6,3:4) K(6,6)]

k =

     37800         0        18900
         0     50400        12600
     18900     12600        25200

»  f=[-20 ; 0 ; 0]

f =

   -20
     0
     0

» u=k\f

u =

  1.0e-003 *

   -0.9259
   -0.1984
    0.7937
```

Il est maintenant clair que le déplacement vertical au noeud 2 est 0.9259m (vers le bas), tandis que les rotations aux nœuds 2 et 3 sont 0,1984 rad (sens horaire) et 0,7937 rad (sens anti-horaire), respectivement.

Étape 6 - Post-traitement:

Dans cette étape, nous obtenons les réactions au niveau des nœuds 1 et 3, les efforts tranchants et les moments dans chaque élément de poutre à l'aide de MATLAB comme suit. Nous avons d'abord mis en place le vecteur de déplacement nodal U global, puis nous calculons le vecteur global de force nodale F.

```
» U=[0 ; 0 ; u(1) ; u(2) ; 0 ; u(3)]

U =

  1.0e-003 *

        0
        0
   -0.9259
   -0.1984
        0
    0.7937

» F=K*U

F =

    13.7500
    15.0000
   -20.0000
        0
     6.2500
    -0.0000
```

Ainsi, les réactions au noeud 1 sont une force verticale de 13,75 kN (vers le haut) et un moment de 15 kN.m (sens anti-horaire) tandis que la réaction au nœud 3 est une force verticale de 6,25 kN (vers le haut). Il est clair que la condition d'équilibre statique est satisfaite. Ensuite, nous mettons en place les vecteurs déplacements nodaux U1 et U2 de l'élément, puis nous calculons les vecteurs forces nodales f1 et f2 de l'élément en faisant appels à la fonction « BeamForces » de MATLAB.

```
» u1=[U(1) ; U(2) ; U(3) ; U(4)]

u1 =

  1.0e-003 *

        0
        0
   -0.9259
   -0.1984

» f1= BeamForces (k1,u1)

f1 =

   13.7500
   15.0000
  -13.7500
   12.5000

»  u2=[U(3) ; U(4) ; U(5) ; U(6)]

u2 =

  1.0e-003 *

   -0.9259
   -0.1984
        0
    0.7937

» f2= BeamForces (k2,u2)

f2 =

   -6.2500
  -12.5000
    6.2500
   -0.0000
```

Enfin, nous appelons les fonctions MATLAB « BeamShearDiagram » et « BeamMomentDiagram » pour tracer le diagramme des efforts tranchants et le diagramme des moments fléchissants, pour chaque élément.

5. Application

Considérons le treillis plan représenté sur la figure. 5.8.

Données :

E = 210GPa et A = 1×10^{-4} m^2.

Déterminer:

1. la matrice de rigidité globale de la structure.
2. le déplacement horizontal au nœud 2.
3. les déplacements horizontal et vertical au nœud 3.
4. les réactions aux nœuds 1 et 2.
5. La contrainte dans chaque élément.

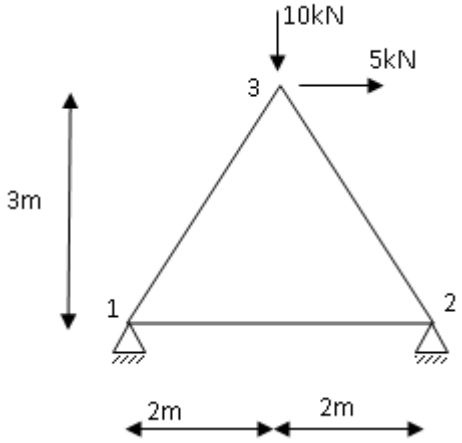


Figure 5.8: Treillis à trois éléments

Chapitre V : Calcul des structures sous effets dynamique et sismique par MATLAB.

1. Introduction

Un des mouvements les plus importants observés dans la nature est le mouvement oscillatoire, en particulier le mouvement harmonique : oscillations d'un pendule, d'une masse attachée à un ressort, d'un bâtiment, etc.

Dans le cas des oscillations de systèmes mécaniques conservatifs isolés, on parle d'oscillations libres ; en présence de frottement, l'amplitude des oscillations décroît et on observe des oscillations amorties. Si les oscillations sont entretenues par une action extérieure, on parle d'oscillations forcées.

Un système réel comprend généralement plusieurs masses reliées entre elles par des éléments de types ressort et amortisseur ce qui nous ramène à la notion de systèmes à plusieurs degrés de liberté.

Pour comprendre les principes de base, une suite de programmes informatiques dans **MATLAB** est donnée pour calculer les structures sous effets dynamique et sismique.

2. Systèmes à 1 degré de liberté libres non amortis

2.1 Équations du mouvement.

Soit une masse m supportée par un ressort k. On parle d'oscillations non amorties quand l'amortissement est nul, c'est-à-dire c=0.

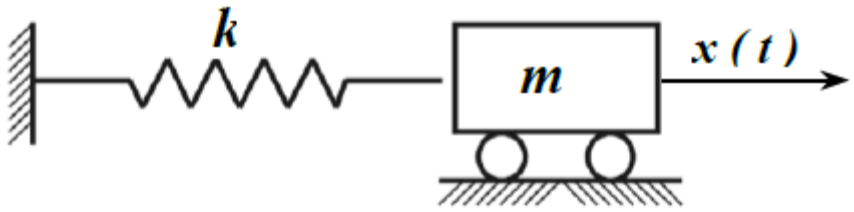


Figure 4.1 : Système non amorti

Bilan des forces en position déformée :

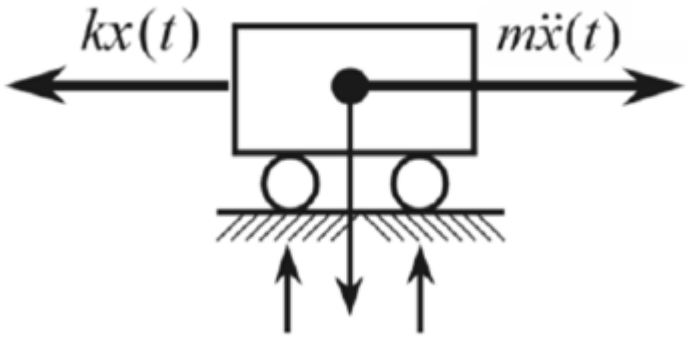


Figure 4.2: Forces présentes dans un système non amorti

Ainsi, selon la loi de Newton :

$$\sum F_x = m\ddot{x}(t)$$

$$-kx(t) = m\ddot{x}(t)$$

$$m\ddot{x}(t) + kx(t) = 0 \qquad (1)$$

En posant

$$\omega_n = \sqrt{\frac{k}{m}}$$

l'équation (1) devient :

$$\ddot{x} + \omega_n^2 x = 0$$

La solution générale de l'équation :

$$x(t) = \sqrt{x_0^2 + \left[\frac{V_0}{\omega_n}\right]^2} \cos\left[\omega_n t - \text{arctg}\left[\frac{V_0}{\omega_n\ x_0}\right]\right]$$

2.2 Calcul de la réponse libre à 1 ddl

Soit un système masse-ressort où k= 50 N/m; m = 1.5 kg, excité par une vitesse initiale de 1m/s.

La fonction suivante ssdl1 écrite en MATLAB permet de calculer les réponses vibratoires d'un système non amorti à 1 ddl,

```
function[a,b,c,d,e]= ssdl(m,k,x0,v0,tf)
%ssdl(m,k,x0,v0,tf) trace la réponse libre. m et k  représentent
respectivement la masse et la raideur du système.les arguments x0 et
vo représentent les conditions initiales en déplacement et en
vitesse. tf représente le temps d'observation
w=sqrt(k/m) ;% pulsation naturelle non amortie
t=0:tf/1000:tf;
A=sqrt(((v0)^2+(x0*w)^2)/w^2);%Amplitude de l'exponentielle
Phi=atan2(x0*w, v0);% phase
x=A*sin(w*t+Phi); % réponse non amortie
plot(t,x)
xlabel('Temps (s)')
ylabel('Déplacement (m)')
title ('Réponse temporelle libre')
```

Après exécution, on obtient :

```
» ssdl1 (1.5,50,0,1,10)
```

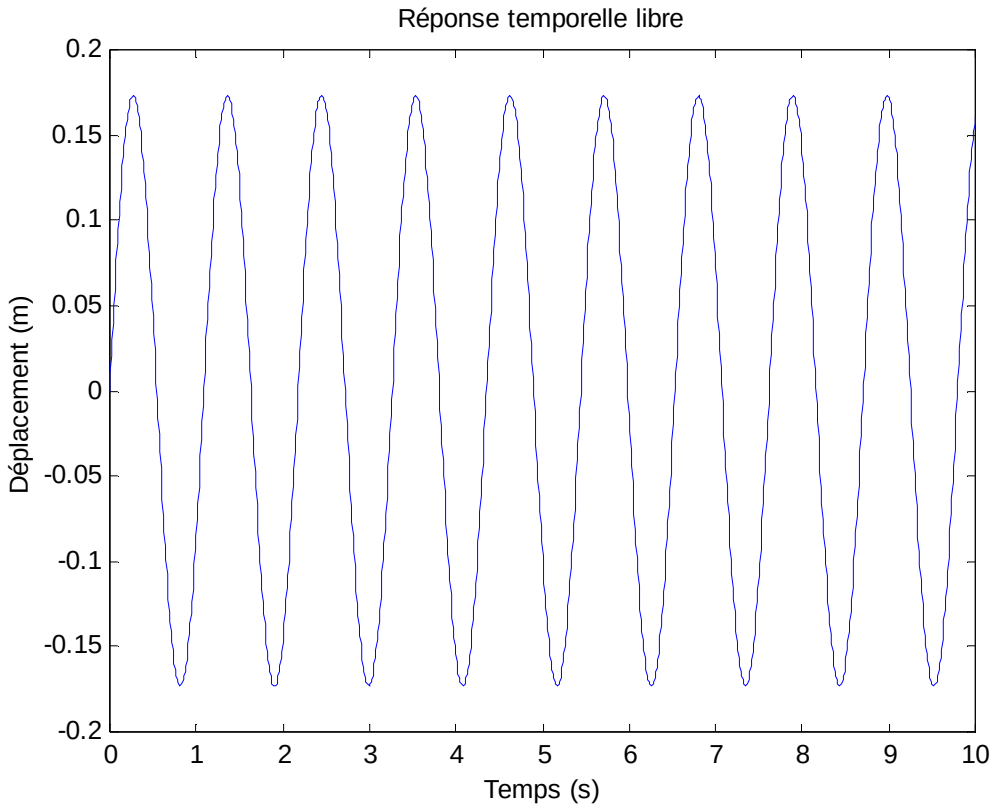


Figure 4. 3: Réponse temporelle libre

3. Systèmes à 1 degré de liberté libres amortis

3.1 Équations du mouvement.

Soit une masse m supportée par un ressort k et un amortisseur c.

L'amortisseur de type visqueux oppose une force proportionnelle à la vitesse du déplacement:

Fb = -Cx'(t).

- Le ressort s'oppose au déplacement par une force: FB = -Kx(t)
- La vibration est libre. La force d'excitation est donc nulle.

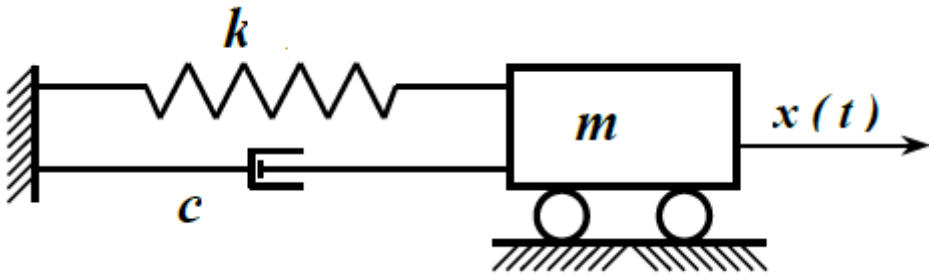


Figure 4.4: Système masse ressort amortisseur

Bilan des forces en position déformée :

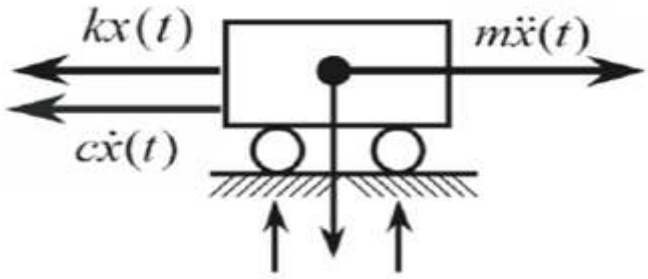


Figure 4.5: Forces et action de la masse présentes dans un système amorti

Ainsi, selon la loi de Newton, l'équation du mouvement s'écrit :

$$\sum F_x = m\ddot{x}(t)$$

$$-kx(t) - c\dot{x}(t) = m\ddot{x}(t)$$

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

Qui peut s'écrire aussi :

$$\ddot{x}(t) + 2\lambda\dot{x} + \omega_n^2 x = 0 \quad (2)$$

Où

$$2\lambda = \frac{c}{m}$$

Avec :

λ : Facteur d'amortissement[s^{-1}]

La solution de l'équation est de type :

$$x(t) = A\, e^{rt} \quad (3)$$

$$x(t) = C_1\, e^{r_1 t} + C_2\, e^{r_2 t} \quad (4)$$

C^2- 4MK < 0: Sous amortissement (Les solutions sont complexes)

Les deux formulations pour x(t) deviennent respectivement :

$$x(t) = e^{-\xi\omega_n t}\left(Ae^{i\omega_n t\sqrt{1-\xi^2}} + Be^{-i\omega_n t\sqrt{1-\xi^2}}\right)$$

$$A = \frac{\dot{x}(0) + \left(\xi + \sqrt{\xi^2 - 1}\right)\omega_n * x(0}{2\omega_n\sqrt{\xi^2 - 1}}$$

$$B = \frac{-\dot{x}(0) - \left(\xi - \sqrt{\xi^2 - 1}\right)\omega_n * x(0}{2\omega_n\sqrt{\xi^2 - 1}}$$

C^2- 4MK > 0: Sur- amortissement (Les solutions sont réelles)

Dans ce cas, les racines de l'équation caractéristique (3) sont réelles et le système s'approche lentement de sa position d'équilibre plutôt que de vibrer.

C^2-4KM = 0 : Amortissement critique

Ce cas représente la frontière entre les deux régimes précédents. Cette fois-ci, les racines de l'équation (3) sont égales à λ ; le système ne vibre pas et s'approche rapidement de sa position d'équilibre.

3.2 Calcul de la réponse libre amortie à 1 ddl

Soit un système masse-ressort-amortisseur où k = 163 N/m; c = 2.5 N.s/m; m = 1 kg, excité par une vitesse initiale de 1m/s.

La fonction suivante ssdl2 écrite en **MATLAB** permet de calculer les réponses vibratoires d'un système à 1 ddl, selon le type d'amortissement.

```
function[a,b,c,d,e,f]= ssdl2(m,c,k,x0,v0,tf)
%ssdl(m,c,k,x0,v0,tf) trace la réponse libre. m,c et k  représentent
respectivement la masse, l'amortissent et la raideur du système.les
arguments x0 et vo représentent les conditions initiales en déplacement
et en vitesse. tf représente le temps d'observation
w=sqrt(k/m) ;% pulsation naturelle
z=c/2/w/m;%calcul du rapport d'amortissement
wd=w*sqrt(1-z^2);% pulsation naturelle amortie

t=0:tf/1000:tf;
if z<1
A=sqrt(((v0+z*w*x0)^2+(x0*wd)^2)/wd^2);%Amplitude de l'exponontielle
Phi=atan2(x0*wd, v0+z*w*x0);% phase
x=A*exp(-z*w*t).*sin(wd*t+Phi); % réponse sous amortie
elseif z==1
a1=x0;
a2=v0+w*x0;
x=(a1+a2*t).*exp(-w*t);% réponse critique
else
a1=(-v0+(-z+sqrt(z^2-1))*w*x0)/2/w/sqrt(z^2-1);
a2=(v0+(z+sqrt(z^2-1))*w*x0)/2/w/sqrt(z^2-1);
x=exp(-z*w*t).*(a1*exp(-w*sqrt(z^2-1)*t))+(a2*exp(w*sqrt(z^2-1)*t)); %
réponse sur amortie
end
plot(t,x)
xlabel('Temps(s)')
ylabel('Déplacement(m)')
title ('Réponse temporelle libre')
```

Après exécution, on obtient :

```
» ssdl2(1,2.5,163,0,1,10)
```

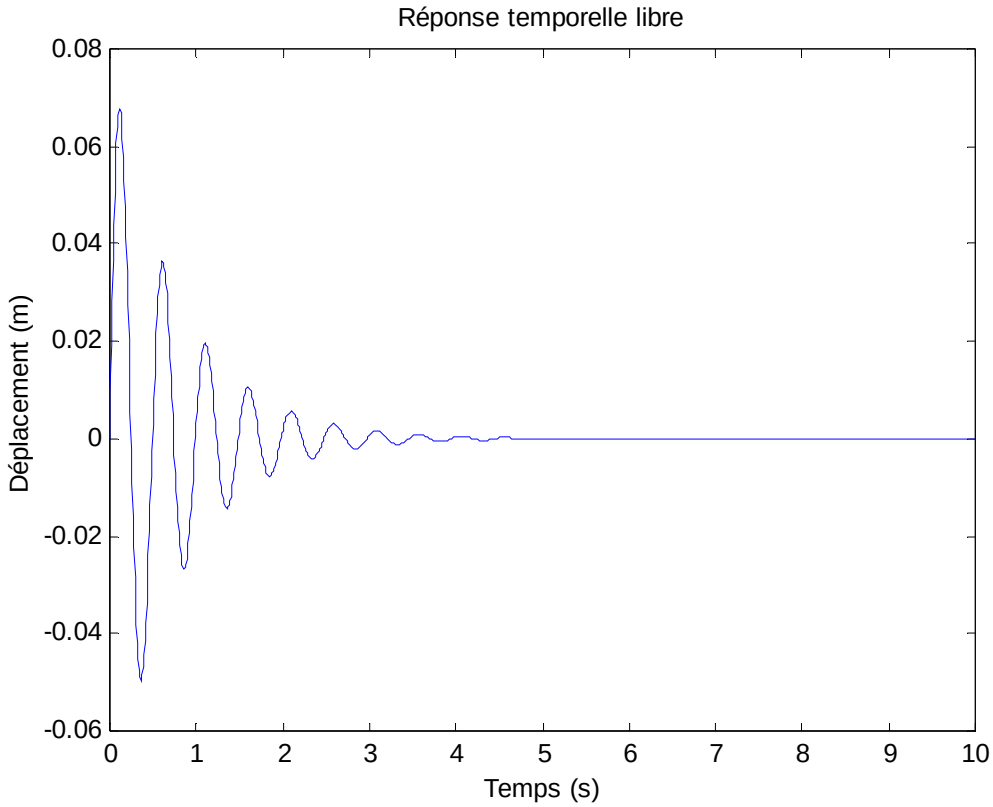


Figure 4.6: réponse temporelle libre amortie

4. Calcul d'une structure sous effets dynamiques (Systèmes à plusieurs degrés de liberté)

Un système réel comprend généralement plusieurs masses reliées entre elles par des éléments de types ressort et amortisseur.

Le nombre de paramètres indépendants nécessaires pour déterminer la position relative de chaque masse est appelé « nombre de degrés de liberté » ; un système comportant N masses susceptibles de se déplacer dans un plan possède donc 2N degrés de liberté.

Pour chaque degré de liberté, on établira les conditions d'équilibre, comportant les effets dus à l'accélération et ceux dus aux autres actions extérieures du système (Newton), au point considéré.

Intéressons-nous au cadre à deux degrés de liberté illustré à figure 4.5 afin de développer les équations dans le cas de structures à plusieurs degrés de liberté.

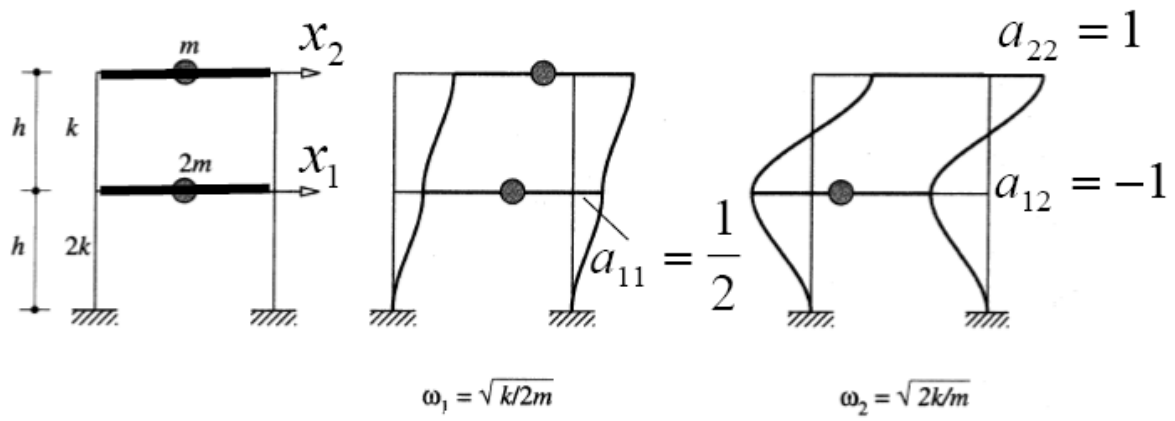


Figure 4.7: Cadre à deux degrés de liberté

Les équations du mouvement pour chacun des deux degrés de liberté de la structure de la figure 4.7 , en cas d'oscillations libres non amorties, s'expriment de la manière suivante :

$$M\ddot{x} + Kx = 0$$

Celles-ci peuvent aussi s'exprimer sous forme matricielle :

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

Si M et K sont constantes dans le temps, le système d'équations couplées peut être découplé et transformé en un système d'oscillateurs simples. Les modes propres s'obtiennent en annulant le déterminant du système afin d'obtenir les solutions non triviales :

$$|-\omega_n^2 M + K| = 0$$

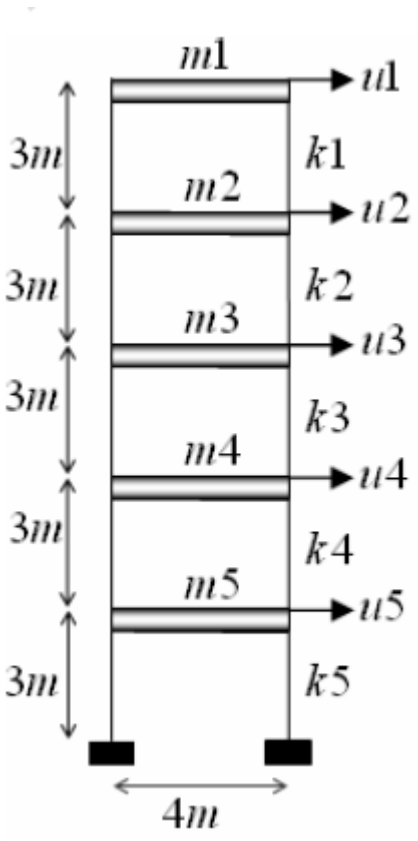
Les déplacements relatifs x s'expriment alors en coordonnées modales par le changement de variables suivant :

$$x = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = AZ$$

Où A est la matrice des vecteurs modaux. Ses lignes sont formées des vecteurs propres. Le vecteur z est le vecteur des coordonnées modales.

Exemple :

La figure suivante illustre un portique en 2D de 5 niveaux. Il est constitué de colonnes carrées de section (60 x 60cm²) et de poutres infiniment rigides (I = ∞) avec un module d'élasticité E = 2. 10^6 kN / m².



• La masse totale de chaque niveau (m = 100t).

• Le portique est soumis à un spectre de réponse tel que défini dans le RPA 2003 avec les paramètres de conception suivants: Zone sismique II.a, sol rocheux et un facteur d'amortissement (ζ = 0,05).

Évaluer les points suivants:

(A).Fréquences et modes propres ;

(B).Périodes correspondantes à chaque mode de vibration ;

(C).Accélérations correspondantes à chaque période ;

(D).Déplacements correspondants à chaque mode de vibration ;

(E).Déplacement maximum en fonction de la combinaison modale (SRSS) ;

(F).Facteurs de participation modale.

Solution :

Le processus de décomposition modale est illustré à la Figure 4.6.Un système à *n* degrés de liberté est traité comme *n* oscillateurs simples.

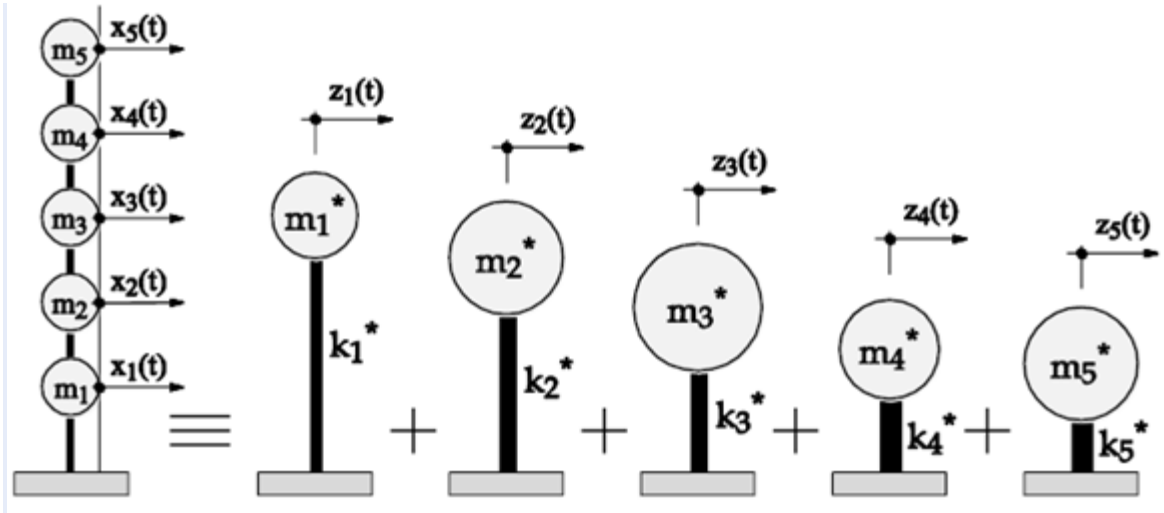


Figure 4.8: Décomposition modale

1. Définition de la matrice globale de masse :

$$M=\begin{pmatrix} m_1 & 0 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 & 0 \\ 0 & 0 & m_3 & 0 & 0 \\ 0 & 0 & 0 & m_4 & 0 \\ 0 & 0 & 0 & 0 & m_5 \end{pmatrix} = \begin{pmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 100 \end{pmatrix} t$$

```
» % Définir la matrice globale de masse
» M = [100 0 0 0 0;
       0 100 0 0 0;
       0 0 100 0 0;
       0 0 0 100 0;
       0 0 0 0 100]
```

2. Définition de la matrice globale de rigidité :

$$K_c = \frac{12EI_C}{h^3}$$

$$I_c = \frac{a^4}{12} = \frac{0.6^4}{12} = 0.0108m^4$$

La rigidité de chaque étage est:

$$K_I = \sum K_c = 2K_c = \frac{24EI_C}{h^3} = \frac{24\times 2\times 10^6 \times 0.0108}{3^3} = 19200 KN/m$$

La matrice de rigidité de la structure est donnée par :

$$K=\begin{pmatrix} K_1+K_2 & -K_2 & 0 & 0 & 0 \\ -K_2 & K_2+K_3 & -K_2 & 0 & 0 \\ 0 & -K_3 & K_3+K_4 & -K_3 & 0 \\ 0 & 0 & -K_4 & K_4+K_5 & -K_4 \\ 0 & 0 & 0 & -K_5 & K_5 \end{pmatrix} = 19200 \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} KN/m$$

```
» % Définir la matrice  de rigidité
» K = 19200.*[2 -1 0 0 0;
              -1 2 -1 0 0;
              0 -1 2 -1 0;
              0 0 -1 2 -1;
              0 0 0 -1 1]
```

3. Calcul des fréquences et modes propres :

Basé sur la théorie de la dynamique des structures, **le calcul des fréquences et modes propres** est déterminé par la résolution de l'équation suivante :

$$\left[K-\omega^2 M\right]\Phi=0$$

```
» [Modes,Omega]=eig(inv(M)*K)
```

$$\Omega=\begin{pmatrix} 132.5335 & 0 & 0 & 0 & 0 \\ 0 & 329.3511 & 0 & 0 & 0 \\ 0 & 0 & 15.5547 & 0 & 0 \\ 0 & 0 & 0 & 543.5194 & 0 \\ 0 & 0 & 0 & 0 & 707.0414 \end{pmatrix}$$

$$\Phi=\begin{pmatrix} 0.4557 & -0.5969 & 0.1699 & 0.5485 & 0.3260 \\ 0.5969 & -0.1699 & 0.3260 & -0.4557 & -0.5485 \\ 0.3260 & 0.5485 & 0.4557 & -0.1699 & 0.5969 \\ -0.1699 & -0.3260 & 0.5485 & 0.5969 & -0.4557 \\ -0.5485 & -0.4557 & 0.5969 & -0.3260 & 0.1669 \end{pmatrix}$$

4. Calcul des fréquences :

```
» % Calcul  des fréquences
» Freq = zeros(5) ;
» for i=1:5
  Freq(i,i)= Omega(i,i)^0.5
  end
```

$$\omega=\begin{pmatrix} 11.5123 & 0 & 0 & 0 & 0 \\ 0 & 18.1480 & 0 & 0 & 0 \\ 0 & 0 & 3.9439 & 0 & 0 \\ 0 & 0 & 0 & 23.3135 & 0 \\ 0 & 0 & 0 & 0 & 26.5902 \end{pmatrix} \text{ rad/sec}$$

On obtient:

$$\Phi_1=\begin{pmatrix}0.4557\\0.5969\\0.3260\\-0.1699\\-0.5485\end{pmatrix},\ \Phi_2=\begin{pmatrix}-0.5969\\-0.1699\\0.5485\\-0.3260\\-0.4557\end{pmatrix},\ \Phi_3=\begin{pmatrix}0.1699\\0.3260\\0.4557\\0.5485\\0.5969\end{pmatrix},\ \Phi_4=\begin{pmatrix}0.5485\\-0.4557\\-0.1699\\0.5969\\-0.3260\end{pmatrix},\ \Phi_5=\begin{pmatrix}0.3260\\-0.5485\\0.5969\\-0.4557\\0.1669\end{pmatrix}$$

ω_1=11.5123rad/sec ω_2=18.1480rad/sec ω_3=3.9439rad/sec ω_4=23.13135 rad/sec ω_5=26.5902 rad/sec

5. Calcul des périodes correspondant à chaque mode de vibration :

$$T=\frac{2\pi}{\omega}$$

```
» % Calcul  des périodes
» T = zeros(5);
» for i=1:5
  T (i,i) = 2 * pi /Freq(i,i)
  end
```

On obtient:

$$T=\begin{pmatrix}0.5458 & 0 & 0 & 0 & 0\\ 0 & 0.3462 & 0 & 0 & 0\\ 0 & 0 & 1.5931 & 0 & 0\\ 0 & 0 & 0 & 0.2695 & 0\\ 0 & 0 & 0 & 0 & 0.2363\end{pmatrix}\ \text{sec}$$

6. Détermination des accélérations correspondantes à chaque période :

L'action sismique est représentée par le spectre de calcul suivant (RPA 2003) :

$$\frac{S_a}{g} = \begin{cases} 1.25A\left(1+\frac{T}{T_1}\left(2.5\eta\frac{Q}{R}-1\right)\right) & 0 \le T \le T_1 \\ 2.5\eta(1.25A)\left(\frac{Q}{R}\right) & T_1 \le T \le T_2 \\ 2.5\eta(1.25A)\left(\frac{Q}{R}\right)\left(\frac{T_2}{T}\right)^{2/3} & T_2 \le T \le 3.0s \\ 2.5\eta(1.25A)\left(\frac{T_2}{3}\right)^{2/3}\left(\frac{3}{T}\right)^{5/3}\left(\frac{Q}{R}\right) & T > 3.0s \end{cases}$$

A : coefficient d'accélération de zone

η : facteur de correction d'amortissement (quant l'amortissement est différent de 5%)

$\eta = 7/2+\xi \geq$ **0.7**

ξ : pourcentage d'amortissement critique

R : coefficient de comportement de la structure

T1, T2 : périodes caractéristiques associées à la catégorie de site

Q : facteur de qualité

La détermination de l'action sismique dépend des paramètres suivants :

Zone sismique II.a

Groupe d'usage du bâtiment 2 } $\rightarrow$A=0.15

Q=1.25

R=3.5 (Portiques autostables avec remplissages en maçonnerie rigide)

ξ=5% $\rightarrow\eta$ = 1

```
function [a,b,c,d,e]=Sa(T1,T2,A,Q,R,ETA)
Sa=zeros(5)
T=[1.5931 0 0 0 0;
   0 0.5458 0 0 0;
   0 0 0.3462 0 0;
   0 0 0 0.2695 0;
   0 0 0 0  0.2363]

for i=1:5
   if (T(i,i)>=0) & (T(i,i)<T1)
      Sa(i,i)=(1.25 * A * (1+ T(i,i)/T1 * ((2.5 * ETA * Q/R) -1))) * 9.81
   else
      if T(i,i)>=T1 & T(i,i)<T2
         Sa(i,i)=(1.25 * A * 2.5 * ETA * Q/R )*9.81
      else
          if T(i,i)>=T2 & T(i,i)<=3
             Sa(i,i)=(1.25 * A * 2.5 * ETA * Q/R* (T2/T(i,i))^2/3 )*9.81
          else
            Sa(i,i)=(1.25 * A * 2.5 * ETA * Q/R* (T2/3)^2/3* (3/T(i,i))^5/3
)*9.81
          end
      end
   end
end
```

Sol rocheux (S_1)→T_1=0.15s et T_2=0.35s

Après exécution:

```
» Sa(0.15,0.35,0.15,1.25,3.5,1)
```

On obtient:

$$S_a = \begin{pmatrix} 0.2251 & 0 & 0 & 0 & 0 \\ 0 & 1.6423 & 0 & 0 & 0 \\ 0 & 0 & 0.0264 & 0 & 0 \\ 0 & 0 & 0 & 1.6423 & 0 \\ 0 & 0 & 0 & 0 & 1.6423 \end{pmatrix} \text{m/sec}^2$$

7. Détermination des déplacements correspondants à chaque mode de vibration :

La matrice des déplacements est donnée par l'expression suivante:

$$U = \phi \frac{L}{m^*} \frac{S_a}{\Omega}$$

Où:

$$L = \phi^T M\{1\} \quad m^* = \phi^T M \phi$$

Calcul de L:

```
» LL=Modes'*M*[1;1;1;1;1]
» for i=1:5
  L(i,i)=LL(i,1)
  end
```

Calcul de m*:

```
» ModalMass = Modes'* M * Modes
```

8. Calcul des déplacements :

```
» U_Modal = Modes*(L/ModalMass)*(Sa/Omegas)
```

9. Détermination du déplacement maximum en fonction de la combinaison modale (SRSS) :

$$U_{max} = \sqrt{\sum_{i=1}^{n} (U_i)^2} = \sqrt{(U_1)^2 + (U_2)^2 + (U_3)^2 + (U_4)^2 + (U_5)^2}$$

```
» for i=1:5
  s = 0
  for j=1:5
  s = s + U_Modal(i,j)^2
  end
  U_Max(i,1) = s^0.5
  end
```

Les facteurs de participation modale représentent l'interaction entre la forme et le mode de la distribution spatiale de la charge externe.

Ces facteurs sont donnés par :

MPF=L/m*

```
» MPF=L/ModalMass
```

5. Application

Ecrire le code MATLAB qui permet de calculer les réponses vibratoires d'un système forcé à 1 ddl

Références bibliographiques.

Essential MATLAB for engineers and scientists.

HAHN, Brian et VALENTINE, Daniel. Brian D. Hahn, Daniel T. Valentine

Newnes, 2007.

PRACTICAL MATLAB. BASICS FOR ENGINEERS
KALECHMAN, Misza.

CRC Press, 2008.

A GUIDE TO MATLAB for Beginners and Experienced Users
HUNT, Brian R., LIPSMAN, Ronald L., et ROSENBERG, Jonathan M.

Cambridge university press, 1995.

Numerical methods in engineering with MATLAB®.
Jaan Kiusalaas.

Cambridge university press, 2 eme edition, 2010.

Méthodes numériques: algorithmes, analyse et applications.
QUARTERONI, Alfio Maria, SACCO, Riccardo, et SALERI, Fausto.

Springer Science & Business Media, 2008.

Introduction to MATLAB with Numerical Preliminaries
STANOYEVITCH, Alexander.

Wiley Interscience, 2005.

MATLAB for Engineers
Holly Moore.

Pearson Education, 2012.

Structural dynamics of earthquake engineering
Theory and application using MATHEMATICA and MATLAB

RAJASEKARAN, Sundaramoorthy.

Elsevier, 2009.

The Finite Element Method Using MATLAB

KWON, Young W. et BANG, Hyochoong

CRC press, 2000.

Simulation des vibrations mécaniques par Matlab, Simulink et Ansys

THOMAS, Marc et LAVILLE, Frédéric.
PUQ, 2007.

Cours Méthode des Éléments Finis

Abdelghani SEGHIR

Université A. Mira, Béjaia, Algérie

Cours Dynamique des structures

I. Smith et P. Lestuzzi

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Ecole Polytechnique Fédérale de LAUSANE

Support du cours LCS : Langage du calcul scientifique (Matlab)

Département ST

Université Abderrahmane MIRA de Bejaia

Cours Langage

Z.Mansouri

Université de Skikda 20 août 55

Cours Analyse Numérique Appliquée

M. Mignotte

Université de Montréal