REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABOU—BAKR—BELKAID TLEMCEN
FACULTE DE TECHNOLOGIE
DEPARTEMENT DE GENIE BIOMEDICAL
LABORATOIRE DE GENIE BIOMEDICAL

Polycopié de travaux pratiques

Présenté Par

Melle BAAKEK YETTOU NOUR EL HOUDA

MÉTHODES NUMÉRIQUES

2eme année LMD : tronc commun GBM

Avant-propos

Ce polycopié de travaux pratiques de méthodes numériques a été élaboré pour la formation des étudiants de deuxième année tronc commun LMD de la filière Génie Biomédical. Il est constitué de quatre textes de travaux pratiques qui vont permettre aux étudiants de mieux comprendre le côté pratique de certaines notions théoriques déjà vues pendant les séances des cours et des travaux dirigés.

Nous avons organisé notre polycopié en quatre TP : le premier TP concerne l'algèbre des matrices, le deuxième concerne la résolution d'un système d'équations linéaires à plusieurs variables. Alors que le troisième TP concerne la résolution d'équations non linéaires avec une seule variable, et le dernier TP concerne les techniques numériques de dérivations et d'intégrations d'une fonction.

Nous avons terminé notre polycopié par une bibliographie qui contient les différentes ressources utilisées dans l'ouvrage.

Table de matière

Table de figures

Table des tableaux

Introduction

Au fil des ans, nous avons appris à résoudre des équations mathématiques à l'aide de diverses méthodes algébriques. Ces méthodes incluent la méthode de substitution et la méthode de réduction (élimination). D'autres méthodes algébriques incluent la formule quadratique et la factorisation.

En Algèbre, nous avons appris aussi que la résolution des équations non linéaires, ou des systèmes d'équations linéaires peut être implémentée en utilisant des techniques sous forme d'algorithmes. Cependant, lorsque ces méthodes échouent, nous utilisons le concept de méthode numérique.

Les méthodes numériques sont utilisées pour approximer les solutions des équations lorsqu'elles ne peuvent pas être déterminées exactement par des méthodes algébriques. Ils construisent des approximations successives qui convergent vers la solution exacte d'une équation ou d'un système d'équations.

Dans le premier TP nous avons décrit d'une manière générale l'algèbre des matrices. Dans le deuxième TP nous nous somme concentré sur la résolution d'un système d'équations linéaires à plusieurs variables en utilisant la méthode de Gauss Jordan.

Dans le troisième TP, nous nous sommes concentrés sur la résolution d'équations non linéaires n'impliquant qu'une seule variable. Nous avons utilisé trois méthodes : méthode de dichotomie, méthode de point fixe, et méthode de Newton Raphson.

Dans le quatrième TP, nous avons décrit quatre techniques d'intégration d'une fonction f(x): méthode de trapèze, règle 1/3 de Simpson, règle 3/8 de Simpson, et méthode de Newton cotes.

Après la description de chacune de ces méthodes, nous avons utilisé le logiciel Matlab pour résoudre un exemple pour chaque méthode.

TPN1. Algèbre des matrices

1.1. Introduction

Le mot MATLAB est une abréviation du mot « Matrix Laboratory ». C'est un langage de programmation, permettant d'exprimer directement les variables et les opérateurs fondamentaux (addition, multiplication, …etc) sous forme de tableaux (vecteurs) et de matrices.

C'est un langage de programmation multi-paradigme propriétaire et un environnement de calcul numérique développé par MathWorks. MATLAB permet de faire des manipulations matricielles, le traçage des fonctions et des données, la mise en œuvre d'algorithmes, la création d'interfaces utilisateur et l'interfaçage avec des programmes écrits dans d'autres langages.

1.2. Opérations matricielles

Considérez l'expression MATLAB suivante:

C = A + B

Si A et B sont des scalaires (matrices 1x1), C sera un scalaire égal à leur somme. Si A et B sont des vecteurs lignes de longueurs identiques, C sera un vecteur ligne de même longueur, chaque élément étant égal à la somme des éléments correspondants de A et B. Si A et B sont de taille (n x m) matrices, il en sera de même pour C, avec chaque élément égal à la somme des éléments correspondants de A et B.

En bref, le symbole "+" signifie "effectuer une addition de matrice". Mais que faire si A et B sont de tailles incompatibles?

Sans surprise, MATLAB affiche la déclaration suivante:

```
??? Error using ==> +
        Matrix dimensions must agree.
```

1.3 Déclarations d'affectation

MATLAB utilise un modèle commun dans de nombreux langages de programmation pour affecter la valeur d'une expression à une variable. Le nom de la variable est placé à gauche d'un signe égal et l'expression à droite. L'expression est évaluée et le résultat est affecté au nom de la variable. Dans MATLAB, il n'est pas nécessaire de déclarer une variable avant de lui affecter une valeur. Si une valeur a été précédemment affectée à une variable, la nouvelle valeur remplace la précédente.

1.4. Affichage des valeurs

Pour voir le contenu d'une variable, tapez simplement son nom.
si vous tapez:

```
C = A + B
```

Pour éviter l'affichage du contenu de la variable, mettez un point-virgule à la fin de toute instruction d'affectation. Par exemple:

```
C = A + B;
```

1.5. Initialisation des matrices

Si une matrice est suffisamment petite, on peut fournir des valeurs initiales en les tapant simplement. Par exemple:

```
a = 15;
b = [12 3 1];
c = [61; 52; 14];
d = [12 3 1; 61 52 14];
```

Dans ce cas là, a est un scalaire, b est un vecteur ligne (1 x 3), c est un vecteur colonne (3 x 1), et d est une matrice (2 x 3). Ainsi, tapez "d" produit:

```
d =

    12     3     1
    61    52    14
```

Le système d'indication du contenu de la matrice est très simple. Les valeurs séparées par des espaces doivent être sur la même ligne; ceux séparés par des points-virgules doivent être sur des lignes séparées. Toutes les valeurs sont placées entre crochets.

1.6. Création des sous matrices

Le schéma général d'initialisation des matrices peut être étendu pour inclure des matrices en tant que composants. Par exemple:

```
a = [11 12 23];
b = [41 6 21];
c = [a b];
```

donne:

```
c =
    11    12    23    41     6    21
```

Tandis que: `d = [a; b]`
donne:

```
d =
    11    12    23
    41     6    21
```

Les matrices peuvent facilement être "collées" ensemble de cette manière. Les tailles des matrices doivent être compatibles. Si ce n'est pas le cas, MATLAB vous le dira.

1.7. Utilisation des sous matrices

On souhaite souvent ne référencer qu'une partie d'une matrice. MATLAB fournit des moyens simples et puissants de le faire.

Pour référencer une partie d'une matrice, donnez le nom de la matrice suivi de parenthèses avec des expressions indiquant la partie souhaitée. Le cas le plus simple se présente lorsqu'un seul élément est souhaité. Par exemple, en utilisant d dans la section précédente:

`d (1,2)` vaut `12`
`d (2,1)` vaut `41`

Dans tous les cas, la première expression entre parenthèses indique la ligne (ou les lignes), tandis que la deuxième expression indique la ou les colonnes. C. à. d. pour indiquer un élément on donne le numéro de la ligne et de la colonne (la position).

Lorsqu'on veut rechercher plus d'un élément d'une matrice, à titre d'exemple, pour indiquer «toutes les lignes», utilisez le signe deux points (:) pour la première expression. Pour indiquer «toutes les colonnes», utilisez deux points (:) pour la deuxième expression:

```
 d (1, :) =
        11    12    23
    d (:, 2) =
        12
        6
```

En fait, vous pouvez utiliser n'importe quelle expression de cette manière. Par exemple:

```
d (2, [2 3]) =
6    21

d (2, [3 2]) =
         21     6
```

Les variables peuvent également être utilisées comme "indices". Donc:
Si

```
>> z = [2 3]

z =

     2     3
```

Alors,

```
>> d(2,z)

ans =

     6    21
```

Les deux points (:) est utilisée aussi pour produire une chaîne d'entiers consécutifs. Par exemple, la déclaration: `x = 3: 5`, produit :

```
    x =
                3   4   5
```

Donc:

`d (1, 1: 2)` est égal à

```
     11  12
```

1.8. Chaînes de caractères (string)

Une variable dans MATLAB est de deux types: numérique ou chaîne. Une matrice de chaîne de caractères est comme n'importe quelle matrice, sauf que les éléments qu'elle contient sont interprétés comme des nombres ASCII. Pour créer une variable chaîne de caractères, mettez une chaîne de caractères entre deux guillemets "simples" (en fait, des apostrophes):

```
>> stg = 'Ceci est une chaîne' ;
```

Puisqu'une variable chaîne de caractère est un vecteur ligne, il est possible de créer une liste de chaînes de caractère en créant une matrice dans laquelle chaque ligne est une chaîne de caractère distincte. Comme pour toutes les matrices standard, les lignes doivent être de la même longueur. Donc:

la déclaration : `x = ['ab'; 'cd']`

produit:

```
x =
     ab
     cd
```

tandis que : `x = ['ab' 'cd']`

produit: `abcd`

1.9. Opérations de matrice et de vecteurs

1.9.1. Opérations matricielles

La transposition matricielle est aussi simple que d'ajouter un premier (apostrophe) au nom de la matrice. Donc:

Si `x = [1  2  3]`, alors :

```
x' =
     1
     2
     3
```

Pour additionner deux matrices de même taille, utilisez le signe plus (+). Pour soustraire une matrice d'une autre de même taille, utilisez un signe moins (-).

Il existe un cas dans lequel l'addition ou la soustraction fonctionne lorsque les composants sont de tailles différentes. Si l'un est un scalaire, il est ajouté ou soustrait de tous les éléments de l'autre.

La multiplication matricielle est indiquée par un astérisque (*). A une exception près, les règles habituelles s'appliquent: les dimensions intérieures des deux opérandes doivent être les mêmes. La seule exception autorisée couvre le cas dans lequel l'un des éléments est un scalaire.

Dans ce cas, la valeur scalaire est multipliée par chaque élément de la matrice, ce qui donne une nouvelle matrice de même taille [1- 4].

MATLAB fournit deux notations pour la «division matricielle» (/ et inv) qui fournissent des solutions rapides aux problèmes d'équation simultanée ou de régression linéaire [1- 4].

1.9.2. Opérations sur les vecteurs

Pour indiquer une opération de vecteur (élément par élément), faites précéder un opérateur standard d'un point (.). Donc:

```
Si   x = 1     2     3
et       y = 4     5     6
Alors :  x.*y =
           4    10    18
```

Vous pouvez diviser tous les éléments d'une matrice par les éléments correspondants d'une autre, produisant une matrice de même taille, comme dans:

```
C = A ./ B
```

Dans chaque cas, l'un des opérandes peut être un scalaire. Cela s'avère pratique lorsque vous souhaitez élever tous les éléments d'une matrice à une puissance. Par exemple:

```
Si x =  1     2     3
```

Alors :

```
x.^2 =
      1     4     9
```

Les opérations de vecteurs MATLAB incluent la multiplication (. *), La division (./) et l'exponentiation (. ^). L'addition et la soustraction de vecteurs ne sont pas nécessaires (et en fait ne

sont pas autorisées), car elles dupliqueraient simplement les opérations d'addition et de soustraction de matrice [1 – 4].

1.10. Utilisation des fonctions

MATLAB a un certain nombre de fonctions intégrées. Certaines fournissent une réponse (matricielle); d'autres en fournissent deux ou plus.
Vous pouvez utiliser n'importe quelle fonction dans une expression. Si elle renvoie une réponse, cette réponse sera utilisée. La fonction *sum* fournit une somme [1 – 4]:

```
Si   x =
          1
          2
          3
```

puis la déclaration:

```
      y =sum(x) + 10
```

produira:

```
      y =
         16
```

Certaines fonctions, telles que max, fournissent plus d'une réponse. Si une telle fonction est incluse dans une expression, seule la première réponse sera utilisée. Par exemple:

Si `x = 1 4 3`

Alors la déclaration: `z = 10 + max(x)` produira: `z =`

```
14
```

Pour obtenir toutes les réponses d'une fonction, utilisez une instruction d'affectation multiple dans laquelle les variables qui doivent recevoir les réponses sont répertoriées à gauche du signe égal (=), entre crochets, et la fonction est sur la droite. Par exemple:
Si `x = 1 4 3`
Alors la déclaration : `[y n] = max(x)` produira: `y =`

```
     4
n =
     2
```

Dans ce cas, y est la valeur maximale de x et n indique la position dans laquelle elle a été trouvée.

De nombreuses fonctions intégrées de MATLAB, telles que *sum*, *min*, *max* et *mean*, ont des interprétations naturelles lorsqu'elles sont appliquées à un vecteur. Si une matrice est donnée

comme argument à une telle fonction, sa procédure est appliquée séparément à chaque colonne et le résultat est un vecteur ligne.

Si x =

```
     1     2     3
     4     5     6
```

Alors :

```
sum(x) =
     5     7     9
```

Certaines fonctions ne fournissent aucune réponse en soi. Par exemple, pour tracer un vecteur y par rapport à un vecteur x, utilisez simplement l'instruction: *plot(x,y)*

Notez que dans ce cas, deux arguments (les éléments entre parenthèses après le nom de la fonction) ont été fournis en tant qu'entrées de la fonction. Chaque fonction nécessite un nombre spécifique d'entrées. Cependant, certains ont été programmés(surdéfinies) pour réagir de manière appropriée selon diverses situations. Par exemple, pour tracer y vs (1,2,3 ...), vous pouvez utiliser l'instruction: *plot(y)*

Il existe de nombreuses fonctions intégrées dans MATLAB. Parmi eux, les suivants [1 – 4]:

Instruction	Description des fonctionnalités
ones	ones matrix
zeros	zeros matrix
size	size of a matrix
diag	diagonal elements of a matrix
inv	matrix inverse
rand	uniformly distributed random numbers
randn	normally distributed random numbers
cumprod	cumulative product of elements
cumsum	cumulative sum of elements
max	largest component
min	smallest component
sum	sum of elements
mean	average or mean value
median	median value
std	standard deviation
sort	sort in ascending order
find	find indices of nonzero entries
corrcoef	correlation coefficients
cov	covariance matrix

Tableau 1.1 : *Fonctions prédéfinies dans MATLAB*

1.11. Opérations logiques et relationnelles sur les matrices

MATLAB propose six opérateurs relationnels:

Opérateur relationnels	Signification
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	equal
~=	Not equal

Tableau 1.2 : *Opérateurs relationnels dans MATLAB*

Notez bien :

1. la différence entre la double égalité (A == B) et la simple égalité (A = B). La première est une relation logique, alors que la seconde une déclaration d'affectation.
2. Chaque fois que MATLAB rencontre un opérateur relationnel, il produit un (1) si l'expression est vraie et un zéro (0) si l'expression est fausse. Donc le code:

`x = 1 < 3` produit: `x=1`, tandis que `x = 1 > 3` produit: `x=0`

Les opérateurs relationnels peuvent être utilisés sur des matrices, à condition qu'ils soient de la même taille. Les opérations sont effectuées élément par élément, ce qui donne une matrice avec des uns dans les positions pour lesquelles la relation était vraie et des zéros dans les positions pour lesquelles la relation était fausse [1- 4]. Donc:

Si `A =`

```
    1    2
    3    4
```

et `B =`

```
    3    1
    2    2
```

Le code: `C = A > B` produit:

```
C =
     0     1
     1 1
     2
```

Un ou les deux opérandes connectés par un opérateur relationnel peuvent être un scalaire. Donc:

si `A =`

```
     1     2
     3     4
```

Le code: `C = A > 2` produit:

```
C =
     0     0
     1     1
```

On peut également utiliser des opérateurs logiques dont il existe trois:

- & : and
- | : or
- ~ : not

Chacun fonctionne avec des matrices élément par élément et se conforme aux règles ordinaires de la logique, traitant tout élément non nul comme vrai et tout élément nul comme faux.

- Les opérateurs relationnels et logiques sont fréquemment utilisés avec l'instruction If (décrite ci-dessous) et les variables scalaires, comme dans les autres langages de programmation [1 – 4].

1.12. Tri des matrices

Pour trier une matrice par ordre croissant, utilisez la fonction de tri (*sort*). Si l'argument est un vecteur, le résultat sera un nouveau vecteur avec les éléments dans l'ordre souhaité. S'il s'agit d'une matrice, le résultat sera une nouvelle matrice dans laquelle chaque colonne contiendra le contenu de la colonne correspondante de l'ancienne matrice, par ordre croissant. Notez que dans ce dernier cas, chaque colonne est en effet, triée séparément [1 – 4]. Donc:

Si `x =`

```
     1     5
     3     2
     2     8
```

Le code : `y=sort(x)` produit :

```
y =
     1     2
     2     5
     3     8
```

Pour obtenir un enregistrement des lignes à partir desquelles chacun des éléments triés provient, utilisez une affectation multiple pour obtenir la deuxième sortie de la fonction. Pour le cas ci-dessus [1 – 4]:

Le code: `[y r] = sort(x)` produit :

```
r =
     1     2
     3     1
     2     3
```

Le deuxième élément de la liste triée de la colonne 1 provenait de la ligne 3, etc.

1.13. Contrôle du flux d'exécution :

Il est possible de faire beaucoup de choses dans MATLAB en exécutant simplement des instructions impliquant des expressions matricielles, les unes après les autres. Cependant, il existe des cas dans lesquels on doit simplement substituer un ordre non séquentiel. Pour faciliter cela, MATLAB propose trois méthodes relativement standard pour contrôler le déroulement du programme: les boucles For, les boucles While, et les instructions If [1 – 4].

1.13.1. Les boucles

L'utilisation la plus courante d'une boucle For se produit lorsqu'un ensemble d'instructions doit être répété un nombre fixe de fois, comme dans:

```
for j= 1: n
      .......
   end
```

Exemple:

```
x=1
for i=1:5
    x=x*i
end
```

Alors que la boucle While contient des instructions à exécuter tant qu'une condition déclarée reste vraie, comme dans:

```
while x > 0.5
      .......
   end
```

```
x=1;
while x<10
    x=x+2
end
```

Il est crucial qu'à un moment donné, une instruction soit exécutée qui rendra la condition de l'instruction While fausse. Si ce n'est pas le cas, vous avez créé une boucle infinie - une boucle qui continuera jusqu'à ce que vous arrêtez le programme.

Pour la lisibilité, il est parfois utile de créer des variables comme TRUE et FALSE, puis de les utiliser dans une boucle While. Par exemple:

```
true = 1==1;
   false = 1==0;
   .....
   done = false;
   while not done
        ........
   end
```

Dans la boucle While, il devrait y avoir une instruction qui, à un moment donné, sera définie égale à true [1 – 4].

1.13.2. Instructions conditionnelles If

Une instruction If fournit une méthode pour exécuter certaines instructions si une condition est vraie et d'autres instructions (ou aucune) si la condition est fausse.

```
if x > 0.15
          ........
       else
          .......
     end
```

Dans ce cas, si x est supérieur à 0,15, le premier ensemble d'instructions sera exécuté; sinon, le deuxième ensemble sera exécuté.

Exemple :

```
a=80;
if a==10
    fprintf('la valeur de a est 10\n')
elseif (a==20)
     fprintf('la valeur de a est 20\n')
elseif (a==30)
     fprintf('la valeur de a est 30\n')
else
     fprintf('aucune valeur ne correspond \n')
     fprintf('la valeur exacte de a est: %d\n', a)
end
```

Une version plus simple omet la "section else", comme dans:

```
if x > 0.15
          ........
    end
```

Ici, les instructions seront exécutées si (mais seulement si) x dépasse 0,15.

1.13.3. Nesting (imbrication).

Ces trois structures permettent une imbrication dans laquelle un type de structure se trouve dans un autre. Par exemple:

```
for j = 1:n
        for k = 1:n
          if x(j,k) > 0.5
              x(j,k) = 1.5;
          end
       end
   end
```

1.14. Fonctions d'écriture

La puissance de MATLAB entre vraiment en jeu lorsque vous ajoutez vos propres fonctions pour améliorer le langage. Une fois qu'un fichier m de fonction est écrit, débogué et placé dans un répertoire approprié, il fait partie à toutes fins pratiques de votre version de MATLAB.

Un fichier de fonction commence par une ligne déclarant la fonction, ses arguments et ses sorties. Suivent les instructions nécessaires pour produire les sorties à partir des entrées (arguments) [1 – 4].

```
function [sortie1,sortie2, ...] = nom_fonction(entree1,entree2, ...)
.
.
.
end
```

Voici un exemple simple qui calcule et renvoie en sortie le produit de deux scalaires passés en paramètre, on écrira dans le fichier produit.m :

```
function [p] = produit(a,b)
 p = a*b;
end
```

Il est important de noter que les noms des arguments d'entrée et de sortie utilisés dans un fichier de fonction sont des variables strictement locales qui n'existent que dans la fonction elle-même. Ainsi dans un programme, on pourrait écrire l'instruction:

```
r = produit(3,16);
```

Le premier vecteur (ici scalaire) de la liste d'arguments de cette instruction c'est (3) sera affectée au premier argument de la fonction (ici, a) tandis que le second vecteur (scalaire) de l'instruction appelante (16) sera affectée au second vecteur dans la fonction (ici b). Il n'est pas nécessaire que les noms soient les mêmes à aucun égard. De plus, la fonction ne peut en aucun cas modifier les arguments d'origine. Il ne peut renvoyer des informations que via sa sortie.

Cette fonction renvoie une seule sortie, appelée (p) en interne. Cependant, la matrice résultante sera remplacée par l'argument entier « r» dans n'importe quelle expression.

Si une fonction doit renvoyer deux arguments ou plus, attribuez-leur simplement des noms dans la ligne de déclaration, comme dans:

```
function [val1, val2] = produit(a,b)
val1 = a*b;
val2= val1*a ;
end
```

TPN2. Résolution des systèmes linéaires
Méthode de Gauss Jordan

2.1. Système d'équations linéaires

Un système d'équations linéaires est un groupe d'équations linéaires avec divers facteurs inconnus.

Soient $A \in M_{n,n}(IR)$, $x, b \in IR^n$ et l'équation $Ax = b$.

On cherche x qui vérifié l'équation avec A et b donnés.

Un système de n équations linéaires à n inconnues est de la forme suivante:

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

Où les a_{ij} sont les coefficients du système, les x_i les inconnues et les b_i les termes constants.

Remarque:

1. le système à une solution unique si et seulement si $detA \neq 0$ [1].

Exemple :

$$\begin{cases} x + y = 4 \\ 3x + 5y = 1 \end{cases}$$

2. Si la matrice A est singulière($\det(A) = 0$), alors on dit que le système est dégénéré ou indéterminé (impossible).

Exemple :

$$\begin{cases} 4x + 6y = 3 \\ 8x + 12y = 7 \end{cases}$$

Pour résoudre un tel système, nous devons représenter un système d'équations linéaires à l'aide d'une matrice augmentée c.à.d. alignez toutes les variables d'un côté de l'équation, avec les constantes de l'autre côté. Ensuite, formez une matrice de nombres avec juste les coefficients et les constantes des équations.

Les techniques les plus populaires pour résoudre ce type d'équations linéaires sont :

- Règle de Cramer
- Décomposition LU
- Décomposition de Choleskey
- Elimination de Gauss

La méthode de Gauss (l'élimination gaussienne), également connue sous le nom de réduction de ligne, est un algorithme d'algèbre linéaire permettant de transformer le système $Ax = b$ en un système $= c$, ensuite résoudre ce dernier. Il est généralement compris comme une séquence d'opérations effectuées sur la matrice de coefficients correspondante [5].
Pour effectuer une réduction de ligne sur une matrice, on utilise une séquence d'opérations élémentaires de ligne pour modifier la matrice jusqu'à ce que le coin inférieur gauche de la matrice soit rempli de zéros, autant que possible. Il existe trois types d'opérations élémentaires [5 - 6]:

- Permutation de deux lignes ou deux colonnes,
- Multiplier ou diviser une ligne par un nombre différent de zéro,
- Ajouter ou retrancher à une ligne un certain nombre de fois une autre ligne

Il est important de noter que lors du calcul en utilisant la méthode de Gauss, si la matrice a au moins une ligne zéro avec le côté droit non nul (colonne de termes constants), le système d'équations est alors incohérent. L'ensemble de solutions d'un tel système d'équations linéaires n'existe pas [5 – 6].

2.2. Les différentes formes d'une matrice

Il existe trois différentes formes de la matrice A à résoudre ; la matrice diagonale, la matrice triangulaire, et la matrice quelconque.

2.2.1. Premier cas : un cas simple « A est une matrice diagonale »

$$\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \ddots & \vdots \\ 0 & 0 & a_{ii} & 0 & 0 \\ \vdots & \ddots & 0 & \ddots & 0 \\ 0 & \dots & 0 & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

La solution de cette équation est : $x_i = \frac{b_i}{a_{ii}}, \quad 1 \leq i \leq n$

Algorithme :

Pour i=1 jusqu'à n % (i décrit tous les indice des lignes et de colonnes).

$x_i \leftarrow \frac{b_i}{a_{ii}}$ % (x_i *reçoit la valeur* $\frac{b_i}{a_{ii}}$)

finpour % fin de la boucle pour

2.2.2. Deuxième cas : « A est de forme triangulaire »

$$\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ \vdots & \ddots & 0 & \ddots & \vdots \\ a_{i1} & \ddots & a_{ii} & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ a_{n1} & \dots & a_{ni} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

La solution de cette équation est :

$x_1 = \frac{b_1}{a_{11}}$

$x_i = \frac{1}{a_{ii}}(b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j}_{S})$

Algorithme :

$x_1 \leftarrow \frac{b_1}{a_{11}}$ % x_1 *reçoit la valeur* $\frac{b_1}{a_{11}}$ *(la valeur de la première variable)*

$pour\ i = 2\ jusqu'à\ n$ *% (i décrit tous les indices de lignes).*

$S \leftarrow b_i$ *% (initialiser la somme S à la valeur* b_i*)*

$pour\ j = 1\ jusqu'à\ i-1$ *% (j décrit tous les indices de colonnes).*

$S \leftarrow S - a_{ij}x_j$ *% (la somme S reçoit la valeur initiale* $-a_{ij}x_j$ *)*

$finpour$ *% fin de la boucle pour*

$x_i \leftarrow \frac{S}{a_{ii}}$ *% (la solution* x_i *reçoit la valeur* $\frac{S}{a_{ii}}$ *)*

$finpour$ *% fin de la boucle pour*

2.2.3. Troisième cas : « A est de forme quelconque »

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \ddots & a_{ii} & \ddots & b_i \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ni} & \cdots & b_n \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

Pour résoudre ce système on doit faire tout d'abord la triangulation de la matrice A, afin d'obtenir une matrice triangulaire notée (AA), (voir ci-dessous), ensuite on utilise l'algorithme suivant :

$$\underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & a_{ii} & \ddots & a_{in} \\ \vdots & \ddots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & a_{nn} \end{bmatrix}}_{AA} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

La solution de cette équation est :

$x_n = \frac{b_n}{a_{nn}}$

$x_i = \frac{1}{a_{ii}}(b_i - \underbrace{\sum_{j=i+1}^{n} a_{ij}x_j}_{S})$

Algorithme :

$x_n \leftarrow \frac{b_n}{a_{nn}}$ *% x_n reçoit la valeur $\frac{b_n}{a_{nn}}$ (initialisation de la première solution)*

$pour\ i = n-1\ jusqu'à\ 1$ *% (i décrit tous les indices des lignes).*

$S \leftarrow b_i$ *% (initialiser la somme S à la valeur b_i)*

$pour\ j = i+1\ \ jusqu'à\ \ n$ *% (j décrit tous les indices des colonnes).*

$S \leftarrow S - a_{ij}x_j$ *% (la somme S reçoit la valeur initiale $-a_{ij}x_j$)*

$finpour$ *% fin de la boucle pour*

$x_i \leftarrow \frac{S}{a_{ii}}$ *% (la solution x_i reçoit la valeur $\frac{S}{a_{ii}}$)*

$finpour$ *% fin de la boucle pour*

2.3. ***Triangulation d'une matrice :***

La triangulation d'une matrice A contient 3 étapes [5 – 7]:

1ere étape :

On sélectionne le premier élément de la première ligne qui représente le pivot (1) de la première ligne.

Exemple :

Pivot

$$\begin{cases} -7x_1 & -3x_2 & +3x_3 & = & 12 \\ 2x_1 & +2x_2 & +2x_3 & = & 0 \\ -x_1 & -4x_2 & +3x_3 & = & -9 \end{cases}$$

2eme étape :

Pour éliminer les coefficients de x_1 de la première ligne, on effectue des combinaisons linéaires d'équations.

La Ième ligne (i) devient $(i) \leftarrow (i) - \frac{a_{i1}}{a_{11}}$

et les coefficients de la ligne (i) seront remplacés par $a_{ij} \leftarrow a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}$

avec, $2 \ll i \ll n$

$2 \ll j \ll n$

Exemple :

$$\begin{cases} -7x_1 & -3x_2 & +3x_3 & = & 12 \\ 2x_1 & +2x_2 & +2x_3 & = & 0 \\ -x_1 & -4x_2 & +3x_3 & = & -9 \end{cases}$$

$$eq(2) = eq(2) - \frac{a_{21}}{pivot} eq(1)$$

$$\begin{cases} -7x_1 & -3x_2 & +3x_3 & = & 12 \\ 0x_1 & +\frac{8}{7}x_2 & +\frac{6}{7}x_3 & = & \frac{24}{7} \\ -x_1 & -4x_2 & +3x_3 & = & -9 \end{cases}$$

3eme étape:

La combinaison linéaire doit être appliquée à tous les éléments de la ligne i :

$$a_{ij}^{(k)} \leftarrow a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \qquad k+1 \leq j \leq n$$

$$b_i^{(k)} \leftarrow b_i^{(k-1)} - \frac{a_{ik}^{(k+1)}}{a_{kk}^{(k-1)}} b_k$$

Exemple :

$$\begin{cases} -7x_1 & -3x_2 & +3x_3 & = & 12 \\ 2x_1 & +2x_2 & +2x_3 & = & 0 \\ -x_1 & -4x_2 & +3x_3 & = & -9 \end{cases}$$

$$eq(2) = eq(2) - \frac{a_{21}}{pivot} eq(1)$$

$$\begin{cases} -7x_1 & -3x_2 & +3x_3 & = & 12 \\ 0x_1 & +\frac{8}{7}x_2 & +\frac{6}{7}x_3 & = & \frac{24}{7} \\ -x_1 & -4x_2 & +3x_3 & = & -9 \end{cases}$$

$$eq(3) = eq(3) - \frac{a_{31}}{pivot} eq(1)$$

$$\begin{cases} -7x_1 & -3x_2 & -3x_3 & = & 12 \\ 0x_1 & +\frac{8}{7}x_2 & +\frac{6}{7}x_3 & = & \frac{24}{7} \\ 0x_1 & -\frac{25}{7}x_2 & +\frac{18}{7}x_3 & = & -\frac{61}{7} \end{cases}$$

2.4. Algorithme du pivot de Gauss

1. Triangulation

```
AB=(A,b)      % la matrice  AB regroupe les deux matrices A et b
pour k=1 jusqu'à n-1  % k décrit tous les indices des lignes et des colonnes
  si AB(k,k)==0   % vérification du premier élément de la matrice AB si il est différent de zéro.
    % si le pivot =0, chercher une autre ligne pivot
    pivot_trouve=0;
    i=k+1;    % incrémenter l'indice pour aller à la ligne suivante
    tant que pivot_trouvet=0 et i<=n  % condition de l'indice i, si il est plus petit que la taille de la matrice AB, alors on fait les étapes suivantes.
        si AB(i,k)<>0  % si le premier élément de la nouvelle ligne est différent de zéro, alors on fait une permutation entre cette ligne et la ligne précédente
                permuter ligne K et ligne I de AB
                pivot_trouve=1
        sinon
        i=i+1; % sinon, on passe à la ligne suivante
        finsi
    finsi
    si i>n  % si i est plus grand que la taille de la matrice AB, alors on déclare qu'il n y a pas de pivot.
        ecrire('pas de ligne pivot');
        exit;
    finsi
  finsi
  %triangulation continue
  pour i=k+1 jusqu'à n
    ligne i =ligne i- ligne k * AB(i,k)/AB(k,k)
  finpour
fin
```

2.5. ***Code Matlab***

Utiliser le programme suivant pour résoudre ce système d'équations

```
a=[1 2 3;2 1 2;3 1 -4];b=[-5;-5;6];

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


%autre solution pour la triangulation

function X=gauss(A,b)

%triangulation d'une matrice quelconque

n=size(A,1);

if n~=size(b,1)

   printf('problème de taille entre A et b');
```

```
end
AB=[A,b];
for k=1:n-1
   if AB(k,k)==0
      arret=0;
      i=k+1;
      while arret==0 && i<=n
         if AB(i,k)~=0
            ech=AB(k,:);
            AB(k,:)=AB(i,:);
            AB(i,:)=ech
            arret=1;
         else
            i=i+1;
         end
      end;
      if i>n
         fprintf('pas de ligne pivot');
         exit;
      end
   end
   %triangulation continue
   for i=k+1:n
      AB(i,:)=AB(i,:)-AB(k,:)*AB(i,k)/AB(k,k);
   end
end
X=AB;
end
```

2.6. Travail demandé

Ecrire un programme Matlab qui permet de résoudre les systèmes d'équations suivantes :

a) $a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ et b $= \begin{bmatrix} 3 \\ 6 \\ 5 \end{bmatrix}$

b) $a = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$ et b $= \begin{bmatrix} 3 \\ 6 \\ 5 \end{bmatrix}$

c) $a = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 2 & 0 \\ 3 & -1 & 3 \end{bmatrix}$ et b $= \begin{bmatrix} 3 \\ 6 \\ 5 \end{bmatrix}$

d) $a = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 2 & 4 \\ 3 & -1 & 3 \end{bmatrix}$ et b $= \begin{bmatrix} 3 \\ 6 \\ 5 \end{bmatrix}$

e) $a = \begin{bmatrix} 12 & -2 & 3 \\ 1 & 10 & 2 \\ 3 & 1 & 15 \end{bmatrix}$ et $b = \begin{bmatrix} 18 \\ 16 \\ 52 \end{bmatrix}$

f) $a = \begin{bmatrix} 3 & 4 & -2 & 2 \\ 4 & 9 & -3 & 5 \\ -2 & -3 & 7 & 6 \\ 1 & 4 & 6 & 7 \end{bmatrix}$ et $b = \begin{bmatrix} 2 \\ 8 \\ 10 \\ 2 \end{bmatrix}$

TPN3: Résolution des équations non linéaires

3.1. Introduction :

Une équation est dite non linéaire lorsqu'elle implique des termes de degré supérieur à 1 dans la quantité inconnue. Ces termes peuvent être polynomiaux ou susceptibles d'être décomposés [7 – 8].

Dans certains cas, il est possible de trouver les racines exactes de la fonction f(x)=0, par exemple lorsque f(x) est un polynôme quadratique, ou cubique, sinon, on essaie de trouver des solutions utilisant certains méthodes numériques (des méthodes itératives). Le principe de ces méthodes de résolution consistent à partir d'un point arbitraire - le plus proche possible de la solution recherchée - et d'arriver à la solution progressivement à travers des tests successifs [7 – 8].

Dans ce TP on va voir trois méthodes : méthode de dichotomie, méthode de point fixe, et méthode de Newton Raphson.

3.2. Méthode de dichotomie

La méthode de dichotomie est une méthode numérique utilisée pour résoudre des équations avec une seule inconnue. Le mot dichotomie est composé de deux mots (dicho = deux, tomie = coupe). Il exprime clairement le principe de la méthode.

Soit l'équation $f(x) = 0$ continue sur l'intervalle [a, b]. La fonction f(x) prend des valeurs de signes différent aux extrémités de cet intervalle ($f(a)f(b) < 0$), alors cette équation admet une solution (s) tel que $s \in [a, b]$. [5-10]

3.2.1. Algorithme de dichotomie

Pour trouver s approximativement dans l'intervalle [a, b] :

1- Trouver a et b tels que $f(a)f(b) \leq 0$
2- On devise [a, b] en deux ($m = \frac{1}{2}(a + b)$) et on calcule la valeur de $f(m)$.
3- On prend les deux intervalles [a, m] et [m, b] et parmi eux on sélectionne pour la dichotomie suivante celui aux extrémités duquel les valeurs de la fonction différents en signe. C .à. d. si $f(a)f(m) \leq 0$, poser $b = m$, (s est dans [a, m]), sinon poser $a = m$ (s est dans [m, b])
4- Reprendre à partir de (1) tant que $|b - a| \geq \varepsilon$

```
Tant que (abs (b-a) > epsilon)  % test d'arrêt
m=(a+b)/2
si (f(a) * f(m)) < 0)
b=m  %s est dans [a,m]
sinon
a = m  % s est dans [m,b]
Fin si
Fin tant que
```

Cette méthode est relativement lente. Par contre elle converge dans tous les cas où la fonction change de signe au voisinage de la racine (ce qui exclut les racines de multiplicités paires).

Exemple

Les points successifs pour les premiers pas d'itération apparaissent sur la figure 1 pour la fonction $f(x) = e^{-2x} - cosx - 3,$ avec a=-2 et b=2.

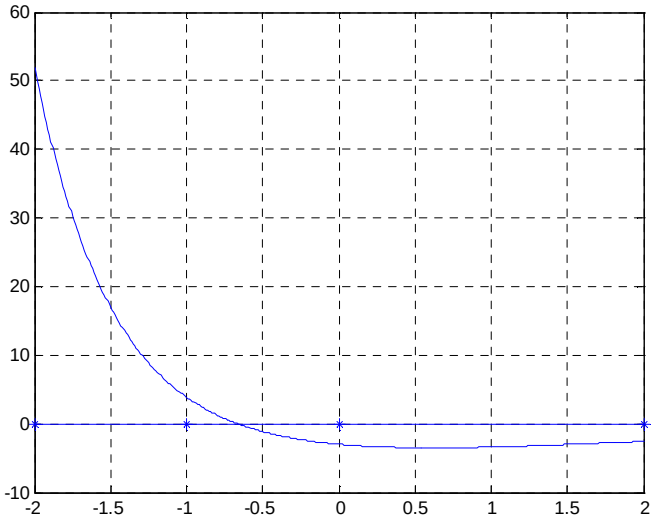

Figure 3.1: Méthode de dichotomie

```
clear all, clc
a=-2, b=2,
x=[a:0.01:b];
f=inline('exp(-2*x)-cos(x)-3')
v=[-2 -1 0 2];
y= [0 0 0 0];
plot(x,f(x)), grid, hold on, scatter(v,y,40,'b','*')
plot(v,y)
```

Exemple :

Ecrire un script Matlab permettant de calculer la racine de la fonction $f(x) = 2\sin(x) - x$ située dans l'intervalle [1, 6] par la méthode de dichotomie.
Indications : On prendra un test d'arrêt de la forme $|x_{n+1} - x_n| < 10^{-4}$.

3.2.2. Code Matlab

```
a=1, b=6
fa=2*sin(a)-a;
fb=2*sin(b)-b;
eps=0.0001;
nb=0;
 fprintf('nb | x | f(x) \n')
 fprintf('-------------------------------------\n')
 if fa*fb<0
while (b-a)>eps
 nb=nb+1;
 x=(a+b)/2;
 fx=2*sin(x)-x;
 if fa*fx<0
b=x;
 else
 a=x;
 end
 fprintf('%i | %10.5f | %10.5f \n',nb,x,fx)
end;
end
```

3.2.3. Travail demandé :

Ecrire un script Matlab permettant de calculer les racines des fonctions suivantes en utilisant la méthode de dichotomie:

Indications : On prendra un test d'arrêt de la forme $|x_{n+1} - x_n| < \varepsilon$.

1) $f(x) = x^3 - 2x - 5$, dans l'intervalle [0, 3], $\varepsilon = 10^{-6}$
2) $f(x) = 1 - xe^x$, [0.5, 1], $\varepsilon = 10^{-10}$
3) $f(x) = e^x + 3\sqrt{x} - 2$, [0, 1], $\varepsilon = 10^{-10}$

3.3. Méthode de point fixe

3.3.1. Principe de la méthode

Le principe de la méthode est basé sur la transformation de l'équation f(x)=0 en une équation équivalente g(x)=x, avec $x \in [a, b]$, ensuite la construction d'une suite itérative partant d'une valeur initiale $x_0 \in [a, b]$, ensuite on calcule $x_1 = g(x_0)$, puis on calcul $x_2 = g(x_1)$, est ainsi de suite,

approchant de plus en plus la racine exacte, jusqu'à la stabilisation du processus $x_n \approx g(x_{n-1})$, avec une précision demandée (ε) qui est appelée test d'arrêt [5 – 10].

Remarque :

1) La valeur initiale x_0 pouvant être n'importe quel point de l'intervalle [a, b]. On termine les itérations lorsque la condition suivante est vérifiée.

$$\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < \varepsilon$$

2) La fonction g est une fonction dépendante de f non unique comme le montre l'exemple suivant [5 – 10].

Exemple 1

Si $f(x) = \sin(2x) - 1 + x = 0$; la fonction g peut être
$g_1(x) = 1 - \sin(2x), \quad x \in IR$

Ou
$g_2(x) = \frac{1}{2}\arcsin(1 - x), \quad 0 \leq x \leq 1$

Le code Matlab suivant permet de tracer les représentations graphiques de ces fonctions (f(x), g1(x), et g2(x)), et la droite y = x :

Code Matlab

```
x = [0:0.001:1];
f = inline('sin(2*x)-1 + x');
g1 = inline('1-sin(2*x)');
g2 = inline('1/2*(asin(1-x))');
h = inline('x');
plot(x, f(x), '--.b', x, g1(x), '-.b', x, g2(x), '--b', x,
h(x),'b');
legend('f', 'y=1-\sin(2x)', 'y=1/2*(Arcsin(1-x))', 'y=x');
grid on;
ylabel('y(x)');
xlabel('x');
```

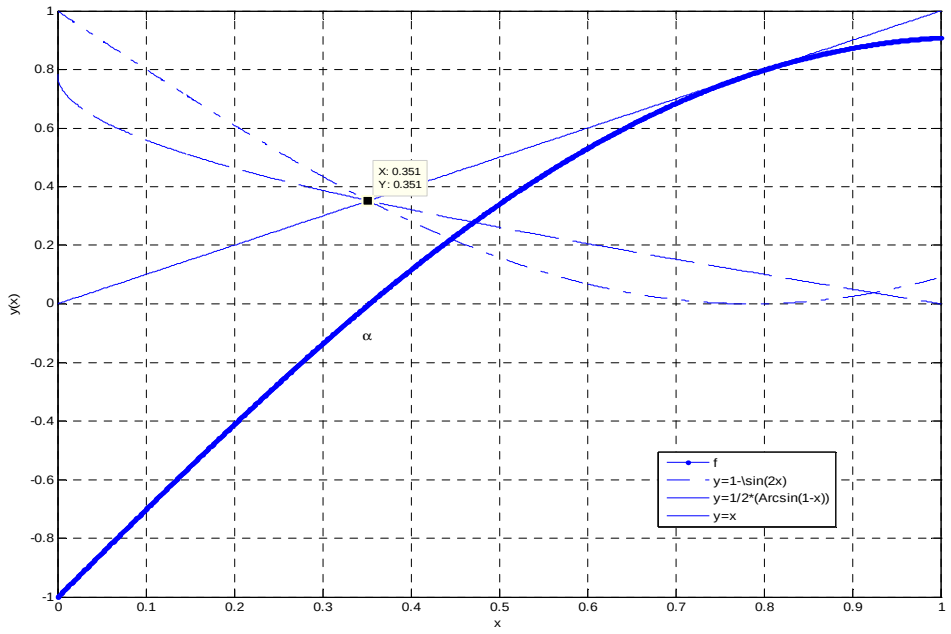

Figure 3.2. Représentation graphique des fonctions (f(x), g1(x), et g2(x)), et la droite y = x

On remarque que f admet un zéro unique dans l'intervalle [0; 1] et que les graphes des fonctions y = x; y = 1-sin(2x); et y = 0.5(arcsin(1-x)) se coupent en un seul point (α, α)=(0.351,0.351).

Exemple 2

Soit l'équation suivante

$$x^2 - x - 1 = 0 \qquad (3.1)$$

Nous avons deux racines :

$x = \frac{1 \pm \sqrt{5}}{2}$,

$x_1 = 1.61803398, \qquad x_2 = -0.61803398$

L'équation *(1)* peut s'écrire sous la forme suivante :

$x^2 = x + 1 \qquad (3.2)$

$g_1(x) = \frac{x+1}{x} = 1 + \frac{1}{x}$

$g_2(x) = x^2 - 1$

$g_3(x) = \sqrt{x+1}$

On peut prendre par exemple $g_1(x) = 1 + \frac{1}{x}$ ➔ $x_{n+1} = 1 + \frac{1}{x_n}$

On veut trouver une solution pour l'équation (3.1) avec une précision $\varepsilon = 10^{-3}$.

Prenant par exemple une solution initiale x_0=1, car x_0 ne peut pas être 0.

Alors $x_1=g(x_0)$ =1+1=2, puis on calcule les termes suivants jusqu'à ou on obtient la précision demandée (voir tableau 3.1). Dans notre exemple, on remarque que la huitième itération a une erreur relative de 0.00008 ce qui signifie qu'on a la précision demandée. La solution approchée pour l'équation (1) est donc 1.6177.

n	0	1	2	3	4	5	6	7	8
x_n	1	2	1.5	1.6666	1.6	1.625	1.6154	1.619	1.6177
$\left\lvert\frac{x_n - x_{n-1}}{x_{n-1}}\right\rvert$		1	0.5	0.1111	0.04	0.0156	0.0059	0.0022	0.00008

Tableau 3.1 : *Valeurs des solutions et test d'arrêt pour la fonction $g_1(x)$.*

Contre-exemple :

On va prendre maintenant la fonction : $g_2(x) = x^2 - 1$ ➔ $x_{i+1} = x_i^2 - 1$,

n	0	1	2	3	4	5	6
x_n	1	0	-1	0	-1	0	-1
$\left\lvert\frac{x_n - x_{n-1}}{x_{n-1}}\right\rvert$		-1	ID	-1	ID	-1	ID

Tableau 3.2 : *Valeurs des solutions et test d'arrêt pour la fonction $g_2(x)$.*

D'après le tableau 3.2, on remarque que la solution continuera à fluctuer entre 0 et -1 et ne convergera jamais.

Critères sur g(x)

Pour que la méthode du point fixe soit applicable, il faut que la fonction g(x) réponde aux critères suivants [5-10] :

$$\begin{cases} g \text{ est continue et dérivable sur } [a,b] \\ g \text{ prend ses valeurs en } [a,b] \\ \exists\, M \in [0,1] : \forall\, x \epsilon\, [a,b]\ \ |g'(x)| \leq M \end{cases}$$

3.3.2. Algorithme de la méthode du point fixe

Etant donnée une équation *f(x)=0*, une précision ε et un intervalle [a,b], l'algorithme de la méthode du point fixe s'énonce comme suit [5-10] :

1- Ecrire l'équation f(x)=0, sous la forme x=g(x) avec g(x) répond aux trois critères cités ci-dessus.

2- Choisir une solution initiale $x_0 \in [a, b]$

3- Calculer $x_1 = g(x_0)$

4- Tant que $\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| > \varepsilon$, calculer le terme $x_{n+1} = g(x_n)$

3.3.3. Code Matlab

```
% ce programme résoudre l'équation x^2 - x - 1 = 0 en utilisant la méthode du
point fixe dans l'intervalle [1 2]
clear all
clc
% choix de g(x)
% f(x)= 0 ==> x^2-x-1=0
%  x^2-x-1=0 ==> x=1+1/x ; % donc on peut prendre g(x)= 1+1/x
g=inline('1+1./x');

% la solution initiale x(1)=1
x(1)=input('entrer la solution initiale x1:');

% calcul de la deuxième solution approximative x(2) est qui sera plus
proche à  la solution analytique.
x(2)=g(x(1));
n=2;
precision=10^(-3);
while ((x(n)-x(n-1))/x(n-1))> precision
    x(n+1)=g(x(n));
    n=n+1
end
x'
```

3.3.4. Travail demandé

Trouver la solution approchées des trois équations f(x), en appliquant la méthode de point fixe sur les fonctions g(x) suivantes :

1) $f(x) = 2\sin(x) - x$ $\quad [a, b] = [1.5, 2],\ \varepsilon = 10^{-6},\ \ x_0 = 0.5$
   a) $g_1(x) = 2\sin(x)$
   b) $g_2(x) = \arcsin(\frac{x}{2})$

2) $f(x) = e^x - x - 2$ $\quad$ [a, b]=[1, 2], $\varepsilon = 10^{-6},\quad x_0 = 1$
   a) $g_1(x) = e^x - 2$
   b) $g_2(x) = \ln(2 + x)$

3) $f(x) = x^2 - x - 1$ $\quad$ [a, b]= [-1, 2], $\quad \varepsilon = 10^{-3},\quad x_0 = 1$, ensuite $x_0 = -0.5$
   a) $g_1(x) = x^2 - 1$

b) $g_2(x) = \sqrt{x+1}$

c) $g_3(x) = 1 + \frac{1}{x}$

Le processus converge t-il vers la solution approchées ?
Comparer le nombre d'itérations pour les fonctions g(x) pour chaque équation f(x).

3.4. *Méthode de Newton Raphson*

En analyse numérique, la méthode de Newton, également connue sous le nom de méthode Newton – Raphson, du nom d'Isaac Newton et Joseph Raphson. C'est un algorithme de recherche de racines qui produit successivement de meilleures approximations aux racines (ou zéros) d'une fonction à valeur réelle.

Comme hypothèse initiale, cette méthode ne nécessite qu'un seul point de départ approprié x0. À (x0, f (x0)) une tangente à f (x) = 0 est dessinée (voir figure 3.3). L'équation de cette tangente est donnée par l'équation (3.3) [5-10].

$$y = f'(x_0)(x - x_0) + f(\mathrm{x_0}) \qquad (3.3)$$

Le point d'intersection, de cette tangente avec l'axe des abscisses (y = 0) est considéré comme la prochaine approximation de la racine de f (x) = 0. Donc, en substituant y = 0 dans l'équation tangente, nous obtenons :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Si $\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < \varepsilon$ (ε *est une précision donnée*), nous avons une racine approximative acceptable de f (x) = 0, sinon nous remplaçons x_0 par x_1 , et dessinons une tangente à la fonction f (x) = 0 au point $(x_1, f(x_1))$ et considérons son intersection, avec l'axe des x comme une approximation améliorée de la racine de f (x) = 0.

Si $\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| > \varepsilon$, nous itérons le processus ci-dessus jusqu'à ce que les critères de convergence soient satisfaits. Cette description géométrique de la méthode peut être clairement visualisée dans la figure ci-dessous (figure 3.3):

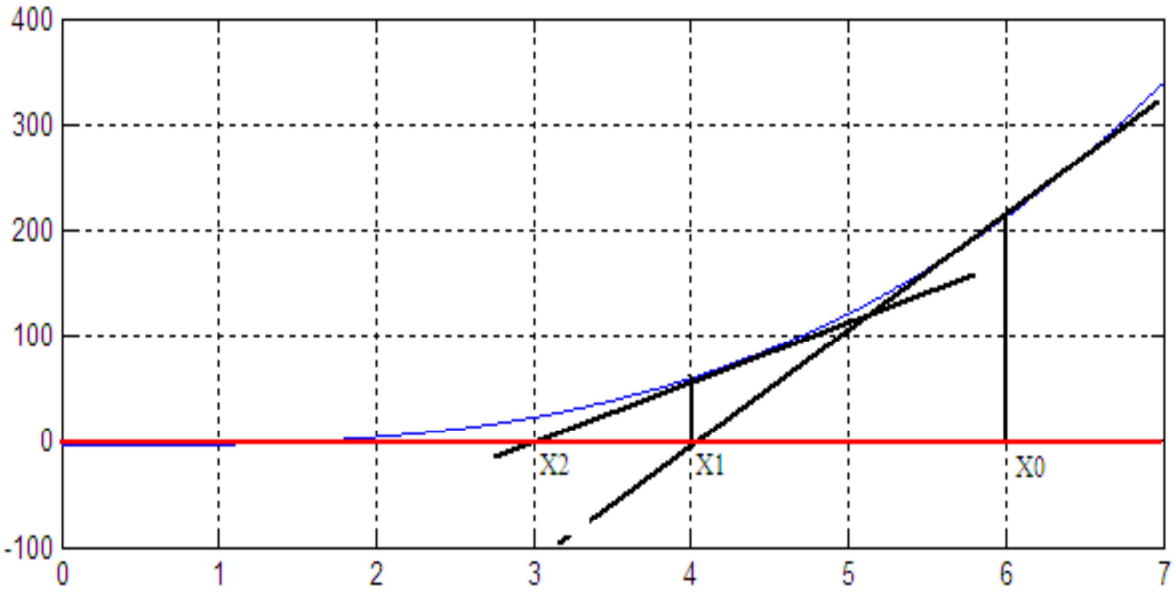

Figure 3.3: Principe de la méthode de Newton Raphson

Le choix de la solution initiale a une grande importance lorsqu'on calcule la solution d'une équation par la méthode de Newton.

Les différentes étapes impliquées dans le calcul de la racine de f (x) = 0 par la méthode Newton Raphson sont décrites dans l'algorithme ci-dessous.

3.4.1. Algorithme:

1. Entrer la formule de f.
2. Entrer la formule de sa dérivée.
3. Donner une solution initiale x_0 qui doit être proche de la solution recherchée. Dans cette étape on peut tracer la courbe de f pour donner une estimation de la solution initiale x_0.
4. Calculer les termes $\boldsymbol{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$
5. Répéter la 4eme étape, jusqu'à la stabilisation du processus avec la précision demandée (ε) qui est appelée par 'test d'arrêt' $\left|\frac{x_n - x_{n-1}}{x_{n-1}}\right| < \varepsilon$.

Cette méthode peut être dérivée directement par le développement de Taylor au voisinage de la racine r. L'approximation de départ x0 de la fonction f(x)=0 doit être correctement choisie pour que l'approximation de la série de Taylor du premier ordre de f(x0+h), au voisinage de x0 mène à une meilleure approximation de xi c'est-à-dire :

$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \cdots = 0$

$h \ll 1$, negligeant h^2 ➔ $f(x_0) + hf'(x_0) = 0$

i.e $h = -\frac{f(x_0)}{f'(x_0)}$, avec $h = x_1 - x_0$, donc :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Calculant les approximations $x_2, x_3, \ldots\ x_n, \ldots$ etc en utilisant la formule suivante :

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

Exemple résolu :

Donner une valeur approchée de l'équation suivante:

$x^3 - 3x - 5 = 0$, $x \in [2,3]$, avec $\varepsilon = 10^{-3}$

Solution :

On pose : $f(x) = x^3 - 3x - 5$
$f(2) = -3$
$f(3) = 16$ ➔ $x_0 = 2$
On a :
$f'(x) = 3x^2 - 3$

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

n	0	1	2	3	4
x_n	2	2.333	2.2805	2.279	2.2790
$\left\lvert\frac{x_n - x_{n-1}}{x_{n-1}}\right\rvert$		0.1665	0.0225	$6.5775.10^{-4}$	0

Tableau 3.3 : *Valeurs des solutions et test d'arrêt pour la fonction f(x).*

D'après le tableau (3.3), on remarque que la 4ème itération (n=3) répond à la précision demandée.

3.4.2. Code Matlab

```
% ce programme résoudre l'équation x.^3-3.*x-5=0 en utilisant la
méthode de Newton Raphson dans l'intervalle [2, 3].
% la solution initiale X1=2.
clear all
clc
f=inline('x.^3-3.*x-5');
f1= inline('3.*x.^2-3');
x(1)=input('entrer la valeur de la solution initiale');
x(2)=x(1)-(f(x(1))/f1(x(1)));
n=2;
precision=10^(-4);
while ((x(n)-x(n-1))/x(n-1))> precision
    x(n+1)=x(n)-(f(x(n))/f1(x(n)));
    n=n+1;
end
x'
```

3. 4. 3. Travail demandé:

Résoudre les équations suivantes par la méthode de Newton Raphson:

$x^3 - 3x - 5 = 0$, $x \in [2,3]$

$cosx - xe^x = 0$, $x \in [0,1]$

$2(x-3) = log_{10}x$, $x \in [3,4]$

TPN4: Intégration numérique

4.1. Introduction

Habituellement, pour trouver l'intégrale d'une fonction f(x), nous utilisons les théorèmes fondamentaux du calcul, où nous devons appliquer les techniques d'intégration. Mais parfois, il est difficile de trouver la primitive d'une fonction, dans le cas des expériences scientifiques, où la fonction doit être déterminée à partir des lectures observées à titre d'exemple. Par conséquent, les méthodes numériques sont utilisées pour approcher l'intégrale dans telles conditions.

Dans ce TP nous avons décrit 4 méthodes d'intégration : méthode de Trapèze, règle 1/3 de Simpson, règle 3/8 de Simpson, et méthode de Newton Cotes.

4.1.1. Méthode de Trapèze

La méthode de trapèze est une méthode qui évalue la zone sous les courbes en divisant la zone totale en plusieurs trapèzes plus petits (figure 4.1). Cette intégration fonctionne en approximant la région sous le graphe d'une fonction comme un trapèze, puis en calculant l'aire [5-10].

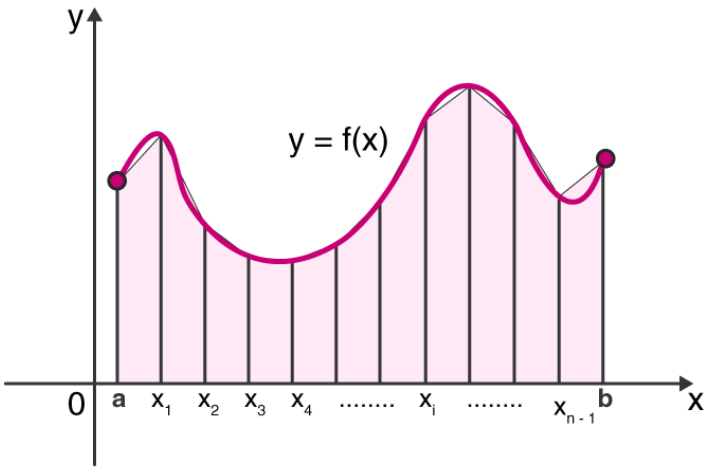

Figure 4.1 : Méthode de Trapèze

Formule de la méthode de trapèze:

Soit f (x) une fonction continue sur l'intervalle [a, b], divisé en n sous-intervalles égaux de taille (Δx).

$\Delta x = \frac{(b-a)}{n}$ tel que : $a = x_0 < x_1 < x_2 < \cdots < x_{n=}b$ (4.4)

La méthode de trapèze est donnée par la formule (4.5):

$$\int_a^b f(x)dx \approx T_n = \frac{\Delta x}{2}\,[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)] \quad (4.5)$$

Avec, $x_i = a + i\Delta x$

Exemple résolu :

Approximer l'aire sous la courbe y = f (x) entre x = 0 et x = 8 en utilisant la règle trapézoïdale avec n = 4 sous-intervalles. Les valeurs de la fonction f (x) est donnée dans le tableau (4.1)

x	0	2	4	6	8
f(x)	3	7	11	9	3

Tableau 4.1 : *Valeurs de la fonction f(x)*

Solution:

La formule de la règle trapézoïdale pour n = 4 sous-intervalles est donnée comme suit:

$T_4 = \left(\frac{\Delta x}{2}\right)[f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + f(x_4)]$

Ici, la largeur du sous-intervalle $\Delta x = \frac{8-0}{2} = 2$

En remplaçant les valeurs du tableau, pour trouver la valeur approximative de l'aire sous la courbe.

$$T_4 = \frac{2}{2}[3 + 2(7) + 2(11) + 2(9) + 3] = 60$$

Par conséquent, la valeur approximative de l'aire sous la courbe à l'aide de la règle trapézoïdale est de 60.

4.1.1.1. Code Matlab :

```
% Calculez ∫_1^5 1/(x+2) dx, par la méthode de Trapèze
clear all, clc,
a=1, b=5, n=2
h=(b-a)/n;
for i=0:n
    x(i+1)=a+i*h;
end
x
Y=1./(x+2);
ans= Y(1)+Y(n+1)
for i=2:n
    ans=ans+(2*Y(i));
end

ans=(h/2)*ans
```

Cependant, la méthode de trapèze ne donne pas une valeur précise. Par contre, la méthode de Simpson donne une approximation plus précise que celle-ci. Ce défaut revient au calcul approché de l'intégrales où nous remplaçons un arc de courbe $(\widehat{M_i M_{i+1}})$ par le segment $[M_i M_{i+1}]$. Cette méthode fort simple à programmer reste cependant très imprécise. Alors que Simpson apporte une correction très efficace en utilisant l'interpolation polynomiale.

4.1.2. Méthode de Simpson

Les règles de Simpson sont plus précises que les autres approximations) numériques et sa formule pour n + 1 subdivisions également espacées est donnée par deux règles : Règle 1/3 et règle 3/8 de Simpson [5-10].

a) Règle 1/3 de Simpson

La règle 1/3 de Simpson est une extension de la règle trapézoïdale dans laquelle l'intégrale est approchée par un polynôme du second ordre.

$$\int_a^b f(x)dx \approx S_n = \frac{\Delta x}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] \quad (4.6)$$

Où n est un nombre pair, $\Delta x = \frac{(b-a)}{n}$ et $x_i = a + i\Delta x$, n : doit être un nombre pair.

Si nous avons $f(x) = y$, qui est également espacé entre $[a, b]$et si :

$a = x_0,\ x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh$ où h est la différence entre les termes.

Nous pouvons dire que $y_0 = f(x_0),\ y_1 = f(x_1),\ y_2 = f(x_2), \ldots,\ y_n = f(x_n),$ sont les valeurs analogues de y avec chaque valeur de x.

Exemple résolu:

Calculez $\int_1^5 \frac{1}{1+x^2} dx$, par la règle ⅓ de Simpson.

Solution:

Divisons l'intervalle [1,5] en quatre parties égales en prenant $h = \frac{5-1}{4} = 1$

x	1	2	3	4	5
$f(x)$	$1/3$	$1/4$	$1/5$	$1/6$	$1/7$

$\int_a^b f(x)dx = \frac{h}{3}[(y_0 + y_n) + 4(y_1 + y_3 + \cdots) + 2(y_2 + y_4 + \cdots)]$ où $h = \frac{b-a}{n} = \frac{5-1}{4} = 1,$
n doit être un nombre pair.

$\int_1^5 \frac{1}{x+2} dx = \frac{1}{3}[\left(\frac{1}{3} + \frac{1}{7}\right) + 4\left(\frac{1}{4} + \frac{1}{6}\right) + 2(\frac{1}{5})]$

$\int_1^5 \frac{1}{x+2} dx = 0.8477$

a.1) Code Matlab

```
% Calculez ∫_1^5 1/(x+2) dx, par la méthode 1/3 de Simpson
clear all,
clc,
a=1
b=5
n=4
h=(b-a)/n;
for i=0:n
    x(i+1)=a+i*h;
end
x
Y=1./(x+2);
ans= Y(1)+Y(n+1)
for i=2:2:n
    ans=ans+(4*Y(i));
end
for i=3:2:n
    ans=ans+(2*Y(i));
end
ans=(h/3)*ans
```

b) Règle 3/8 de Simpson

Une autre méthode d'intégration numérique est la «règle 3/8 de Simpson». Il est entièrement basé sur l'interpolation cubique plutôt que sur l'interpolation quadratique. La règle 3/8 ou trois-huit de Simpson est donnée par [5-10]:

$$\int_a^b f(x)\,dx = \frac{3h}{8}[(y_0 + y_n) + 3(y_1 + y_2 + y_4 + y_5 + \cdots + y_{n-1}) + 2(y_3 + y_6 + y_9 + \cdots + y_{3i} + \cdots y_{n-3})]$$

Avec : $1 \le i \le n$

Exemple résolu:

Calculer l'intégral $\int_0^6 f(x)\,dx = \int_0^6 \frac{dx}{1+x^2}$

Solution :

$$\int_a^b f(x)\,dx = \frac{3h}{8}[(y_0 + y_n) + 3(y_1 + y_2 + y_4 + y_5 + \cdots) + 2(y_3 + y_6 + y_9 + \cdots)]$$

Où $h = \frac{b-a}{n} = \frac{6-0}{6} = 1,$ n doit être un multiple de 3, généralement on prend n=6.

x	0	1	2	3	4	5	6
F(x)	1	0.5	0.2	0.1	0.589	0.0385	0.027

$$\int_0^6 \frac{1}{1+x^2} dx = \frac{3(1)}{8}[(1 + 0.027) + 2(0.1) + 3(0.5 + 0.2 + 0.589 + 0.0385)]$$

$\int_0^6 \frac{1}{1+x^2} dx = 1.3571$

b.1) Code Matlab

```
Calculez ∫_0^6 1/(1+x^2) dx, par la méthode 3/8 de Simpson
clear all, clc,
a=0
b=6
n=6
h=(b-a)/n;
for i=0:n
    x(i+1)=a+i*h;
end
x
Y=1./(1+x.^2);
ans= Y(1)+Y(n+1);
for j=2:3:n-1
    ans=ans+(3*Y(j));
end
for k=3:3:n
    ans=ans+(3*Y(k));
end
for l=4:3:n
    ans=ans+(2*Y(l));
end
I=(3*h/8)*ans
```

4.1.3. Méthode de Newton Cotes

La méthode de Newton cotes est une famille de formules extrêmement utile pour l'intégration numérique.

Afin d'intégrer une fonction f(x) sur un certain intervalle [a, b], on divise cet intervalle en n parties égales de taille ($h = \frac{(b-a)}{n}$), ensuite on remplace la fonction f(x) par le polynôme de Lagrange de degré n ($P_n(x)$), enfin on calcul l'intégral pour approximer l'air sous la courbe.

Les formules résultantes sont appelées formules de Newton -Cotes. Ces formules se diffèrent selon le nombre des sous intervalles (n) [5-10].

1. Si n=1, donc on fait une approximation de la fonction f(x) en 2 points seulement $x_0 = a$, et $x_1 = b$.

$$\int_a^b f(x)dx$$

$$P_1(x) = \frac{(x-x_1)}{(x_0-x_1)} f(x_0) + \frac{(x-x_0)}{(x_1-x_0)} f(x_1) \qquad x_0 = a, \text{ et } x_1 = b.$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx = \int_{x_0}^{x_1} \frac{(x-x_1)}{(x_0-x_1)} f(x_0) + \frac{(x-x_0)}{(x_1-x_0)} f(x_1)$$

$$= \frac{1}{x_1-x_0} \int_{x_0}^{x_1}[(x-x_0)f(x_1)-(x-x_1)f(x_0)]dx$$

Ici les termes $x_0, x_1, f(x_0), f(x_1)$ sont des constantes.

$$\int_a^b f(x)dx = \frac{1}{(x_1-x_0)}[\frac{(x-x_0)^2}{2} f(x_1) - \frac{(x-x_0)^2}{2} f(x_0)]_{x_0}^{x_1}$$

$$= \frac{1}{x_1-x_0}[\frac{(x_1-x_0)^2}{2} f(x_1) + \frac{(x_0-x_1)^2}{2} f(x_0)], \qquad (x_0-x_1)^2 = (x_1-x_0)^2$$

$$\boxed{\int_a^b f(x)dx = \frac{x_1-x_0}{2}[f(x_1)+f(x_0)]}$$

Formule Trapézoïdale

$$\boxed{\int_a^b f(x)dx = \frac{h}{2}[f(x_1)+f(x_0)]} \qquad \text{avec } h = x_1 - x_0$$

2. Si n=2, donc il y a 3 points : (x_0, x_1, x_2)

$$P_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_2}[\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)]dx$$

$$\boxed{\int_a^b f(x)dx = \frac{h}{3}(f(x_0)+4f(x_1)+f(x_2))}$$

Formule 1/3 de Simpson

3. Si n=3, donc il y a 4 points : (x_0, x_1, x_2, x_3)

$$\boxed{\int_a^b f(x)dx = \frac{3h}{8}(f(x_0)+3f(x_1)+3f(x_2)+f(x_3)}$$

Formule 3/8 de Simpson

Exemple résolu:

Calculer l'intégral de la fonction suivante :

$$\int_0^{\frac{\pi}{2}} \sin(x)dx \approx 1 - \frac{1}{\sqrt{2}} \approx 0.29289322$$

Solution :

1. **R.T :**

$$\mathrm{h} = \frac{\pi}{4}$$

$$\int_0^{\frac{\pi}{2}} \sin(x)dx = \frac{\frac{\pi}{4}}{2}\left[\sin(0) + \sin\left(\frac{\pi}{4}\right)\right] = 0.27768018$$

2. **R .1/3 S :**

$$\mathrm{h} = \frac{\pi}{8}$$

$$\int_0^{\frac{\pi}{2}} \sin(x)dx = \frac{\frac{\pi}{8}}{3}\left[\sin(0) + 4\sin\left(\frac{\pi}{8}\right) + \sin(\frac{\pi}{4})\right] = 0.29293264$$

3. **R. 3/8 S :**

$$\mathrm{h} = \frac{\pi}{12}$$

$$\int_0^{\frac{\pi}{2}} \sin(x)dx = \frac{3\frac{\pi}{12}}{8}\left[\sin(0) + 3\sin\left(\frac{\pi}{12}\right) + 3\sin\left(\frac{\pi}{6}\right) + \sin(\frac{\pi}{4})\right] = 0.2929107$$

4.1.4. Travail demandé

Ecrire le code Matlab permettant le calcul de l'intégration des fonctions ci-dessous, en appliquant les règles suivantes:

a) Règle des trapèzes
b) Règle de 1/3 de Simpson
c) Règle de 3/8 de Simpson

$$I_1 = \int_{0.1}^{1.3} 5xe^{-2x}\,dx$$

$$I_2 = \int_0^4 x^2 dx$$

$$I_3 = \int_0^{\frac{\pi}{4}} \sin(x)dx$$

d) Comparer les résultats obtenus par les différentes méthodes.

References:

[1] PETER I. KATTAN, "*MATLAB for Beginners: A Gentle Approach*", Petra Books, Smashwords Edition, ISBN: 978-1438203096, April 2008.

[2] STORMY ATTAWAY, "*Matlab: A Practical Introduction to Programming and Problem Solving"*, College of Engineering, Boston University, Boston, MA, Butterworth-Heinemann Inc, ISBN-10: 9780128045251, ISBN-13 : 978-0128045251, 2016 .

[3] www. Tutorialspoint.com/ MATLAB numerical computing, date: 13/01/2021.

[4] https://goalkicker.com/MATLABBook, date: 13/01/2021.

[5] WALTER GAUTSCHI, "*Numerical analysis*", Springer New York Dordrecht Heidelberg London, Library of Congress Control Number: 2011941359, Mathematics Subject Classification (2010), ISBN 978-0-8176-8258-3 e-ISBN 978-0-8176-8259-0, DOI 10.1007/978-0-8176-8259-0.

[6] JAMES F. EPPERSON, "*An introduction to numerical methods and analysis*", Mathematical Reviews, Published by John Wiley & Sons, Inc., Hoboken, New Jersey. Published simultaneously in Canada, ISBN 978-1-118-36759-9

[7] ROSTAMK. SAEED, KARWAN H.F. JWAMER, FARAIDUNK. HAMASALH, "*Introduction to Numerical Analysis*", University of Sulaimani, Kurdistan Region- Iraq, 2015, Printed by Pasha print , 2015.

[8] RICHARD. L. BURDEN, J. DOUGLAS FAIRES, "*Numerical analysis*", ninth edition, Brooks/ Cole Cengage learning, 2010.

[9] R. K. JAIN, S. R. K. LYENGAR, "*Numerical methods*", new age international publishers.

[10] RAO V. DUKKIPATI, "*Numerical methods*", Anshan, ISBN- 10: 1848290543, ISBN-13 978-1905740987.