UNIVERSITE ABDELHAMID IBN BADIS MOSTAGANEM
FACULTE DES SCIENCES ET DE LA TECHNOLOGIE
DEPARTEMENT DE GENIE CIVIL

Polycopié
2eme année
Génie Civil & Travaux Public

Travaux Pratiques
Méthodes Numériques

Produit et modifié par : Dr. Yassine ZELMAT
'MCB' au département de : Génie Civil

Année Universitaire : 2022/2023

Avant-Propos

Ce manuel pédagogique est un polycopié de travaux pratiques pour la matière "Méthodes numériques", destiné aux étudiants de la deuxième année de licence en génie civil. Il vise à donner aux étudiants une idée sur l'implémentation en MATLAB de quelques méthodes étudiées dans les différents chapitres du cours de méthodes numériques.

Les objectifs de ce polycopié sont les suivants :

- Présenter les fondements mathématiques du calcul scientifique en analysant les propriétés théoriques des méthodes numériques, tout en illustrant leurs avantages et inconvénients à l'aide d'exemples.
- Utiliser le logiciel de calcul scientifique MATLAB pour implémenter, visualiser et comparer les résultats.

Mr Zelmat Yassine

Résumé

Les méthodes numériques sont des techniques mathématiques et informatiques utilisées pour résoudre des problèmes mathématiques et scientifiques qui ne peuvent pas être résolus analytiquement. Ces méthodes incluent l'optimisation, l'algèbre linéaire, la résolution d'équations différentielles, l'intégration numérique et l'interpolation.

Les méthodes numériques sont utilisées dans de nombreux domaines, tels que l'ingénierie, les sciences physiques, les mathématiques, la finance et l'informatique. Elles ont des avantages, tels que la capacité à fournir des solutions rapides et précises à des problèmes complexes, mais elles peuvent également être sujettes à des erreurs d'arrondi et à des instabilités numériques.

Il est important de comprendre les propriétés et les limites de chaque méthode numérique afin de choisir la méthode appropriée pour résoudre un problème spécifique. En fin de compte, les méthodes numériques ont permis d'accomplir des avancées significatives dans de nombreux domaines scientifiques et technologiques, et continuent de se développer pour relever les défis scientifiques les plus complexes de notre temps.

Mots-clés : Matlab ; Méthodes Numériques ; Méthodes d'approximation ; Equations non linéaires ; Equations différentielles ; Système d’équations linéaires.

Abstract

Numerical methods are mathematical and computational techniques used to solve mathematical and scientific problems that can' not be solved analytically. These methods include optimization, linear algebra, differential equation solving, numerical integration, and interpolation.

Numerical methods it used in many fields, such as engineering, physical sciences, mathematics, finance, and computer science. They have advantages, such as the ability to provide fast and accurate solutions to complex problems, but they can also be prone to rounding errors and numerical instabilities.

It is important to understand the properties and limitations of each numerical method in order to choose the appropriate method to solve a specific problem. Eventually numerical methods have enabled significant advancements in many scientific and technological fields, and continue to develop to address the most complex scientific challenges of our time.

Keywords: Matlab; Numerical Methods; Approximation Methods; Nonlinear Equations; Differential Equations; Linear Equation Systems.

الملخص

الطرق العددية هي تقنيات رياضية وحوسبة تستخدم لحل المشاكل الرياضية والعلمية التي لا يمكن حلها تحليلياً. تشمل هذه الطرق التحسين، الجبر الخطي، حل المعادلات التفاضلية، التكامل العددي والتكامل التكميلي والترابط.

تستخدم الطرق العددية في العديد من المجالات مثل الهندسة والعلوم الفيزيائية والرياضيات والتمويل والحوسبة. لها مزايا، مثل القدرة على توفير حلول سريعة ودقيقة لمشاكل معقدة، ولكنها قد تكون عرضة لأخطاء التقريب وعدم الاستقرار العددي.

من المهم فهم خصائص وحدود كل طريقة عددية لاختيار الطريقة المناسبة لحل مشكلة محددة. في نهاية المطاف، ساعدت الطرق العددية على تحقيق تقدم كبير في العديد من المجالات العلمية والتكنولوجية، وما زالت تتطور لمواجهة التحديات العلمية الأكثر تعقيداً في زمننا الحالي.

الكلمات المفتاحية: ماتلاب؛ الأساليب العددية؛ أساليب التقريب؛ المعادلات غير الخطية؛ المعادلات التفاضلية؛ أنظمة المعادلات الخطية.

SOMMAIRE

1.1. LISTE DES FIGURES

Introduction générale

Le calcul scientifique implique le développement, l'analyse et l'application de méthodes mathématiques variées telles que l'analyse, l'algèbre linéaire, la géométrie, la théorie de l'approximation, les équations fonctionnelles, l'optimisation ou le calcul différentiel. Les méthodes numériques sont utiles pour résoudre les problèmes issus de domaines tels que la physique, les sciences biologiques, les sciences de l'ingénieur, l'économie et la finance. Avec l'évolution des ordinateurs et des algorithmes, il est possible de simuler des phénomènes réels en résolvant des problèmes de grande taille.

Ce polycopié est structuré en cinq chapitres. Chaque chapitre comprend un rappel théorique de différentes méthodes, un programme de test et un énoncé des travaux pratiques. Les résultats des tests sont également présentés pour évaluer les performances de chaque méthode.

Dans le premier chapitre, plusieurs méthodes pour résoudre numériquement des équations non linéaires sont exposées, notamment la méthode de dichotomie, la méthode de point fixe et la méthode de Newton.

Le deuxième chapitre traite du problème d'interpolation et d'approximation polynomiale. Les méthodes de Lagrange et de Newton sont étudiées.

Le troisième chapitre est consacré à l'intégration numérique de fonctions. Trois approches sont développées : la méthode des rectangles, la méthode des trapèzes et la méthode de Simpson.

Le quatrième chapitre aborde la résolution numérique des équations différentielles ordinaires. Deux méthodes sont présentées : la méthode d'Euler et la méthode de Runge-Kutta.

Enfin, le cinquième chapitre traite de la résolution numérique des systèmes d'équations linéaires. Trois approches sont exposées : la méthode de Gauss, la méthode de Jacobi et la méthode de Gauss-Seidel.

Résolution numérique des équations non linéaires

1. Introduction :

Le but principal de ce chapitre est de présenter des méthodes permettant d'approximer les racines d'une fonction réelle à une variable réelle, c'est-à-dire de trouver une solution approchée à l'équation non linéaire :

$$f(x) = 0 \qquad \text{(I-1)}$$

La valeur de la variable x qui satisfait cette équation est appelée racine ou solution de l'équation, notée x0. Dans de nombreux cas, la solution exacte ne peut pas être déterminée analytiquement et des méthodes numériques doivent être utilisées. Ainsi, la résolution de cette équation consiste à définir une séquence de valeurs (solutions intermédiaires $x_{0;k}$) qui convergent vers la solution recherchée lorsque k tend vers l'infini. On ne parle pas de solution exacte car elle n'existe souvent pas, mais on cherche une solution qui soit suffisamment proche de la solution exacte avec une certaine précision.

Il existe plusieurs techniques pour résoudre des équations non linéaires, qui diffèrent par leurs principes et leurs taux de convergence. Dans cette section, nous introduisons les méthodes de dichotomie (ou de bissection), de point fixe et de Newton. Nous les présentons dans l'ordre croissant de complexité des algorithmes. Dans le cas de la méthode de dichotomie, la seule information utilisée est le signe de la fonction f aux extrémités des sous-intervalles, tandis que pour les autres algorithmes, les valeurs de la fonction et/ou de ses dérivées sont également prises en compte.

2. La méthode de Dichotomie

La méthode de dichotomie est une méthode numérique pour trouver une approximation d'une racine d'une fonction continue. Cette méthode est également connue sous le nom de méthode de la bissection. Elle consiste à diviser l'intervalle contenant la racine en deux parties égales, puis à déterminer dans quelle partie se trouve la racine. Le processus est répété sur la moitié de l'intervalle contenant la racine jusqu'à ce que l'approximation soit suffisamment précise.

La méthode de dichotomie est basée sur le théorème des valeurs intermédiaires, qui stipule que si une fonction continue change de signe sur un intervalle, alors elle doit avoir au moins une racine sur cet intervalle. En divisant l'intervalle en deux parties égales, la méthode de dichotomie garantit que la fonction change de signe sur au moins une des deux parties. En répétant le processus, l'intervalle contenant la racine est progressivement réduit jusqu'à ce que l'approximation souhaitée soit atteinte.

La méthode de dichotomie est relativement simple à mettre en œuvre, mais elle peut être lente si le nombre d'itérations requis pour atteindre la précision souhaitée est élevé. De plus, elle ne fonctionne que pour les fonctions continues et monotones, ce qui limite son utilité dans certaines situations. Cependant, elle reste une méthode importante en analyse numérique pour la résolution d'équations non linéaires. Elle repose sur les hypothèses suivantes :

- Il existe une solution sur un intervalle $[a, b]$
- La fonction $f(x)$ est monotone (croissante ou décroissante) sur l'intervalle $[a, b]$

Sous ces deux hypothèses, l'inégalité suivante est vérifiée :

$$f(a) * f(b) < 0 \qquad \text{(I-2)}$$

2.1. Principe de la méthode de Dichotomie

Le principe de la méthode de dichotomie est de diviser un intervalle contenant une racine d'une fonction en deux parties égales, puis de déterminer dans quelle partie se trouve la racine. Ensuite, le processus est répété sur la moitié de l'intervalle contenant la racine jusqu'à ce que l'approximation soit suffisamment précise. Voici les étapes de base de la méthode de dichotomie :

1. Choisir un intervalle $[a, b]$ contenant une racine de la fonction *f(x)*.
2. Calculer le point milieu *c= (a+b)/2* de l'intervalle $[a, b]$.
3. Évaluer la fonction *f(c)* au point milieu c.
4. Si *f(c)* est suffisamment proche de zéro (c'est-à-dire si *f(c)* est inférieur à une tolérance donnée), alors c est considéré comme une approximation de la racine et le processus est terminé.
5. Sinon, déterminer dans quelle partie de l'intervalle $[a, c]$ ou $[c, b]$ se trouve la racine en évaluant le signe de *f(a)* et *f(c)* ou *f(c)* et *f(b)*.
6. Répéter le processus sur la partie de l'intervalle contenant la racine jusqu'à ce que l'approximation soit suffisamment précise.

Le processus est répété jusqu'à ce que la précision souhaitée soit atteinte. Le nombre d'itérations nécessaires dépend de la précision souhaitée et de la fonction *f(x)* en question. La méthode de dichotomie est relativement simple à mettre en œuvre et garantit la convergence vers une solution, à condition que la fonction soit continue et monotone sur l'intervalle considéré.

La figure ci-dessous illustré le principe de la méthode de Dichotomie :

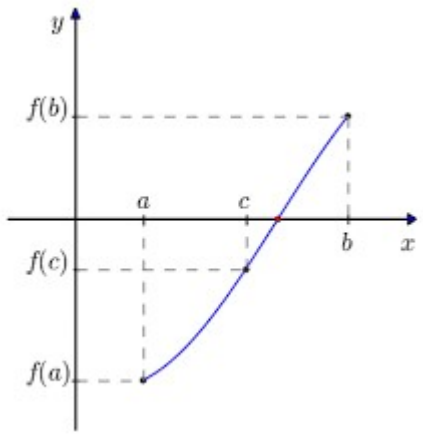

Figure I.1 : Principe de la méthode de Dichotomie

On calcule c par l'expression :

$$c = a + b/2 \qquad \text{(I-3)}$$

- Si $f(c) = 0$, la solution est égale à c, alors arrêter la recherche.
- mettre à jours l'intervalle $[a, b]$ telle que :
  - Si $f(a) * f(c) < 0$, la solution se trouve dans l'intervalle $[a, c]$ alors on prend b=c
  - Si $f(a) * f(s) < 0$ la solution se trouve dans l'intervalle $[c, b]$ alors on prend a=c
- On recommence le calcul de c itérativement jusqu'à ce que : $|b - a| < s$

2.2. La condition d'arrêt

La condition d'arrêt pour la méthode de dichotomie est généralement basée sur la précision souhaitée pour l'approximation de la racine de la fonction. L'idée est de répéter le processus de dichotomie en divisant l'intervalle en deux parties égales jusqu'à ce que la longueur de l'intervalle soit suffisamment petite pour fournir une approximation de la racine avec la précision souhaitée.

La condition d'arrêt typique pour la méthode de dichotomie est que la longueur de l'intervalle $[a, b]$ devienne plus petite qu'une tolérance donnée tol, c'est-à-dire que |b - a| < tol. Autrement dit, la méthode s'arrête lorsque la distance entre les bornes a et b de l'intervalle est inférieure ou égale à la tolérance.

Il est important de choisir une valeur de tol appropriée pour obtenir une approximation précise de la racine de la fonction. Cependant, une tolérance trop petite peut entraîner une augmentation significative du nombre d'itérations nécessaires pour atteindre la précision souhaitée, ce qui peut ralentir l'exécution du programme.

Il est donc recommandé de choisir une valeur de tol qui est suffisamment petite pour obtenir une approximation précise de la racine de la fonction, tout en minimisant le nombre d'itérations nécessaires pour atteindre la précision souhaitée. Le nombre d'itération dépend de la précision s voulue ainsi que l'intervalle initial $[a, b]$ soit :

$$N \geq log(b-a) - log(2s)/log(2) \qquad \text{(I-4)}$$

2.3. Implémenter du programme Matlab

Ce programme utilise une boucle while pour répéter le processus de la méthode de dichotomie jusqu'à ce que la précision souhaitée soit atteinte. La fonction *f(x)* est passée en argument, ainsi que les bornes de l'intervalle $[a, b]$ et la tolérance tol pour l'approximation de la racine.

Le programme vérifie également que la condition initiale *f(a) * f(b)* < 0 est satisfaite, ce qui garantit qu'il existe au moins une racine de la fonction *f(x)* dans l'intervalle $[a, b]$.

À chaque itération de la boucle, le programme calcule le point milieu de l'intervalle $[a, b]$ et évalue la fonction *f(x)* en ce point. Il détermine ensuite dans quelle partie de l'intervalle se trouve la racine et répète le processus sur cette partie jusqu'à ce que l'approximation soit suffisamment précise. Le programme retourne finalement le point milieu de l'intervalle $[a, b]$ comme approximation de la racine.

Un exemple de programme Matlab pour implémenter la méthode de dichotomie.

```
function x = dichotomy(f, a, b, tol)
% f : la fonction dont on cherche une racine
% a : borne inférieure de l'intervalle
% b : borne supérieure de l'intervalle
% tol : la tolérance pour l'approximation de la racine
 % Vérification initiale de la condition initiale : f(a) * f(b) < 0
if f(a) * f(b) >= 0
    error("La condition initiale f(a) * f(b) < 0 n'est pas satisfaite
!");
end
while abs (b-a) > tol
    % Calcul du point milieu de l'intervalle [a, b]
    c = (a + b) / 2;
    % Évaluation de la fonction f(c) au point milieu c
    fc = f(c);
    % Détermination de la nouvelle partie de l'intervalle contenant la
racine
    if fc == 0
    % c est une racine exacte de f
        break;
    elseif f(a) * fc < 0
        b = c;
    else
        a = c;
    end
end
```

3. La méthode de point fixe

La méthode de point fixe est une méthode numérique pour résoudre des équations non linéaires en transformant l'équation en une forme équivalente de la forme x = g(x), où g est une fonction continue et différentiable. La méthode consiste à itérer cette fonction g sur une approximation initiale de la racine jusqu'à ce que la séquence converge vers la racine de l'équation.

La méthode de point fixe peut être considérée comme une généralisation de la méthode de Newton-Raphson, car la méthode de Newton-Raphson peut être vue comme une méthode de point fixe en utilisant la formule de récurrence

$$x = g(x)\ avec\ g(x) = x - f(x) / f'(x) \qquad \text{(I-5)}$$

La méthode de point fixe nécessite une transformation préalable de l'équation pour obtenir la forme $x=g(x)$, qui peut être obtenue par plusieurs méthodes, telles que l'utilisation de la forme normale de Banach ou la transformation de l'équation sous forme intégrale.

La convergence de la méthode de point fixe dépend de la fonction g et de l'approximation initiale de la racine. La méthode converge si la fonction g est contractante sur l'intervalle contenant la racine, c'est-à-dire si elle satisfait la condition $|g'(x)|<1$ pour tout x dans l'intervalle. La convergence peut être accélérée en choisissant une fonction g qui minimise la constante de Lipschitz, qui est définie comme la borne supérieure de $|g'(x)|$ pour tout x dans l'intervalle.

La méthode de point fixe est une méthode simple et efficace pour résoudre des équations non linéaires, en particulier lorsque la méthode de Newton-Raphson est difficile à appliquer ou converge lentement. Cependant, la méthode peut être sensible au choix de la fonction g et de l'approximation initiale de la racine, et elle peut nécessiter un grand nombre d'itérations pour atteindre la précision souhaitée.

3.1. Principe de la méthode de point fixe

Le principe de la méthode de point fixe est de transformer une équation non linéaire $f(x)=0$ en une équation équivalente de la forme $x=g(x)$. Cette transformation est obtenue en isolant x dans l'équation $f(x)=0$ et en définissant $g(x)$ comme la solution de cette équation pour x.

Une fois que l'équation est transformée en $x = g(x)$, la méthode consiste à prendre une approximation initiale de la racine x_0 et à itérer la fonction g sur cette approximation pour

obtenir une séquence x_0, x_1, x_2, ..., x_n, où x_n est l'approximation finale de la racine. Cette séquence est définie par la formule de récurrence suivante :

$$x_{n+1} = g(x\_n) \qquad \text{(I-6)}$$

La méthode converge si la fonction g est contractante sur l'intervalle contenant la racine, c'est-à-dire si elle satisfait la condition $|g'(x)| < 1$ pour tout x dans l'intervalle. Si cette condition est satisfaite, la méthode converge vers la racine unique de l'équation non linéaire, à condition que l'approximation initiale soit suffisamment proche de la racine.

La méthode de point fixe peut être utilisée pour résoudre une grande variété d'équations non linéaires, y compris des équations transcendantes, des systèmes d'équations non linéaires et des équations différentielles non linéaires. La méthode peut être plus simple à mettre en œuvre que d'autres méthodes numériques, telles que la méthode de Newton-Raphson, et peut être utilisée lorsque ces méthodes ne sont pas applicables ou convergent lentement. La figure ci-dessous illustré le principe de la méthode de point fixe :

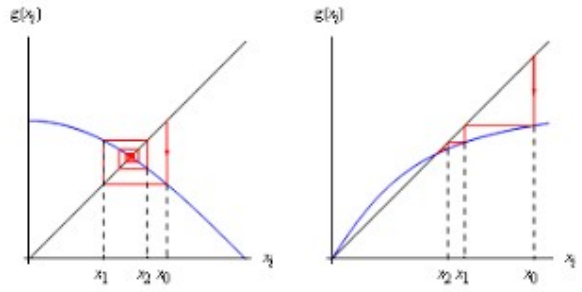

Figure I.2 : Principe de la méthode de point fixe (convergence en spirale et en escalier)

- Le principe de la méthode du point fixe correspond à la recherche du point d'intersection entre les deux fonctions :
- la première fonction est la droite $y = x$
- la deuxième fonction est $y = g(x)$
- On choisit initialement un point x_0, qui sera la première estimée de la solution
- on calcul (x_0), cette dernière valeur sera considérée comme étant la deuxième estimée de la solution soit $x_1 = (x_0)$ a son tour, cette dernière valeur sera injectée dans la fonction (x) et ainsi de suite jusqu'à la satisfaction d'un critère d'arrêt. Le critère d'arrêt soit $| x_1 - x_0| < s$

3.2. Implémenter du programme Matlab

Un exemple de programme Matlab pour implémenter la méthode de point fixe.

```
% Définir la fonction f(x)
f = @(x) cos(x) - x;
% Définir la fonction g(x)
```

```
g = @(x) cos(x);
% Définir la valeur initiale
x0 = 0.5;
% Appliquer la méthode itérative
tol = 1e-6; % Tolérance de convergence
max_iter = 1000; % Nombre maximal d'itérations
x = x0;
for i = 1:max_iter
    x_old = x;
    x = g(x_old);
    if abs(x - x_old) < tol % Vérifier la convergence
        break;
    end
end
% Afficher le résultat
fprintf('La solution est x = %f après %d itérations\n', x, i);
```

Dans cet exemple, on utilise la fonction *cos(x)* comme fonction *g(x)*, ce qui donne *g(x)=cos(x)=x* si et seulement si *cos(x) = x*. La valeur initiale est x_0 = 0.5, et on applique la méthode itérative jusqu'à ce que la différence entre deux itérations successives soit inférieure à la tolérance de convergence tol = 1e-6. Le nombre maximal d'itérations est fixé à max_{iter} = 1000 pour éviter une boucle infinie en cas de non-convergence. Finalement, on affiche la solution trouvée et le nombre d'itérations nécessaires pour atteindre la convergence.

4. La méthode de Newton-Raphson

La méthode de Newton-Raphson est une méthode numérique pour résoudre des équations non linéaires en utilisant des approximations successives de la racine. Elle utilise la formule de récurrence suivante pour itérer une approximation initiale x_0 de la racine :

$$xn + 1 = xn - /frac\{f(xn)\}\{f'(xn)\} \quad \text{(I-7)}$$

Où *f(x)* est la fonction dont la racine est recherchée et *f'(x)* est sa dérivée.

Le principe de la méthode de Newton-Raphson est d'approximer la fonction *f(x)* localement autour de l'approximation initiale de la racine x_0 par une droite tangente à la courbe de *f(x)* à x_0. Cette droite est déterminée par la pente de la dérivée *f'(x)* à x_0 et par le point (x_0, $f(x_0)$). L'intersection de cette droite avec l'axe des *x* donne la prochaine approximation x_1 de la racine. Cette approximation est obtenue en résolvant l'équation de la droite tangente, qui est donnée par :

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0 \quad \text{(I-8)}$$

En général, la méthode de Newton-Raphson converge rapidement vers la racine de *f(x)*, à condition que l'approximation initiale x_0 soit suffisamment proche de la racine et que la dérivée *f'(x)* ne s'annule pas sur l'intervalle contenant la racine. Si la méthode converge, elle

converge quadratiquement, c'est-à-dire que le nombre de chiffres significatifs doubles à chaque itération.

4.1. Principe de la méthode de Newton

Le principe de la méthode de Newton-Raphson est d'approximer la fonction *f(x)* localement autour de l'approximation initiale de la racine x_0 par une droite tangente à la courbe de *f(x)* à x_0. Cette droite est déterminée par la pente de la dérivée *f'(x)* à x_0 et par le point $(x_0, f(x_0))$. L'intersection de cette droite avec l'axe des *x* donne la prochaine approximation x_1 de la racine.

Plus précisément, si x_n est l'approximation actuelle de la racine, la méthode de Newton-Raphson consiste à calculer la prochaine approximation x_{n+1} en utilisant la formule de récurrence suivante :

$$x_{n+1} = x_n - \backslash frac\{f(x_n)\}\{f'(x_n)\} \quad \text{(I-9)}$$

Où *f(x)* est la fonction dont la racine est recherchée et *f'(x)* est sa dérivée.

L'approximation x_{n+1} est la valeur de x pour laquelle la tangente à la courbe de *f(x)* à x_n passe par l'axe des *x*. En d'autres termes, on cherche l'intersection entre la droite tangente à la courbe de *f(x)* en x_n et l'axe des x. Cette intersection est donnée par :

$$f(x_n) + f'(x_n)\,(x_{n+1} - x_n) = 0$$

Il faut noter que la méthode de Newton-Raphson ne garantit pas la convergence vers une solution, et qu'elle peut diverger si la dérivée de la fonction s'annule ou change de signe à proximité de la racine. De plus, il est important de choisir une approximation initiale suffisamment proche de la racine pour assurer la convergence rapide de la méthode.

Le principe de cette méthode est illustré sur la figure I.3 où : On choisit une première estimée x_0, la second estimée est x_1 déterminée par l'intersection de la ligne tangente de la fonction $f(x)$ au point $(x_1; f(x_1))$ et la droite $y = 0$. La troisième estimée x_2 est déterminée par l'intersection de la ligne tangente de la fonction $f(x)$ au point $(x_2; f(x_2))$ et la droite $y = 0$, et ainsi de suite.

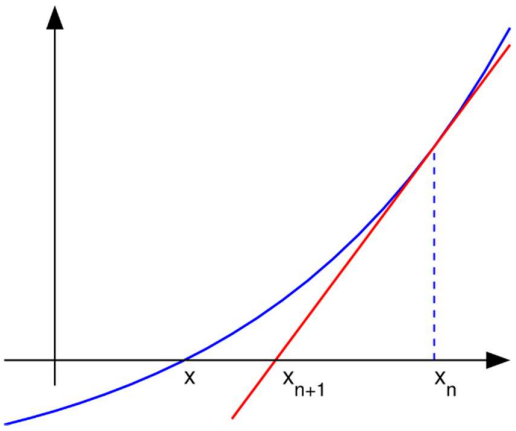

Figure I.3 : Principe de la méthode de Newton

L'algorithme de Newton-Raphson est :

$$x = x - f(x_i)f'(x_i) \qquad \text{(I-10)}$$

Le critère d'arrêt soit $|x_{i+1} - x_i| < s$

4.2. Implémenter du programme Matlab

Voici le code complet pour implémenter la méthode de Newton-Raphson sur MATLAB pour trouver les zéros d'une fonction donnée :

```
function x1 = newton_raphson(f, df, x0, tol, max_iter)
% f : la fonction à trouver les zéros
% df : la dérivée de la fonction f
% x0 : point de départ pour l'itération
% tol : tolérance de convergence
% max_iter : nombre maximum d'itérations autorisées
 for i = 1:max_iter
    f_x0 = f(x0);
    df_x0 = df(x0);
    x1 = x0 - f_x0/df_x0;
    if abs(x1 - x0) < tol
        break;
    end
    x0 = x1;
end

if i == max_iter
    warning('La méthode de Newton-Raphson n''a pas convergé');
end
```

Pour utiliser cette fonction, il faut appeler la fonction Newton-Raphson en fournissant les arguments nécessaires :

Dans cet exemple, nous avons choisi la fonction $f(x) = x^2 - 2$ comme exemple. Vous pouvez remplacer cette fonction par n'importe quelle fonction pour laquelle vous voulez trouver les zéros.

```
% Définir la fonction et sa dérivée
f = @(x) x^2 - 2;
```

```
df = @(x) 2*x;
% Définir le point de départ, la tolérance et le nombre maximum
d'itérations
x0 = 1;
tol = 1e-6;
max_iter = 100;
% Appeler la fonction newton_raphson
x1 = newton_raphson(f, df, x0, tol, max_iter);
% Afficher le résultat
disp(['La solution est x = ', num2str(x1)])
```

5. Mise en œuvre sous Matlab

Dans ce test, il est demandé de trouver la racine de $f(x) = 2.x^2 - x - 1$ sur l'intervalle [0.5, 1.5] en utilisant la méthode de dichotomie, la méthode de point fixe et la méthode de Newton jusqu'à la convergence avec une précision de 10^{-3} (TOL=10^{-3}).

```
%=======Résolution Numérique des Equations non linéaires F(X)=0======
clear all;
close all;
clc;
disp(['*** Résolution des Equations non linéaires F(X)=0 ***'])
disp(['*** S=1. La Méthode de Dichotomie ***'])
disp(['*** S=2. La Méthode du point fixe ***'])
disp(['*** S=3. La Méthode de NEWTON ***'])
disp(['*** S=4. Pour Quitter ***'])
disp(['*** Tapez S entre <1-4> ***'])
s=input('S =');
switch s
case 1
disp(['*** Début de la Méthode de Dichotomie***'])
disp(['l`intervalle initial de la recherche [a b]'])
a=input('a =');
b=input('b =');
disp(['Donnez Tol; " Tol petite valeur!"'])
tol=input('Tol =');
x=a:(b-a)/100:b;
y=feval('f',x);
figure(1);
plot(x,y,'g')
title('la racine de f(x)=0 par dichotomie')
grid
xlabel('    valeur de x')
ylabel('    valeur de f(x)')
hold
iter= 0;
while abs(b-a)>tol
iter=iter+1;
c=0.5*(a+b)
plot(c,feval('f',c),'k<')
plot(a,feval('f',a),'k>')
yc=feval('f',c);
yd=feval('f',a);
disp(['x1 =',num2str(yc)])
disp(['y1 =',num2str(yd)])
disp([' '])
pause
if yc* yd <=0
b = c;
plot(b,feval('f',b),'b*')
else;
```

```
a = c;
plot(a,feval('f',a),'r*')
end
end
disp('  ')
disp(['-Le nombre d`itération est N=',num2str(iter)])
disp('  ')
disp(['-La valeur de la racine est X*=',num2str(c)])
disp('  ')
disp(['-La valeur de f(X*)=',num2str(feval('f',c))])
case 2
disp(['*** Début de la Méthode du point fixe ***'])
disp(['l`intervalle initial de la recherche [a b]'])
a=input('a =');
b=input('b =');
disp([ 'Donnez une valeur initiale X0 dans [a b]'])
x0=input('x0= ')
disp(['Donnez Tol " Tol petite valeur!'])
tol=input('tol= ')
x=a:(b-a)/100:b;
y=feval('g',x);
figure(2);
plot(x,y,'g')
hold on plot(x,x,'--r')
title('la racine de f(x)=0 par le point fixe')
grid
xlabel('    x')
ylabel('    g(x)')
hold
kp = 0;
x1=feval('g',x0);
while abs(x1 - x0) >= tol x0 = x1;
kp = kp + 1;
x1=feval('g',x0);
hold on
plot(abs(x0),feval('g',x0),'r*')
pause plot(abs(x1),feval('g',x1),'b*')
end
disp('  ')
disp(['-Le nombre d`itération est N=',num2str(kp)])
disp('  ')
disp(['-La valeur de la racine est X*=',num2str(x1)])
disp('  ')
disp(['-La valeur de g(X*)=',num2str(feval('g',x1))])
case 3
disp(['*** Début de la Méthode de NEWTON ***'])
disp(['l`intervalle initial de la recherche [a b]'])
a=input('a =');
b=input('b =');
disp([ 'Donnez une valeur initiale X0 dans [a b]'])
x0=input('x0= ')
disp(['Donnez Tol " Tol petite valeur!'])
tol=input('tol= ')
x=a:(b-a)/100:b;
y=feval('f',x); figure(3); plot(x,y,'g')
title('la racine de f(x)=0 par Newton')
grid
xlabel('    x')
ylabel('    f(x)')
hold
kp = 0;
x1 = x0 - feval('f',x0)/feval('df',x0)
while abs(x1 - x0) >= tol
```

```
x0 = x1
kp = kp+1
x1=x0 - feval('f',x0) / feval('df',x0);
hold on
plot(abs(x0),feval('f',x0),'r*')
pause plot(abs(x1),feval('f',x1),'b*') end
disp('      ')
disp(['-Le nombre d`itération est N=',num2str(kp)])
disp('      ')
disp(['-La valeur de la racine est X*=',num2str(x1)])
disp('      ')
disp(['-La valeur de f(X*)=',num2str(feval('f',x1))])
case 4
quit
otherwise
disp('Erreur! S entre <1-4> ') ;
end
```

6. TP N°1 : Résolution numérique des équations non linéaires

6.1. But du TP

Dans ce TP, nous allons implémenter les algorithmes des méthodes de résolution des équations non linéaires étudiées : la méthode de Dichotomie, la méthode de Point fixe et la méthode de Newton-Raphson.

6.2. Énoncé du TP

Soit l'équation non linéaire : $f(x) = x_2 - 2 = 0$

1) Déclarer la fonction $f(x)$ avec $x = -10 : 0.001 : 10$
2) Tracer le graphe $y = f(x)$ sur un intervalle tel qu'il vous permet de localiser la solution de l'équation.
3) Il est à noter que, les solutions exactes de cette équation sont $x1 = \sqrt{2}$ et $x2 = -\sqrt{2}$ et on veut trouver la première racine $x1$ de cette équation en utilisant :

6.2.1. La méthode de dichotomie

- Quel est le nombre d'opération nécessaire pour atteindre une précision de s = 0.01 si on prend l'intervalle [0, 3] ?
- Ecrire un script qui implémente la méthode de Dichotomie suivant les étapes :
  - Déclarer a, b et s
  - Initialiser un compteur d'itération
  - Ecrire l'algorithme en incrémentant le compteur i à chaque passage de boucle
  - Arrêter la boucle quand la largeur de l'intervalle devient inférieure ou égale à s
  - Afficher la solution calculée ainsi que le nombre d'itérations.
- Faire dérouler le programme et remplir la table ci-dessous :

i	*a*	*b*	*c*	*f(a)*	*f (b)*	*f (c)*	*s*

6.2.2. La méthode de point fixe

- Quelles sont les formes possibles de la fonction g(x) ?
- Quelle est la fonction qui vérifie le théorème précédent, sur l'intervalle [0, 3] ?
- Ecrire un programme Matlab qui donne la solution de cette équation. Prendre s = 0.01 et $x0 = 0$ puis $x0 = 3$. Conclure !

6.2.3. La méthode de Newton-Raphson

- Ecrire un programme Matlab qui donne la solution de cette équation. Prendre s = 0.01 et $x0 = 2$ puis $x0 = 3$. Conclure !

4) Comparer les résultats des différentes méthodes implémentées.

Interpolation et approximation polynômiale de fonctions

1. Introduction :

L'interpolation et l'approximation polynômiale sont des techniques courantes en analyse numérique pour trouver une fonction polynomiale qui passe par un ensemble de points donnés.

L'interpolation polynomiale consiste à trouver un polynôme de degré n-1 (n étant le nombre de points donnés) qui passe exactement par ces points. En d'autres termes, le polynôme doit avoir la même valeur que la fonction à interpoler aux points donnés. Cette technique est souvent utilisée pour l'approximation de fonctions simples, mais peut poser des problèmes si les points sont très éloignés les uns des autres ou si la fonction à interpoler est très complexe.

L'approximation polynomiale, quant à elle, consiste à trouver le polynôme de degré n-1 qui minimise la distance entre la fonction à approximer et le polynôme. Cette technique est souvent utilisée lorsque l'interpolation exacte est impossible ou peu pratique, ou lorsque l'on souhaite éviter les oscillations entre les points. Les méthodes d'approximation polynomiale les plus courantes sont la régression linéaire, la régression polynomiale, la méthode des moindres carrés et la méthode de Lagrange.

En général, l'approximation polynomiale est plus flexible que l'interpolation polynomiale, mais elle peut être plus complexe à mettre en œuvre et à interpréter. Il est important de noter que l'interpolation et l'approximation polynômiale ne sont que des techniques approximatives, et que l'erreur d'approximation peut être importante en dehors de la plage de valeurs où les points sont donnés. Nous instruisons dans ce chapitre aux formes particulières de **Lagrange** et **Newton**.

2. La méthode d'interpolation de Lagrange

La méthode d'interpolation de Lagrange est une technique courante pour trouver un polynôme qui passe exactement par un ensemble de points donnés. Elle est nommée d'après le mathématicien français Joseph Louis Lagrange, qui l'a développée au 18ème siècle.

La méthode de Lagrange consiste à construire un polynôme d'interpolation en combinant une série de polynômes de base. Chaque polynôme de base est construit de sorte qu'il ait une valeur de 1 au point d'interpolation correspondant et une valeur de 0 à tous les autres points. Le polynôme d'interpolation final est alors une somme pondérée de ces polynômes de base, où les poids sont donnés par les valeurs de la fonction à interpoler aux points correspondants.

Pour un ensemble de n points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, le polynôme d'interpolation de Lagrange est donné par :

$$L(x) = y_1 * L_1(x) + y_2 * L_2(x) + \ldots + y_n * L_n(x) \quad \text{(II-1)}$$

Où $L_i(x)$ est le i-ème polynôme de base de Lagrange, qui est donné par :

$$L_i(x) = \prod_{j \neq i} (x - x_j) / (x_i - x_j) \quad \text{(II-2)}$$

En d'autres termes, le polynôme de base $L_i(x)$ a une valeur de 1 au point d'interpolation xi et une valeur de 0 à tous les autres points. Le polynôme d'interpolation final $L(x)$ passe exactement par les n points donnés, car chaque terme dans la somme contribue uniquement à la valeur de $L(x)$ au point correspondant.

La méthode de Lagrange est simple à mettre en œuvre et fournit une interpolation exacte pour un ensemble de points donnés. Cependant, elle peut être sensible aux erreurs d'arrondi lorsque le nombre de points est grand ou lorsque les points sont très proches les uns des autres. De plus, la méthode de Lagrange peut être inefficace pour l'approximation de fonctions complexes, car elle peut conduire à des polynômes de degré élevé.

3. La méthode d'interpolation de Newton

La méthode d'interpolation de Newton est une autre technique courante pour trouver un polynôme qui passe exactement par un ensemble de points donnés. Elle est nommée d'après le mathématicien anglais Isaac Newton, qui a développé la méthode au 17ème siècle.

La méthode de Newton consiste à construire un polynôme d'interpolation en utilisant une série de différences divisées. Les différences divisées sont des coefficients qui sont calculés à partir des valeurs de la fonction à interpoler aux points donnés. Le polynôme d'interpolation final est alors une somme pondérée de ces différences divisées, où les poids sont donnés par des termes de produit de facteurs de différence. Le polynôme résultant est un polynôme de degré n-1 pour n points donnés.

Pour un ensemble de n points $(x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1})$, le polynôme d'interpolation de Newton est donné par :

$$P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \ldots + (x - x_0)(x - x_1)\ldots(x - x_n - 2)f[x_0, x_1, \ldots, x_{n-1}] \quad \text{(II-3)}$$

où $f[x_i]$ est la valeur de la fonction à interpoler au point x_i, et $f[x_i,x_j,\ldots,x_k]$ est la différence divisée de la fonction à interpoler calculée aux points x_i, x_j, ..., x_k. La différence divisée de la fonction à interpoler est définie de manière récursive comme :

$$f[x_i, x_j, \ldots, x_k] = (f[x_j, \ldots, x_k] - f[x_i, \ldots, x_{k-1}]) / (x_i - x_k) \quad \text{(II-4)}$$

En d'autres termes, chaque différence divisée est calculée comme la différence entre les différences divisées de l'étape précédente, divisée par la distance entre les points correspondants.

La méthode de Newton est également simple à mettre en œuvre et fournit une interpolation exacte pour un ensemble de points donnés. Elle peut être plus stable que la méthode de Lagrange pour un grand nombre de points, car elle évite de calculer plusieurs fois les mêmes produits de facteurs de différence. Cependant, elle peut être plus complexe que la méthode de Lagrange à mettre en œuvre et à interpréter, car elle utilise des différences divisées et des termes de produit de facteurs de différence.

4. La méthode d'interpolation de Tchebychev

La méthode d'interpolation de Tchebychev est une technique d'interpolation numérique qui vise à minimiser l'erreur d'interpolation dans un intervalle donné. Elle est basée sur les polynômes de Tchebychev, qui sont des polynômes orthogonaux définis sur l'intervalle [-1,1]. Elle est nommée d'après le mathématicien russe Pafnouti Tchebychev, qui a développé la méthode au 19ème siècle. La méthode de Tchebychev consiste à choisir les points d'interpolation de sorte qu'ils soient les n racines du n^{ième} polynôme de Tchebychev. Ces points sont donnés par :

$$x_i = cos((2i - 1)\pi / (2n)),\ pour\ i = 1,2,\ldots,n. \quad \text{(II-5)}$$

Ces points sont choisis de manière à minimiser la somme des carrés des erreurs d'interpolation sur l'intervalle donné, pour une fonction continue *f(x)*. En effet, en choisissant ces points d'interpolation, on minimise la norme de l'erreur d'interpolation dans l'espace des fonctions continues. Le polynôme d'interpolation final est alors donné par :

$$P_n(x) = a_0 T_0(x) + a_1 T_1(x) + \ldots + a_{n-1} T_{n-1}(x) \quad \text{(II-6)}$$

Où $T_0(x)$, $T_1(x)$, ..., $T_{n-1}(x)$ sont les polynômes de Tchebychev de degré 0, 1, ..., n-1, respectivement. Les coefficients a_0, a_1,..., a_{n-1} sont déterminés en résolvant un système d'équations linéaires, qui est obtenu en imposant la condition que le polynôme d'interpolation passe par les valeurs de la fonction *f(x)* aux points d'interpolation.

La méthode de Tchebychev est efficace pour approximer des fonctions continues sur des intervalles donnés. Elle peut être plus précise que les méthodes d'interpolation de Lagrange ou de Newton, en particulier pour les fonctions oscillantes. Cependant, elle peut être plus complexe à mettre en œuvre, car elle nécessite de calculer les polynômes de Tchebychev et

de résoudre un système d'équations linéaires pour déterminer les coefficients du polynôme d'interpolation.

5. Mise en œuvre sous Matlab

1) Méthode d'interpolation de Lagrange

Dans ce test, il est demandé de trouver le polynôme d'interpolation qui interpole les points d'appui *(x_0, y_0)= (0, 1), (x_1, y1)= (1, 2) et (x_2, y2)= (2, 5)*

2) Méthode d'interpolation de Tchebychev

Soit la fonction : $f(x) = 1/(1 + x2)$

Et les nœuds d'interpolation suivants :

$x_0 = -5$, $x_1 = -4$, $x_2 = -3$, $x_3 = -2$, $x_4 = -1$, $x_5 = 0$, $x_6 = 1$, $x_7 = 2$, $x_8 = 3$, $x_9 = 4$, $x_{10} = 5$.

Tracer, le polynôme de Tchebychev, les points d'interpolation et la fonction *f(x)*.

```
%==============Interpolation et Approximation de Fonctions==============
clear all;
close all;
clc;
disp(['*** Interpolation et Approximation de Fonctions ***'])
disp(['*** S=1. La Méthode d''interpolation de Lagrange ***'])
disp(['*** S=2. La Méthode d''interpolation de Tchebychev***'])
 disp(['*** S=3. Pour Quitter ***'])
disp(['*** Tapez S entre <1-3> ***'])
s=input('S =');
switch s
    case 1
disp(['*** Début de la Méthode d''interpolation de Lagrange ***'])
 x(1)=0;
x(2)=1;
x(3)=2;
n=3;
%POLYNOME DE DEGRE 2
y(1)=1;
y(2)=2;
y(3)=5;
interv=1000;
dx=(x(3)-x(1))/interv;
xvar=x(1):dx:x(3);
polyn=0;
col={'+k','+r','+m'};
for i=1:n lag=1;
    for j = 1:n
        if(i~=j)
            lag=(xvar-x(j))./(x(i)-x(j)).*lag;
        end
    end
    figure(1);
    plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
    hold on;
    polyn=polyn+lag.*y(i);
end
hold on
plot(xvar,polyn,'LineWidth',1);
```

```
hold on; xlabel('x');
ylabel('y')
axis([-0.5 2.5 -0.5 5.5])
title('Interpolation de Lagrange')
p=2;
coeff=polyfit(xvar,polyn,p)
case 2
disp(['*** Début de la Méthode d''interpolation de Tchebychev ***'])
 format long
x(1)=-5;
x(2)=-4;
x(3)=-3;
n=11;
%POLYNOME DE DEGRE n-1
 x(4)=-2;
x(5)=-1;
x(6)=0;x(7)=1;
x(8)=2;x(9)=3;
x(10)=4;
x(11)=5;
y=1./(1+x.^2);
interv = 1000;
dx = (x(11)-x(1))/interv;
xvar=x(1):dx:x(11);
polyn=0;
x=(x(11)+x(1))/2+((x(11)-x(1))/2)*cos((2*(n-(1:n))+1)*pi/(2*n));
xChe=x;
%LES NOEUDS xi ANNULANT LES POLYNOMES DE TCHEBYCHEV :
%CECI AFIN DE MINIMISER L'ECART ENTRE Pn(x) ET LA FONCTION f(x)
for i = 1:n
    lag = 1;
    for j = 1:n
        if (i~=j)
            lag=(xvar-xChe(j))/(xChe(i)-xChe(j))*lag;
        end
    end
    figure(1);
    plot(xChe(i),y(i),'+k','MarkerSize',12,'LineWidth',1);
    hold on;
    polyn = polyn+lag.*y(i);
end
hold on
plot(xvar, polyn,'b','LineWidth',1) ;
hold on; xlabel('x');
ylabel('y')
title('Interpolation de Tchebychev')
p=10;
coeff=polyfit(xvar,polyn,p);
evalp=polyval(coeff,xvar);
hold on;
plot(xvar,1./(1+xvar.^2),'r')
case 3
    otherwise
disp('Erreur! S entre <1-3> ');
end
```

6. TP N°2 : Interpolation et approximation polynômiale

Durant ce TP, nous allons implémenter sous Matlab des algorithmes d'interpolation étudiés pendant le cours de méthodes numériques : la méthode de Lagrange et la méthode de Newton

Pendant les travaux pratiques de mesures on a effectué la caractérisation d'une thermistance.

Les résultats expérimentaux ont donnés la caractéristique reportée sur le tableau :

Température (25^0C)	15	20	30	40	50	60
Résistance (Ω)	15.11	14.04	9.28	6.44	4.44	2.9

a) Tracer la courbe résistance en fonction de température ?

b) Interpolation de Lagrange

- Déterminer d'abord ce polynôme de façon analytique.
- Ecrire un algorithme sous MATLAB permettant l'implémentation de la méthode de Lagrange.
- Déterminer la valeur estimée de la résistance à la température $T = 35^0C$

c) Interpolation par le polynôme de Newton

- Déterminer le degré du polynôme de Newton qui passe par tous ces points ?
- Donner l'expression du polynôme de Newton correspondant ?
- Réaliser un algorithme sous MATLAB permettant l'implémentation de la méthode de Newton ?
- Quelle est la valeur estimée de la résistance à la température $T = 35^0C$?

Intégration numérique de fonctions

1. Introduction

L'intégration numérique de fonctions est une méthode permettant d'approximer la valeur d'une intégrale à l'aide d'une série de calculs mathématiques. Cette méthode est souvent utilisée lorsque l'intégrale ne peut pas être calculée analytiquement, ou lorsque l'expression de la fonction intégrante est trop complexe pour être intégrée de manière analytique.

Il existe plusieurs méthodes d'intégration numérique, chacune ayant ses propres avantages et inconvénients. Voici quelques-unes des méthodes les plus courantes :

- La méthode des rectangles : cette méthode consiste à diviser l'intervalle d'intégration en plusieurs sous-intervalles de largeur égale, puis à approximer l'intégrale en utilisant la hauteur du rectangle formé par chaque sous-intervalle. Cette méthode est simple à mettre en œuvre, mais elle peut conduire à une approximation grossière de l'intégrale si la fonction est très irrégulière.
- La méthode du point milieu : cette méthode consiste à approximer l'intégrale en utilisant la valeur de la fonction au point milieu de chaque sous-intervalle. Cette méthode est plus précise que la méthode des rectangles, mais elle reste sensible à la régularité de la fonction.
- La méthode de Simpson : cette méthode consiste à approximer l'intégrale en utilisant une combinaison linéaire de la valeur de la fonction en trois points différents dans chaque sous-intervalle. Cette méthode est plus précise que les méthodes précédentes, mais elle est également plus complexe à mettre en œuvre.

Il est important de noter que l'erreur d'approximation dépend de la méthode d'intégration numérique choisie et de la taille des sous-intervalles utilisés. Pour minimiser l'erreur, il est donc important de choisir la méthode la plus appropriée pour la fonction à intégrer et de choisir la taille des sous-intervalles en fonction de la régularité de la fonction.

En pratique, on est souvent amener à calculer l'intégrale définie d'une fonction f continue sur $[a, b]$ dans R, définie par $I(f)=\int_a f(x)dx$ peut se révéler très laborieux, ou tout simplement impossible à atteindre. Par conséquent, on fait appel à des méthodes numériques, afin de calculer une approximation de $I(f)$. Dans ces méthodes numériques, la fonction f, est remplacée par une somme finie. Dans ce chapitre, nous allons étudier et implémenter, sous Matlab, quelques méthodes usuelles (point milieu, trapèze et Simpson) dédiées à l'intégration numérique.

Supposons qu'on veuille intégrer une fonction *f(x)* sur un intervalle [a, b]. Puisque l'intégrale est la limite de sommes de Riemann, l'idée la plus évidente pour approximer numériquement le résultat qu'on cherche est de calculer des sommes de Riemann en espérant qu'elles s'approcheront de manière désirée de la réponse exacte à condition de prendre les sous-intervalles suffisamment petits.

Découpons donc l'intervalle [a, b] en n sous-intervalles $[x_{i-1}, x_i]$ de largeur :

$$\Delta x = \frac{b-a}{n} \qquad \text{(III-1)}$$

1.1. La méthode des Rectangles

La méthode des rectangles est une méthode d'intégration numérique qui consiste à diviser l'intervalle d'intégration en plusieurs sous-intervalles de largeur égale, puis à approximer l'intégrale en utilisant la hauteur du rectangle formé par chaque sous-intervalle.

Plus précisément, pour une fonction f(x) à intégrer sur l'intervalle [a,b], la méthode des rectangles consiste à diviser cet intervalle en n sous-intervalles de largeur égale h=(b-a)/n, et à approximer l'intégrale par la somme des aires des rectangles de hauteur f(xi) pour i variant de 0 à n-1, où xi est le point d'abscisse a+i*h dans l'intervalle considéré.

Ainsi, l'approximation de l'intégrale I de f(x) sur l'intervalle [a,b] par la méthode des rectangles est donnée par :

$$I \approx h(f(x_0) + f(x_1) + \ldots + f(x_{n-1})) \qquad \text{(III-2)}$$

Où $x_i = a + i*h$ pour i variant de 0 à n-1.

Il existe plusieurs variantes de la méthode des rectangles, selon la façon dont on choisit la hauteur du rectangle. Par exemple, on peut choisir :

La méthode des rectangles à gauche, qui consiste à utiliser la valeur de la fonction $f(x_i)$ à gauche de chaque sous-intervalle pour déterminer la hauteur du rectangle, la méthode des rectangles à droite, qui consiste à utiliser la valeur de la fonction $f(x_i)$ à droite de chaque sous-intervalle pour déterminer la hauteur du rectangle, la méthode des rectangles du point milieu, qui consiste à utiliser la valeur de la fonction $f((x_i+x_{i+1})/2)$ au point milieu de chaque sous-intervalle pour déterminer la hauteur du rectangle.

Il est important de noter que la méthode des rectangles peut conduire à une approximation grossière de l'intégrale si la fonction est très irrégulière, car elle n'utilise qu'une seule valeur de la fonction par sous-intervalle pour déterminer la hauteur du rectangle.

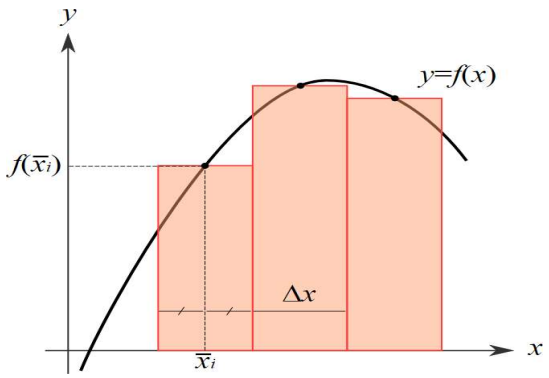

Figure III.1 : Principe de la méthode des Rectangles

1.2. La méthode des Trapèzes

La méthode des trapèzes est une méthode d'intégration numérique qui consiste à approximer l'intégrale d'une fonction *f(x)* sur un intervalle [a, b] en remplaçant la courbe de la fonction par une série de trapèzes de base $[x_i,x_{i+1}]$ et de hauteur $f(x_i)$ et $f(x_{i+1})$, où xi est un point de subdivision de l'intervalle [a, b].

La méthode des trapèzes est plus précise que la méthode des rectangles, car elle utilise deux points de la fonction par sous-intervalle pour déterminer la hauteur du trapèze au lieu d'un seul point pour la méthode des rectangles.

Plus précisément, pour une fonction *f(x)* à intégrer sur l'intervalle [a, b], la méthode des trapèzes consiste à diviser cet intervalle en n sous-intervalles de largeur égale $h=(b-a)/n$, et à approximer l'intégrale par la somme des aires des trapèzes formés par les segments reliant les points $(x_i,f(x_i))$ et $(x_{i+1},f(x_{i+1}))$ pour i variant de 0 à n-1, où xi est le point d'abscisse a+i*h dans l'intervalle considéré.

Ainsi, l'approximation de l'intégrale *I* de *f(x)* sur l'intervalle [a, b] par la méthode des trapèzes est donnée par :

$$I \approx (h/2) * (f(x0) + 2f(x1) + 2f(x2) + \ldots + 2f(xn-2) + f(x_n)) \qquad \text{(III-3)}$$

Où $x_i=a+i*h$ pour i variant de 0 à n.

Il est important de noter que la méthode des trapèzes reste sensible à la régularité de la fonction, car elle approxime la courbe de la fonction par des segments droits. Cependant, elle est plus précise que la méthode des rectangles pour les fonctions qui varient rapidement.

Le principe de l'approximation d'une intégrale par la méthode des trapèzes est illustré dans la figure(III.2). La formule générale de l'intégrale par la méthode des trapèzes s'écrit :

$$I = h \sum n \, (f(x_i) + f(x_i + 1)) \quad i=1 \qquad \text{(III-4)}$$

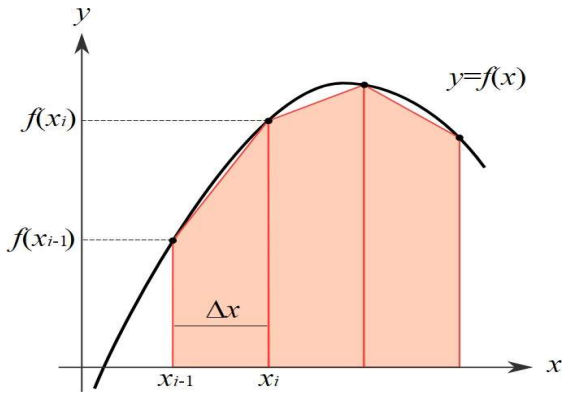

Figure III.2 : Principe de la méthode des trapèzes

1.3. La méthode de Simpson

La méthode de Simpson est une méthode d'intégration numérique qui permet d'obtenir une approximation plus précise de l'intégrale d'une fonction *f(x)* sur un intervalle [a, b] que les méthodes des rectangles et des trapèzes. Cette méthode consiste à remplacer la courbe de la fonction par des arcs de paraboles, ce qui permet de mieux approcher la forme réelle de la courbe.

Plus précisément, pour une fonction *f(x)* à intégrer sur l'intervalle [a, b], la méthode de Simpson consiste à diviser cet intervalle en n sous-intervalles de largeur égale $h=(b-a)/n$, et à approximer l'intégrale par la somme des aires des paraboles qui passent par les points $(x_{i-1},f(x_{i-1}))$, $(x_i, f(x_i))$, et $(x_i+1,f(x_{i+1}))$ pour i variant de 1 à n-1, où x_i est le point d'abscisse a+i*h dans l'intervalle considéré.

Ainsi, l'approximation de l'intégrale I de *f(x)* sur l'intervalle [a, b] par la méthode de Simpson est donnée par :

$$I \approx (h/3) * (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_n) + \ldots + 2f(x_n - 2) + 4f(x_n - 1) + f(x_n)) \quad \text{(III-5)}$$

Où $x_i = a + i*h$ pour i variant de 0 à n.

Il est important de noter que la méthode de Simpson est plus précise que les méthodes des rectangles et des trapèzes, car elle utilise des arcs de paraboles pour approximer la courbe de la fonction. Cependant, cette méthode nécessite un nombre pair de sous-intervalles pour être appliquée, ce qui peut être un inconvénient dans certaines situations. Le principe de l'approximation d'une intégrale par la méthode de Simpson est illustré dans la figure ci-dessous :

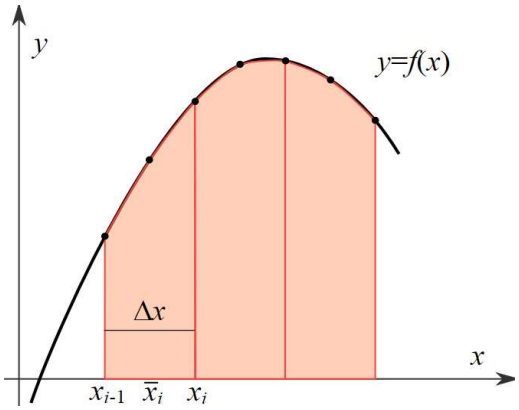

Figure III.3 : Principe de la méthode de Simpson

2. Mise en œuvre sous Matlab

Dans ce test, il est demandé de calculer par la méthode des rectangles, des trapèzes puis par celle de Simpson l'intégral

```
%=============== Intégration numérique de Fonctions ===============
clear all;
close all;
clc;
a=input('a =');b=input('b =');n=input('n =');h = (b - a)/n
disp(['h=',num2str(h)])
disp(['*** Intégration numérique de f(x) = (1/3)*x^2 ***'])
disp(['***  S=1. La Méthode des Rectangles ***'])
disp(['*** S=2. La Méthode des Trapèzes ***'])
disp(['*** S=3. La Méthode de Simpson ***'])
disp(['***  S=4. Pour Quitter ***'])
disp(['*** Tapez S entre <1-4> ***'])
s=input('S =');
switch scase 1
disp(['*** Début de la méthode des Rectangles***'])
R1 = 0;
R2 = 0;
f= inline ('(1/3)*x^2');
for i=1:n
  R1 = R1 + h * f(a + (i - 1) * h);
  end
for j =1:n
  R2 = R2 + h * f(a + j * h);
  end
if R2 > R1
      Rmin = R1;
      Rmax = R2;
for j =1:n
  R2 = R2 + h * f(a + j * h);
  end
if R2 > R1
      Rmin = R1;
      Rmax = R2;
end
```

3. TP N° 3 : Intégration numérique de fonctions

3.1. But du TP

Le but de ce TP est le calcul numérique d'une intégrale définie en utilisant les méthodesdu point milieu, des trapèzes et de Simpson.

3.2. Énoncé du TP

On se propose de calculer l'intégrale définie :

$$I = \int_0^3 ln(2x + 1)\, dx$$

- ✓ Ecrire un programme qui calcule cette intégrale en utilisant les méthodes du point milieu, du trapèze et de Simpson avec n=10.
- ✓ Calculer la valeur exacte de l'intégrale et comparer les résultats de chaque méthode, conclure.
- ✓ Refaire l'exécution avec n=150
- ✓ Étudier l'influence du nombre de sous-intervalles (n) sur l'erreur d'intégration
- ✓ Appliquez les mêmes étapes pour l'intégrale :

$$I = \int_0^{2\pi} \cos(x)\, dx$$

Résolution numérique des équations différentielles

1. Introduction

La résolution numérique des équations différentielles consiste à trouver une solution approchée d'une équation différentielle à l'aide de méthodes numériques. Ces méthodes sont nécessaires car il n'est souvent pas possible de trouver une solution exacte pour de nombreuses équations différentielles complexes.

Les méthodes numériques les plus couramment utilisées pour la résolution numérique des équations différentielles sont les méthodes d'Euler, de Runge-Kutta et de la méthode des différences finies.

La méthode d'Euler est une méthode simple qui consiste à avancer la solution d'un pas de temps à la fois en utilisant la pente locale de la solution à chaque étape. Cette méthode est relativement peu précise et peut produire des solutions instables ou divergentes pour des problèmes complexes.

Les méthodes de Runge-Kutta sont des méthodes plus avancées qui utilisent des approximations de la solution à différents moments dans le pas de temps pour améliorer la précision de la solution. Les méthodes de Runge-Kutta d'ordre élevé, telles que la méthode de Runge-Kutta d'ordre 4, sont couramment utilisées car elles offrent une bonne précision tout en restant relativement simples à implémenter.

La méthode des différences finies est une méthode qui consiste à discrétiser l'équation différentielle en approximant les dérivées par des différences finies. Cette méthode peut être utilisée pour résoudre des équations différentielles à plusieurs dimensions et peut être plus précise que les méthodes d'Euler ou de Runge-Kutta dans certains cas.

Enfin, il est important de noter que la résolution numérique des équations différentielles est une discipline complexe qui nécessite une expertise et une expérience pour être appliquée correctement. Les méthodes numériques doivent être choisies et paramétrées avec soin pour garantir la précision et la stabilité de la solution.

Le but de ce chapitre est de calculer la solution sur l’intervalle du problème de Cauchy

2. La méthode d’Euler

La méthode d'Euler est une méthode numérique utilisée pour résoudre des équations différentielles ordinaires. Elle consiste à approximer la solution en avançant d'un pas de temps à la fois, en utilisant la pente de la tangente à la courbe de la solution à chaque étape.

Plus précisément, la méthode d'Euler consiste à diviser l'intervalle de temps sur lequel la

solution est définie en un certain nombre de pas de temps égaux, et à calculer une approximation de la solution à chaque pas de temps. Cette approximation est calculée en utilisant la valeur de la solution et de sa dérivée (ou vitesse de variation) à l'instant initial, et en utilisant la formule suivante :

$$y(t+h) = y(t) + h * f(t, y(t)) \quad \text{(IV-1)}$$

Où *y(t)* est la valeur de la solution à l'instant t, h est la taille du pas de temps, et *f(t, y(t))* est la dérivée de la solution à l'instant t.

La méthode d'Euler est simple à implémenter et à comprendre, mais elle peut être imprécise pour certaines équations différentielles et nécessite souvent des pas de temps très petits pour obtenir une bonne précision. Des méthodes plus sophistiquées, comme la méthode de Runge-Kutta, sont souvent préférées pour des problèmes plus complexes.

3. La méthode de Runge-Kutta, d'ordre 4

La méthode de Runge-Kutta d'ordre 4 est une méthode numérique pour résoudre des équations différentielles ordinaires. Cette méthode est plus précise que la méthode d'Euler et est largement utilisée dans les applications scientifiques et techniques pour résoudre les problèmes de simulation et de modélisation.

La méthode de Runge-Kutta d'ordre 4 est une méthode de pas unique, ce qui signifie qu'elle calcule la solution à chaque instant de temps sans avoir besoin de connaître la solution à des instants précédents. Elle utilise quatre évaluations de la fonction dérivée de la solution pour calculer une approximation plus précise de la solution à chaque pas de temps.

Plus précisément, la méthode de Runge-Kutta d'ordre 4 utilise la formule suivante pour calculer l'approximation de la solution à l'instant $t+h$:

$$y(t + h) = y(t) + (1/6)\,(k1 + 2\,k2 + 2 * k3 + k4) * h \quad \text{(IV-2)}$$

Où

$$\begin{aligned} k1 &\; f\big(t, y(t)\big) \\ k2 &= f\left(t + \frac{h}{2}, y(t) + k1 * \frac{h}{2}\right) \\ k3 &= f\left(t + \frac{h}{2}, y(t) + k2 * \frac{h}{2}\right) \\ k4 &= f(t + h, y(t) + k3 * h) \end{aligned} \quad \text{(IV-3)}$$

f est la fonction dérivée de la solution à l'instant *t* et *y(t)* est la valeur de la solution à l'instant t.

La méthode de Runge-Kutta d'ordre 4 est plus précise que la méthode d'Euler car elle utilise une combinaison pondérée de la fonction dérivée en plusieurs points pour calculer chaque approximation de la solution. Cela permet d'obtenir une précision plus élevée avec des pas de temps plus grands, ce qui peut réduire le temps de calcul nécessaire pour résoudre les équations différentielles.

4. Mise en œuvre sous Matlab

Soit le problème de Cauchy

$$\begin{cases} y' = y - \frac{2x}{y} \\ y(0) = 1 \end{cases} \qquad \text{(IV-4)}$$

On désire approcher, effectuant le calcul avec quatre (4) décimales, la solution de (1) en x=0.5 à l'aide de la méthode d'Euler et celle de Runge-Kutta, en subdivisant l'intervalle [0, 1] en 50 parties égales.

La solution exacte étant $y = \sqrt{2.x\ 1}$, on estimera alors les résultats obtenus.

```
%============ Résolution numérique des équations différentielles===========
clear all;
close all;
clc;
a=input('a =');
b=input('b =');
n=input('n =');
h = (b - a)/n;
t=a:h:b;
disp(['h=',num2str(h)])
disp(['*Résolution numérique d''équation différentielle f=y-(2.x/y)*'])
disp(['***    S=1. Pour la Méthode d''Euler ***'])
disp(['*** S=2. Pour la Méthode de Runge-Kutta ***'])
disp(['*** S=3. Pour les deux méthodes ***'])
disp(['***    S=4. Pour Quitter ***'])
disp(['*** Tapez S entre <1-4> ***'])
s=input('S =');
switch s
case 1
disp(['*** Début de la méthode de Runge-Kutta ***'])
%%%%%%%%%%%%%%Trace de la solution exacte %%%%%%%%%%%%%%
yex=sqrt(2*t+1);
figure('color',[1 1 1])
plot(t,yex,'*-g')
%%%%%%%%%%%%%%METHODE Runge-Kutta %%%%%%%%%%%%%%
dydt=inline('y-(2.*t/y)','t','y');
epsilon=0.0001;u(1)=1+epsilon;
for i=1:n-1
  u1(i)=u(i);u2(i)=u(i)+h/2*dydt(t(i),u1(i));
  u3(i)=u(i)+h/2*dydt(t(i)+h/2,u2(i));u4(i)=u(i)+h*dydt(t(i)+h/2,u3(i));
  u(i+1)=u(i)+h/6*(dydt(t(i),u1(i))+2*dydt(t(i)+h/2,...
  u2(i))+2*dydt(t(i)+h/2,u3(i))+dydt(t(i+1),u4(i)));
  end
```

```
  hold on
  plot(t(1:end-1),u,'*-r')
  xlabel('                    tn')
  ylabel('                    Un')
  legend('Exacte','Runge-Kutta')
  grid
  %%%%%%%%%%%%%METHODE d'Euler %%%%%%%%%%%%%
case 2
disp(['*** Début de la méthode d''Euler ***'])
%%%%%%%%%%%%%Trace de la solution exacte%%%%%%%%%%%%
yex=sqrt(2*t+1);
figure('color',[1 1 1])
plot(t,yex,'*-g')
dydt = inline('y-(2.*t/y)','t','y');
epsilon=0.0001;u(1)=1+epsilon;
  for i=1:n-1
  u(i+1)=u(i)+h/2*(dydt(t(i),u(i)));
  end
hold on
  plot(t(1:end-1),u,'*-b')
  xlabel('                    tn')
  ylabel('                    Un')
  legend('Exacte','Euler')
  grid
case 3
disp(['*** Pour les deux méthodes ***'])
%%%%%%%%%%%%%Trace de la solution exacte%%%%%%%%%%%%
yex=sqrt(2*t+1);
figure('color',[1 1 1])
plot(t,yex,'*-g')
%%%%%%%%%%%%%METHODE de Runge-Kutta%%%%%%%%%%%%%
dydt=inline('y-(2.*t/y)','t','y');
epsilon=0.0001;u(1)=1+epsilon;
for i=1:n-1
  u1(i)=u(i);u2(i)=u(i)+h/2*dydt(t(i),u1(i));
  u3(i)=u(i)+h/2*dydt(t(i)+h/2,u2(i));u4(i)=u(i)+h*dydt(t(i)+h/2,u3(i));
  u(i+1)=u(i)+h/6*(dydt(t(i),u1(i))+2*dydt(t(i)+h/2,...
  u2(i))+2*dydt(t(i)+h/2,u3(i))+dydt(t(i+1),u4(i)));
  end
  hold on
  plot(t(1:end-1),u,'*-r')
  %%%%%%%%%%%%%METHODE d'Euler %%%%%%%%%%%%%
  for i=1:n-1
  u(i+1)=u(i)+h/2*(dydt(t(i),u(i)));
  end
hold on
  plot(t(1:end-1),u,'*-b')
  xlabel('                    tn')
  ylabel('                    Un')
  legend('Exacte','Runge-Kutta','Euler')
  grid
case 4
  otherwise
            disp('Erreur! la valeur de S entre <1-4> ');
end
```

5. TP N° 4 : Résolution numérique des équations différentielles

5.1. But du TP

Le but de ce TP est l'implémentation de la méthode d'Euler et la méthode de Range Kutta pour la résolution d'équations différentielles.

5.2. Énoncé du TP

Soit l'équation différentielle (a)

$$\dot{y} = y\,/1 + t^2 \qquad \text{(a)}$$

Avec $y(0) = 1, t \in [0\ 0.4]$ et un pas d'intégration $h = 0.2$

- Calculer la solution exacte de l'équation (a).
- Résoudre numériquement l'équation (a), par le biais de la méthode d'Euler et de Runge-Kutta.
- Afficher sur la même figure, la solution exacte ainsi que les solutions estimées.
- Comparer la solution exacte avec les approximations précédentes. Conclure !

Résolution numérique des systèmes d'équations linéaires

1. Introduction

La résolution numérique des systèmes d'équations linéaires est une technique mathématique importante utilisée dans de nombreux domaines, y compris les sciences, l'ingénierie, l'économie et l'informatique. Cette technique est utilisée pour trouver des solutions numériques à des systèmes d'équations linéaires qui ne peuvent pas être résolus analytiquement.

Il existe plusieurs méthodes pour résoudre numériquement des systèmes d'équations linéaires, mais l'une des plus courantes est la méthode de Gauss-Jordan. Cette méthode consiste à transformer le système d'équations linéaires en une forme échelonnée réduite, où chaque ligne a un nombre croissant de zéros à gauche. À partir de cette forme, les solutions du système peuvent être déterminées en remontant de la dernière équation à la première.

Une autre méthode populaire est la méthode de décomposition LU. Cette méthode consiste à décomposer la matrice de coefficients du système en deux matrices triangulaires inférieure et supérieure. Cette décomposition permet de résoudre rapidement des systèmes d'équations linéaires différents en utilisant les mêmes matrices triangulaires.

En outre, il existe d'autres techniques telles que la méthode de Jacobi, la méthode de Gauss-Seidel, la méthode de relaxation et la méthode de gradient conjugué, qui peuvent être utilisées pour résoudre des systèmes d'équations linéaires.

2. La méthode directe LU

La méthode directe LU (Lower-Upper) est une méthode courante de résolution numérique des systèmes d'équations linéaires. Elle consiste à décomposer la matrice de coefficients du système en deux matrices triangulaires inférieure et supérieure, afin de résoudre plus rapidement des systèmes d'équations linéaires différents en utilisant les mêmes matrices triangulaires. Voici les étapes pour résoudre numériquement un système d'équations linéaires avec la méthode directe LU :

1. Écrire le système d'équations sous forme matricielle $A*X = B$, où A est la matrice de coefficients, X est le vecteur inconnu à résoudre et B est le vecteur résultant.

2. Décomposer la matrice A en une matrice triangulaire inférieure L et une matrice triangulaire supérieure U, en utilisant la méthode de décomposition LU. Cette méthode consiste à effectuer des opérations élémentaires sur les lignes de la matrice A jusqu'à ce qu'elle soit transformée en deux matrices triangulaires L et U.

3. Résoudre le système d'équations linéaires en deux étapes : tout d'abord, résoudre $LY = B$ pour trouver Y, où Y est un vecteur intermédiaire, puis résoudre $UX = Y$ pour trouver X.

4. La résolution des systèmes d'équations linéaires triangulaires inférieurs ou supérieurs se fait par substitution en avant ou en arrière.

La méthode directe LU est efficace pour résoudre des systèmes d'équations linéaires de taille moyenne, mais elle peut être coûteuse en termes de temps de calcul pour les systèmes d'équations linéaires de grande taille. Cependant, elle est stable et précise, et elle est utilisée dans de nombreux domaines, y compris les sciences, l'ingénierie et l'informatique.

3. La méthode de Gauss

La méthode de Gauss, également connue sous le nom d'élimination de Gauss, est une méthode algébrique pour résoudre des systèmes d'équations linéaires en utilisant des opérations élémentaires sur les lignes d'une matrice augmentée.

La méthode consiste à écrire les équations linéaires sous forme de matrice augmentée, où les coefficients des variables sont représentés dans une matrice et les constantes sont placées dans une colonne séparée. Ensuite, on applique une série d'opérations élémentaires sur les lignes de la matrice augmentée pour transformer la matrice en une forme échelonnée réduite par rapport aux coefficients de variables.

Les opérations élémentaires comprennent l'addition d'une ligne à une autre ligne multipliée par un scalaire, l'échange de deux lignes et la multiplication d'une ligne par un scalaire non nul. En appliquant ces opérations, on obtient une matrice augmentée dans laquelle les coefficients des variables sont triangulaires supérieurs.

Ensuite, on utilise une méthode de substitution arrière pour trouver les valeurs de variables en partant de la dernière équation et en remontant vers la première. Ainsi, on peut obtenir la solution du système d'équations linéaires.

La méthode de Gauss est très efficace pour résoudre des systèmes d'équations linéaires de petite et moyenne taille, mais peut devenir très coûteuse en termes de temps et de calculs pour les grands systèmes. Cependant, de nombreuses variantes ont été développées pour améliorer son efficacité, telles que la méthode de Gauss-Jordan et la décomposition LU.

4. La méthode de Jacobi

La méthode de Jacobi est une méthode itérative pour résoudre des systèmes d'équations linéaires. Elle a été développée par le mathématicien allemand Carl Gustav Jacobi au 19ème siècle.

La méthode de Jacobi consiste à réécrire le système d'équations linéaires sous la forme d'une équation matricielle de la forme $Ax=b$, où A est une matrice carrée de coefficients et b est un vecteur colonne de constantes. Ensuite, on utilise une itération récurrente pour approximer la solution de l'équation $Ax=b$.

La méthode de Jacobi part de l'hypothèse que chaque composante du vecteur solution x est initialisée à zéro. À chaque itération, on calcule une nouvelle estimation du vecteur solution en utilisant la formule suivante :

$$x_i^{(k+1)} = \left(b_i - \sum_{j \neq i} a_{ij} * x_j^k\right) / a_{ii} \qquad \text{(V-1)}$$

Où $x_i^{(k+1)}$ est la nouvelle estimation de la ième composante du vecteur solution à l'itération $k+1$, a_{ij} sont les coefficients de la matrice A et x_j^k est l'estimation courante de la jème composante du vecteur solution à l'itération k.

Cette méthode itérative est répétée jusqu'à ce que la différence entre les estimations successives de x soit suffisamment petite.

La méthode de Jacobi est simple et facile à implémenter, mais elle peut être assez lente pour converger vers une solution précise pour de grands systèmes. Des variantes plus efficaces ont donc été développées, telles que la méthode de Gauss-Seidel et la méthode de relaxation successive

5. La méthode de Gauss-Seidel

La méthode de Gauss-Seidel est une méthode itérative pour résoudre des systèmes d'équations linéaires. Elle est similaire à la méthode de Jacobi, mais elle utilise une approche plus efficace en utilisant des valeurs actualisées dès qu'elles sont disponibles.

Comme la méthode de Jacobi, la méthode de Gauss-Seidel commence par réécrire le système d'équations linéaires sous la forme d'une équation matricielle de la forme $Ax=b$. Ensuite, elle utilise une itération récurrente pour approximer la solution de l'équation.

La méthode de Gauss-Seidel utilise la nouvelle estimation $x_i^{(k+1)}$ dès qu'elle est disponible pour calculer la nouvelle estimation de la composante suivante, $x_j^{(k+1)}$, au lieu

d'attendre que toutes les estimations soient calculées comme dans la méthode de Jacobi. La formule pour la méthode de Gauss-Seidel est la suivante :

$$x_i^{(k+1)} = \left(b_i - \sum_{j<i} a_{ij} * x_j^{(k+1)} - \sum_{j>i} a_{ij} * x_j^k\right) / a_{ii} \qquad \text{(IV-2)}$$

Où $x_i^{(k+1)}$ est la nouvelle estimation de la ième composante du vecteur solution à l'itération k+1, a_ij sont les coefficients de la matrice A, $x_j^{(k+1)}$ est la nouvelle estimation de la jème composante du vecteur solution à l'itération $k+1$, et x_j^k est l'estimation courante de la jème composante du vecteur solution à l'itération k.

La méthode de Gauss-Seidel est généralement plus rapide que la méthode de Jacobi car elle utilise les nouvelles estimations dès qu'elles sont disponibles. Cependant, elle ne fonctionne pas toujours pour tous les systèmes d'équations linéaires, et elle peut converger lentement pour certains systèmes mal conditionnés. Dans ce cas, il peut être préférable d'utiliser d'autres méthodes itératives telles que la méthode de relaxation successive.

6. Mise en œuvre sous Matlab

Pour le système suivant, on va utiliser la méthode directe, de Jacobi et de Gauss-Seidel (en considérant que le vecteur initiale est X_0=(0,0,0)):

```
%========== Résolution numérique des systèmes d'équations
linéaires======clear all;clc;
%Initialisation de la matrice
du systèmea=[4 -1 1;4 -8 1;-2
1 5];
%Initialisation du vecteur
des donnéesb=[7 ;-21 ;15];
%Nombre de variables et
d'équationsn=3;
%Le
vect
eur
X0X0
=
[0;0
;0];
disp(['*Résolution numérique de l''équation linéaire
A.X=B *'])disp(['*** S=1. Pour la méthode directe A\B
***'])
disp(['*** S=2. Pour la méthode directe LU
***']) disp(['*** S=3. Pour la méthode de
Gauss ***']) disp(['*** S=4. Pour la méthode
de Jacobi ***']) disp(['*** S=5. Pour la
méthode de Gauss-Seidel ***'])disp(['***  S=6.
Pour Quitter ***'])
disp(['*** Tapez S entre <1-6>
***'])s=input('S =');
switch s
case 1
```

```
disp(['*** Début de la méthode directe A/B ***'])
X=a \b
case 2
disp(['*** Début de la méthode directe LU ***'])
[L,U,X] = decompositionLU(a,b)
%La fonction decompositionLU.m
case 3
disp(['*** Début de la méthode de Gauss ***'])
%Formation de la matrice
augmentéeA=[a b];
for k=1:n-1
for i=k+1:n
A(i,:)=A(i,:)-(A(i,k)/A(k,k))*A(k,:);
end
endA
%Extraction de la solution du système
d'équationsfor 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=x'

case 4
disp(['*** Début de la méthode de Jacobi ***'])
[X,N] = jacobi(a,b,X0,50,0.0001) %La fonction jacobi.m
%50 Nombre maximal d'itérations
% N le nombre d'itération
% 0.0001 la précision
case 5
disp(['*** Début de la méthode de Gauss-Seidel
***']) [X,N] = gseidel(a,b,X0,50,0.0001) %La
fonction gseidel.m
case 6quit
    otherwise
        disp('Erreur! la valeur de S entre <1-6> ');
end
        %%================La fonction jacobi.m ================
function [X,niter] =
jacobi(A,b,X0,nmax,tol)n = length(b);
X = X0;
for niter = 1:nmax
% Calculer
l'itération suivante
for i = 1:n
j = [1:i-1,i+1:n];
somme = A(i,j)*X0(j);
X(i) = (b(i)-somme)/A(i,i);
end
% Tester la convergence
if norm(X-X0) < tol return
end
% L'ancien X0 devient le nouveau X
X0 = X;
```

```
end
% En cas de divergence
        %================La fonction gseidel.m ================
function [X,niter] =
gseidel(A,b,X0,nmax,tol)n = length(b);
X = X0;
for niter = 1:nmax % Calculer l'itération suivante
for i = 1:n
somme1 = A(i,1:i-1)*X(1:i-1);
somme2 = A(i,i+1:n)*X0(i+1:n);
X(i)=(b(i)-somme1-somme2)/A(i,i);
end
% Tester la convergence
if norm(X-X0) < tol
return
end
% L'ancien X0 devient le nouveau X
X0 = X;
end
% En cas de divergence
disp('Pas de convergence')

La fonction decompositionLU.m============================
function [L,U,x] = decompositionLU(A,b)
n = length(b); L = zeros(n,n); U = zeros(n,n); x = zeros(n,1);
%Factorisation LU      ------------------------------------------------
for i = 1:n ,
U(i,i) = 1;
endfor k = 1:n
for i = k:n
L(i,k) = A(i,k)-L(i,1:k-1)*U(1:k-1,k);
end
for j = k+1:n
U(k,j) = (A(k,j)-L(k,1:k-1)*U(1:k-1,j))/L(k,k);
end
end
------%--------       Résoudre Ly = b-----------------------------------------
y =
zeros(1,n);
for i = 1:n
somme = sum(L(i,1:i-1).*y(1:i-1));
y(i) = (b(i)-somme)/L(i,i);
end
------%--------       Résoudre Ux = y-----------------------------------------
for i = n:-1:1
somme =
U(i,i+1:n)*x(i+1:n);
x(i) = y(i)-somme;
end
        %================La fonction jacobi.m ================
function [X,niter] =
jacobi(A,b,X0,nmax,tol)n = length(b);
X = X0;
for niter = 1:nmax
% Calculer l'itération
suivante
for i = 1:n
j = [1:i-1,i+1:n];
somme = A(i,j)*X0(j);
X(i) = (b(i)-
somme)/A(i,i);
```

```
end
% Tester la
convergenceif
norm(X-X0) <
tol return
end
% L'ancien X0 devient le nouveau
X
X0 = X;
end
% En cas de divergence
        %================La fonction gseidel.m ================
function [X,niter] =
gseidel(A,b,X0,nmax,tol)n = length(b); X =
X0;
for niter = 1:nmax
% Calculer
l'itération suivante
for i = 1:n
somme1 = A(i,1:i-
1)*X(1:i-1);
somme2 = A(i,i+1:n)*X0(i+1:n);
X(i)=(b(i)-somme1-somme2)/A(i,i);
end
% Tester la
convergenceif
norm(X-X0) <
tol return
end
% L'ancien X0 devient le
nouveau XX0 = X;
end
% En cas de divergence
disp('Pas de convergence')
```

7. TP N° 4 : Résolution numérique des équations différentielles

7.1. But du TP

Durant ce TP, nous allons implémenter les méthodes numériques de résolution des systèmes d'équations linéaires (Méthode de Gauss, de Jacobi et de Gauss-Seidel).

7.2. Énoncé du TP

Soit le système linéaire suivant : $A * x = B$

Où $A = \begin{bmatrix} 10 & 7 & 5 \\ 7 & 8 & 6 \\ 8 & 9 & 5 \end{bmatrix}$; $B = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$

- Calculer x = inv(A) * B.
- Résoudre ce système en utilisant la méthode de Gauss, de Jacobi et de Gauss-Seidel.
- Comparer les résultats obtenus. Conclure !

Conclusion Générale

Conclusions générales

Les méthodes numériques sont des outils essentiels pour résoudre des problèmes mathématiques et scientifiques qui ne peuvent pas être résolus analytiquement. Ces méthodes impliquent l'utilisation de techniques mathématiques et informatiques pour approximer des solutions exactes ou pour calculer des valeurs numériques pour des problèmes complexes.

Les méthodes numériques comprennent un large éventail de techniques, telles que l'optimisation, l'algèbre linéaire, la résolution d'équations différentielles, l'intégration numérique et l'interpolation. Les méthodes numériques sont utilisées dans de nombreux domaines, tels que l'ingénierie, les sciences physiques, les mathématiques, la finance et l'informatique.

Les méthodes numériques ont leurs avantages et leurs limites. Elles peuvent fournir des solutions précises et rapides à des problèmes complexes, mais elles peuvent également être sujettes à des erreurs d'arrondi et à des instabilités numériques. Il est important de comprendre les propriétés et les limites de chaque méthode numérique afin de choisir la méthode appropriée pour résoudre un problème spécifique.

En fin de compte, les méthodes numériques sont des outils puissants qui ont permis d'accomplir des avancées significatives dans de nombreux domaines scientifiques et technologiques. Avec l'avènement de technologies de calcul avancées, les méthodes numériques continuent d'évoluer et de se développer pour relever les défis scientifiques les plus complexes de notre temps.

Références Bibliographique

REFERENCES BIBLIOGRAPHIQUE

1. "Numerical Recipes: The Art of Scientific Computing" par W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery.
2. "Numerical Mathematics and Computing" par E. Cheny et D. Kincaid.
3. "Numerical Methods for Engineers" par S. C. Chapra and R. P. Canale.
4. "Introduction to Numerical Methods in Differential Equations" par M. H. Holmes.
5. "Numerical Methods Using MATLAB" par J. H. Mathews and K. D. Fink.
6. "Applied Numerical Methods with MATLAB for Engineers and Scientists" par S. C. Chapra.
7. "Numerical Analysis" par R. L. Burden and J. D. Faires.
8. "Numerical Analysis: Mathematics of Scientific Computing" par D. Kincaid and W. Cheney.
9. "An Introduction to Numerical Methods: A MATLAB Approach" par R. J. Schilling et S. R. Harris.
10. "A First Course in Numerical Methods" par U. M. Ascher et C. Greif.
11. "Numerical Methods in Engineering with MATLAB" par J. Kiusalaas.
12. "Numerical Analysis and Scientific Computing" par J. F. Blowey et S. J. Cox.
13. "Numerical Methods for Engineers and Scientists: An Introduction with Applications Using MATLAB" par A. Gilat et V. Subramaniam.
14. "Applied Numerical Methods with MATLAB for Engineers and Scientists" par S. C. Chapra.
15. "Numerical Methods for Scientists and Engineers" par R. W. Hamming.
16. "Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms" par A. Greenbaum et T. P. Chartier.
17. "Numerical Analysis: Theory and Practice" par G. Dahlquist et A. Björck.
18. "Numerical Methods in Scientific Computing, Volume 1" par A. Quarteroni et R. Sacco.
19. "Numerical Methods for Scientists and Engineers" par R. B. Potts et R. L. Miller.
20. "Numerical Methods: Algorithms and Applications" par M. H. Schultz et W. L. Wade.