République Algérienne Démocratique et Populaire

Ministère de l'enseignement Supérieur et de la Recherche scientifique

Université Echahid Hamma Lakhdar – EL OUED

Faculté de la Technologie

Département de Génie Electrique

Polycopié destiné aux étudiants de Master I-S-1

en génie électrique

Matière:

Méthode numériques appliquées et optimisation

Réalisé par :

Dr. ALLAG Meriem

Septembre 2024

Résumé

Ce document est un support pédagogique du cours destiné aux étudiants de Master première année assurés au département de ***Génie électrique*** (Faculté de Technologie) pour les quatre spécialités: ***Commande électrique, Machine électrique , Réseaux électriques et Robotiques***. Dans ce polycopié de cours on s'intéresse, en première partie a un certain nombre de méthodes itératives utilisées pour la résolution des systèmes d'équations linéaires, des équations non linéaires, la résolution numérique des équations différentielle et les formules de quadrature, Trapèze et Simpson pour l'intégration numérique. Ensuite la résolution des équations aux dérivées partielles en utilisant la méthode des différences finies (MDF) et la méthode des éléments finis (MEF). En deuxième partie, on s'intéresse aux techniques d'optimisation ; dont l'optimisation de point de vue athématique consiste à rechercher le minimum ou le maximum d'une fonction avec ou sans contraintes, cependant on limite souvent l'optimisation à une recherche de minimum. Plusieurs méthodes seront étudiées comme la méthode de gradient optimal, gradient conjugué, méthode de Newton…etc. Le tous est regroupé sous le terme générique de ***''Méthodes Numériques appliquées et optimisation''***. A la fin de ce cours, l'étudient obtient des connaissances solides sur différentes méthodes numériques et technique d'optimisation que par la suite sera capable de les implémenter en langage de programmation tel que Matlab.

Mot de clé : EDO : équation différentielle ordinaire, EDP :équation différentielle partielle , MDF :Méthode des différences finies , MVF : Méthode des volumes finis ,

MEF : Méthode des éléments finis .

Objectifs de l'enseignement:

L'objectif de cet enseignement est de présenter les outils nécessaires à l'analyse numérique et à l'optimisation, avec ou sans contraintes, des systèmes physiques, dans le domaine de l'ingénierie.

Connaissances préalables recommandées:

Mathématique, programmation, maitrise de l'environnement MATLAB.

Contenu de la matière :

Chapitre I : Rappels sur quelques méthodes numériques (4 semaines)

1. Résolution des systèmes d'équations linéaires et non linéaires par les méthodes itératives (Méthode de Jacobi, Méthode de Gauss-Seidel, Méthode de Newton Raphson ,point fixe….

2. Intégration et différentiation numérique.

3.Méthodes de résolution d'équations différentielles ordinaires (EDO): Méthodes d'Euler ; Méthodes de Runge-Kutta ; Méthode d'Adams.

4.Résolution des système d'EDO.

Chapitre II : Equations aux dérivées partielles (EDP) (6 semaines)

1. Introduction et classifications des problèmes aux dérivées partielles et des conditions aux limites;
2. Méthode des différences finies (MDF)
3. Méthode des volumes finis (MVF)
4. Méthode des éléments finis (MEF).

Chapitre III : Techniques d'optimisation (6 semaines)

1. Définition et formulation d'un problème d'optimisation.
2. Optimisation unique et multiple avec ou sans contraintes.
3. Algorithmes d'optimisation sans contraintes (Méthodes déterministes, Méthodes stochastiques).
4. Traitement des contraintes (Méthodes de transformation, Méthodes directes).

Introduction sur les Objectifs du Cours d'Analyse Numérique et d'Optimisation

L'analyse numérique et l'optimisation sont des domaines fondamentaux dans le cadre de l'ingénierie moderne. Ils fournissent les outils et les méthodes nécessaires pour résoudre des problèmes complexes qui ne peuvent pas être résolus analytiquement ou qui nécessitent des solutions optimales pour des systèmes physiques. Ce cours a pour objectif de présenter les concepts et techniques essentiels de l'analyse numérique et de l'optimisation, en mettant l'accent sur leur application pratique dans des systèmes physiques avec ou sans contraintes. Dans ce résumé, nous explorerons les principaux objectifs de ce cours, les compétences qu'il vise à développer, et l'importance de ces outils dans le domaine de l'ingénierie.

1. Introduction à l'Analyse Numérique et à l'Optimisation

1.1 Importance de l'Analyse Numérique

L'analyse numérique est une branche des mathématiques qui développe, analyse et applique des algorithmes pour résoudre des problèmes mathématiques par des moyens numériques. Ces problèmes comprennent la résolution d'équations différentielles, l'intégration numérique, l'interpolation, et bien d'autres. Dans le contexte de l'ingénierie, l'analyse numérique est cruciale car elle permet de traiter des modèles mathématiques complexes qui décrivent le comportement des systèmes physiques.

Objectifs liés à l'analyse numérique :

- **Comprendre les méthodes numériques de base** : Le cours vise à enseigner les méthodes numériques fondamentales pour la résolution d'équations algébriques et différentielles, telles que les méthodes de Newton-Raphson, les méthodes de différences finies, et les méthodes de Runge-Kutta.
- **Développer des compétences en programmation** : L'utilisation des logiciels de calcul et de programmation, tels que MATLAB, Python ou autres, est essentielle pour mettre en œuvre et tester les méthodes numériques. Les étudiants apprendront à développer et à utiliser des programmes informatiques pour résoudre des problèmes d'analyse numérique.
- **Analyser les erreurs et la stabilité** : Un objectif clé est de comprendre les sources d'erreur dans les méthodes numériques (erreurs de troncature, erreurs d'arrondi) et d'évaluer la stabilité des algorithmes. Cette compréhension est essentielle pour garantir la précision et la fiabilité des solutions numériques.

1.2 Importance de l'Optimisation

L'optimisation est le processus de modification des entrées d'un système ou d'un processus pour atteindre les meilleures performances possibles, selon un critère donné. Dans

l'ingénierie, l'optimisation permet de concevoir des systèmes plus efficaces, de réduire les coûts, d'améliorer la qualité et de minimiser les ressources utilisées.

Objectifs liés à l'optimisation :

- **Formuler des problèmes d'optimisation** : Le cours enseigne comment formuler des problèmes d'optimisation en définissant des fonctions objectif et des contraintes. Cette compétence est cruciale pour transformer des problèmes d'ingénierie réels en problèmes d'optimisation mathématique.
- **Explorer les techniques d'optimisation** : Les étudiants apprendront différentes techniques d'optimisation, y compris l'optimisation linéaire et non linéaire, l'optimisation avec contraintes et sans contraintes, et les méthodes heuristiques. Les algorithmes comme la programmation linéaire (simplexe), les méthodes de gradient, et les algorithmes génétiques seront couverts.
- **Appliquer l'optimisation aux systèmes physiques** : Un objectif pratique est d'appliquer les techniques d'optimisation pour résoudre des problèmes spécifiques dans des systèmes physiques, tels que l'optimisation des structures mécaniques, des systèmes électriques, et des processus de production.

2. Méthodes et Techniques d'Analyse Numérique

2.1. Méthodes de Résolution d'Équations :

Les équations jouent un rôle central dans les mathématiques, la physique, l'ingénierie et d'autres disciplines scientifiques. Résoudre des équations, qu'elles soient linéaires, non linéaires, algébriques, ou différentielles, est essentiel pour comprendre les comportements et prédire les résultats de divers systèmes. Voici un aperçu des principales méthodes de résolution d'équations.

2.1. Équations Linéaires

Les équations linéaires sont parmi les plus simples et les plus couramment rencontrées. Elles prennent la forme générale où la relation entre les variables est proportionnelle. Les méthodes de résolution incluent :

- **Méthode de Substitution** : Utilisée principalement pour les systèmes de deux équations ou plus, elle consiste à résoudre une des équations pour une variable, puis substituer cette expression dans les autres équations.
- **Méthode d'Élimination (ou Combinaison Linéaire)** : Cette méthode consiste à combiner les équations pour éliminer une variable, simplifiant ainsi le système à une seule équation à une seule inconnue.

- **Méthode de Matrices (Méthode de Gauss-Jordan)** : Pour les systèmes plus complexes, les matrices sont utilisées pour organiser les équations et appliquer des opérations qui mènent à une solution.

2.2. Équations Non Linéaires

Les équations non linéaires, qui incluent des termes quadratiques, exponentiels, logarithmiques, ou trigonométriques, sont plus complexes à résoudre :

- **Méthode de la Bissection** : Une méthode numérique simple utilisée pour trouver les racines d'une équation non linéaire, en divisant l'intervalle en deux et en sélectionnant la sous-intervalle où un changement de signe indique une racine.
- **Méthode de Newton-Raphson** : Une méthode itérative efficace pour résoudre des équations non linéaires en approximant la solution. Elle nécessite le calcul de la dérivée de la fonction, ce qui la rend rapide mais dépendante des conditions initiales.
- **Méthode de Secante** : Similaire à Newton-Raphson, mais n'utilise pas de dérivée. Elle est plus simple mais peut converger moins rapidement

2.3. Techniques d'Intégration Numérique

L'intégration numérique est une technique pour calculer l'intégrale d'une fonction lorsque la forme analytique de l'intégrale est inconnue ou difficile à obtenir. Elle est couramment utilisée dans divers domaines, tels que la physique, l'ingénierie, et les sciences économiques, pour résoudre des problèmes pratiques où les données sont souvent disponibles sous forme discrète. Voici un aperçu des principales techniques d'intégration numérique.

Méthodes de quadrature : Le cours couvre des méthodes telles que la méthode des trapèzes et la méthode de Simpson. Ces techniques sont utilisées pour estimer les intégrales définies à partir de points d'échantillonnage discrets

1. Méthode des Trapèzes

La méthode des trapèzes est l'une des techniques les plus simples d'intégration numérique. Elle consiste à approximer la région sous la courbe par une série de trapèzes et à additionner les aires de ces trapèzes.

Avantages: Simple à mettre en œuvre et efficace pour des fonctions lisses et continues.

Inconvénients: Moins précise pour des fonctions fortement non linéaires ou oscillantes et nécessite un grand nombre de points pour des approximations précises.

2. Méthode de Simpson

La méthode de Simpson est une technique d'ordre supérieur qui utilise des polynômes de degré deux pour approximer les segments de la fonction. Il s'agit d'une amélioration significative par rapport à la méthode des trapèzes.

Avantages : Plus précise que la méthode des trapèzes pour un nombre comparable de points et particulièrement efficace pour des fonctions polynomiales de bas degré.

Inconvénients : Plus complexe à mettre en œuvre et requiert un nombre pair de segments.

En conclusion : Les techniques d'intégration numérique offrent des outils puissants pour estimer les intégrales de fonctions complexes, chaque méthode ayant ses propres avantages et limites. Le choix de la méthode dépend de la nature de la fonction, de la précision requise et de la complexité acceptable du calcul. En combinant différentes techniques, il est possible de résoudre efficacement une large gamme de problèmes pratiques.

2.4. Résolution des Équations Différentielles : Principales Méthodes

Les équations différentielles sont essentielles pour modéliser les comportements dynamiques dans des domaines variés, tels que la physique, l'ingénierie, et la biologie. Elles décrivent comment une variable évolue en fonction d'une autre, généralement le temps. Les méthodes de résolution peuvent être analytiques ou numériques, ces dernières étant cruciales lorsque les solutions analytiques sont inaccessibles ou trop compliquées.

1. Méthode d'Euler
La méthode d'Euler est une technique basique pour résoudre des équations différentielles ordinaires (EDO). Elle avance étape par étape, en utilisant la dérivée pour calculer la prochaine valeur de la solution. Bien que simple à utiliser, cette méthode peut manquer de précision et devenir instable avec de grands pas de temps, nécessitant des ajustements fins pour améliorer l'exactitude.

2. Méthodes de Runge-Kutta
Les méthodes de Runge-Kutta, notamment la méthode de quatrième ordre (RK4), sont largement préférées pour leur équilibre entre précision et effort de calcul. En effectuant plusieurs estimations intermédiaires à chaque pas, RK4 améliore significativement l'approximation des solutions par rapport à la méthode d'Euler. RK4 est particulièrement adaptée aux problèmes où des approximations plus simples échouent.

3. Méthodes Implicites

Dans les systèmes raides, où les variations rapides peuvent entraîner l'instabilité des méthodes explicites comme Euler et Runge-Kutta, les méthodes implicites sont utilisées. Par exemple, la méthode de Backward Euler résout les équations en tenant compte des valeurs futures,

offrant ainsi une meilleure stabilité pour les systèmes à changement rapide. Cependant, ces méthodes sont plus complexes et exigent plus de calculs.

4. Méthode de Taylor (Ordre 2 et 4)

Les méthodes de Taylor utilisent les dérivées successives de la fonction pour construire une approximation plus précise de la solution. Les méthodes de Taylor de second et quatrième ordre prennent en compte plusieurs dérivées, permettant une approximation plus fine et précise des solutions. Bien qu'elles offrent une meilleure précision, elles peuvent être difficiles à implémenter pour des équations complexes nécessitant de nombreuses dérivées.

En conclusion les techniques de résolution des équations différentielles varient en fonction de la complexité des équations et des besoins de précision et de stabilité. Le choix de la méthode dépend du type de système à modéliser et des contraintes de calcul, chaque méthode offrant un équilibre entre simplicité, précision, et stabilité.

3. Introduction et Classifications des EDP

Le cours commence par une introduction aux EDP, en expliquant leur rôle dans la modélisation des processus dynamiques. Les EDP sont classées selon différents types, notamment par ordre (premier, second, etc.) et par leur nature (elliptiques, paraboliques, hyperboliques). L'importance des conditions aux limites, qui spécifient les valeurs de la solution sur les frontières du domaine de résolution, est également abordée. Ces conditions aux limites sont cruciales pour déterminer des solutions uniques et physiquement significatives.

Méthodes de Résolution des EDP :
Le cours explore trois principales méthodes numériques pour résoudre les EDP :

- **Méthode des Différences Finies (MDF) :** Cette méthode convertit les EDP en systèmes d'équations algébriques en utilisant des approximations par différences pour les dérivées. La MDF est utile pour des grilles régulières et des problèmes simples.
- **Méthode des Volumes Finis (MVF) :** Adaptée aux problèmes de conservation, la MVF divise le domaine en petits volumes et applique des bilans de conservation à chaque volume. Cette méthode est particulièrement efficace pour les problèmes de dynamique des fluides et de transfert de chaleur.
- **Méthode des Éléments Finis (MEF) :** La MEF subdivise le domaine en éléments plus petits et utilise des fonctions de forme pour approximer la solution. Elle est largement utilisée pour résoudre des problèmes avec des géométries complexes et des conditions aux limites variées.

Introduction sur les Objectifs du Cours d'Analyse Numérique et d'Optimisation

En combinant théorie et application pratique, ce cours vise à doter les étudiants des compétences nécessaires pour comprendre et appliquer efficacement les techniques de résolution des EDP dans divers contextes scientifiques et d'ingénierie.

4. Techniques et Stratégies d'Optimisation

4.1 Optimisation Linéaire

L'optimisation linéaire traite des problèmes où la fonction objectif et les contraintes sont linéaires. Ce type d'optimisation est courant dans la recherche opérationnelle et l'économie.

- **Algorithme du simplexe** : Le cours détaille cet algorithme fondamental pour résoudre des problèmes d'optimisation linéaire. Les étudiants apprendront à formuler des problèmes de programmation linéaire et à les résoudre à l'aide de l'algorithme du simplexe.
- **Dualité en programmation linéaire** : La dualité est un concept important qui relie chaque problème d'optimisation linéaire à un problème dual. Le cours explique comment utiliser la dualité pour interpréter les solutions et améliorer l'efficacité de la résolution.

4.2 Optimisation Non Linéaire

L'optimisation non linéaire traite des problèmes avec des fonctions objectif ou des contraintes non linéaires. Ces problèmes sont plus complexes que l'optimisation linéaire et nécessitent des techniques spécifiques.

- **Méthodes de gradient** : Les méthodes de gradient et de gradient conjugué sont utilisées pour minimiser des fonctions non linéaires différentiables. Le cours explore l'usage de ces méthodes, en mettant l'accent sur les techniques de descente de gradient pour atteindre les minima locaux.
- **Méthodes de Newton et quasi-Newton** : Ces méthodes améliorent la convergence par rapport aux méthodes de gradient en utilisant des informations sur la courbure de la fonction objectif. Le cours couvre l'implémentation de ces méthodes et les stratégies pour surmonter les défis liés à l'inversion des matrices Hessiennes.

4.3 Optimisation avec Contraintes

Les problèmes d'optimisation dans l'ingénierie incluent souvent des contraintes qui doivent être satisfaites simultanément avec la minimisation ou la maximisation de la fonction objective.

- **Conditions de Karush-Kuhn-Tucker (KKT)** : Les conditions KKT généralisent les conditions de Lagrange pour les problèmes avec des contraintes d'égalité et d'inégalité.

Le cours enseigne comment utiliser ces conditions pour identifier les solutions optimales dans des contextes non linéaires.

- **Méthodes des multiplicateurs de Lagrange** : Utilisées pour incorporer des contraintes dans les problèmes d'optimisation, ces méthodes permettent de transformer un problème contraint en un problème non contraint. Le cours explore les applications des multiplicateurs de Lagrange dans l'optimisation de systèmes physiques complexes.

5. Applications Pratiques en Ingénierie

L'objectif final de ce cours est d'appliquer les concepts et techniques d'analyse numérique et d'optimisation à des problèmes réels d'ingénierie. Voici quelques exemples d'applications pratiques :

- **Optimisation de la conception structurelle** : Utiliser des méthodes numériques pour analyser et optimiser la conception de structures mécaniques afin de minimiser le poids tout en respectant les contraintes de résistance et de déformation.
- **Simulation et contrôle des systèmes dynamiques** : Appliquer des techniques d'intégration numérique pour simuler le comportement des systèmes dynamiques, tels que les véhicules autonomes ou les systèmes robotiques, et utiliser des techniques d'optimisation pour améliorer la performance du contrôle.
- **Optimisation de la gestion de l'énergie** : Utiliser des techniques d'optimisation pour améliorer l'efficacité énergétique dans les systèmes de production et de distribution d'énergie, en minimisant les coûts tout en répondant aux contraintes de demande et d'approvisionnement.
- **Planification de la production et de la logistique** : Appliquer des modèles d'optimisation linéaire et non linéaire pour planifier et gérer les opérations de production, la chaîne d'approvisionnement, et la distribution, afin de maximiser l'efficacité et minimiser les coûts opérationnels.

Conclusion

L'analyse numérique et l'optimisation sont des outils essentiels pour résoudre des problèmes complexes en ingénierie. Ce cours vise à doter les étudiants des compétences nécessaires pour utiliser ces outils de manière efficace, en leur fournissant une compréhension approfondie des méthodes et des techniques qui sous-tendent l'analyse numérique et l'optimisation. En mettant l'accent sur les applications pratiques, le cours prépare les étudiants à relever les défis du monde réel, en leur permettant de concevoir des systèmes plus efficaces, plus robustes, et plus optimaux

Chapitre I : Rappels sur quelques méthodes numériques

1. Résolution des systèmes d'équations linéaires par les méthodes itératives (Méthode de Jacobi, Méthode de Gauss-Seidel)
2. Résolution d'une équation non linéaires par les méthodes itératives , Méthode de bissection ,sécante , Newton Raphson ,point fixe….
3. Résolution des systémes des équation non linéaires par les méthodes itératives , Newton Raphson ,point fixe
4. . Intégration et différentiation numérique.
5. Méthodes de résolution d'équations différentielles ordinaires (EDO): Méthodes d'Euler ; Méthodes de Runge-Kutta ; Méthode d'Adams.
6. Résolution des système d'EDO.

Chapitre I : Rappels sur quelques méthodes numériques

1. Résolution d'Équations Linéaires par les méthodes itératives

Les équations linéaires sont omniprésentes dans les mathématiques et les sciences appliquées. Elles se présentent sous la forme d'un système d'équations où chaque équation est une combinaison linéaire des variables. Dans de nombreux cas, ces systèmes sont trop grands pour être résolus efficacement par des méthodes directes comme l'élimination de Gauss ou la méthode de Gauss-Jordan. Les méthodes itératives offrent une alternative efficace pour ces situations, en particulier pour les systèmes creux ou de grande taille. Cet essai explore en détail deux méthodes itératives principales pour résoudre des systèmes d'équations linéaires : la méthode de Jacobi et la méthode de Gauss-Seidel, illustrées par plusieurs exemples concrets.

Les méthodes itératives commencent avec une estimation initiale de la solution et améliorent cette estimation en utilisant un processus répétitif. Contrairement aux méthodes directes, qui visent à trouver la solution exacte en un nombre fini d'étapes, les méthodes itératives continuent jusqu'à ce que l'erreur soit inférieure à un seuil prédéfini.

Pourquoi utiliser des méthodes itératives ?

1. **Efficacité pour les grands systèmes** : Les méthodes itératives sont souvent plus rapides et nécessitent moins de mémoire pour les systèmes avec un grand nombre d'équations.
2. **Adaptabilité aux systèmes creux** : Les systèmes avec de nombreux zéros dans leur matrice de coefficients sont bien adaptés aux méthodes itératives, car seules les valeurs non nulles doivent être stockées et manipulées.
3. **Flexibilité** : Ces méthodes peuvent facilement être ajustées pour répondre à des critères de précision différents, ce qui les rend utiles pour des applications nécessitant des approximations rapides.

1.1 Méthode de Jacobi

Description de la Méthode de Jacobi

La méthode de Jacobi est une technique simple et intuitive pour résoudre des systèmes d'équations linéaires. Elle repose sur l'idée de séparer chaque équation du système et de résoudre chaque variable indépendamment en utilisant les valeurs actuelles des autres variables.

Algorithme de la Méthode de Jacobi

Soit un système de nxn équations linéaires sous la forme :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Ce système peut être écrit sous forme matricielle :$Ax = B$

où A est une matrice $n \times n$, x est un vecteur de taille n, et b est un vecteur de taille n. Supposons que A puisse être décomposée en $D + R$, où D est une matrice diagonale contenant les éléments diagonaux de A et R est la matrice des termes restants. Le système peut être écrit comme :

$$Dx = b - Rx$$

L'algorithme de Jacobi se définit alors par l'itération suivante :

$$x^{k+1} = D^{-1}(b - Rx^k)$$

Exemple 1 : Système de Deux Équations à Deux Inconnues

Considérons le système d'équation suivant :$\begin{cases} \boldsymbol{4x + y = 5} \\ \boldsymbol{2x + 3y = 5} \end{cases}$

Étape 1 : Formuler les itérations de Jacobi :

Étape 1 : Formuler les itérations de Jacobi.

Écrivons chaque équation pour isoler les variables :

Pour la première équation :

$$x = \frac{1}{4}(5 - y)$$

Pour la deuxième équation :

$$y = \frac{1}{3}(5 - 2x)$$

Étape 2 : Choisir une estimation initiale.

Choisissons $x^{(0)} = 0$ et $y^{(0)} = 0$.

Étape 3 : Effectuer les itérations.

Itération 1 :

$$x^{(1)} = \frac{1}{4}(5 - 0) = 1.25$$

$$y^{(1)} = \frac{1}{3}(5 - 2 \times 0) = \frac{5}{3} \approx 1.67$$

Itération 2 :

$$x^{(2)} = \frac{1}{4}(5 - 1.67) = 0.833$$

$$y^{(2)} = \frac{1}{3}(5 - 2 \times 1.25) = 0.833$$

Les itérations continuent jusqu'à ce que les valeurs de x ety convergent vers une solution stable.

Avantages et Inconvénients de la Méthode de Jacobi

Avantages :

- **Simplicité** : Facile à comprendre et à implémenter.
- **Parallélisable** : Chaque équation peut être résolue indépendamment, ce qui permet une mise en œuvre parallèle efficace.

Inconvénients :

- **Convergence lente** : Peut nécessiter de nombreuses itérations pour converger, en particulier si la matrice A n'est pas strictement diagonale dominante.
- **Stabilité** : Moins stable que la méthode de Gauss-Seidel, car elle repose sur des valeurs antérieures plutôt que mises à jour.

1.2. Méthode de Gauss-Seidel

Description de la Méthode de Gauss-Seidel

La méthode de Gauss-Seidel améliore la méthode de Jacobi en utilisant les valeurs mises à jour des variables dès qu'elles sont disponibles. Cela signifie que chaque nouvelle estimation de la variable est utilisée immédiatement dans les calculs des autres variables dans la même itération, ce qui peut accélérer la convergence.

Algorithme de la Méthode de Gauss-Seidel

L'algorithme de Gauss-Seidel est similaire à celui de Jacobi, sauf qu'il utilise les valeurs mises à jour dans le calcul :

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}\right)$$

Exemple 2 : Système de Deux Équations à Deux Inconnues

Considérons le système d'équation suivant : $\begin{cases} \mathbf{4x + y = 5} \\ \mathbf{2x + 3y = 5} \end{cases}$

Étape 1 : Formuler les itérations de Gauss-Seidel.

Pour la première équation :

$$x^{(k+1)} = \frac{1}{4}(5 - y^{(k)})$$

Pour la deuxième équation :

$$y^{(k+1)} = \frac{1}{3}(5 - 2x^{(k+1)})$$

Étape 2 : Choisir une estimation initiale.

Choisissons $x^{(0)} = 0$ et $y^{(0)} = 0$.

Étape 3 : Effectuer les itérations.

Itération 1 :

$$x^{(1)} = \frac{1}{4}(5 - 0) = 1.25$$

$$y^{(1)} = \frac{1}{3}(5 - 2 \times 1.25) = 0.833$$

Itération 2 :

$$x^{(2)} = \frac{1}{4}(5 - 0.833) = 1.042$$

$$y^{(2)} = \frac{1}{3}(5 - 2 \times 1.042) = 0.972$$

Les itérations continuent jusqu'à convergence. On remarque souvent que la méthode de Gauss-Seidel converge plus rapidement que la méthode de Jacobi.

Avantages et Inconvénients de la Méthode de Gauss-Seidel

Avantages :

- **Convergence rapide** : Utilise les valeurs mises à jour dès qu'elles sont disponibles, ce qui accélère souvent la convergence.
- **Plus stable** : Tendance à être plus stable que Jacobi, particulièrement pour les matrices diagonales dominantes.

Inconvénients :

- **Dépendance des variables** : Les calculs ne sont pas parallélisables, ce qui peut limiter l'efficacité sur les architectures parallèles.
- **Convergence conditionnelle** : Peut ne pas converger pour toutes les matrices. La convergence est assurée si la matrice est strictement diagonale dominante.

Conclusion

Les méthodes itératives comme Jacobi et Gauss-Seidel sont des outils puissants pour résoudre des systèmes d'équations linéaires, en particulier pour les grands systèmes et ceux avec des matrices creuses. Elles offrent une alternative pratique et souvent plus rapide aux méthodes directes, surtout dans les applications nécessitant des approximations rapides ou une solution approchée. La méthode de Gauss-Seidel, en particulier, tend à converger plus rapidement et est plus stable, ce qui en fait le choix préféré dans de nombreuses situations. Cependant, la méthode de Jacobi reste utile dans les cas où la parallélisation est possible et avantageuse. Une compréhension approfondie de ces méthodes permet aux mathématiciens, ingénieurs, et scientifiques de choisir l'approche la plus efficace et appropriée pour leurs problèmes spécifiques.

Exemple 1 considérons le même système précédemment

Étape 1 : Formulation des itérations de Gauss-Seidel

Les équations peuvent être réécrites sous forme itérative :

$$x^{(k+1)} = \frac{1}{4}(5 - y^{(k)})$$

$$y^{(k+1)} = \frac{1}{3}(6 - x^{(k+1)})$$

```
% Initialisation
x = 0; % Estimation initiale pour x
y = 0; % Estimation initiale pour y
n_iterations = 10; % Nombre d'itérations
% Itérations de Jacobi
for k = 1:n_iterations
    x_new = (5 - y) / 4;
    y_new = (6 - x) / 3;
    x = x_new;
  y = y_new;
    fprintf('Iteration %d: x = %f, y = %f\n', k, x, y);
end
```

Exemple 2 : Résolution d'un Système de Trois Équations à Trois Inconnues

Considérons le système suivant :

$$\begin{cases} 10x + 2y - z = 27 \\ -3x - 6y + 2z = -61.5 \\ x + y + 5z = -21.5 \end{cases}$$

Étape 1 : Formulation des itérations de Jacobi

Les équations peuvent être réécrites sous forme itérative :

$$x^{(k+1)} = \frac{1}{10}(27 - 2y^{(k)} + z^{(k)})$$

$$y^{(k+1)} = \frac{1}{-6}(-61.5 + 3x^{(k)} - 2z^{(k)})$$

$$z^{(k+1)} = \frac{1}{5}(-21.5 - x^{(k)} - y^{(k)})$$

```
% Initialisation
x = 0; y = 0; z = 0; % Estimations initiales
n_iterations = 20; % Nombre d'itérations
% Itérations de Jacobi
for k = 1:n_iterations
    x_new = (27 - 2*y + z) / 10;
    y_new = (-61.5 + 3*x - 2*z) / -6;
    z_new = (-21.5 - x - y) / 5;
    x = x_new;
    y = y_new;
    z = z_new;
    fprintf('Iteration %d: x = %f, y = %f, z = %f\n', k, x, y, z);
end
```

Exemple 3 :(Gauss Seidel Method)

```
% Initialisation
x = 0; % Estimation initiale pour x
y = 0; % Estimation initiale pour y
```

```
n_iterations = 10; % Nombre d'itérations

% Itérations de Gauss-Seidel

for k = 1:n_iterations

   x = (5 - y) / 4;

   y = (6 - x) / 3;

   fprintf('Iteration %d: x = %f, y = %f\n', k, x, y);

end
```

Exemple 4 : Gauss Seidel

Exemple 4 : Résolution d'un Système de Trois Équations à Trois Inconnues

Considérons le même système que pour la méthode de Jacobi :

$$\begin{cases} 10x + 2y - z = 27 \\ -3x - 6y + 2z = -61.5 \\ x + y + 5z = -21.5 \end{cases}$$

Étape 1 : Formulation des itérations de Gauss-Seidel

Les équations peuvent être réécrites sous forme itérative :

$$x^{(k+1)} = \frac{1}{10}(27 - 2y^{(k)} + z^{(k)})$$

$$y^{(k+1)} = \frac{1}{-6}(-61.5 + 3x^{(k+1)} - 2z^{(k)})$$

$$z^{(k+1)} = \frac{1}{5}(-21.5 - x^{(k+1)} - y^{(k+1)})$$

Programme MATLAB :

```
% Initialisation

x = 0; y = 0; z = 0; % Estimations initiales

n_iterations = 20; % Nombre d'itérations

% Itérations de Gauss-Seidel

for k = 1:n_iterations

   x = (27 - 2*y + z) / 10;

   y = (-61.5 + 3*x - 2*z) / -6;

   z = (-21.5 - x - y) / 5;

   fprintf('Iteration %d: x = %f, y = %f, z = %f\n', k, x, y, z);

end
```

Chapitre I : Rappels sur quelques méthodes numériques

2. Résolution des systèmes des équations Non Linéaires

2.1 Résolution d'équations non linéaires(a une variable)

L'objet essentiel de ce chapitre est l'approximation des racines d'une fonction réelle d'une variable réelle, c'est-à-dire la résolution approchée du problème suivant :
Étant donné une équation non linéaire, à une seul variable, est définie par :

$$f: R \to R$$
$$f(x) = 0$$

La valeur de la variable ***x*** qui vérifie cette égalité est appelée ***solution*** (ou racine) de l'équation, elle est notée ***x0***. Dans beaucoup de cas, on doit recourir aux méthodes numériques car la solution ne peut pas être déterminée analytiquement.
Il existe plusieurs techniques permettant de résoudre l'équation non linéaire, ces méthodes se distinguent par leurs principes et leurs vitesses de convergence. Nous introduisons dans cette section la méthodes ***dichotomie*** (ou de bissection) et de ***Newton*** et de celle de ***sécante*** ***Puis la méthode de*** point fixe

2.1. 1. La Méthode de la Bissection : Une Approche Approfondie

La méthode de la bissection, également connue sous le nom de méthode de dichotomie, est une technique classique et fondamentale pour trouver les racines d'équations non linéaires. C'est l'une des méthodes les plus simples et les plus robustes pour résoudre des équations de la forme f(x)=0, où f est une fonction continue. La méthode est basée sur le théorème des valeurs intermédiaires et fonctionne en réduisant l'intervalle de recherche de manière itérative jusqu'à ce qu'une précision souhaitée soit atteinte.

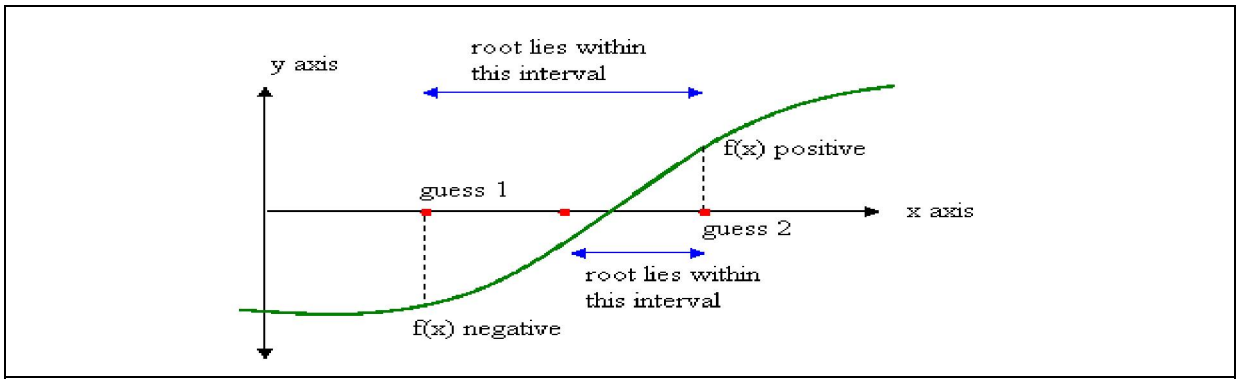


FIG 01 :le principe de la métode bissection

1. **Concept Fondamental de la Méthode de Bissection**

Chapitre I : Rappels sur quelques méthodes numériques

Le principe de la méthode de la bissection repose sur une idée simple : si une fonction continue $f(x)$ change de signe entre deux points a et b (c'est-à-dire $f(a) \times f(b) < 0$), alors il existe au moins une racine dans l'intervalle $[a, b]$.

Étapes de la Méthode de la Bissection

1. **Définir l'intervalle initial** : Choisir deux points a et b tels que $f(a) \times f(b) < 0$. Cela signifie que la fonction change de signe entre a et b, suggérant l'existence d'une racine dans cet intervalle.
2. **Calculer le point médian** : Trouver le point médian c de l'intervalle $[a, b]$:

$$c = \frac{a+b}{2}$$

3. **Évaluer la fonction au point médian** :
   - Si $f(c) = 0$, alors c est la racine exacte.
   - Si $f(c) \times f(a) < 0$, cela signifie que la racine se trouve dans l'intervalle $[a, c]$. Dans ce cas, remplacez b par c.
   - Si $f(c) \times f(b) < 0$, cela signifie que la racine se trouve dans l'intervalle $[c, b]$. Dans ce cas, remplacez a par c.

4. **Répéter le processus** : Continuer à répéter les étapes ci-dessus en réduisant l'intervalle de moitié à chaque itération jusqu'à ce que l'intervalle soit suffisamment petit (précision souhaitée).

2. Avantages et Limites de la Méthode de la Bissection

Avantages

- **Simplicité** : La méthode est simple à comprendre et à implémenter.
- **Robustesse** : Elle garantit la convergence tant que la fonction est continue sur l'intervalle initial et que la condition de changement de signe est satisfaite.
- **Fiabilité** : Elle ne dépend pas de la dérivée de la fonction, ce qui la rend applicable même aux fonctions non dérivables.

Limites

- **Lenteur** : La méthode de la bissection est relativement lente car elle divise l'intervalle par deux à chaque itération, ce qui nécessite un grand nombre d'itérations pour atteindre une précision élevée.
- **Nécessité de changement de signe** : La méthode nécessite un changement de signe de la fonction sur l'intervalle, ce qui peut ne pas être présent dans toutes les situations.

3. Mise en Œuvre de la Méthode de la Bissection en Matlab

La mise en œuvre de la méthode de la bissection peut être réalisée facilement à l'aide de Matlab. Voici comment cela peut être fait à travers des exercices pratiques.

Exercice 1 : Trouver la racine de $f(x) = x^2 - 4$

Problème : Trouver une racine de l'équation $f(x) = x^2 - 4$ en utilisant la méthode de la bissection sur l'intervalle $[1, 3]$.

Solution avec Matlab :

```
% Définir la fonction
f = @(x) x^2 - 4;
% Définir l'intervalle initial
a = 1;
b = 3;
% Définir la tolérance
tol = 1e-6;
% Implémenter la méthode de la bissection
while (b - a) / 2 > tol
    c = (a + b) / 2; % Point médian
    if f(c) == 0
        break; % c est la racine exacte
    elseif f(a) * f(c) < 0
        b = c; % La racine est dans [a, c]
    else
        a = c; % La racine est dans [c, b]
    end
end
% Résultat
root = (a + b) / 2;
disp(['La racine approximée est : ', num2str(root)]);
```

Explication

1. La fonction $f(x) = x^2 - 4$ est définie.
2. L'intervalle initial est choisi comme $[1, 3]$.
3. Une tolérance de 10^{-6} est définie pour arrêter l'itération.
4. La boucle `while` continue jusqu'à ce que l'intervalle soit suffisamment petit.
5. À chaque itération, le point médian c est calculé, et l'intervalle est mis à jour en fonction du signe de $f(c)$.

Exercice 2 : Résolution d'une Équation Trigonométrique

Problème : Trouver une racine de l'équation $f(x) = \cos(x) - x$ sur l'intervalle $[0, \pi/2]$.

Solution avec Matlab :

```
% Définir la fonction
f = @(x) cos(x) - x;
% Définir l'intervalle initial
a = 0;
b = pi/2;
% Définir la tolérance
tol = 1e-6;
% Implémenter la méthode de la bissection
while (b - a) / 2 > tol
   c = (a + b) / 2; % Point médian
   if f(c) == 0
      break; % c est la racine exacte
   elseif f(a) * f(c) < 0
      b = c; % La racine est dans [a, c]
   else
      a = c; % La racine est dans [c, b]
   end
end
% Résultat
root = (a + b) / 2;
disp(['La racine approximée est : ', num2str(root)]);
```

Explication :

1. La fonction $f(x) = \cos(x) - x$ est définie.
2. L'intervalle initial est choisi comme $[0, \pi/2]$, car on sait que la fonction change de signe dans cet intervalle.
3. La méthode de la bissection est appliquée pour réduire l'intervalle à chaque itération jusqu'à ce que la tolérance soit atteinte.

4. Exercice Avancé : Application à une Fonction Non Linéaire Complexe

Problème : Trouver la racine de $f(x) = x^3 - 2x + 1$ sur l'intervalle $[-2, 0]$.

Solution avec Matlab :

```
% Définir la fonction
f = @(x) x^3 - 2*x + 1;
% Définir l'intervalle initial
a = -2;
```

```
b = 0;
% Définir la tolérance
tol = 1e-6;
% Implémenter la méthode de la bissection
while (b - a) / 2 > tol
   c = (a + b) / 2; % Point médian
   if f(c) == 0
      break; % c est la racine exacte
   elseif f(a) * f(c) < 0
      b = c; % La racine est dans [a, c]
   else
      a = c; % La racine est dans [c, b]
   end
end
% Résultat
root = (a + b) / 2;
disp(['La racine approximée est : ', num2str(root)]);
```

Explication :

1. La fonction $f(x) = x^3 - 2x + 1$ est définie.
2. L'intervalle initial est choisi comme $[-2, 0]$, un intervalle dans lequel la fonction change de signe.
3. À chaque itération, le point médian est calculé, et l'intervalle est réduit en fonction du signe de la fonction à ce point.

5. Analyse et Comparaison des Résultats

Après avoir mis en œuvre ces exercices, on peut analyser les résultats obtenus et comprendre l'efficacité de la méthode de la bissection. Comparons les résultats avec des solutions analytiques (lorsqu'elles sont disponibles) ou avec d'autres méthodes numériques comme la méthode de Newton-Raphson.

- **Précision** : La méthode de la bissection peut atteindre une précision élevée avec un nombre suffisant d'itérations. Cependant, elle est généralement plus lente que les méthodes qui utilisent des informations supplémentaires sur la fonction, comme les dérivées.
- **Convergence** : La convergence de la méthode de la bissection est garantie sous les conditions de continuité et de changement de signe, contrairement à la méthode de Newton-Raphson qui peut ne pas converger si la dérivée s'annule ou si le point initial est mal choisi.

En conclusion, La méthode de la bissection est une méthode puissante pour trouver les racines des équations non linéaires. Elle est simple, robuste et garantit la convergence sous des conditions spécifiques. En combinant une compréhension théorique avec des exercices pratiques, les étudiants peuvent développer une intuition forte pour l'application de cette méthode à une variété de problèmes. Les exemples de Matlab fournis montrent comment implémenter cette méthode de manière efficace, fournissant des compétences analytiques essentielles pour résoudre des problèmes non linéaires dans différents domaines. En utilisant des outils comme Matlab, les étudiants peuvent explorer les effets de différents paramètres sur la précision et la convergence, les préparant à aborder des problèmes plus complexes dans leurs études et leurs carrières professionnelles

Travaux dirigés (Méthode de la bissection)

Exercice 1: Trouver la racine de $f(x) = x^2 - 4$ sur l'intervalle [1, 3]

1. **Initialisation:** $a = 1, b = 3$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = \frac{1+3}{2} = 2$.
3. **Évaluation:** $f(2) = 2^2 - 4 = 0$.
   - Puisque $f(2) = 0$, 2 est la racine de l'équation.

Exercice 2: Trouver la racine de $f(x) = x^3 - x - 2$ sur l'intervalle [1, 2]

1. **Initialisation:** $a = 1, b = 2$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = \frac{1+2}{2} = 1.5$.
3. **Évaluation:** $f(1.5) = 1.5^3 - 1.5 - 2 = -0.125$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $a = 1.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 3: Trouver la racine de $f(x) = \cos(x) - x$ sur l'intervalle [0, 1]

1. **Initialisation:** $a = 0, b = 1$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 0.5$.
3. **Évaluation:** $f(0.5) = \cos(0.5) - 0.5 \approx 0.377$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 0.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 4: Trouver la racine de $f(x) = x^3 - 2x^2 - 5$ sur l'intervalle [2, 3]

1. **Initialisation:** $a = 2, b = 3$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 2.5$.
3. **Évaluation:** $f(2.5) = 2.5^3 - 2 \times 2.5^2 - 5 = -1.875$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 2.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 5: Trouver la racine de $f(x) = \ln(x) - 1$ sur l'intervalle [2, 3]

1. **Initialisation:** $a = 2, b = 3$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 2.5$.
3. **Évaluation:** $f(2.5) = \ln(2.5) - 1 \approx -0.083$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 2.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 6: Trouver la racine de $f(x) = x^2 - e^x$ sur l'intervalle [0, 1]

1. **Initialisation:** $a = 0, b = 1$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 0.5$.
3. **Évaluation:** $f(0.5) = 0.5^2 - e^{0.5} \approx -0.351$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 0.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 7: Trouver la racine de $f(x) = \tan(x) - x$ sur l'intervalle [4, 5]

1. **Initialisation:** $a = 4, b = 5$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 4.5$.
3. **Évaluation:** $f(4.5) = \tan(4.5) - 4.5 \approx -3.637$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 4.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

Exercice 8: Trouver la racine de $f(x) = e^x - 3x$ sur l'intervalle [0, 1]

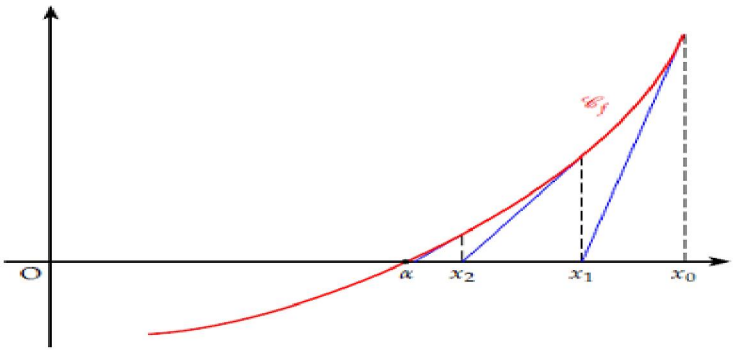
1. **Initialisation:** $a = 0, b = 1$.
2. **Calcul de la valeur moyenne:** $c = \frac{a+b}{2} = 0.5$.
3. **Évaluation:** $f(0.5) = e^{0.5} - 3 \times 0.5 \approx 0.132$.
4. **Nouveau Intervalle:** Puisque $f(a) \times f(c) < 0$, prendre $b = 0.5$.
5. **Répétition:** Continuez de la même manière jusqu'à obtenir une racine avec la précision souhaitée.

2.1.2 La Méthode de Newton-Raphson :

La méthode de Newton-Raphson, aussi appelée méthode de la tangente, est une technique puissante pour trouver des approximations des racines d'équations non linéaires. Cette méthode s'appuie sur la formule suivante :

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

Elle nécessite une estimation initiale proche de la racine réelle et que la fonction soit dérivable à proximité de cette racine. Sa popularité tient à sa rapidité de convergence lorsque les conditions sont idéales.



Exercice 1 : Résolution de $x^2 - 2 = 0$

Nous débutons avec x0=1.5, une estimation initiale proche de la racine carrée de 2. Ce choix illustre bien la capacité de convergence rapide de la méthode de **Newton-Raphson.**

Implémentation en MATLAB

```
f = @(x) x^2 - 2;  % Fonction
df = @(x) 2*x;     % Dérivée de la fonction
x0 = 1.5;          % Valeur initiale
tol = 1e-6;        % Tolérance pour la convergence
max_iter = 100;    % Nombre maximum d'itérations
x = x0;
for i = 1:max_iter
   x_new = x - f(x)/df(x);  % Application de la formule de Newton-Raphson
   if abs(x_new - x) < tol  % Test de la convergence
      break;
   end
   x = x_new;            % Mise à jour de la valeur de x
end
fprintf('La racine trouvée est : %f\n', x);
```

Analyse des résultats

Cette section discute chaque itération pour illustrer comment la valeur de x converge vers la racine réelle $\sqrt{2}$,et comment l'estimation initiale influence la vitesse de convergence.

Exercice 2 : Résolution de cos(x)−x=0

Pour cet exercice, x0 = 0.5 est choisi, basé sur une observation graphique de la fonction suggérant que la racine se trouve près de 0.74.

Implémentation en MATLAB

```
f = @(x) cos(x) - x;   % Fonction
df = @(x) -sin(x) - 1; % Dérivée de la fonction
x0 = 0.5;              % Valeur initiale
x = x0;
for i = 1:max_iter
   x_new = x - f(x)/df(x);
   if abs(x_new - x) < tol
      break;
   end
   x = x_new;
end
fprintf('La racine trouvée est : %f\n', x);
```

Exercice 3 : Résolution de $e^x - 5x + 3 = 0$

x0=0 est sélectionné comme point de départ, permettant d'explorer l'efficacité de Newton-Raphson sur une fonction avec des termes exponentiels.

Implémentation en MATLAB

```
f = @(x) exp(x) - 5*x + 3; % Fonction
df = @(x) exp(x) - 5;      % Dérivée de la fonction
x0 = 0;                    % Valeur initiale
x = x0;
for i = 1:max_iter
   x_new = x - f(x)/df(x);
   if abs(x_new - x) < tol
      break;
   end
   x = x_new;
end
fprintf('La racine trouvée est : %f\n', x);
```

Discussion : Analyse approfondie des itérations, des défis tels que les dérivées proches de zéro, et des techniques pour assurer une convergence stable.

Conclusion : La méthode de Newton-Raphson est démontrée à travers ces exercices comme un outil extrêmement efficace pour trouver des racines d'équations non linéaires, à condition que l'estimation initiale soit appropriée et que la fonction soit bien comportée. Cette discussion souligne l'importance de choisir judicieusement les valeurs initiales et expose les implications pratiques de la méthode dans divers contextes professionnels et académiques. L'encouragement à pratiquer et à explorer d'autres configurations est essentiel pour renforcer la compréhension et la maîtrise de la méthode.

Chapitre I : Rappels sur quelques méthodes numériques

Exercices de travaux dirigés (Newton-Raphson)

Exercice 1: Trouver la racine de $f(x) = x^2 - 2$

1. **Initialisation:** Choisissez $x_0 = 1.5$.
2. **Formule:** $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. **Calcul de la dérivée:** $f'(x) = 2x$.
4. **Première itération:** $x_1 = 1.5 - \frac{1.5^2-2}{2\times1.5} = 1.4167$.
5. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 2: Trouver la racine de $f(x) = x^3 - x - 1$

1. **Initialisation:** Choisissez $x_0 = 1$.
2. **Formule:** $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. **Calcul de la dérivée:** $f'(x) = 3x^2 - 1$.
4. **Première itération:** $x_1 = 1 - \frac{1^3-1-1}{3\times1^2-1} = 1.5$.
5. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 3: Trouver la racine de $f(x) = \cos(x) - x$

1. **Initialisation:** Choisissez $x_0 = 0.5$.
2. **Formule:** $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. **Calcul de la dérivée:** $f'(x) = -\sin(x) - 1$.
4. **Première itération:** $x_1 = 0.5 - \frac{\cos(0.5)-0.5}{-\sin(0.5)-1} \approx 0.736$.
5. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 4: Trouver la racine de $f(x) = e^x - 3x^2$

1. **Initialisation:** Choisissez $x_0 = 1$.
2. **Formule:** $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. **Calcul de la dérivée:** $f'(x) = e^x - 6x$.
4. **Première itération:** $x_1 = 1 - \frac{e^1-3\times1^2}{e^1-6\times1} \approx 0.85$.
5. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 5: Trouver la racine de $f(x) = x^4 - x - 10$

1. Initialisation: Choisissez $x_0 = 2$.
2. Formule: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. Calcul de la dérivée: $f'(x) = 4x^3 - 1$.
4. Première itération: $x_1 = 2 - \frac{2^4-2-10}{4\times 2^3-1} \approx 1.78$.
5. Continuez les itérations jusqu'à obtenir une précision souhaitée.

Exercice 6: Trouver la racine de $f(x) = \ln(x) - 1$

1. Initialisation: Choisissez $x_0 = 2$.
2. Formule: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. Calcul de la dérivée: $f'(x) = \frac{1}{x}$.
4. Première itération: $x_1 = 2 - \frac{\ln(2)-1}{1/2} \approx 1.387$.
5. Continuez les itérations jusqu'à obtenir une précision souhaitée.

Exercice 7: Trouver la racine de $f(x) = x^5 - x^3 + 4$

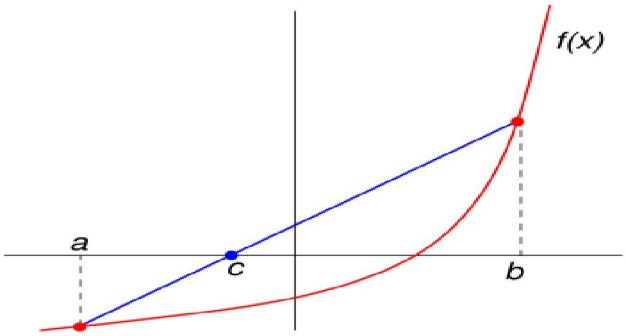
1. Initialisation: Choisissez $x_0 = 1$.
2. Formule: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. Calcul de la dérivée: $f'(x) = 5x^4 - 3x^2$.
4. Première itération: $x_1 = 1 - \frac{1^5-1^3+4}{5\times 1^4-3\times 1^2} \approx 0.636$.
5. Continuez les itérations jusqu'à obtenir une précision souhaitée.

Exercice 8: Trouver la racine de $f(x) = \sin(x) - x/2$

1. Initialisation: Choisissez $x_0 = 1$.
2. Formule: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
3. Calcul de la dérivée: $f'(x) = \cos(x) - 1/2$.
4. Première itération: $x_1 = 1 - \frac{\sin(1)-1/2}{\cos(1)-1/2} \approx 0.86$.
5. Continuez les itérations jusqu'à obtenir une précision souhaitée.

2.1.2 Méthode de la Sécante

La méthode de la sécante est une approximation de Newton-Raphson qui n'exige pas le calcul de la dérivée. Elle utilise deux approximations initiales de la racine pour estimer la pente de la sécante.



Algorithme :

1. Commencer avec deux estimations initiales x_0 et x_1
2. Appliquer :$\boldsymbol{x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}}$
3. Répéter jusqu'à ce que la valeur converge suffisamment.

Exercice 1: Trouver la racine de $f(x) = x^2 - 3$

1. Initialisation: Choisissez $x_0 = 1$ et $x_1 = 2$.
2. Formule: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. Première itération: $x_2 = 2 - \frac{2^2-3}{(2^2-3)-(1^2-3)} \approx 1.5$.
4. Continuez les itérations jusqu'à obtenir une précision souhaitée.

Exercice 2: Trouver la racine de $f(x) = x^3 - 2x + 1$

1. Initialisation: Choisissez $x_0 = -2$ et $x_1 = 0$.
2. Formule: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. Première itération: $x_2 = 0 - \frac{0^3-2\times0+1}{(0^3-2\times0+1)-((-2)^3-2\times-2+1)} \approx -0.2$.
4. Continuez les itérations jusqu'à obtenir une précision souhaitée.

Exercice 3: Trouver la racine de $f(x) = e^x - 2x$

1. Initialisation: Choisissez $x_0 = 0$ et $x_1 = 1$.
2. Formule: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. Première itération: $x_2 = 1 - \frac{e^1-2\times1}{(e^1-2\times1)-(e^0-2\times0)} \approx 0.5$.

4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 4: Trouver la racine de $f(x) = \ln(x) - 2$

1. **Initialisation**: Choisissez $x_0 = 2$ et $x_1 = 3$.
2. **Formule**: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. **Première itération**: $x_2 = 3 - \frac{\ln(3)-2}{(\ln(3)-2)-(\ln(2)-2)} \approx 2.4$.
4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 5: Trouver la racine de $f(x) = x^4 - 16$

1. **Initialisation**: Choisissez $x_0 = 1$ et $x_1 = 3$.
2. **Formule**: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. **Première itération**: $x_2 = 3 - \frac{3^4-16}{(3^4-16)-(1^4-16)} \approx 2.5$.
4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 6: Trouver la racine de $f(x) = \sin(x) - 0.5$

1. **Initialisation**: Choisissez $x_0 = 0$ et $x_1 = 1$.
2. **Formule**: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. **Première itération**: $x_2 = 1 - \frac{\sin(1)-0.5}{(\sin(1)-0.5)-(\sin(0)-0.5)} \approx 0.523$.
4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 7: Trouver la racine de $f(x) = x^2 - e^x$

1. **Initialisation**: Choisissez $x_0 = 0$ et $x_1 = 1$.
2. **Formule**: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. **Première itération**: $x_2 = 1 - \frac{1^2-e^1}{(1^2-e^1)-(0^2-e^0)} \approx 0.75$.
4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

Exercice 8: Trouver la racine de $f(x) = x^3 - 2x + 2$

1. **Initialisation**: Choisissez $x_0 = -1$ et $x_1 = 0$.
2. **Formule**: $x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$.
3. **Première itération**: $x_2 = 0 - \frac{0^3-2\times0+2}{(0^3-2\times0+2)-((-1)^3-2\times-1+2)} \approx -0.5$.
4. **Continuez les itérations** jusqu'à obtenir une précision souhaitée.

En conclusion, Chaque méthode de résolution d'équations non linéaires présente des avantages spécifiques. La bissection est simple et fiable, Newton-Raphson est rapide mais nécessite une bonne valeur initiale et la dérivée de la fonction, tandis que la sécante est flexible mais moins stable. Le choix de la méthode dépend du problème spécifique, des

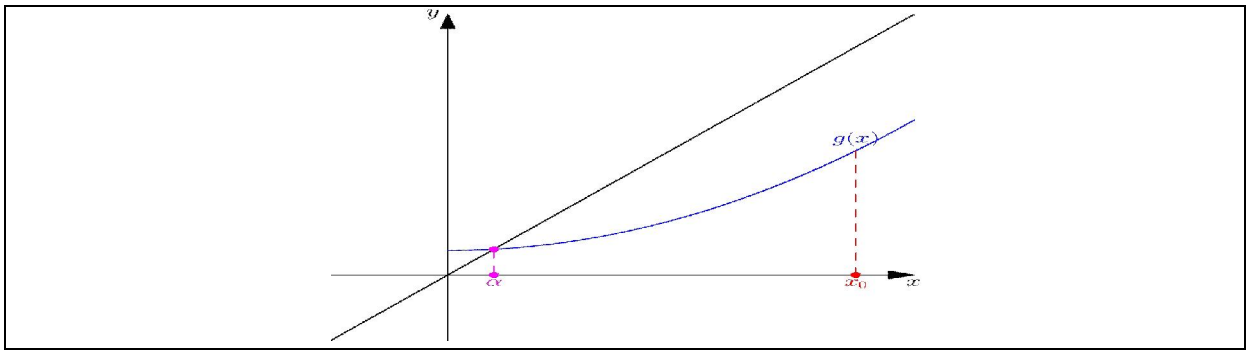
connaissances sur la fonction concernée, et des ressources disponibles pour le calcul des dérivées. Ces méthodes sont fondamentales en analyse numérique et sont largement appliquées dans les recherches scientifiques, l'ingénierie et l'économie pour résoudre des problèmes complexes modélisés par des équations non linéaires

2.1.4 La méthode de point fixe :

Définition Soit g une fonction continue sur$[a, b]$. On appelle point fixe de la fonction g tout point $x\epsilon[a, b]$ vérifiant $g(x) = x$.

- Soit $g: [a, b] \rightarrow [a, b]$ une fonction continue. Alors la fonction $g(x)$ admet au moins un point fixe dans$[a, b]$.
- Pour approcher les racines de $f(x) = 0$ par la méthode du point fixe on cherche donc une fonction g telle que

f (x) = 0 ⇐⇒ g(x) = x



Convergence

Soit g: [a, b] ∈ R une fonction donnée tels que :

a) g est une contraction stricte sur [a, b] Si ġ vérifie$|g'(x)| < 1, \forall x \in [a, b]$

b) g([a, b]) C [a, b]c'est-à-dire $\forall$ x ∈ [a, b], g(x) ∈ [a, b]

Alors

1. La fonction g(x) admet un unique point fixe x_ dans [a, b].
2. Pour tout $x_0 \in [a, b]$,la suite $(x_n)_{n\in N}$ définie par : $x_{n+1} = g(x_n)$,
converge vers x*lorsque n → ∞.

Exercice 1 : Trouver les solutions de l'équation non linéaire suivante en utilisant la méthode des points fixes : $x - \cos(x) = 0$

Solution avec Matlab :

Étape 1 : Réécriture en Forme de Point Fixe

La fonction équivalente $g(x) = x = \cos(x)$

Étape 2 : Initialisation et Paramètres de l'Algorithme

On choisit un point initial et on définit les tolérances.

```
 Programme matlab
% Point initial
x = 0.5;
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;
% Définir la fonction g(x)
g = @(x) cos(x);
```

Étape 3 : Implémentation de la Méthode des Points Fixes

On itère pour mettre à jour la valeur de x à chaque étape.

```
% Implémenter la méthode des points fixes
while abs(g(x) - x) > tol && iter < max_iter
   x = g(x); % Mettre à jour x
   iter = iter + 1; % Incrémenter le compteur d'itérations
end
% Afficher le résultat
disp(['La solution est : ', num2str(x)]);
```

Étape 4 : Explication Pas à Pas

1. **Initialisation** : On commence avec x=0.5
2. **Itération** : La nouvelle valeur de x est calculée comme x_{K+1}=cos(x_k)
3. **Critère d'Arrêt** : La boucle continue jusqu'à ce que la différence entre $|x_{K+1} - x_k|$ soit inférieure à la tolérance.

Exercice 2: Résoudre l'équation suivante en utilisant la méthode des points fixes :

$$x = \frac{1}{2} ln(4 - x^2)$$

Solution avec Matlab :

Étape 1 : Réécriture en Forme de Point Fixe $x = g(x) = \frac{1}{2} ln(4 - x^2)$

Étape 2 : Initialisation et Paramètres de l'Algorithme

On choisit un point initial et on définit les tolérances.

```
% Point initial
x = 0.5;
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;
% Définir la fonction g(x)
g = @(x) 0.5 * log(4 - x^2);
```

Étape 3 : Implémentation de la Méthode des Points Fixes

On itère pour mettre à jour la valeur de x à chaque étape.

```
% Implémenter la méthode des points fixes
while abs(g(x) - x) > tol && iter < max_iter
    x = g(x); % Mettre à jour x
    iter = iter + 1; % Incrémenter le compteur d'itérations
end
% Afficher le résultat
disp(['La solution est : ', num2str(x)]);
```

Étape 4 : Explication Pas à Pas

1. **Initialisation** : On commence avec x=0.5
2. **Itération** : La nouvelle valeur de x est calculée comme $x_{k+1} = \frac{1}{2} ln(4 - x^2{}_k)$.
3. **Critère d'Arrêt** : La boucle continue jusqu'à ce que la différence entre $|x_{K+1} - x_k|$ soit inférieure à la tolérance.

2.2 Résolution des systèmes des équations non linéaires

Nous pouvons adapter la méthode de point fixe utilisé pour la résolution d'une équation non linéaire a un système des équations non linéaire

$$\boldsymbol{F(X) = 0}$$

$$\begin{cases} f_1(x_1, x_2, x_3, \cdots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \cdots, x_n) = 0 \end{cases}$$

2.2.1Point fixe

Par extraction d'une seul variable 'une équation de facon à obtenir les schéma suivante :

$$\begin{cases} x_1 = g_1(x_1, x_2, \dots, x_n) \\ x_2 = g_2(x_1, x_2, \dots, x_n) \\ \vdots \\ x_n = g_n(x_1, x_2, \dots, x_n) \end{cases}$$

En suite on passe au schéma de récurrence suivante :$X^{k+1} = G(X^k)$

$$\begin{cases} x_1^{k+1} = g_1({x_1}^k, {x_2}^k, \dots, {x_n}^k) \\ x_1^{k+1} = g_1({x_1}^k, {x_2}^k, \dots, {x_n}^k) \\ \vdots \\ x_n^{k+1} = g_n({x_1}^k, {x_2}^k, \dots, {x_n}^k) \end{cases}$$

Avec $X^0 = \begin{pmatrix} x_1^0 \\ x_2^0 \\ \vdots \\ x_n^0 \end{pmatrix}$ Connu ou donnée

Pou améliore la convergence il est possile d'obtenir une autre configuation basé sur l'utilisation des nouvelles valeurs de $x_j$dans le cas ou $j > i$ c-à-d :

$$X^{k+1} = G(X^k, X^{k+1})$$

$$\begin{cases} x_1^{k+1} = g_1({x_1}^k, {x_2}^k, \dots, {x_n}^k) \\ x_1^{k+1} = g_1({x_1}^{k+1}, {x_2}^k, \dots, {x_n}^k) \\ \vdots \\ x_n^{k+1} = g_n({x_1}^{k+1}, {x_2}^{k+1}, \dots, {x_{n-1}}^k, {x_n}^k) \end{cases}$$

Critère de convergence

Dans le cas $n > 1$ La condition de convergence est étendue au système de condition suivante :

$$\begin{cases} \left|\frac{\partial g_1}{\partial x_1}\right|_{X^k} + \left|\frac{\partial g_2}{\partial x_1}\right|_{X^k} + \left|\frac{\partial g_3}{\partial x_1}\right|_{X^k} + \dots + \left|\frac{\partial g_n}{\partial x_1}\right|_{X^k} \; et \\ \left|\frac{\partial g_1}{\partial x_2}\right|_{X^k} + \left|\frac{\partial g_2}{\partial x_2}\right|_{X^k} + \left|\frac{\partial g_3}{\partial x_2}\right|_{X^k} + \dots + \left|\frac{\partial g_n}{\partial x_2}\right|_{X^k} \; et \\ \vdots \\ \left|\frac{\partial g_1}{\partial x_n}\right|_{X^k} + \left|\frac{\partial g_2}{\partial x_n}\right|_{X^k} + \left|\frac{\partial g_3}{\partial x_n}\right|_{X^k} + \dots + \left|\frac{\partial g_n}{\partial x_n}\right|_{X^k} \; et \end{cases}$$

Exercice 1 : Système Non Linéaire à Deux Variables

Problème : Trouver les solutions du système d'équations non linéaires suivant :

$$\begin{cases} x_1 - \cos(x_2) = 0 \\ x_2 - \sin(x_1) = 0 \end{cases} \qquad X^0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Solution avec Matlab :

Étape 1 : Réécriture en Forme de Point Fixe

Le système est sous la forme $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} g_1(x_1, x_2) \\ g_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} \cos(x_2) \\ \sin(x_1) \end{bmatrix}$

Étape 2 : Initialisation et Paramètres de l'Algorithme

On choisit un point initial et on définit les tolérances.

```
% Point initial
x = [0.5; 0.5];
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;
% Définir la fonction g(x)
g = @(x) [cos(x(2)); sin(x(1))];
```

Étape 3 : Implémentation de la Méthode des Points Fixes

On itère pour mettre à jour les valeurs de x à chaque étape.

```
% Implémenter la méthode des points fixes
while norm (g(x) - x) > tol && iter < max_iter
   x = g(x); % Mettre à jour x
   iter = iter + 1; % Incrémenter le compteur d'itérations
end
% Afficher le résultat
disp(['La solution est : (', num2str(x(1)), ', ', num2str(x(2)), ')']);
```

Exercice 2 : Système Non Linéaire à Trois Variables Complexe

Problème : Trouver les solutions du système non linéaire suivant :

$$\begin{cases} x_1 = \cos(x_2 x_3) \\ x_2 = \sin(x_1 x_3) \\ x_3 = (x_2 x_1) \end{cases}$$

Solution avec Matlab :

Étape 1 : Réécriture en Forme de Point Fixe

Le système est réécrit comme : $X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} g_1(x_1, x_2x_3) \\ g_2(x_1, x_2x_3) \\ g_3(x_1, x_2x_3) \end{bmatrix} = \begin{bmatrix} \cos(x_2x_3) \\ \sin(x_1x_3) \\ x_1x_2 \end{bmatrix}$

Étape 2 : Initialisation et Paramètres de l'Algorithme

On choisit un point initial et on définit les tolérances.

```
% Point initial
x = [0.5; 0.5; 0.5];
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;
% Définir la fonction g(x)
g = @(x) [cos(x(2)* x(3)); sin(x(1) * x(3)); x(1) * x(2)];
```

Étape 3 : Implémentation de la Méthode des Points Fixes

On itère pour mettre à jour les valeurs de x à chaque étape.

```
% Implémenter la méthode des points fixes
while norm(g(x) - x) > tol && iter < max_iter
   x = g(x); % Mettre à jour x
   iter = iter + 1; % Incrémenter le compteur d'itérations
end
% Afficher le résultat
disp(['La solution est : (', num2str(x(1)), ', ', num2str(x(2)), ', ', num2str(x(3)), ')']);
```

Étape 4 : Explication Pas à Pas

1. **Initialisation** : Le point initial est choisi comme (0.5,0.5,0.5)
2. **Itération** : Chaque nouvelle valeur de x1, x2, et x3 est calculée en utilisant les fonctions logarithmiques, trigonométriques et algébriques.
3. **Critère d'Arrêt** : L'algorithme continue jusqu'à ce que la différence entre les itérations successives soit inférieure à la tolérance.

Analyse et Discussion

Ces exercices démontrent comment la méthode des points fixes peut être appliquée pour résoudre des systèmes non linéaires à deux et trois variables. L'utilisation de Matlab permet une mise en œuvre simple et efficace, facilitant l'itération sur les points fixes et l'analyse des résultats.

2.2.2 Newton Raphson (système des équation non linéaire)

La méthode de Newton pour la résolution d'un système d'équations est une généralisation de l'algorithme de Newton pour la recherche du Zéro d'une fonction unidimensionnelle. Le fait qu'un algorithme pour un problème unidimensionnel puisse être efficacement généralisé au cas multidimensionnel est une situation exceptionnelle .La méthode de Newton est souvent aussi appelée méthode de Newton -Raphson.

Soit les systèmes d'équation NL suivante :

$$\boldsymbol{F(X) = 0}$$

$$\begin{cases} \boldsymbol{f_1(x_1, x_2, x_3, \cdots, x_n) = 0} \\ \boldsymbol{f_2(x_1, x_2, x_3, \cdots, x_n) = 0} \\ \vdots \\ \boldsymbol{f_n(x_1, x_2, x_3, \cdots, x_n) = 0} \end{cases}$$

On peut étendre la méthode de newton au cas vectoriel

$$\begin{cases} \textbf{Pose } \boldsymbol{X^{k+1} = X^k + \delta X^k} \\ \textbf{Résoudre } \boldsymbol{\delta X^k = -\left[J_F(X^k)\right]^{-1} F(X^k)} \end{cases}$$

Avec $J_F(X^k) = \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \cdots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \cdots & \partial f_2/\partial x_n \\ \vdots & \vdots & \vdots \\ \partial f_n/\partial x_1 & \partial f_n/\partial x_2 \cdots & \partial f_n/\partial x_n \end{bmatrix}$ La matrice Jacobiennec

Avec X^k étant donnée.

Algorithme de la Méthode de newton-Raphson

1. **Initialisation** : Choisir un point initial $\mathbf{x}_0$.
2. **Calcul de la Fonction et de la Jacobienne** : Calculer le vecteur de fonctions $\mathbf{F}(\mathbf{x}_k)$ et la matrice Jacobienne $\mathbf{J}(\mathbf{x}_k)$.
3. **Mise à Jour du Point** : Mettre à jour le point suivant en utilisant :

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k)$$

4. **Critère d'Achat** : Répéter jusqu'à ce que $\|\mathbf{x}_{k+1} - \mathbf{x}_k\|$ soit inférieur à une tolérance prédéfinie ou que $\|\mathbf{F}(\mathbf{x}_{k+1})\|$ soit proche de zéro.

Exemples Résolus avec Matlab

Exercice 1 : Système Non Linéaire Couplé Simple

Problème : Résoudre le système d'équations non linéaires :

$$\begin{cases} x_1^2 + x_2^2 - 4 = 0 \\ x_1 x_2 - 1 = 0 \end{cases}$$

Solution avec Matlab :

```
% Définir la fonction et sa Jacobienne
F = @(x) [x(1)^2 + x(2)^2 - 4; x(1)*x(2) - 1];
J = @(x) [2*x(1), 2*x(2); x(2), x(1)];
% Point initial
x = [1.5; 1.5];
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;
% Implémenter la méthode de Newton-Raphson
while norm(F(x)) > tol && iter < max_iter
    x = x - J(x)\F(x);
    iter = iter + 1;
end
```

```
% Résultat

disp(['La solution est : (', num2str(x(1)), ', ', num2str(x(2)), ')']);
```

Explication Pas à Pas :

1. **Définition des fonctions et de la Jacobienne** : Les fonctions f1 et f2 sont définies, ainsi que leurs dérivées partielles pour former la matrice Jacobienne.
2. **Point Initial** : Le point initial choisi est (1.5,1.5).
3. **Mise à Jour du Point** : À chaque itération, le point est mis à jour en utilisant la formule de Newton-Raphson.
4. **Critère d'Arrêt** : L'algorithme continue jusqu'à ce que le résidu soit inférieur à la tolérance.

Exercice 2 : Système de Deux Équations Non Linéaires

Problème : Trouver les solutions du système :

$$\begin{cases} e^{x_1} - x_2^2 = 0 \\ x_1^2 + x_2^2 - 1 = 0 \end{cases}$$

Solution avec Matlab :

```
% Définir la fonction et sa Jacobienne
F = @(x) [exp(x(1)) - x(2)^2; x(1)^2 + x(2)^2 - 1];
J = @(x) [exp(x(1)), -2*x(2); 2*x(1), 2*x(2)];
% Point initial
x = [0.5; 0.5];
% Paramètres de l'algorithme
tol = 1e-6;
max_iter = 100;
iter = 0;

% Implémenter la méthode de Newton-Raphson
while norm(F(x)) > tol && iter < max_iter
    x = x - J(x)\F(x);
    iter = iter + 1;
end
% Résultat
disp(['La solution est : (', num2str(x(1)), ', ', num2str(x(2)), ')']);
```

Explication Pas à Pas :

1. **Définition des fonctions** : Les équations données sont converties en fonctions Matlab, et la matrice Jacobienne est calculée.

2. **Point Initial et Itération** : Un point initial est choisi, et les itérations sont effectuées en mettant à jour xxx en utilisant l'inverse de la Jacobienne.
3. **Critère de Convergence** : L'algorithme vérifie si les conditions de convergence sont satisfaites à chaque itération.

3. Intégrations numériques : Méthode des Trapèzes et Méthode de Simpson

Parmi les nombreuses techniques numériques disponibles pour traiter ces problèmes d'intégrations numériques, la méthode des trapèzes et la méthode de Simpson sont particulièrement utiles pour estimer les intégrales définies à partir de points d'échantillonnage discrets. Cet essai examine ces méthodes en détail, présente leurs principes fondamentaux et illustre leur application à travers plusieurs exemples.

L'intégration est une opération mathématique fondamentale permettant de calculer la surface sous une courbe définie par une fonction continue. Cependant, il est souvent difficile, voire impossible, de trouver des solutions exactes pour les intégrales de fonctions non linéaires complexes. Les méthodes numériques fournissent une approche pour estimer ces intégrales à l'aide de points d'échantillonnage discrets.

Applications des Méthodes Numériques d'Intégration

Les méthodes numériques d'intégration, comme celles des trapèzes et de Simpson, sont utilisées dans divers domaines :

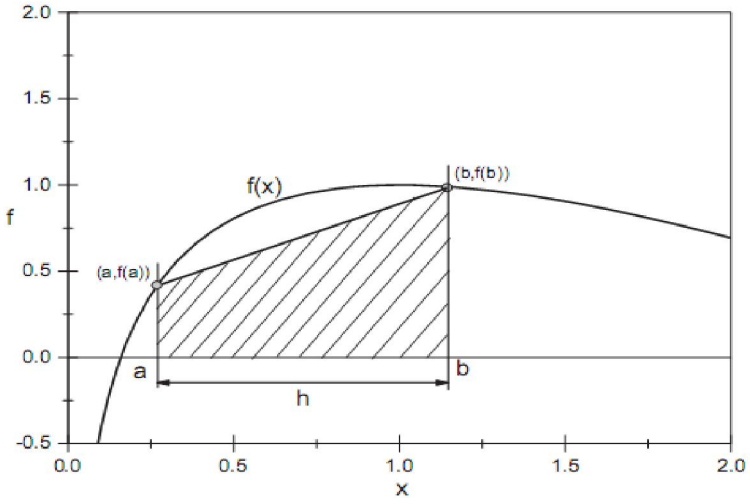
1. **Physique** : Calcul des déplacements, des travaux, et des énergies potentielles où les forces varient de manière non linéaire.
2. **Ingénierie** : Évaluation des charges, des tensions, et des courants dans des systèmes complexes.
3. **Économie** : Calcul des profits, des coûts, et des utilités où les relations ne sont pas linéaires.
4. **Biologie** : Modélisation de la croissance des populations et de la diffusion de substances chimiques.

3.1. Méthode des Trapèzes

Principe de la Méthode des Trapèzes

La méthode des trapèzes est une technique d'intégration numérique simple qui approxime l'aire sous une courbe en divisant la région sous la courbe en une série de trapèzes. Chaque trapèze est formé en reliant deux points successifs sur la courbe par une ligne droite.

Chapitre I : Rappels sur quelques méthodes numériques



Pour une fonction continue $f(x)$ sur un intervalle $[a, b]$, l'intégrale peut être approximée par :

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2} \Delta x$$

où $\Delta x = \frac{b-a}{n}$ est la largeur de chaque sous-intervalle et $x_0, x_1, \ldots, x_n$ sont les points d'échantillonnage.

Exemple 1 : Intégration d'une Fonction Quadratique

Intégrons la fonction quadratique simple $f(x) = x^2$ sur l'intervalle $[0, 1]$ en utilisant la méthode des trapèzes.

Étape 1 : Discrétisation de l'intervalle

Supposons que nous utilisons $n = 4$ sous-intervalles. Alors, $\Delta x = \frac{1-0}{4} = 0.25$.

Les points d'échantillonnage sont : $x_0 = 0, x_1 = 0.25, x_2 = 0.5, x_3 = 0.75, x_4 = 1$.

Étape 2 : Application de la formule des trapèzes

Calculons les valeurs de la fonction en ces points :

$$f(0) = 0^2 = 0, \quad f(0.25) = 0.25^2 = 0.0625, \quad f(0.5) = 0.5^2 = 0.25, \quad f(0.75) = 0.75^2 = 0.5625, \quad f(1) = 1^2 = 1$$

Appliquons la formule :

$$\int_0^1 x^2\,dx \approx \frac{0 + 0.0625}{2} \times 0.25 + \frac{0.0625 + 0.25}{2} \times 0.25 + \frac{0.25 + 0.5625}{2} \times 0.25 + \frac{0.5625 + 1}{2} \times 0.25$$

$$= 0.0078125 + 0.0390625 + 0.1015625 + 0.1953125 = 0.34375$$

Programme MATLAB :

```
% Paramètres
a = 0; % borne inférieure
b = 1; % borne supérieure
```

```
n = 4; % nombre de sous-intervalles
dx = (b - a) / n;
% Points d'échantillonnage
x = a:dx:b;
f = x.^2;
% Application de la méthode des trapèzes
integral_approx = 0;
for i = 1:n
   integral_approx = integral_approx + (f(i) + f(i+1)) * dx / 2;
end
fprintf('Approximation de l''intégrale par la méthode des trapèzes : %f\n', integral_approx);
```

Résultat attendu :

L'approximation de l'intégrale par la méthode des trapèzes est 0.34375, ce qui est proche de la valeur exacte $\frac{1}{3} \approx 0.3333$.

Exemple 2 : Intégration d'une Fonction Exponentielle

Considérons l'intégrale de $f(x) = e^x$ sur l'intervalle $[0, 1]$ en utilisant $n = 4$ sous-intervalles.

Étape 1 : Discrétisation de l'intervalle

Les points d'échantillonnage sont : $x_0 = 0, x_1 = 0.25, x_2 = 0.5, x_3 = 0.75, x_4 = 1$.

Étape 2 : Application de la formule des trapèzes

Calculons les valeurs de la fonction en ces points :

$$f(0) = 1, \quad f(0.25) = e^{0.25} \approx 1.284, \quad f(0.5) = e^{0.5} \approx 1.649, \quad f(0.75) = e^{0.75} \approx 2.117, \quad f(1) = e^1 \approx 2.718$$

Appliquons la formule :

$$\int_0^1 e^x \, dx \approx \frac{1 + 1.284}{2} \times 0.25 + \frac{1.284 + 1.649}{2} \times 0.25 + \frac{1.649 + 2.117}{2} \times 0.25 + \frac{2.117 + 2.718}{2} \times 0.25$$

Programme MATLAB :

```
% Paramètres
a = 0; % borne inférieure
b = 1; % borne supérieure
n = 4; % nombre de sous-intervalles
dx = (b - a) / n;
% Points d'échantillonnage
x = a:dx:b;
f = exp(x);
```

```
% Application de la méthode des trapèzes
integral_approx = 0;
for i = 1:n
    integral_approx = integral_approx + (f(i) + f(i+1)) * dx / 2;
end
fprintf('Approximation de l''intégrale par la méthode des trapèzes : %f\n',
integral_approx);
```

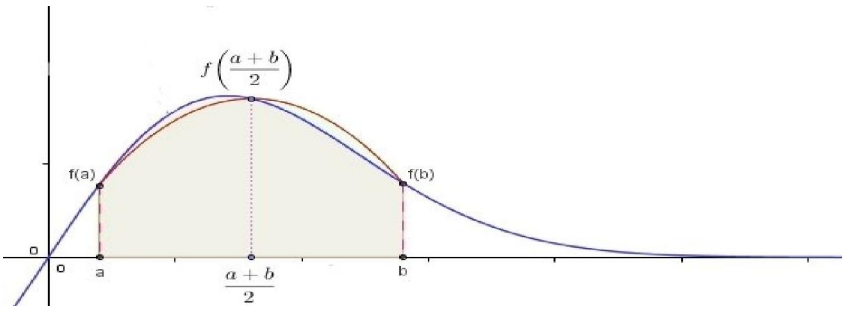
Résultat attendu :

L'approximation de l'intégrale par la méthode des trapèzes est environ 1.718, qui est proche de la valeur exacte $e - 1 \approx 1.7183$.

3.2. Méthode de Simpson

Principe de la Méthode de Simpson

La méthode de Simpson offre une approximation plus précise que la méthode des trapèzes en utilisant des paraboles pour approximer les segments de la courbe. Il existe deux variantes principales de la méthode de Simpson : Simpson 1/3 et Simpson 3/8. La méthode Simpson 1/3 est plus couramment utilisée et s'applique sur des intervalles pairs.



Pour une fonction continue $f(x)$ sur un intervalle $[a, b]$ avec n sous-intervalles (où n est pair), l'intégrale est approximée par :

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left[f(x_0) + 4\sum_{i\ \text{impairs}} f(x_i) + 2\sum_{i\ \text{pairs}} f(x_i) + f(x_n)\right]$$

Exemple 3 : Intégration d'une Fonction Quadratique

Réutilisons l'intégration de $f(x) = x^2$ sur $[0, 1]$ mais cette fois en utilisant la méthode de Simpson avec $n = 4$.

Étape 1 : Discrétisation de l'intervalle

$\Delta x = 0.25$, points : $x_0 = 0, x_1 = 0.25, x_2 = 0.5, x_3 = 0.75, x_4 = 1$.

Étape 2 : Application de la formule de Simpson

$$\int_0^1 x^2\,dx \approx \frac{0.25}{3}[0 + 4 \times (0.0625 + 0.5625) + 2 \times 0.25 + 1]$$

$$= \frac{0.25}{3}[0 + 2.5 + 0.5 + 1] = 0.3333$$

Programme MATLAB :

```
% Paramètres
a = 0; % borne inférieure
b = 1; % borne supérieure
n = 4; % nombre de sous-intervalles (doit être pair)
dx = (b - a) / n;
% Points d'échantillonnage
x = a:dx:b;
f = x.^2;
% Application de la méthode de Simpson
integral_approx = f(1) + f(end);
for i = 2:2:n
   integral_approx = integral_approx + 4 * f(i);
end
for i = 3:2:n-1
   integral_approx = integral_approx + 2 * f(i);
end
integral_approx = integral_approx * dx / 3;
fprintf('Approximation de l''intégrale par la méthode de Simpson : %f\n',
integral_approx);
```

Exemple 4 : Intégration d'une Fonction Exponentielle

Considérons l'intégration de $f(x) = e^x$ sur $[0, 1]$ avec $n = 4$ sous-intervalles.

Étape 1 : Discrétisation de l'intervalle

Les points d'échantillonnage sont : $x_0 = 0, x_1 = 0.25, x_2 = 0.5, x_3 = 0.75, x_4 = 1$.

Étape 2 : Application de la formule de Simpson

Calculons les valeurs de la fonction en ces points : $f(0) = 1, f(0.25) = 1.284, f(0.5) = 1.649, f(0.75) = 2.117, f(1) = 2.718$.

$$\int_0^1 e^x\,dx \approx \frac{0.25}{3}\,[1 + 4 \times (1.284 + 2.117) + 2 \times 1.649 + 2.718]$$

Programme Matlab

```
% Paramètres
a = 0; % borne inférieure
b = 1; % borne supérieure
n = 4; % nombre de sous-intervalles (doit être pair)
dx = (b - a) / n;
% Points d'échantillonnage
x = a:dx:b;
```

```
f = exp(x);
% Application de la méthode de Simpson
integral_approx = f(1) + f(end);
for i = 2:2:n
   integral_approx = integral_approx + 4 * f(i);
end
for i = 3:2:n-1
   integral_approx = integral_approx + 2 * f(i);
end
integral_approx = integral_approx * dx / 3;
fprintf('Approximation de l''intégrale par la méthode de Simpson : %f\n',
integral_approx);
```

Résultat attendu :

L'approximation de l'intégrale par la méthode de Simpson est environ 1.718, proche de la valeur exacte $e - 1 \approx 1.7183$.

V.4. Comparaison des Méthodes des Trapèzes et de Simpson

- **Précision** : La méthode de Simpson est généralement plus précise que la méthode des trapèzes pour le même nombre de sous-intervalles, car elle utilise des paraboles au lieu de lignes droites.
- **Complexité** : La méthode de Simpson nécessite un nombre pair de sous-intervalles et est légèrement plus complexe à implémenter, mais elle offre une meilleure approximation pour des fonctions non linéaires.
- **Applications** : Les deux méthodes sont couramment utilisées en pratique. La méthode des trapèzes est utilisée pour des approximations rapides, tandis que la méthode de Simpson est préférée pour une meilleure précision.

Conclusion :

Les méthodes des trapèzes et de Simpson sont des techniques puissantes pour l'estimation numérique des intégrales définies, particulièrement utiles pour les fonctions non linéaires complexes. Ces méthodes offrent des solutions pratiques dans divers domaines, des sciences physiques à l'économie, et sont des outils essentiels dans la boîte à outils des analystes et ingénieurs. Les exemples et les programmes MATLAB fournis démontrent comment appliquer ces méthodes pour obtenir des approximations précises des intégrales de fonctions, renforçant ainsi la compréhension et l'application de ces techniques numériques.

4. La différentiation numérique

Chapitre I : Rappels sur quelques méthodes numériques

La différentiation numérique est une technique essentielle utilisée pour approximer la dérivée d'une fonction en utilisant des valeurs discrètes. Elle est particulièrement utile lorsque les dérivées analytiques ne sont pas facilement calculables ou lorsque seules des données discrètes sont disponibles. Voici une présentation approfondie de cette méthode, suivie d'exercices détaillés pour enrichir les compétences analytiques des étudiants et leur capacité à résoudre des problèmes non linéaires.

II. Compréhension de la Différentiation Numérique

1. **Concept Fondamental** : La différentiation numérique consiste à estimer la dérivée d'une fonction en utilisant les valeurs de la fonction à des points spécifiques. La dérivée première d'une fonction $f(x)$ est approximée à l'aide de différences finies.
2. **Formules de Différences Finies** :
   - **Différence Finie Avant (Forward Difference)** :

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

   - **Différence Finie Arrière (Backward Difference)** :

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

   - **Différence Finie Centrale (Central Difference)** :

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

   Ici, h est un petit incrément appelé pas de discrétisation.
3. **Ordre d'Approximation** : La précision de l'approximation dépend du choix de h. En général, une valeur plus petite de h offre une approximation plus précise, mais des valeurs trop petites peuvent entraîner des erreurs dues à l'arrondi numérique.
4. **Différentiation de Second Ordre** : Pour approximer la dérivée seconde, on utilise :

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

)

Application

Exercice 1 : Approximation de la Dérivée Première

Chapitre I : Rappels sur quelques méthodes numériques

Problème : Approximer la dérivée première de la fonction f(x)=sin (x) en x=π4x = \frac{\pi}{4}x=4π en utilisant la différence finie centrale avec h=0.1h = 0

Exercice 1 : Approximation de la Dérivée Première

Problème : Approximer la dérivée première de la fonction $f(x) = \sin(x)$ en $x = \frac{\pi}{4}$ en utilisant la différence finie centrale avec $h = 0.1$.

Solution Pas à Pas :

1. **Calculer les valeurs de la fonction :**

$$f(\frac{\pi}{4} + 0.1) = \sin(\frac{\pi}{4} + 0.1)$$

2. **Appliquer la formule de différence finie centrale :**

$$f'(\frac{\pi}{4}) \approx \frac{\sin(\frac{\pi}{4} + 0.1) - \sin(\frac{\pi}{4} - 0.1)}{2 \times 0.1}$$

3. **Calculer les résultats** : En utilisant une calculatrice ou un logiciel, on trouve :

$$\sin(\frac{\pi}{4} + 0.1) \approx 0.789$$

$$\sin(\frac{\pi}{4} - 0.1) \approx 0.697$$

$$f'(\frac{\pi}{4}) \approx \frac{0.789 - 0.697}{0.2} = 0.46$$

Exercice 2 : Approximation de la Dérivée Seconde

Problème : Approximer la dérivée seconde de la fonction $f(x) = e^x$ en $x = 1$ en utilisant la différence finie centrale avec $h = 0.01$.

Solution Pas à Pas :

1. **Calculer les valeurs de la fonction :**

$$f(1 + 0.01) = e^{1.01}$$

$$f(1 - 0.01) = e^{0.99}$$

$$f(1) = e^1$$

2. **Appliquer la formule de différence finie pour la dérivée seconde :**

$$f''(1) \approx \frac{e^{1.01} - 2e^1 + e^{0.99}}{(0.01)^2}$$

3. **Calculer les résultats** : En utilisant une calculatrice ou un logiciel :

$$e^{1.01} \approx 2.745$$

$$e^{0.99} \approx 2.691$$

$$e^{1} \approx 2.718$$

$$f''(1) \approx \frac{2.745 - 2 \times 2.718 + 2.691}{0.0001} \approx 2.72$$

En conclusionn , la différentiation numérique est un outil puissant pour approximer les dérivées de fonctions lorsque des solutions analytiques ne sont pas possibles ou pratiques. Les exercices ci-dessus montrent comment appliquer des formules de différences finies pour approximer les dérivées première et seconde. En pratiquant ces techniques, les étudiants peuvent améliorer leur compréhension des concepts fondamentaux et développer des compétences analytiques solides pour résoudre des problèmes non linéaires dans leurs domaines respectifs.

Pour fournir une compréhension approfondie de la différentiation numérique et enrichir les compétences analytiques des étudiants, nous allons aborder cinq exercices résolus en utilisant Matlab. Ces exercices sont conçus pour démontrer comment les techniques de différentiation numérique peuvent être appliquées pour résoudre des problèmes non linéaires dans divers domaines.

Application de la Différentiation Numérique

La différentiation numérique est une méthode pour estimer les dérivées d'une fonction à partir de valeurs discrètes. C'est une technique importante dans le calcul numérique, surtout lorsqu'une expression analytique de la dérivée n'est pas disponible. Les techniques courantes incluent les différences finies avant, arrière, et centrale.

Exercice 1 : Approximation de la Dérivée Première

Problème : Approximer la dérivée première de la fonction $f(x) = \sin(x)$ en $x = \frac{\pi}{4}$ en utilisant la différence finie centrale.

Solution avec Matlab :

```
% Définir les paramètres
h = 0.01;
x = pi/4;
% Définir la fonction
f = @(x) sin(x);
    % Calculer la dérivée approximative
```

```
f_prime_approx = (f(x + h) - f(x - h)) / (2 * h);
% Afficher le résultat
disp(['La dérivée approximée de sin(x) en x = pi/4 est : ', num2str(f_prime_approx)]);
```

.

Exercice 2 : Approximation de la Dérivée Seconde

Problème : Approximer la dérivée seconde de la fonction $f(x) = e^x$ en $x = 1$ en utilisant la différence finie centrale.

Solution avec Matlab :

```
% Définir les paramètres
h = 0.01;
x = 1;
% Définir la fonction
f = @(x) exp(x);
% Calculer la dérivée seconde approximative
f_double_prime_approx = (f(x + h) - 2 * f(x) + f(x - h)) / (h^2);
% Afficher le résultat
disp(['La dérivée seconde approximée de e^x en x = 1 est : ',
num2str(f_double_prime_approx)]);
```

Exercice 3 : Estimation de la Dérivée Première pour des Données Discrètes

Problème : Données $x = [1, 2, 3, 4, 5]$ et $y = [2.7, 5.8, 6.6, 7.5, 8.9]$. Estimer la dérivée première en $x = 3$ en utilisant la différence finie centrale.

Solution avec Matlab :

```
% Définir les données
x = [1, 2, 3, 4, 5];
y = [2.7, 5.8, 6.6, 7.5, 8.9];
% Trouver l'indice pour x = 3
idx = find(x == 3);
% Utiliser la différence finie centrale
dy_dx = (y(idx+1) - y(idx-1)) / (x(idx+1) - x(idx-1));
% Afficher le résultat
disp(['La dérivée approximée en x = 3 est : ', num2str(dy_dx)]);
```

.

Exercice 4 : Approximation de la Dérivée Première pour une Fonction Non Linéaire

Problème : Approximer la dérivée première de $f(x) = \ln(x)$ en $x = 2$ en utilisant la différence finie avant avec $h = 0.05$.

Solution avec Matlab :

```
% Définir les paramètres
h = 0.05;
x = 2;
% Définir la fonction
f = @(x) log(x);
% Calculer la dérivée approximative
f_prime_approx = (f(x + h) - f(x)) / h;
% Afficher le résultat
disp(['La dérivée approximée de ln(x) en x = 2 est : ', num2str(f_prime_approx)]);
```

Exercice 5 : Approximation de la Dérivée Seconde pour une Fonction Sinusoïdale

Problème : Approximer la dérivée seconde de $f(x) = \cos(x)$ en $x = 0$ en utilisant la différence finie centrale avec $h = 0.01$.

Solution avec Matlab :

```
% Définir les paramètres
h = 0.01;
x = 0;
% Définir la fonction
f = @(x) cos(x);
% Calculer la dérivée seconde approximative
f_double_prime_approx = (f(x + h) - 2 * f(x) + f(x - h)) / (h^2);
% Afficher le résultat
disp(['La dérivée seconde approximée de cos(x) en x = 0 est : ', num2str(f_double_prime_approx)]);
```

Conclusion

Ces exercices montrent comment la différentiation numérique peut être appliquée à différentes fonctions, y compris des fonctions sinusoïdales, exponentielles et logarithmiques, ainsi que des données discrètes. L'utilisation de Matlab facilite la mise en œuvre et la visualisation de ces concepts, aidant ainsi les étudiants à développer une compréhension pratique des méthodes numériques pour résoudre des problèmes non linéaires.

Chapitre I : Rappels sur quelques méthodes numériques

Travaux dirigés I

Méthode des Trapèzes

The Trapezoidal Rule approximates the integral of a function by dividing the area under the curve into trapezoids rather than rectangles.

Exercice 1: Approximer $\int_0^1 x^2\,dx$ avec 2 sous-intervalles

1. **Intervalle de l'intégration:** $[0, 1]$.
2. **Nombre de sous-intervalles:** $n = 2$.
3. **Largeur des sous-intervalles:** $h = \frac{b-a}{n} = \frac{1-0}{2} = 0.5$.
4. **Points d'évaluation:** $x_0 = 0, x_1 = 0.5, x_2 = 1$.
5. **Valeurs de la fonction:** $f(x_0) = 0^2 = 0, f(x_1) = 0.5^2 = 0.25, f(x_2) = 1^2 = 1$.
6. **Approximation par la méthode des trapèzes:**

$$I \approx \frac{h}{2}[f(x_0) + 2f(x_1) + f(x_2)] = \frac{0.5}{2}[0 + 2 \times 0.25 + 1] = 0.375$$

Exercice 2: Approximer $\int_0^\pi \sin(x)\,dx$ avec 4 sous-intervalles

1. **Intervalle de l'intégration:** $[0, \pi]$.
2. **Nombre de sous-intervalles:** $n = 4$.
3. **Largeur des sous-intervalles:** $h = \frac{\pi-0}{4} = \frac{\pi}{4}$.
4. **Points d'évaluation:** $x_0 = 0, x_1 = \frac{\pi}{4}, x_2 = \frac{\pi}{2}, x_3 = \frac{3\pi}{4}, x_4 = \pi$.
5. **Valeurs de la fonction:**
   - $f(x_0) = \sin(0) = 0$,
   - $f(x_1) = \sin(\frac{\pi}{4}) = \frac{\sqrt{2}}{2}$,
   - $f(x_2) = \sin(\frac{\pi}{2}) = 1$,
   - $f(x_3) = \sin(\frac{3\pi}{4}) = \frac{\sqrt{2}}{2}$,
   - $f(x_4) = \sin(\pi) = 0$.
6. **Approximation par la méthode des trapèzes:**

$$I \approx \frac{h}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + f(x_4)] = \frac{\pi/4}{2}[0 + 2\frac{\sqrt{2}}{2} + 2 \times 1 + 2\frac{\sqrt{2}}{2} + 0] = \frac{\pi}{8}[2 + 2\sqrt{2}]$$

Exercice 3: Approximer $\int_1^2 \frac{1}{x}\,dx$ avec 3 sous-intervalles

1. **Intervalle de l'intégration:** $[1, 2]$.
2. **Nombre de sous-intervalles:** $n = 3$.
3. **Largeur des sous-intervalles:** $h = \frac{2-1}{3} = \frac{1}{3}$.
4. **Points d'évaluation:** $x_0 = 1, x_1 = 1.333, x_2 = 1.667, x_3 = 2$.
5. **Valeurs de la fonction:**
   - $f(x_0) = \frac{1}{1} = 1$,
   - $f(x_1) = \frac{1}{1.333} \approx 0.75$,
   - $f(x_2) = \frac{1}{1.667} \approx 0.6$,
   - $f(x_3) = \frac{1}{2} = 0.5$.
6. **Approximation par la méthode des trapèzes:**

$$I \approx \frac{h}{2}[f(x_0) + 2f(x_1) + 2f(x_2) + f(x_3)] = \frac{1/3}{2}[1 + 2 \times 0.75 + 2 \times 0.6 + 0.5] = \frac{1}{6}[3.7] \approx 0.6167$$

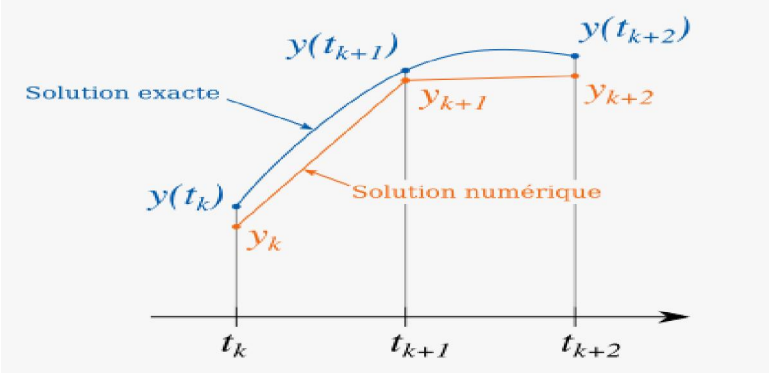
Exercice 4: Approximer $\int_1^2 x^3\,dx$ avec 2 sous-intervalles

5. Méthodes numériques pour les Équations Différentielles Ordinaires (EDO)

Les équations différentielles ordinaires (EDO) modélisent de nombreux phénomènes dans divers domaines tels que la physique, l'ingénierie, la biologie et l'économie. Souvent, ces équations ne peuvent pas être résolues analytiquement, et des méthodes numériques comme la méthode d'Euler sont nécessaires pour approximativement déterminer leur solution. Ce cours détaillé présente la méthode d'Euler, une méthode simple mais fondamentale pour résoudre les EDO.

1. Principes de la Méthode d'Euler

La méthode d'Euler est l'une des approches numériques les plus simples pour résoudre des EDO. Elle est basée sur l'approximation locale d'une fonction par une ligne droite, utilisant la dérivée de la fonction.



Formulation :

Considérons une EDO de la forme : $\begin{cases} \frac{dy}{dt} = f(t,y) \\ y(t_0) = y_0 \end{cases}$

où :

- y est la fonction inconnue,
- t est la variable indépendante,
- $f(t, y)$ est une fonction donnée qui définit l'EDO,
- y_0 est la condition initiale, c'est-à-dire la valeur de y à $t = t_0$.

La méthode d'Euler utilise l'approximation suivante pour estimer la valeur de y à un temps futur t_{n+1} à partir de la valeur actuelle y_n :

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

où h est le pas de temps, défini comme la différence entre t_{n+1} et t_n.

Algorithme de la Méthode d'Euler :

1. **Initialisation** : Choisir les valeurs initiales t_0 et y_0, et définir un pas de temps h.
2. **Itération** : Pour chaque n, calculer :

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

$$t_{n+1} = t_n + h$$

3. Répéter l'étape 2 pour l'intervalle de temps souhaité.

2. Exemples Pratiques et Exercices Résolus

Pour illustrer la méthode d'Euler, nous allons examiner quatre exemples pratiques, du plus simple au plus complexe, chacun étant suivi d'une résolution détaillée.

Exemple 1 : Exponentielle Dégressive Simple

Considérons l'EDO suivante :

$$\frac{dy}{dt} = -y, \quad y(0) = 1$$

Cette équation modélise une décroissance exponentielle, telle que la désintégration radioactive.

- **Objectif** : Approximons la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.1$.

Étapes de calcul :

1. **Initialisation** : $t_0 = 0, y_0 = 1, h = 0.1$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:

$$y_1 = y_0 + h \cdot f(t_0, y_0) = 1 + 0.1 \cdot (-1) = 0.9$$

   - Pour $n = 1$:

$$y_2 = y_1 + h \cdot f(t_1, y_1) = 0.9 + 0.1 \cdot (-0.9) = 0.81$$

   - Pour $n = 2$:

$$y_3 = y_2 + h \cdot f(t_2, y_2) = 0.81 + 0.1 \cdot (-0.81) = 0.729$$

Conclusion : La solution approximative pour $t = 0.3$ est $y \approx 0.729$. La solution exacte est $y(t) = e^{-t}$, et pour $t = 0.3$, $y \approx 0.7408$. L'erreur est donc de l'ordre de 0.0118, montrant que la méthode d'Euler a une précision limitée.

Exemple 2 : Croissance avec une Source Constante

Considérons l'EDO suivante :

$$\frac{dy}{dt} = 2t, \quad y(0) = 1$$

Cette équation modélise un phénomène où la variable dépendante croît avec la source linéaire $2t$.

- **Objectif** : Approximons la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.2$.

Étapes de calcul :

1. **Initialisation** : $t_0 = 0, y_0 = 1, h = 0.2$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:

$$y_1 = y_0 + h \cdot f(t_0, y_0) = 1 + 0.2 \cdot (2 \times 0) = 1$$

   - Pour $n = 1$:

$$y_2 = y_1 + h \cdot f(t_1, y_1) = 1 + 0.2 \cdot (2 \times 0.2) = 1.08$$

- Pour $n = 2$:

$$y_3 = y_2 + h \cdot f(t_2, y_2) = 1.08 + 0.2 \cdot (2 \times 0.4) = 1.24$$

Conclusion : La solution approximative pour $t = 0.4$ est $y \approx 1.24$. La solution exacte est $y(t) = t^2 + 1$, donc pour $t = 0.4$, $y = 1.16$. L'erreur est de 0.08.

Exemple 3 : Oscillateur Harmonie

Considérons l'EDO suivante :

$$\frac{dy}{dt} = \cos(t), \quad y(0) = 0$$

Cette équation modélise un mouvement oscillatoire, comme un oscillateur harmonique.

- **Objectif** : Approximons la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.1$.

Étapes de calcul :

1. **Initialisation** : $t_0 = 0$, $y_0 = 0$, $h = 0.1$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:

$$y_1 = y_0 + h \cdot f(t_0, y_0) = 0 + 0.1 \cdot \cos(0) = 0.1$$

   - Pour $n = 1$:

$$y_2 = y_1 + h \cdot f(t_1, y_1) = 0.1 + 0.1 \cdot \cos(0.1) \approx 0.1995$$

   - Pour $n = 2$:

$$y_3 = y_2 + h \cdot f(t_2, y_2) = 0.1995 + 0.1 \cdot \cos(0.2) \approx 0.2946$$

Conclusion : La solution approximative pour $t = 0.2$ est $y \approx 0.2946$. La solution exacte est $y(t) = \sin(t)$, donc pour $t = 0.2$, $y \approx 0.1987$. L'erreur est donc de l'ordre de 0.0959, illustrant encore une fois la nature approximative de la méthode d'Euler.

Exemple 4 : Système Raide et Méthode d'Euler

Considérons l'EDO suivante :

$$\frac{dy}{dt} = -15y, \quad y(0) = 1$$

Cette équation modélise un système raide, qui pourrait apparaître dans des phénomènes chimiques ou thermodynamiques.

- **Objectif** : Approximons la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.05$.

Étapes de calcul :

1. **Initialisation** : $t_0 = 0$, $y_0 = 1$, $h = 0.05$.

2. **Calcul des points successifs :**
   - Pour $n = 0$:

$$y_1 = y_0 + h \cdot f(t_0, y_0) = 1 + 0.05 \cdot (-15 \times 1) = 0.25$$

   - Pour $n = 1$:

$$y_2 = y_1 + h \cdot f(t_1, y_1) = 0.25 + 0.05 \cdot (-15 \times 0.25) = 0.0625$$

   - Pour $n = 2$:

$$y_3 = y_2 + h \cdot f(t_2, y_2) = 0.0625 + 0.05 \cdot (-15 \times 0.0625) = 0.015625$$

Conclusion : La solution approximative pour $t = 0.1$ est $y \approx 0.015625$. La solution exacte est $y(t) = e^{-15t}$, donc pour $t = 0.1$, $y \approx 0.2231$. L'erreur est donc importante, ce qui montre la limite de la méthode d'Euler pour les systèmes raides.

4.Analyse de la Méthode d'Euler

Avantages :

- **Simplicité** : La méthode d'Euler est facile à comprendre et à mettre en œuvre.
- **Faible coût de calcul** : Peu de calculs sont nécessaires à chaque itération.

Inconvénients :

- **Précision limitée** : La méthode d'Euler est d'ordre 1, ce qui signifie que l'erreur globale est proportionnelle au pas de temps h.
- **Instabilité** : Pour des systèmes raides, la méthode d'Euler peut devenir instable, produisant des solutions incorrectes.

Améliorations possibles :

- Utiliser des pas de temps plus petits pour améliorer la précision, au coût d'une augmentation du nombre d'itérations.
- Utiliser des méthodes d'ordre supérieur, comme Runge-Kutta, pour une meilleure précision avec des pas de temps similaires.
- Pour des systèmes raides, des méthodes implicites, comme l'Euler implicite ou les méthodes de backward Euler, peuvent être plus appropriées.

Conclusion

La méthode d'Euler est une méthode fondamentale et simple pour résoudre numériquement les EDO. Elle permet de comprendre les bases des méthodes numériques et sert de point de départ pour des méthodes plus sophistiquées. Cependant, en raison de ses limitations en termes de précision et de stabilité, il est important de reconnaître les situations où elle est appropriée et celles où des méthodes plus avancées sont nécessaires.

Chapitre I : Rappels sur quelques méthodes numériques

En travaillant à travers les exemples et exercices de ce cours, les étudiants peuvent acquérir une compréhension approfondie de la méthode d'Euler, de ses avantages, de ses limites et de ses applications.

Exercices Pratiques et Résolus ou des *(TP)* : *DE LA METHODE D'EULER AVEC MATLAB*

Nous allons explorer des exemples pratiques, chacun accompagné d'un programme MATLAB pour implémenter la solution numérique.

Exercice 1 : Système Raide

EDO :

$$\frac{dy}{dt} = -15y, \quad y(0) = 1$$

- **Objectif** : Approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.05$.

Étapes de Calcul :

1. **Initialisation** : $t_0 = 0, y_0 = 1, h = 0.05$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:

$$y_1 = y_0 + h \cdot f(t_0, y_0) = 1 + 0.05 \cdot (-15 \times 1) = 0.25$$

   - Pour $n = 1$:

$$y_2 = y_1 + h \cdot f(t_1, y_1) = 0.25 + 0.05 \cdot (-15 \times 0.25) = 0.0625$$

   - Continuer jusqu'à $t = 1$.

Programme MATLAB :

```
% Exercice 1 : Système raide

t0 = 0 ;
y0 = 1;
h = 0.05;
t = t0:h:1;
y = zeros(size(t));
y(1) = y0;
for n = 1:length(t)-1
   y(n+1) = y(n) + h * (-15 * y(n));
end
plot(t, y, '-o');
xlabel('t');
```

```
ylabel('y');
title('Méthode d''Euler : Système raide');
```

Exercice 2 : Croissance Logistique

EDO :

$$\frac{dy}{dt} = y(1-y), \quad y(0) = 0.1$$

- **Objectif** : Approximer la solution sur l'intervalle $t \in [0, 5]$ avec un pas de temps $h = 0.1$.

Étapes de Calcul :

1. **Initialisation** : $t_0 = 0, y_0 = 0.1, h = 0.1$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:
   $$y_1 = y_0 + h \cdot f(t_0, y_0) = 0.1 + 0.1 \cdot (0.1 \times (1 - 0.1)) = 0.109$$
   - Pour $n = 1$:
   $$y_2 = y_1 + h \cdot f(t_1, y_1) = 0.109 + 0.1 \cdot (0.109 \times (1 - 0.109)) \approx 0.1181$$
   - Continuer jusqu'à $t = 5$.

Programme MATLAB :

```
% Exercice 2 : Croissance logistique
t0 = 0;
y0 = 0.1;
h = 0.1;
t = t0:h:5;
y = zeros(size(t));
y(1) = y0;
for n = 1:length(t)-1
    y(n+1) = y(n) + h * (y(n) * (1 - y(n)));
end
plot(t, y, '-o');
xlabel('t');
ylabel('y');
title('Méthode d''Euler : Croissance logistique');
```

Exercice 3 : Mouvement d'un Projectile sous Gravité

EDO :

$$\frac{dy}{dt} = v, \quad \frac{dv}{dt} = -g, \quad y(0) = 0, \quad v(0) = 10$$

où $g = 9.81$ est l'accélération due à la gravité.

- **Objectif** : Approximer la position et la vitesse d'un projectile sur l'intervalle $t \in [0, 2]$ avec un pas de temps $h = 0.1$.

Étapes de Calcul :

1. **Initialisation** : $t_0 = 0$, $y_0 = 0$, $v_0 = 10$, $h = 0.1$.
2. **Calcul des points successifs** :
   - Pour $n = 0$:

$$v_1 = v_0 + h \cdot (-g) = 10 - 0.1 \cdot 9.81 = 9.019$$

$$y_1 = y_0 + h \cdot v_0 = 0 + 0.1 \cdot 10 = 1$$

   - Pour $n = 1$:

$$v_2 = v_1 + h \cdot (-g) = 9.019 - 0.1 \cdot 9.81 = 8.038$$

$$y_2 = y_1 + h \cdot v_1 = 1 + 0.1 \cdot 9.019 = 1.9019$$

   - Continuer jusqu'à $t = 2$.

Programme MATLAB :

```
% Exercice 3 : Mouvement d'un projectile sous gravité
t0 = 0;
y0 = 0;
v0 = 10;
g = 9.81;
h = 0.1;
t = t0:h:2;
y = zeros(size(t));
v = zeros(size(t));
y(1) = y0;
v(1) = v0;
for n = 1:length(t)-1
   v(n+1) = v(n) - h * g;
   y(n+1) = y(n) + h * v(n);
end
subplot(2,1,1);
plot(t, y, '-o');
xlabel('t');
ylabel('y');
```

```
title('Position du projectile (Méthode d''Euler)');
subplot(2,1,2);
plot(t, v, '-o');
xlabel('t');
ylabel('v');
title('Vitesse du projectile (Méthode d''Euler)');
```

B :Méthodes Implicites et la Méthode de Taylor pour les Systèmes d'Équations Différentielles Couplées

Nous explorons les méthodes implicites et la méthode de Taylor (ordre 2 et 4) pour résoudre des systèmes d'équations différentielles ordinaires (EDO) couplées. Nous traiterons des exemples pratiques tels que des moteurs électriques et des pendules couplés. Chaque méthode sera illustrée avec cinq exercices résolus en détail, accompagnés de programmes MATLAB.

1. Méthodes Implicites pour les Systèmes Couplés

Les méthodes implicites sont robustes et stables, particulièrement adaptées aux systèmes raides, où les solutions changent rapidement. Contrairement aux méthodes explicites, les méthodes implicites nécessitent la résolution d'équations simultanées à chaque étape, ce qui peut nécessiter des techniques itératives comme la méthode de Newton-Raphson.

1.1. Méthode d'Euler Implicite pour les Systèmes Couplés

Formulation :

Pour un système de deux EDO couplées :

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2)$$
$$\frac{dy_2}{dt} = f_2(t, y_1, y_2)$$

La méthode d'Euler implicite est formulée comme :

$$y_{1,n+1} = y_{1,n} + h \cdot f_1(t_{n+1}, y_{1,n+1}, y_{2,n+1})$$
$$y_{2,n+1} = y_{2,n} + h \cdot f_2(t_{n+1}, y_{1,n+1}, y_{2,n+1})$$

Application :

Exercice 1 : Moteur Électrique Simple

Considérons un modèle simplifié d'un moteur électrique représenté par un système d'équations :

$$\frac{di}{dt} = \frac{V - Ri - L\frac{d\omega}{dt}}{L}$$

$$\frac{d\omega}{dt} = \frac{Ki - D\omega}{J}$$

- V : tension d'entrée
- R : résistance
- L : inductance
- K : constante de moteur
- D : coefficient de frottement
- J : inertie

Conditions Initiales : $i(0) = 0, \omega(0) = 0.$

Objectif : Utiliser Euler implicite pour approximer la solution sur $t \in [0, 0.5]$ avec $h = 0.1$.

1. **Initialisation** : $t_0 = 0, i_0 = 0, \omega_0 = 0, h = 0.1, V = 10, R = 1, L = 0.5, K = 0.1, D = 0.1, J = 0.01.$
2. **Résolution itérative** pour i_{n+1} et ω_{n+1} à chaque étape.

Programme MATLAB :

```
% Méthode d'Euler Implicite pour un moteur électrique

t0 = 0;
i0 = 0;
omega0 = 0;
h = 0.1;
V = 10; R = 1; L = 0.5; K = 0.1; D = 0.1; J = 0.01;
t = t0:h:0.5;
i = zeros(size(t));
omega = zeros(size(t));
i(1) = i0;
omega(1) = omega0;

for n = 1:length(t)-1
    i_n = i(n);
    omega_n = omega(n);
    % Méthode itérative pour trouver i_{n+1} et omega_{n+1}
    i_n1 = (V - R*i_n) / (R + L/h);
    omega_n1 = (K*i_n1 - D*omega_n) / (J + D*h);
    i(n+1) = i_n1;
    omega(n+1) = omega_n1;
end
plot(t, i, '-o', t, omega, '-x');
```

```
xlabel('t');
ylabel('Courant et Vitesse');
legend('Courant (i)', 'Vitesse (\omega)');
title('Méthode d''Euler Implicite pour un moteur électrique');
```

Exercice 2 : Pendules Couplés

Considérons deux pendules couplés dont les mouvements sont décrits par :

$$\frac{d^2\theta_1}{dt^2} = -g\sin(\theta_1) - k(\theta_1 - \theta_2)$$

$$\frac{d^2\theta_2}{dt^2} = -g\sin(\theta_2) - k(\theta_2 - \theta_1)$$

- g : accélération due à la gravité
- k : constante de couplage

Conditions Initiales : $\theta_1(0) = 0.1$, $\theta_2(0) = 0.2$, $\frac{d\theta_1}{dt}(0) = 0$, $\frac{d\theta_2}{dt}(0) = 0$.

Objectif : Utiliser Euler implicite pour approximations sur $t \in [0, 10]$, $h = 0.2$.

1. **Initialisation** : $t_0 = 0$, $\theta_{1,0} = 0.1$, $\theta_{2,0} = 0.2$, $\omega_{1,0} = 0$, $\omega_{2,0} = 0$, $h = 0.2$, $g = 9.81$, $k = 0.5$.
2. **Résolution itérative** pour $\theta_{1,n+1}$, $\theta_{2,n+1}$, $\omega_{1,n+1}$, $\omega_{2,n+1}$.

Programme MATLAB :

```
% Méthode d'Euler Implicite pour des pendules couplés
t0 = 0;
theta1_0 = 0.1;
theta2_0 = 0.2;
omega1_0 = 0;
omega2_0 = 0;
h = 0.2;
g = 9.81;
k = 0.5;
t = t0:h:10;
theta1 = zeros(size(t));
theta2 = zeros(size(t));
omega1 = zeros(size(t));
omega2 = zeros(size(t));
theta1(1) = theta1_0;
theta2(1) = theta2_0;
omega1(1) = omega1_0;
omega2(1) = omega2_0;
```

```
for n = 1:length(t)-1
    theta1_n = theta1(n);
    theta2_n = theta2(n);
    omega1_n = omega1(n);
    omega2_n = omega2(n);
    % Méthode itérative pour trouver theta_{1,n+1} et theta_{2,n+1}
    theta1_n1 = theta1_n + h * omega1_n;
    theta2_n1 = theta2_n + h * omega2_n;
    omega1_n1 = omega1_n - h * (g * sin(theta1_n1) + k * (theta1_n1 - theta2_n1));
    omega2_n1 = omega2_n - h * (g * sin(theta2_n1) + k * (theta2_n1 - theta1_n1));
    theta1(n+1) = theta1_n1;
    theta2(n+1) = theta2_n1;
    omega1(n+1) = omega1_n1;
    omega2(n+1) = omega2_n1;
end

plot(t, theta1, '-o', t, theta2, '-x');
xlabel('t');
ylabel('\theta');
legend('\theta_1', '\theta_2');
title('Méthode d''Euler Implicite pour des pendules couplés');
```

1.3. Méthode du Trapèze pour les Systèmes Couplés

La méthode du trapèze est une méthode implicite d'ordre 2 qui utilise une moyenne pondérée des valeurs aux points actuels et futurs.

Formulation :

Pour un système de deux EDO couplées :

Pour un système de deux EDO couplées :

$$y_{1,n+1} = y_{1,n} + \frac{h}{2}[f_1(t_n, y_{1,n}, y_{2,n}) + f_1(t_{n+1}, y_{1,n+1}, y_{2,n+1})]$$

$$y_{2,n+1} = y_{2,n} + \frac{h}{2}[f_2(t_n, y_{1,n}, y_{2,n}) + f_2(t_{n+1}, y_{1,n+1}, y_{2,n+1})]$$

Exercice 3 : Moteur Électrique avec Frottement Variable

Un modèle de moteur électrique où la force de frottement dépend de la vitesse est :

$$\frac{di}{dt} = \frac{V - Ri - L\frac{d\omega}{dt}}{L}$$

$$\frac{d\omega}{dt} = \frac{Ki - D(\omega)\omega}{J}$$

où $D(\omega) = D_0 + D_1\omega^2$.

Conditions Initiales : $i(0) = 0, \omega(0) = 0$.

Objectif : Utiliser la méthode du trapeze pour approximations sur $t \in [0, 0.5]$, $h = 0.1$.

1. **Initialisation :** $t_0 = 0, i_0 = 0, \omega_0 = 0, h = 0.1, V = 10, R = 1, L = 0.5, K = 0.1, D_0 = 0.1, D_1 = 0.01, J = 0.01$.
2. **Résolution itérative** pour i_{n+1} et ω_{n+1}.

Programme MATLAB :

```
% Méthode du trapeze pour un moteur électrique avec frottement variable
t0 = 0;
i0 = 0;
omega0 = 0;
h = 0.1;
V = 10; R = 1; L = 0.5; K = 0.1; D0 = 0.1; D1 = 0.01; J = 0.01;
t = t0:h:0.5;
i = zeros(size(t));
omega = zeros(size(t));
i(1) = i0;
omega(1) = omega0;

for n = 1:length(t)-1
   i_n = i(n);
   omega_n = omega(n);
   % Méthode itérative pour trouver i_{n+1} et omega_{n+1}
   i_n1 = (V - R*i_n) / (R + L/h);
   omega_n1 = (K*i_n1 - (D0 + D1*omega_n^2)*omega_n) / (J + (D0 + D1*omega_n^2)*h);
   i(n+1) = i_n1;
   omega(n+1) = omega_n1;
end
plot(t, i, '-o', t, omega, '-x');
xlabel('t');
```

```
ylabel('Courant et Vitesse');
legend('Courant (i)', 'Vitesse (\omega)');
title('Méthode du trapeze pour un moteur électrique avec frottement variable');
```

C : Méthode de Taylor pour les Systèmes Couplés

La méthode de Taylor utilise des dérivées successives pour estimer les valeurs futures des variables. Plus l'ordre est élevé, plus l'approximation est précise.

C.1. Méthode de Taylor d'Ordre 2 pour les Systèmes Couplés

Formulation :

Pour un système de deux EDO couplées :

$$y_{1,n+1} = y_{1,n} + hf_1(t_n, y_{1,n}, y_{2,n}) + \frac{h^2}{2} f_1'(t_n, y_{1,n}, y_{2,n})$$

$$y_{2,n+1} = y_{2,n} + hf_2(t_n, y_{1,n}, y_{2,n}) + \frac{h^2}{2} f_2'(t_n, y_{1,n}, y_{2,n})$$

Exercice 4 : Système de Pendules Couplés avec Amortissement

Considérons deux pendules couplés avec amortissement :

$$\frac{d^2\theta_1}{dt^2} = -g\sin(\theta_1) - k(\theta_1 - \theta_2) - b\frac{d\theta_1}{dt}$$

$$\frac{d^2\theta_2}{dt^2} = -g\sin(\theta_2) - k(\theta_2 - \theta_1) - b\frac{d\theta_2}{dt}$$

- g : accélération due à la gravité
- k : constante de couplage
- b : coefficient d'amortissement

Conditions Initiales : $\theta_1(0) = 0.1$, $\theta_2(0) = 0.2$, $\frac{d\theta_1}{dt}(0) = 0$, $\frac{d\theta_2}{dt}(0) = 0$.

Objectif : Utiliser la méthode de Taylor d'ordre 2 pour approximations sur $t \in [0, 10]$, $h = 0.2$.

1. **Initialisation** : $t_0 = 0$, $\theta_{1,0} = 0.1$, $\theta_{2,0} = 0.2$, $\omega_{1,0} = 0$, $\omega_{2,0} = 0$, $h = 0.2$, $g = 9.81$, $k = 0.5$, $b = 0.05$.
2. **Calcul** en incluant les dérivées successives.

Programme MATLAB :

```
% Méthode de Taylor d'ordre 2 pour des pendules couplés avec amortissement
t0 = 0;
theta1_0 = 0.1;
```

```
theta2_0 = 0.2;
omega1_0 = 0;
omega2_0 = 0;
h = 0.2;
g = 9.81;
k = 0.5;
b = 0.05;
t = t0:h:10;
theta1 = zeros(size(t));
theta2 = zeros(size(t));
omega1 = zeros(size(t));
omega2 = zeros(size(t));
theta1(1) = theta1_0;
theta2(1) = theta2_0;
omega1(1) = omega1_0;
omega2(1) = omega2_0;

for n = 1:length(t)-1
   theta1_n = theta1(n);
   theta2_n = theta2(n);
   omega1_n = omega1(n);
   omega2_n = omega2(n);
   % Calcul des dérivées
   f1 = omega1_n;
   f2 = -g * sin(theta1_n) - k * (theta1_n - theta2_n) - b * omega1_n;
   f1_prime = -g * cos(theta1_n) * omega1_n - k * (omega1_n - omega2_n) - b * f1;
   f2_prime = -g * cos(theta2_n) * omega2_n - k * (omega2_n - omega1_n) - b * f2;

   theta1(n+1) = theta1_n + h * f1 + (h^2 / 2) * f1_prime;
   theta2(n+1) = theta2_n + h * f2 + (h^2 / 2) * f2_prime;
   omega1(n+1) = omega1_n + h * f1 + (h^2 / 2) * f1_prime;
   omega2(n+1) = omega2_n + h * f2 + (h^2 / 2) * f2_prime;
end

plot(t, theta1, '-o', t, theta2, '-x');
xlabel('t');
ylabel('\theta');
legend('\theta_1', '\theta_2');
title('Méthode de Taylor d''ordre 2 pour des pendules couplés avec amortissement');
```

C.2. Méthode de Taylor d'Ordre 4 pour les Systèmes Couplés

Formulation :

Pour un système de deux EDO couplées :

$$y_{1,n+1} = y_{1,n} + hf_1(t_n, y_{1,n}, y_{2,n}) + \frac{h^2}{2}f_1'(t_n, y_{1,n}, y_{2,n}) + \frac{h^3}{6}f_1''(t_n, y_{1,n}, y_{2,n}) + \frac{h^4}{24}f_1'''(t_n, y_{1,n}, y_{2,n})$$

$$y_{2,n+1} = y_{2,n} + hf_2(t_n, y_{1,n}, y_{2,n}) + \frac{h^2}{2}f_2'(t_n, y_{1,n}, y_{2,n}) + \frac{h^3}{6}f_2''(t_n, y_{1,n}, y_{2,n}) + \frac{h^4}{24}f_2'''(t_n, y_{1,n}, y_{2,n})$$

Exercice 5 : Moteur Électrique avec Amortissement Non Linéaire

Modèle d'un moteur avec un amortissement non linéaire :

$$\frac{di}{dt} = \frac{V - Ri - L\frac{d\omega}{dt}}{L}$$

$$\frac{d\omega}{dt} = \frac{Ki - D(\omega)\omega}{J}$$

où $D(\omega) = D_0 + D_1\omega^2$.

Conditions Initiales : $i(0) = 0, \omega(0) = 0$.

Objectif : Utiliser la méthode de Taylor d'ordre 4 pour approximations sur $t \in [0, 0.5]$, $h = 0.1$.

1. **Initialisation :** $t_0 = 0, i_0 = 0, \omega_0 = 0, h = 0.1, V = 10, R = 1, L = 0.5, K = 0.1, D_0 = 0.1, D_1 = 0.01, J = 0.01$.
2. **Calcul** en incluant les dérivées successives.

Programme MATLAB :

```
% Méthode de Taylor d'ordre 4 pour un moteur électrique avec amortissement non linéaire
t0 = 0;
i0 = 0;
omega0 = 0;
h = 0.1;
V = 10; R = 1; L = 0.5; K = 0.1; D0 = 0.1; D1 = 0.01; J = 0.01;
t = t0:h:0.5;
i = zeros(size(t));
omega = zeros(size(t));
i(1) = i0;
omega(1) = omega0;

for n = 1:length(t)-1
    i_n = i(n);
    omega_n = omega(n);
    % Calcul des dérivées successives
    f1 = (V - R * i_n - L * (-D0 * omega_n - D1 * omega_n^3)) / L;
    f1_prime = -R/L + K / (L * J) * i_n;
    f1_double_prime = -2 * D1 * omega_n;
    f1_triple_prime = -2 * D1;
```

```
    omega_n1 = omega_n + h * f1 + (h^2 / 2) * f1_prime + (h^3 / 6) * f1_double_prime + (h^4 / 24) * f1_triple_prime;
    i(n+1) = i_n + h * f1 + (h^2 / 2) * f1_prime + (h^3 / 6) * f1_double_prime + (h^4 / 24) * f1_triple_prime;
    omega(n+1) = omega_n1;
end

plot(t, i, '-o', t, omega, '-x');
xlabel('t');
ylabel('Courant et Vitesse');
legend('Courant (i)', 'Vitesse (\omega)');
title('Méthode de Taylor d''ordre 4 pour un moteur électrique avec amortissement non linéaire');
```

Conclusion

Les méthodes implicites et la méthode de Taylor (ordre 2 et 4) offrent des outils puissants pour la résolution des EDO couplées, adaptées à des systèmes raides ou nécessitant des solutions précises. En utilisant ces méthodes, nous pouvons modéliser des phénomènes complexes tels que les moteurs électriques et les pendules couplés avec une grande précision. Les programmes MATLAB fournis démontrent l'application pratique de ces méthodes dans des scénarios réels

D :Méthodes de Runge-Kutta pour les Équations Différentielles Ordinaires (EDO)

Les méthodes de Runge-Kutta sont parmi les plus utilisées pour la résolution numérique des équations différentielles ordinaires (EDO). Elles offrent un bon compromis entre précision et complexité de calcul. Ce cours se concentrera sur les méthodes de Runge-Kutta d'ordre 2 (RK2) et d'ordre 4 (RK4), qui sont les plus couramment employées dans les applications scientifiques et d'ingénierie.

D.1.Introduction aux Méthodes de Runge-Kutta

Les méthodes de Runge-Kutta sont des méthodes itératives qui utilisent plusieurs évaluations de la fonction de dérivée pour améliorer l'estimation de la solution. Elles permettent de réduire l'erreur globale par rapport aux méthodes plus simples comme la méthode d'Euler.

Pour une EDO de la forme :

$$\begin{cases} \dfrac{dy}{dt} = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

Les méthodes de Runge-Kutta calculent la valeur de y à des points discrets en utilisant des combinaisons pondérées des dérivées de la fonction.

D.2. Méthode de Runge-Kutta d'ordre 2 (RK2)

La méthode RK2, également appelée méthode du point médian ou méthode de Heun, améliore la méthode d'Euler en calculant la pente au milieu de l'intervalle de temps.

Formulation de la Méthode RK2

1. **Calcul de la première estimation de la pente** (comme dans la méthode d'Euler) :

$$k_1 = f(t_n, y_n)$$

2. **Calcul de la seconde estimation de la pente** en utilisant k_1 pour trouver une meilleure estimation :

$$k_2 = f(t_n + h, y_n + h \cdot k_1)$$

3. **Mise à jour de la solution** avec une moyenne des deux pentes :

$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2)$$

Exemple Pratique de la Méthode RK2

Problème : Considérons l'EDO $\frac{dy}{dt} = t + y$, avec la condition initiale $y(0) = 1$. Nous allons utiliser RK2 pour approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.2$.

1. **Initialisation** : $t_0 = 0, y_0 = 1, h = 0.2$.
2. **Calcul des valeurs successives** :
   - Pour $n = 0$:

$$k_1 = f(0, 1) = 0 + 1 = 1$$
$$k_2 = f(0.2, 1 + 0.2 \cdot 1) = 0.2 + 1.2 = 1.4$$
$$y_1 = 1 + \frac{0.2}{2} \times (1 + 1.4) = 1 + 0.2 \times 1.2 = 1.24$$

   - Pour $n = 1$:

$$k_1 = f(0.2, 1.24) = 0.2 + 1.24 = 1.44$$
$$k_2 = f(0.4, 1.24 + 0.2 \cdot 1.44) = 0.4 + 1.528 = 1.928$$
$$y_2 = 1.24 + \frac{0.2}{2} \times (1.44 + 1.928) = 1.24 + 0.2 \times 1.684 = 1.5768$$

   - Continuer cette procédure jusqu'à $t = 1$.

Programme MATLAB pour la Méthode RK2 :

```
% Méthode de Runge-Kutta d'ordre 2 (RK2)

t0 = 0;
y0 = 1;
```

```
h = 0.2;
t = t0:h:1;
y = zeros(size(t));
y(1) = y0;
for n = 1:length(t)-1
    k1 = t(n) + y(n);
    k2 = (t(n) + h) + (y(n) + h * k1);
    y(n+1) = y(n) + h/2 * (k1 + k2);
end
plot(t, y, '-o');
xlabel('t');
ylabel('y');
title('Méthode de Runge-Kutta d''ordre 2 (RK2)');
```

D.3. Méthode de Runge-Kutta d'ordre 4 (RK4) : La méthode RK4 est plus précise que RK2 car elle utilise quatre évaluations de la pente à chaque pas de temps, ce qui lui permet de mieux approximer la solution.

Formulation de la Méthode RK4

La méthode RK4 est plus précise que RK2 car elle utilise quatre évaluations de la pente à chaque pas de temps, ce qui lui permet de mieux approximer la solution.

Formulation de la Méthode RK4

1. Calcul des quatre pentes :

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_n + h, y_n + h \cdot k_3)$$

2. Mise à jour de la solution :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Exemple Pratique de la Méthode RK4

Problème : Considérons l'EDO $\frac{dy}{dt} = y - t^2 + 1$, avec la condition initiale $y(0) = 0.5$. Utilisons RK4 pour approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.2$.

1. **Initialisation** : $t_0 = 0, y_0 = 0.5, h = 0.2$.
2. **Calcul des valeurs successives** :
   - Pour $n = 0$:

$$k_1 = f(0, 0.5) = 0.5 - 0^2 + 1 = 1.5$$

$$k_2 = f(0.1, 0.5 + 0.1 \cdot 1.5) = 0.65 + 1 = 1.65$$

$$k_3 = f(0.1, 0.5 + 0.1 \cdot 1.65) = 0.665 + 1 = 1.665$$

$$k_4 = f(0.2, 0.5 + 0.2 \cdot 1.665) = 0.833 + 1 = 1.833$$

$$y_1 = 0.5 + \frac{0.2}{6}(1.5 + 2 \times 1.65 + 2 \times 1.665 + 1.833) \approx 0.829$$

   - Continuer cette procédure jusqu'à $t = 1$.

Programme MATLAB pour la Méthode RK4 :

```
% Méthode de Runge-Kutta d'ordre 4 (RK4)
t0 = 0;
y0 = 0.5;
h = 0.2;
t = t0:h:1;
y = zeros(size(t));
y(1) = y0;

for n = 1:length(t)-1
    k1 = y(n) - t(n)^2 + 1;
    k2 = (y(n) + 0.5 * h * k1) - (t(n) + 0.5 * h)^2 + 1;
    k3 = (y(n) + 0.5 * h * k2) - (t(n) + 0.5 * h)^2 + 1;
    k4 = (y(n) + h * k3) - (t(n) + h)^2 + 1;
    y(n+1) = y(n) + h/6 * (k1 + 2*k2 + 2*k3 + k4);
end

plot(t, y, '-o');
xlabel('t');
ylabel('y');
title('Méthode de Runge-Kutta d''ordre 4 (RK4)');
```

Chapitre I : Rappels sur quelques méthodes numériques

Exercice 1 : Utiliser RK2 pour résoudre l'EDO $\frac{dy}{dt} = y + t^2$, avec la condition initiale $y(0) = 1$, sur l'intervalle $t \in [0, 2]$ avec un pas de temps $h = 0.2$.

Solution :

1. Initialisation : $t_0 = 0, y_0 = 1, h = 0.2$.
2. Calculer k_1 et k_2 :
   - $k_1 = y_n + t_n^2$
   - $k_2 = y_n + h \cdot k_1 + (t_n + h)^2$
3. Mise à jour de la solution :

$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2)$$

4. Programme MATLAB :

```
% Exercice 1 : RK2 avec EDO y' = y + t^2
t0 = 0;
y0 = 1;
h = 0.2;
t = t0:h:2;
y = zeros(size(t));
y(1) = y0;

for n = 1:length(t)-1
    k1 = y(n) + t(n)^2;
    k2 = (y(n) + h * k1) + (t(n) + h)^2;
    y(n+1) = y(n) + h/2 * (k1 + k2);
end

plot(t, y, '-o');
xlabel('t');
ylabel('y');
title('Méthode de Runge-Kutta d''ordre 2 (RK2) - Exercice 1');
```

Exercice 2 : Utiliser RK4 pour résoudre l'EDO $\frac{dy}{dt} = \sin(t) - y$, avec $y(0) = 0$, sur $t \in [0, \pi]$ avec un pas $h = 0.1$.

Solution :

1. Initialisation : $t_0 = 0, y_0 = 0, h = 0.1$.
2. Calcul des pentes k_1, k_2, k_3, k_4 :
   - $k_1 = \sin(t_n) - y_n$
   - $k_2 = \sin(t_n + \frac{h}{2}) - (y_n + \frac{h}{2}k_1)$
   - $k_3 = \sin(t_n + \frac{h}{2}) - (y_n + \frac{h}{2}k_2)$
   - $k_4 = \sin(t_n + h) - (y_n + h \cdot k_3)$
3. Mise à jour de la solution :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

4. Programme MATLAB :

```
% Exercice 2 : RK4 avec EDO : y' = sin(t) - y
t0 = 0;
y0 = 0;
h = 0.1;
t = t0:h:pi;
y = zeros(size(t));
y(1) = y0;

for n = 1:length(t)-1
    k1 = sin(t(n)) - y(n);
    k2 = sin(t(n) + 0.5 * h) - (y(n) + 0.5 * h * k1);
    k3 = sin(t(n) + 0.5 * h) - (y(n) + 0.5 * h * k2);
    k4 = sin(t(n) + h) - (y(n) + h * k3);
    y(n+1) = y(n) + h/6 * (k1 + 2*k2 + 2*k3 + k4);
end

plot(t, y, '-o');
xlabel('t');
ylabel('y');
title('Méthode de Runge-Kutta d''ordre 4 (RK4) - Exercice 2');
```

Conclusion : Les méthodes de Runge-Kutta, notamment RK2 et RK4, sont des outils puissants pour la résolution numérique des EDO. Elles offrent une meilleure précision que la méthode d'Euler, tout en étant relativement simples à implémenter. RK4, en particulier, est largement utilisée en raison de son équilibre entre précision et coût de calcul. En comprenant

et en appliquant ces méthodes à travers des exercices pratiques, les étudiants peuvent acquérir une base solide pour aborder des problèmes plus complexes dans des applications réelles.

E :Méthodes de Runge-Kutta 4 pour les Équations Différentielles Couplées

Les méthodes de Runge-Kutta sont parmi les plus utilisées pour la résolution numérique des équations différentielles ordinaires (EDO). Dans ce cours, nous allons explorer l'application des méthodes de Runge-Kutta, notamment la méthode de quatrième ordre (RK4), pour résoudre des systèmes d'équations différentielles couplées. Nous couvrirons six exercices résolus, allant du simple au complexe, en utilisant MATLAB pour implémenter les solutions numériques.

Formulation de la Méthode RK2

Pour un système d'équations différentielles couplées de la forme :

Pour un système de trois équations différentielles couplées :

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2, y_3)$$

$$\frac{dy_2}{dt} = f_2(t, y_1, y_2, y_3)$$

$$\frac{dy_3}{dt} = f_3(t, y_1, y_2, y_3)$$

Les étapes de la méthode RK2 sont :

1. Calculer les premières estimations des pentes (k_1) pour chaque équation :

$$k_{1,i} = f_i(t_n, y_{1,n}, y_{2,n}, y_{3,n})$$

2. Calculer les secondes estimations des pentes (k_2) en utilisant k_1 pour estimer la valeur au point intermédiaire :

$$k_{2,i} = f_i\left(t_n + h, y_{1,n} + h \cdot k_{1,1}, y_{2,n} + h \cdot k_{1,2}, y_{3,n} + h \cdot k_{1,3}\right)$$

3. Mettre à jour les valeurs des fonctions :

$$y_{i,n+1} = y_{i,n} + \frac{h}{2}(k_{1,i} + k_{2,i})$$

Formulation de la Méthode RK4

Pour les mêmes équations couplées :

1. Calculer les quatre pentes :

$$k_{1,i} = f_i(t_n, y_{1,n}, y_{2,n}, y_{3,n})$$

$$k_{2,i} = f_i\left(t_n + \frac{h}{2}, y_{1,n} + \frac{h}{2}k_{1,1}, y_{2,n} + \frac{h}{2}k_{1,2}, y_{3,n} + \frac{h}{2}k_{1,3}\right)$$

$$k_{3,i} = f_i\left(t_n + \frac{h}{2}, y_{1,n} + \frac{h}{2}k_{2,1}, y_{2,n} + \frac{h}{2}k_{2,2}, y_{3,n} + \frac{h}{2}k_{2,3}\right)$$

$$k_{4,i} = f_i\left(t_n + h, y_{1,n} + hk_{3,1}, y_{2,n} + hk_{3,2}, y_{3,n} + hk_{3,3}\right)$$

2. Mettre à jour les valeurs des fonctions :

$$y_{i,n+1} = y_{i,n} + \frac{h}{6}(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})$$

Exercices Résolus avec Explications et Programmes MATLAB

Exercice 1 : Système de Trois Équations Linéaires Couplées (RK2)

Équations :

$$\frac{dy_1}{dt} = y_1 + y_2 + y_3$$

$$\frac{dy_2}{dt} = 2y_1 - y_2 + y_3$$

$$\frac{dy_3}{dt} = y_1 - y_2 + 2y_3$$

Conditions Initiales : $y_1(0) = 1, y_2(0) = 0, y_3(0) = -1.$

Objectif : Utiliser RK2 pour approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.1.$

Solution :

1. **Initialisation** : $t_0 = 0, y_{1,0} = 1, y_{2,0} = 0, y_{3,0} = -1, h = 0.1.$
2. **Calcul des valeurs successives** avec les étapes RK2 pour chaque équation.

Programme MATLAB :

```
% Exercice 1 : Système de trois équations linéaires couplées (RK2)
t0 = 0;
y1_0 = 1;
y2_0 = 0;
y3_0 = -1;
h = 0.1;
t = t0:h:1;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y1(n) + y2(n) + y3(n);
    k1_2 = 2*y1(n) - y2(n) + y3(n);
    k1_3 = y1(n) - y2(n) + 2*y3(n);

    k2_1 = (y1(n) + h*k1_1) + (y2(n) + h*k1_2) + (y3(n) + h*k1_3);
    k2_2 = 2*(y1(n) + h*k1_1) - (y2(n) + h*k1_2) + (y3(n) + h*k1_3);
    k2_3 = (y1(n) + h*k1_1) - (y2(n) + h*k1_2) + 2*(y3(n) + h*k1_3);

    y1(n+1) = y1(n) + h/2 * (k1_1 + k2_1);
    y2(n+1) = y2(n) + h/2 * (k1_2 + k2_2);
    y3(n+1) = y3(n) + h/2 * (k1_3 + k2_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations linéaires couplées - RK2');
```

Exercice 2 : Système de Trois Équations Non Linéaires Couplées (RK2)

Équations :

$$\frac{dy_1}{dt} = y_2 - y_1 y_3$$

$$\frac{dy_2}{dt} = y_1 + y_3^2$$

$$\frac{dy_3}{dt} = y_1 y_2 - y_3$$

Conditions Initiales : $y_1(0) = 1, y_2(0) = 2, y_3(0) = 1.$

Objectif : Utiliser RK2 pour approximer la solution sur l'intervalle $t \in [0, 2]$ avec un pas de temps $h = 0.2.$

Solution :

1. Initialisation : $t_0 = 0, y_{1,0} = 1, y_{2,0} = 2, y_{3,0} = 1, h = 0.2.$
2. Calcul des valeurs successives avec les étapes RK2 pour chaque équation.

Programme MATLAB :

```
% Exercice 2 : Système de trois équations non linéaires couplées (RK2)
t0 = 0;
y1_0 = 1;
y2_0 = 2;
y3_0 = 1;
h = 0.2;
t = t0:h:2;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y2(n) - y1(n)*y3(n);
    k1_2 = y1(n) + y3(n)^2;
    k1_3 = y1(n)*y2(n) - y3(n);

    k2_1 = (y2(n) + h*k1_2) - (y1(n) + h*k1_1)*(y3(n) + h*k1_3);
    k2_2 = (y1(n) + h*k1_1) + (y3(n) + h*k1_3)^2;
    k2_3 = (y1(n) + h*k1_1)*(y2(n) + h*k1_2) - (y3(n) + h*k1_3);
```

```
    y1(n+1) = y1(n) + h/2 * (k1_1 + k2_1);
    y2(n+1) = y2(n) + h/2 * (k1_2 + k2_2);
    y3(n+1) = y3(n) + h/2 * (k1_3 + k2_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations non linéaires couplées - RK2');
```

Exercice 3 : Système de Trois Équations Linéaires Couplées (RK4)

Équations :

$$\frac{dy_1}{dt} = y_2 + 2y_3$$

$$\frac{dy_2}{dt} = 3y_1 + y_3$$

$$\frac{dy_3}{dt} = y_1 + 4y_2$$

Conditions Initiales : $y_1(0) = 0, y_2(0) = 1, y_3(0) = 2.$

Objectif : Utiliser RK4 pour approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.1$.

Solution :

1. Initialisation : $t_0 = 0, y_{1,0} = 0, y_{2,0} = 1, y_{3,0} = 2, h = 0.1.$
2. Calcul des valeurs successives avec les étapes RK4 pour chaque équation.

Programme MATLAB :

```
% Exercice 3 : Système de trois équations linéaires couplées (RK4)
t0 = 0;
y1_0 = 0;
y2_0 = 1;
y3_0 = 2;
h = 0.1;
t = t0:h:1;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
```

```
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y2(n) + 2*y3(n);
    k1_2 = 3*y1(n) + y3(n);
    k1_3 = y1(n) + 4*y2(n);

    k2_1 = (y2(n) + 0.5*h*k1_2) + 2*(y3(n) + 0.5*h*k1_3);
    k2_2 = 3*(y1(n) + 0.5*h*k1_1) + (y3(n) + 0.5*h*k1_3);
    k2_3 = (y1(n) + 0.5*h*k1_1) + 4*(y2(n) + 0.5*h*k1_2);

    k3_1 = (y2(n) + 0.5*h*k2_2) + 2*(y3(n) + 0.5*h*k2_3);
    k3_2 = 3*(y1(n) + 0.5*h*k2_1) + (y3(n) + 0.5*h*k2_3);
    k3_3 = (y1(n) + 0.5*h*k2_1) + 4*(y2(n) + 0.5*h*k2_2);

    k4_1 = (y2(n) + h*k3_2) + 2*(y3(n) + h*k3_3);
    k4_2 = 3*(y1(n) + h*k3_1) + (y3(n) + h*k3_3);
    k4_3 = (y1(n) + h*k3_1) + 4*(y2(n) + h*k3_2);

    y1(n+1) = y1(n) + h/6 * (k1_1 + 2*k2_1 + 2*k3_1 + k4_1);
    y2(n+1) = y2(n) + h/6 * (k1_2 + 2*k2_2 + 2*k3_2 + k4_2);
    y3(n+1) = y3(n) + h/6 * (k1_3 + 2*k2_3 + 2*k3_3 + k4_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations linéaires couplées - RK4');
```

Exercice 4 : Système de Trois Équations Non Linéaires Couplées (RK4)

Équations :

$$\frac{dy_1}{dt} = y_2 - y_1^2 + y_3$$
$$\frac{dy_2}{dt} = y_1 y_2 - y_3^2$$
$$\frac{dy_3}{dt} = y_1 + y_2^2$$

Conditions Initiales : $y_1(0) = 1, y_2(0) = 2, y_3(0) = 0.$

Objectif : Utiliser RK4 pour approximer la solution sur l'intervalle $t \in [0, 2]$ avec un pas de temps $h = 0.2$.

Solution :

1. **Initialisation :** $t_0 = 0, y_{1,0} = 1, y_{2,0} = 2, y_{3,0} = 0, h = 0.2.$
2. **Calcul des valeurs successives** avec les étapes RK4 pour chaque équation.

Programme Matlab

```
% Exercice 4 : Système de trois équations non linéaires couplées (RK4)
t0 = 0;
y1_0 = 1;
y2_0 = 2;
y3_0 = 0;
h = 0.2;
t = t0:h:2;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y2(n) - y1(n)^2 + y3(n);
    k1_2 = y1(n)*y2(n) - y3(n)^2;
    k1_3 = y1(n) + y2(n)^2;

    k2_1 = (y2(n) + 0.5*h*k1_2) - (y1(n) + 0.5*h*k1_1)^2 + (y3(n) + 0.5*h*k1_3);
    k2_2 = (y1(n) + 0.5*h*k1_1)*(y2(n) + 0.5*h*k1_2) - (y3(n) + 0.5*h*k1_3)^2;
    k2_3 = (y1(n) + 0.5*h*k1_1) + (y2(n) + 0.5*h*k1_2)^2;

    k3_1 = (y2(n) + 0.5*h*k2_2) - (y1(n) + 0.5*h*k2_1)^2 + (y3(n) + 0.5*h*k2_3);
    k3_2 = (y1(n) + 0.5*h*k2_1)*(y2(n) + 0.5*h*k2_2) - (y3(n) + 0.5*h*k2_3)^2;
    k3_3 = (y1(n) + 0.5*h*k2_1) + (y2(n) + 0.5*h*k2_2)^2;
```

```
    k4_1 = (y2(n) + h*k3_2) - (y1(n) + h*k3_1)^2 + (y3(n) + h*k3_3);
    k4_2 = (y1(n) + h*k3_1)*(y2(n) + h*k3_2) - (y3(n) + h*k3_3)^2;
    k4_3 = (y1(n) + h*k3_1) + (y2(n) + h*k3_2)^2;

    y1(n+1) = y1(n) + h/6 * (k1_1 + 2*k2_1 + 2*k3_1 + k4_1);
    y2(n+1) = y2(n) + h/6 * (k1_2 + 2*k2_2 + 2*k3_2 + k4_2);
    y3(n+1) = y3(n) + h/6 * (k1_3 + 2*k2_3 + 2*k3_3 + k4_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations non linéaires couplées - RK4');
```

Exercice 5 : Système de Trois Équations Différentielles Couplées avec Interaction Non Linéaire (RK4)

Équations :

$$\frac{dy_1}{dt} = y_1 y_2 + y_3$$

$$\frac{dy_2}{dt} = y_1 - y_2^2 + y_3$$

$$\frac{dy_3}{dt} = y_1^2 - y_2$$

Conditions Initiales : $y_1(0) = 1, y_2(0) = 1, y_3(0) = 1.$

Objectif : Utiliser RK4 pour approximer la solution sur l'intervalle $t \in [0, 2]$ avec un pas de temps $h = 0.2$.

Solution :

1. Initialisation : $t_0 = 0, y_{1,0} = 1, y_{2,0} = 1, y_{3,0} = 1, h = 0.2.$
2. Calcul des valeurs successives avec les étapes RK4 pour chaque équation.

Programme MATLAB :

```
% Exercice 5 : Système de trois équations différentielles couplées avec interaction non linéaire (RK4)
t0 = 0;
y1_0 = 1;
y2_0 = 1;
y3_0 = 1;
```

```
h = 0.2;
t = t0:h:2;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y1(n)*y2(n) + y3(n);
    k1_2 = y1(n) - y2(n)^2 + y3(n);
    k1_3 = y1(n)^2 - y2(n);

    k2_1 = (y1(n) + 0.5*h*k1_1)*(y2(n) + 0.5*h*k1_2) + (y3(n) + 0.5*h*k1_3);
    k2_2 = (y1(n) + 0.5*h*k1_1) - (y2(n) + 0.5*h*k1_2)^2 + (y3(n) + 0.5*h*k1_3);
    k2_3 = (y1(n) + 0.5*h*k1_1)^2 - (y2(n) + 0.5*h*k1_2);

    k3_1 = (y1(n) + 0.5*h*k2_1)*(y2(n) + 0.5*h*k2_2) + (y3(n) + 0.5*h*k2_3);
    k3_2 = (y1(n) + 0.5*h*k2_1) - (y2(n) + 0.5*h*k2_2)^2 + (y3(n) + 0.5*h*k2_3);
    k3_3 = (y1(n) + 0.5*h*k2_1)^2 - (y2(n) + 0.5*h*k2_2);

    k4_1 = (y1(n) + h*k3_1)*(y2(n) + h*k3_2) + (y3(n) + h*k3_3);
    k4_2 = (y1(n) + h*k3_1) - (y2(n) + h*k3_2)^2 + (y3(n) + h*k3_3);
    k4_3 = (y1(n) + h*k3_1)^2 - (y2(n) + h*k3_2);

    y1(n+1) = y1(n) + h/6 * (k1_1 + 2*k2_1 + 2*k3_1 + k4_1);
    y2(n+1) = y2(n) + h/6 * (k1_2 + 2*k2_2 + 2*k3_2 + k4_2);
    y3(n+1) = y3(n) + h/6 * (k1_3 + 2*k2_3 + 2*k3_3 + k4_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations différentielles couplées avec interaction non linéaire - RK4');
```

Exercice 6 : Système de Trois Équations Différentielles Non Linéaires Complexes (RK4)

Équations :

$$\frac{dy_1}{dt} = y_2^2 - y_1 y_3 + \sin(y_1)$$

$$\frac{dy_2}{dt} = y_1^2 - y_2 + y_3^2$$

$$\frac{dy_3}{dt} = y_1 y_2 - \cos(y_3)$$

Conditions Initiales : $y_1(0) = 0, y_2(0) = 1, y_3(0) = 2.$

Objectif : Utiliser RK4 pour approximer la solution sur l'intervalle $t \in [0, 1]$ avec un pas de temps $h = 0.1$.

Solution :

1. Initialisation : $t_0 = 0, y_{1,0} = 0, y_{2,0} = 1, y_{3,0} = 2, h = 0.1.$
2. Calcul des valeurs successives avec les étapes RK4 pour chaque équation.

Programme MATLAB :

```
% Exercice 6 : Système de trois équations différentielles non linéaires complexes (RK4)

t0 = 0;
y1_0 = 0;
y2_0 = 1;
y3_0 = 2;
h = 0.1;
t = t0:h:1;
y1 = zeros(size(t));
y2 = zeros(size(t));
y3 = zeros(size(t));
y1(1) = y1_0;
y2(1) = y2_0;
y3(1) = y3_0;

for n = 1:length(t)-1
    k1_1 = y2(n)^2 - y1(n)*y3(n) + sin(y1(n));
    k1_2 = y1(n)^2 - y2(n) + y3(n)^2;
    k1_3 = y1(n)*y2(n) - cos(y3(n));
```

```
    k2_1 = (y2(n) + 0.5*h*k1_2)^2 - (y1(n) + 0.5*h*k1_1)*(y3(n) + 0.5*h*k1_3) + sin(y1(n)
+ 0.5*h*k1_1);
    k2_2 = (y1(n) + 0.5*h*k1_1)^2 - (y2(n) + 0.5*h*k1_2) + (y3(n) + 0.5*h*k1_3)^2;
    k2_3 = (y1(n) + 0.5*h*k1_1)*(y2(n) + 0.5*h*k1_2) - cos(y3(n) + 0.5*h*k1_3);

    k3_1 = (y2(n) + 0.5*h*k2_2)^2 - (y1(n) + 0.5*h*k2_1)*(y3(n) + 0.5*h*k2_3) + sin(y1(n)
+ 0.5*h*k2_1);
    k3_2 = (y1(n) + 0.5*h*k2_1)^2 - (y2(n) + 0.5*h*k2_2) + (y3(n) + 0.5*h*k2_3)^2;
    k3_3 = (y1(n) + 0.5*h*k2_1)*(y2(n) + 0.5*h*k2_2) - cos(y3(n) + 0.5*h*k2_3);

    k4_1 = (y2(n) + h*k3_2)^2 - (y1(n) + h*k3_1)*(y3(n) + h*k3_3) + sin(y1(n) + h*k3_1);
    k4_2 = (y1(n) + h*k3_1)^2 - (y2(n) + h*k3_2) + (y3(n) + h*k3_3)^2;
    k4_3 = (y1(n) + h*k3_1)*(y2(n) + h*k3_2) - cos(y3(n) + h*k3_3);

    y1(n+1) = y1(n) + h/6 * (k1_1 + 2*k2_1 + 2*k3_1 + k4_1);
    y2(n+1) = y2(n) + h/6 * (k1_2 + 2*k2_2 + 2*k3_2 + k4_2);
    y3(n+1) = y3(n) + h/6 * (k1_3 + 2*k2_3 + 2*k3_3 + k4_3);
end

plot(t, y1, '-o', t, y2, '-x', t, y3, '-.');
xlabel('t');
ylabel('y');
legend('y1', 'y2', 'y3');
title('Système de trois équations différentielles non linéaires complexes - RK4');
```

Conclusion

Les méthodes de Runge-Kutta d'ordre 2 et 4 sont des outils puissants pour la résolution numérique de systèmes d'équations différentielles couplées, qu'ils soient linéaires ou non linéaires. En utilisant les six exercices ci-dessus, les étudiants peuvent se familiariser avec la mise en œuvre de ces méthodes et comprendre leur utilité dans les applications pratiques. Les programmes MATLAB fournis permettent de visualiser facilement les solutions et de confirmer la précision des approximations obtenues.

.

- .

Chapitre II : Equations aux dérivées partielles (EDP)

1. Introduction et classifications des problèmes aux dérivées partielles et des conditions aux limites;
2. Méthodes de résolution des EDP: Méthode des différences finies (MDF)

3. Méthode des volumes finis (MVF);

4. Méthode des éléments finis (MEF).

Chapitre II : Equations aux dérivées partielles (EDP)

Introduction :

Les équations aux dérivées partielles (EDP) sont des outils mathématiques essentiels utilisés pour décrire une grande variété de phénomènes physiques, tels que la diffusion de la chaleur, les vibrations des membranes, la propagation des ondes, et bien d'autres encore. Ces équations impliquent des dérivées partielles de fonctions inconnues par rapport à plusieurs variables indépendantes.

1. Introduction aux Équations aux Dérivées Partielles (EDP)

Une équation aux dérivées partielles est une équation qui contient des dérivées partielles d'une

$$F\left(x_1, x_2, \ldots, x_n, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial^2 u}{\partial x_1^2}, \ldots\right) = 0$$

Où :

- $x_1, x_2, \ldots, x_n$ sont les variables indépendantes.
- $u = u(x_1, x_2, \ldots, x_n)$ est la fonction inconnue.
- $\frac{\partial u}{\partial x_i}$ sont les dérivées partielles premières de u.
- $\frac{\partial^2 u}{\partial x_i^2}$ sont les dérivées partielles secondes de u.

Les EDP jouent un rôle crucial dans la modélisation mathématique des systèmes dynamiques et dans la résolution de problèmes en sciences appliquées et en ingénierie.

2. Classification des Équations aux Dérivées Partielles

Les EDP sont classées selon plusieurs critères, dont l'ordre de l'équation, la linéarité, et le type de l'équation (elliptique, parabolique, hyperbolique).

2.1Classification selon l'ordre :

2.2 Classification selon la linéarité

- **Équations linéaires** : Si u et ses dérivées apparaissent de manière linéaire (pas de termes comme u^2, $\sin(u)$, etc.).

$$a(x,y)\frac{\partial^2 u}{\partial x^2} + b(x,y)\frac{\partial u}{\partial x} + c(x,y)u = d(x,y)$$

- **Équations non linéaires** : Si u ou ses dérivées apparaissent de manière non linéaire.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \quad \text{(Équation de Burgers)}$$

2.3 Classification selon le type (pour les EDP du second ordre)

La classification selon le type repose sur la forme de l'équation caractéristique associée, et cela dépend des valeurs des coefficients devant les termes des dérivées secondes. Supposons l'équation générale du second ordre :

$$a\frac{\partial^2 u}{\partial x^2} + 2b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + \ldots = 0$$

Le discriminant $\Delta = b^2 - ac$ détermine le type d'EDP :

1. **Équations elliptiques** ($\Delta < 0$) : Caractérisent des phénomènes stationnaires et sont souvent associées à des problèmes de valeurs limites. Exemple : L'équation de Laplace.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

2. **Équations paraboliques** ($\Delta = 0$) : Modélisent des processus de diffusion ou de chaleur. Exemple : L'équation de la chaleur.

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2}\right)$$

3. **Équations hyperboliques** ($\Delta > 0$) : Décrivent des phénomènes de propagation d'ondes. Exemple : L'équation des ondes.

$$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2}\right)$$

3. Conditions aux Limites

Pour résoudre une EDP, il est crucial de spécifier les conditions aux limites. Les conditions aux limites définissent les valeurs de la solution ou de ses dérivées sur la frontière du domaine de définition.

3.1 Types de Conditions aux Limites

1. **Condition de Dirichlet** : Spécifie les valeurs de la fonction sur la frontière.

$$u(x,y) = g(x,y) \quad \text{sur } \partial\Omega$$

2. **Condition de Neumann** : Spécifie les valeurs de la dérivée normale de la fonction sur la frontière.

$$\frac{\partial u}{\partial n} = h(x,y) \quad \text{sur } \partial\Omega$$

.

3. **Condition de Robin (ou mixte)** : Combinaison des conditions de Dirichlet et Neumann.

$$au + b\frac{\partial u}{\partial n} = c \quad \text{sur } \partial\Omega$$

3.2 Importance des Conditions aux Limites

- **Physique du Problème** : Les conditions aux limites reflètent les contraintes physiques et les comportements aux frontières du système étudié.
- **Unicité et Existence des Solutions** : Les conditions aux limites sont essentielles pour garantir l'unicité et l'existence des solutions des EDP. Sans conditions appropriées, une EDP pourrait avoir plusieurs solutions ou aucune solution.

Conclusion

Les équations aux dérivées partielles et les conditions aux limites jouent un rôle fondamental dans la modélisation des phénomènes physiques et des systèmes dynamiques. La classification des EDP selon leur ordre, leur linéarité et leur type, ainsi que l'application de conditions aux limites appropriées, sont des étapes essentielles pour analyser et résoudre ces équations. Ces concepts sont cruciaux pour des domaines variés comme la mécanique des fluides, l'électromagnétisme, la thermodynamique, et bien d'autres. La compréhension approfondie des EDP et des conditions aux limites est donc indispensable pour les scientifiques et les ingénieurs qui travaillent avec des modèles mathématiques complexes.

Applications :

Exemples Pratiques et Cas Réels de Classification des Problèmes aux Dérivées Partielles et des Conditions aux Limites

Les équations aux dérivées partielles (EDP) sont fondamentales pour modéliser divers phénomènes physiques et ingénierie. La classification des EDP et l'application des conditions aux limites sont cruciales pour leur résolution. Dans cette section, nous allons explorer des exemples pratiques et réels pour chaque type de classification des EDP (elliptique, parabolique, hyperbolique) et illustrer comment les conditions aux limites influencent la solution. Pour chaque type, trois exemples seront fournis avec des programmes Matlab.

1. Équations Elliptiques

Les équations elliptiques modélisent des phénomènes stationnaires tels que la distribution du potentiel électrique, la déformation élastique d'une plaque, ou la distribution de température en régime stationnaire.

Exemple 1: Équation de Laplace (Problème de Conductivité Thermique)

Exemple 1: Équation de Laplace (Problème de Conductivité Thermique)

Problème: Trouver la distribution de température $u(x, y)$ dans une plaque rectangulaire, où les bords de la plaque sont maintenus à une température constante.

EDP:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Conditions aux limites:

- $u(0, y) = 0$: Température sur le bord gauche
- $u(L, y) = 0$: Température sur le bord droit
- $u(x, 0) = 0$: Température sur le bord inférieur
- $u(x, H) = 100$: Température sur le bord supérieur

Programme Matlab:

```
% Dimensions de la plaque
L = 10; % Longueur
H = 5; % Hauteur
% Grille
nx = 50;
ny = 50;
x = linspace(0, L, nx);
y = linspace(0, H, ny);
dx = x(2) - x(1);
dy = y(2) - y(1);

% Initialisation
u = zeros(nx, ny);

% Conditions aux limites
u(:, end) = 100; % Bord supérieur

% Résolution par méthode de Gauss-Seidel
tolerance = 1e-4;
error = 1;
while error > tolerance
   u_old = u;
   for i = 2:nx-1
      for j = 2:ny-1
         u(i, j) = 0.25 * (u(i+1, j) + u(i-1, j) + u(i, j+1) + u(i, j-1));
      end
```

```
    end
    error = max(max(abs(u - u_old)));
end

% Affichage du résultat
surf(x, y, u')
xlabel('x')
ylabel('y')
zlabel('Temperature')
title('Distribution de Température dans une Plaque')
```

Exemple 2: Équation de Poisson (Distribution de Potentiel Électrostatique)

Problème: Trouver le potentiel électrique $\phi(x, y)$ dans une région plane contenant une charge électrique.

EDP:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -\rho(x, y)$$

où $\rho(x, y)$ est la densité de charge.

Conditions aux limites:

- $\phi(0, y) = 0$: Potentiel sur le bord gauche
- $\phi(L, y) = 0$: Potentiel sur le bord droit
- $\phi(x, 0) = 0$: Potentiel sur le bord inférieur
- $\phi(x, H) = V_0$: Potentiel sur le bord supérieur

Programme Matlab:

```
% Dimensions
L = 10;
H = 5;
V0 = 100;
% Grille
nx = 50;
ny = 50;
x = linspace(0, L, nx);
y = linspace(0, H, ny);
dx = x(2) - x(1);
dy = y(2) - y(1);
% Initialisation
phi = zeros(nx, ny);
rho = ones(nx, ny); % Densité de charge uniforme
```

```
% Conditions aux limites
phi(:, end) = V0; % Bord supérieur
% Résolution par méthode de relaxation
tolerance = 1e-4;
error = 1;
while error > tolerance
   phi_old = phi;
   for i = 2:nx-1
      for j = 2:ny-1
         phi(i, j) = 0.25 * (phi(i+1, j) + phi(i-1, j) + phi(i, j+1) + phi(i, j-1) - dx^2 * rho(i,
j));
      end
   end
   error = max(max(abs(phi - phi_old)));
end
% Affichage du résultat
surf(x, y, phi')
xlabel('x')
ylabel('y')
zlabel('Potentiel')
title('Distribution de Potentiel Électrostatique')
```

Exemple 3: Problème de Membrane (Équation de Laplace)

Problème: Trouver la déformation $u(x, y)$ d'une membrane élastique sous tension constante.

EDP:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Conditions aux limites:

- $u(0, y) = 0$: Bord gauche fixé
- $u(L, y) = 0$: Bord droit fixé
- $u(x, 0) = 0$: Bord inférieur fixé
- $u(x, H) = f(x)$: Bord supérieur déformé selon une fonction $f(x)$

Programme Matlab:

```
% Dimensions de la membrane
L = 10;
H = 5;

% Grille
```

```
nx = 50;
ny = 50;
x = linspace(0, L, nx);
y = linspace(0, H, ny);
dx = x(2) - x(1);
dy = y(2) - y(1);

% Initialisation
u = zeros(nx, ny);

% Conditions aux limites
u(:, end) = sin(pi*x/L); % Bord supérieur, fonction f(x)

% Méthode de relaxation
tolerance = 1e-4;
error = 1;
while error > tolerance
    u_old = u;
    for i = 2:nx-1
        for j = 2:ny-1
            u(i, j) = 0.25 * (u(i+1, j) + u(i-1, j) + u(i, j+1) + u(i, j-1));
        end
    end
    error = max(max(abs(u - u_old)));
end

% Affichage du résultat
surf(x, y, u')
xlabel('x')
ylabel('y')
zlabel('Déformation')
title('Déformation d\'une Membrane Élastique')
```

2. Équations Paraboliques

Les équations paraboliques modélisent des phénomènes de diffusion tels que la diffusion de la chaleur ou de substances chimiques.

Exemple 1: Équation de la Chaleur (Diffusion de la Température)

Exemple 1: Équation de la Chaleur (Diffusion de la Température)

Problème: Trouver la distribution de température $u(x, t)$ dans une barre métallique isolée.

EDP:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Conditions aux limites:

- $u(0, t) = 0$: Température nulle à l'extrémité gauche
- $u(L, t) = 0$: Température nulle à l'extrémité droite
- $u(x, 0) = f(x)$: Distribution initiale de température

Programme Matlab:

```
% Paramètres
L = 10; % Longueur de la barre
T = 1; % Durée de la simulation
alpha = 0.01; % Diffusivité thermique

% Grille
nx = 50;
nt = 100;
x = linspace(0, L, nx);
t = linspace(0, T, nt);
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
u = zeros(nx, nt);
u(:, 1) = sin(pi*x/L); % Condition initiale

% Méthode explicite pour la diffusion de la chaleur
for n = 1:nt-1
   for i = 2:nx-1
      u(i, n+1) = u(i, n) + alpha*dt/dx^2 * (u(i+1, n) - 2*u(i, n) + u(i-1, n));
   end
end

% Affichage du résultat
surf(x, t, u')
xlabel('Position x')
```

```
ylabel('Temps t')
zlabel('Température')
title('Diffusion de la Température dans une Barre Métallique')
```

Exemple 2: Diffusion de Polluant (Transport de Masse)

Problème: Étudier la diffusion d'un polluant dans une rivière.

EDP:

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$$

Conditions aux limites:

- $C(0,t) = C_0$: Concentration initiale à la source
- $C(L,t) = 0$: Concentration nulle en aval
- $C(x,0) = 0$: Concentration initiale dans la rivière

Programme Matlab:

```
% Paramètres
L = 50; % Longueur de la rivière
T = 10; % Durée de la simulation
D = 0.1; % Coefficient de diffusion
% Grille
nx = 100;
nt = 200;
 = linspace(0, L, nx);
t = linspace(0, T, nt);
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
C = zeros(nx, nt);
C(1, :) = 1; % Concentration initiale à la source

% Méthode explicite pour la diffusion de polluant
for n = 1:nt-1
   for i = 2:nx-1
      C(i, n+1) = C(i, n) + D*dt/dx^2 * (C(i+1, n) - 2*C(i, n) + C(i-1, n));
   end
end

% Affichage du résultat
surf(x, t, C')
```

```
xlabel('Position x')
ylabel('Temps t')
zlabel('Concentration de Polluant')
title('Diffusion de Polluant dans une Rivière')
```

Exemple 3: Équation de Convection-Diffusion (Transfert de Chaleur et Masse)

Problème: Simuler le transfert de chaleur et de masse dans un fluide en mouvement.

EDP:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = D\frac{\partial^2 u}{\partial x^2}$$

Conditions aux limites:

- $u(0,t) = u_0$: Condition de température/concentration initiale à l'entrée
- $u(L,t) = 0$: Condition de température/concentration à la sortie
- $u(x,0) = 0$: Condition initiale dans le fluide

Programme Matlab:

```
% Paramètres
L = 10; % Longueur du domaine
T = 2; % Durée de la simulation
D = 0.1; % Diffusivité
v = 1; % Vitesse de convection

% Grille
nx = 50;
nt = 100;
x = linspace(0, L, nx);
t = linspace(0, T, nt);
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
u = zeros(nx, nt);
u(:, 1) = exp(-x); % Condition initiale

% Méthode explicite pour la convection-diffusion
for n = 1:nt-1
```

```
    for i = 2:nx-1
        u(i, n+1) = u(i, n) - v*dt/dx * (u(i, n) - u(i-1, n)) + D*dt/dx^2 * (u(i+1, n) - 2*u(i, n) + u(i-1, n));
    end
end

% Affichage du résultat
surf(x, t, u')
xlabel('Position x')
ylabel('Temps t')
zlabel('Concentration/Température')
title('Transfert de Chaleur et de Masse dans un Fluide en Mouvement')
```

3. Équations Hyperboliques

Les équations hyperboliques modélisent la propagation des ondes et des vibrations.

Exemple 1: Équation des Ondes (Vibrations d'une Corde)

Problème: Trouver les vibrations d'une corde tendue.

EDP:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

Conditions aux limites:

- $u(0, t) = 0$: Corde fixée à l'extrémité gauche
- $u(L, t) = 0$: Corde fixée à l'extrémité droite
- $u(x, 0) = \sin(\pi x/L)$: Déplacement initial
- $\frac{\partial u}{\partial t}(x, 0) = 0$: Vitesse initiale nulle

Programme Matlab:

```
% Paramètres
L = 1; % Longueur de la corde
T = 1; % Durée de la simulation
c = 1; % Vitesse de propagation des ondes

% Grille
nx = 100;
nt = 100;
x = linspace(0, L, nx);
t = linspace(0, T, nt);
```

```
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
u = zeros(nx, nt);
u(:, 1) = sin(pi*x/L); % Condition initiale de déplacement

% Méthode explicite pour l'équation des ondes
for n = 1:nt-1
   for i = 2:nx-1
      if n == 1
         u(i, n+1) = u(i, n) + 0.5 * (c*dt/dx)^2 * (u(i+1, n) - 2*u(i, n) + u(i-1, n));
      else
         u(i, n+1) = 2*u(i, n) - u(i, n-1) + (c*dt/dx)^2 * (u(i+1, n) - 2*u(i, n) + u(i-1, n));
      end
   end
end

% Affichage du résultat
surf(x, t, u')
xlabel('Position x')
ylabel('Temps t')
zlabel('Déplacement')
title('Vibrations d\'une Corde Tendue')
```

Exemple 2: Propagation des Ondes Sonores (Équation des Ondes Acoustiques)

Problème: Simuler la propagation des ondes sonores dans un milieu homogène.

EDP:

$$\frac{\partial^2 p}{\partial t^2} = c^2 \nabla^2 p$$

où p est la pression acoustique.

.**Conditions aux limites**:

- $p = 0$ sur les bords du domaine (condition de Dirichlet)

Programme Matlab:

```
% Paramètres
L = 1; % Longueur du domaine
T = 0.5; % Durée de la simulation
c = 343; % Vitesse du son dans l'air (m/s)
```

```
% Grille
nx = 100;
nt = 100;
x = linspace(0, L, nx);
t = linspace(0, T, nt);
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
p = zeros(nx, nt);
p(50, 1) = 1; % Impulsion initiale au centre

% Méthode explicite pour l'équation des ondes acoustiques
for n = 1:nt-1
   for i = 2:nx-1
      if n == 1
         p(i, n+1) = p(i, n) + 0.5 * (c*dt/dx)^2 * (p(i+1, n) - 2*p(i, n) + p(i-1, n));
      else
         p(i, n+1) = 2*p(i, n) - p(i, n-1) + (c*dt/dx)^2 * (p(i+1, n) - 2*p(i, n) + p(i-1, n));
      end
   end
end

% Affichage du résultat
surf(x, t, p')
xlabel('Position x')
ylabel('Temps t')
zlabel('Pression')
title('Propagation des Ondes Sonores')
```

Exemple 3: Équation de Transport Hyperbolique (Advection Pure)

Problème: Simuler le transport d'un scalair dans un fluide en mouvement.

EDP:

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0$$

Conditions aux limites:

- $u(0,t) = u_0$: Condition initiale à l'entrée
- $u(x,0) = g(x)$: Condition initiale dans le domaine

Programme Matlab:

```
% Paramètres
L = 10; % Longueur du domaine
T = 2; % Durée de la simulation
v = 2; % Vitesse de transport

% Grille
nx = 50;
nt = 100;
x = linspace(0, L, nx);
t = linspace(0, T, nt);
dx = x(2) - x(1);
dt = t(2) - t(1);

% Initialisation
u = zeros(nx, nt);
u(:, 1) = exp(-((x-5)/1).^2); % Condition initiale

% Méthode explicite pour l'équation de transport
for n = 1:nt-1
   for i = 2:nx-1
      u(i, n+1) = u(i, n) - v*dt/dx * (u(i, n) - u(i-1, n));
   end
end

% Affichage du résultat
surf(x, t, u')
xlabel('Position x')
ylabel('Temps t')
zlabel('Concentration')
title('Transport d\'un Scalaire dans un Fluide')
```

Conclusion

Ces exemples détaillés montrent comment les équations aux dérivées partielles de différents types peuvent modéliser divers phénomènes physiques. Les conditions aux limites sont essentielles pour définir correctement les problèmes et garantir des solutions réalistes. Les programmes Matlab fournis illustrent des approches pratiques pour résoudre ces problèmes par des méthodes numériques, offrant un aperçu précieux pour les ingénieurs et scientifiques travaillant avec des EDP dans des applications réelles.

B : Résolution des Équations aux Dérivées Partielles

Les équations aux dérivées partielles (EDP) apparaissent fréquemment dans la modélisation des phénomènes physiques, biologiques et financiers. Résoudre ces équations est crucial pour comprendre les comportements complexes des systèmes. Cet essai explore trois méthodes numériques principales pour résoudre les EDP : la Méthode des Différences Finies (MDF), la Méthode des Volumes Finis (MVF) et la Méthode des Éléments Finis (MEF). Chaque méthode sera présentée en détail avec des exemples et des exercices pour illustrer leur application pratique.

1. Méthode des Différences Finies

1.1. Introduction à la Méthode des Différences Finies

La Méthode des Différences Finies (MDF) est une technique numérique populaire pour résoudre les Équations aux Dérivées Partielles (EDP). Elle est utilisée pour transformer les EDP en systèmes d'équations algébriques, facilitant ainsi la résolution numérique. Cet essai explore en détail la MDF et propose des exercices pratiques avec des programmes MATLAB pour illustrer son application à différents types d'EDP.La MDF repose sur l'idée d'approximer les dérivées continues par des différences finies en utilisant des valeurs de la fonction à des points discrets sur une grille. En substituant ces approximations dans l'EDP, on obtient un système d'équations algébriques pouvant être résolu numériquement.

Approximation des Dérivées

Approximation des Dérivées

Les dérivées d'une fonction $u(x,t)$ sont approximées de la manière suivante :

- **Différence avant (Forward Difference)** pour la dérivée première :

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}$$

- **Différence arrière (Backward Difference)** pour la dérivée première :

$$\frac{\partial u}{\partial x} \approx \frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}$$

- **Différence centrée (Central Difference)** pour la dérivée première (plus précise) :

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x,t)-u(x-\Delta x,t)}{2\Delta x}$$

- **Différence centrée pour la dérivée seconde** :

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{\Delta x^2}$$

Ces approximations permettent de convertir les EDP en équations algébriques qui peuvent être résolues par des méthodes numériques.

.

1.2. Applications de la Méthode des Différences Finies

Exemple 1 : Résolution de l'Équation de la Chaleur en 1D

L'équation de la chaleur, qui modélise la diffusion de la chaleur dans un milieu, est donnée par :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

où $u(x,t)$ est la température à un point x à l'instant t, et α est le coefficient de diffusion thermique.

Discrétisation et Formulation Numérique

1. **Discrétisation de l'espace et du temps** : Supposons que L est la longueur de la barre, et divisons $[0, L]$ en N sous-intervalles de largeur $\Delta x = \frac{L}{N}$. Le temps est divisé en intervalles de Δt.
2. **Formulation discrète de l'équation de la chaleur** :

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

Ce qui donne la relation suivante pour la température au pas de temps suivant :

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

Programme MATLAB pour l'Équation de la Chaleur

Paramètres : Choisissons $L = 1$, $\alpha = 1$, $\Delta x = 0.1$, $\Delta t = 0.005$. Les conditions initiales sont $u(x,0) = \sin(\pi x)$.

```
% Paramètres
L = 1.0;          % Longueur de la barre
alpha = 1.0;      % Coefficient de diffusion thermique
dx = 0.1;         % Espacement spatial
dt = 0.005;       % Pas de temps
x = 0:dx:L;       % Grille spatiale
u = sin(pi * x);  % Condition initiale : température sinusoïdale
u(1) = 0;         % Condition aux limites gauche
u(end) = 0;       % Condition aux limites droite

% Boucle temporelle
n_steps = 100;    % Nombre d'étapes temporelles
for n = 1:n_steps
   u_new = u;     % Stocker les nouvelles valeurs de u
   for i = 2:length(x)-1
      u_new(i) = u(i) + alpha * dt / dx^2 * (u(i+1) - 2*u(i) + u(i-1));
   end
   u = u_new;     % Mettre à jour u
```

```
end

% Affichage des résultats
plot(x, u, 'b-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Distribution de température après 100 itérations');
grid on;
```

Exemple 2 : Résolution de l'Équation de la propagation de l'onde en 1D

L'équation de l onde en 1D est utilisée pour modéliser le déplacement des ondes sur une corde tendue :

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

où c est la vitesse de propagation de la vague.

Discrétisation et Formulation Numérique

1. **Discrétisation de l'espace et du temps** : Comme précédemment, divisons l'intervalle $[0, L]$ en N sous-intervalles et le temps en intervalles de Δt.
2. **Formulation discrète de l'équation de la vague** :

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

Ce qui donne :

$$u_i^{n+1} = 2u_i^n - u_i^{n-1} + \frac{c^2 \Delta t^2}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

Programme MATLAB pour l'Équation de la Vague

Paramètres : Choisissons $L = 1$, $c = 1$, $\Delta x = 0.1$, $\Delta t = 0.005$. Les conditions initiales sont $u(x, 0) = \sin(\pi x)$ et $\frac{\partial u}{\partial t}(x, 0) = 0$.

```
% Paramètres
L = 1.0;          % Longueur de la corde
c = 1.0;          % Vitesse de propagation de l'onde
dx = 0.1;         % Espacement spatial
dt = 0.005;       % Pas de temps
x = 0:dx:L;       % Grille spatiale
u = sin(pi * x);  % Condition initiale : déplacement sinusoïdal
u_old = u;        % Déplacement au temps précédent
u(1) = 0;         % Condition aux limites gauche
u(end) = 0;       % Condition aux limites droite

% Boucle temporelle
```

```
n_steps = 100;     % Nombre d'étapes temporelles
for n = 1:n_steps
   u_new = zeros(size(u));  % Stocker les nouvelles valeurs de u
   for i = 2:length(x)-1
      u_new(i) = 2*u(i) - u_old(i) + (c * dt / dx)^2 * (u(i+1) - 2*u(i) + u(i-1));
   end
   u_old = u;     % Mettre à jour u_old
   u = u_new;     % Mettre à jour u
end

% Affichage des résultats
plot(x, u, 'r-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Déplacement de la corde après 100 itérations');
grid on;
```

Exemple 3 : Résolution de l'Équation de Laplace en 2D

L'équation de Laplace modélise des phénomènes tels que le potentiel électrostatique et la diffusion stationnaire :

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Discrétisation et Formulation Numérique

1. **Discrétisation de l'espace** : Divisons le domaine $[0, L] \times [0, L]$ en une grille $N \times N$ avec des espacements $\Delta x = \Delta y = \frac{L}{N}$.
2. **Formulation discrète de l'équation de Laplace** :

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0$$

D'où :

$$u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{4}$$

Programme MATLAB pour l'Équation de Laplace

Paramètres : Choisissons $L = 1$, $\Delta x = \Delta y = 0.1$. Les conditions aux limites sont $u(x, 0) = u(x, L) = u(0, y) = u(L, y) = 0$, et une source de chaleur à $u(0.5, 0.5) = 100$.

```
.% Paramètres

L = 1.0;          % Taille du domaine
```

```
dx = 0.1;          % Espacement spatial
x = 0:dx:L;        % Grille spatiale en x
y = 0:dx:L;        % Grille spatiale en y
u = zeros(length(x), length(y));  % Initialisation de u
u(round(0.5/dx), round(0.5/dx)) = 100; % Source de chaleur

% Boucle itérative
n_steps = 1000;    % Nombre d'itérations
for n = 1:n_steps
   u_new = u;     % Stocker les nouvelles valeurs de u
   for i = 2:length(x)-1
      for j = 2:length(y)-1
         u_new(i, j) = 0.25 * (u(i+1, j) + u(i-1, j) + u(i, j+1) + u(i, j-1));
      end
   end
   u = u_new;     % Mettre à jour u
end

% Affichage des résultats
imagesc(x, y, u);
colorbar;
title('Solution de l''équation de Laplace après 1000 itérations');
xlabel('x');
ylabel('y');
```

En conclusion ,la Méthode des Différences Finies (MDF) est une technique efficace pour résoudre les Équations aux Dérivées Partielles (EDP) en transformant les dérivées en approximations discrètes. Grâce à des exemples concrets, nous avons démontré comment appliquer la MDF à divers types d'EDP, notamment l'équation de la chaleur, l'équation de la vague, et l'équation de Laplace. Ces exemples montrent comment la MDF peut être mise en œuvre facilement avec MATLAB, fournissant ainsi une approche robuste pour résoudre des problèmes de diffusion de chaleur, de propagation de vagues, et de potentiel stationnaire dans des contextes variés.

Chapitre II : Equations aux dérivées partielles (EDP)

B : Méthode des Volumes Finis (MVF)

2. Introduction à la Méthode des Volumes Finis (MVF)

La Méthode des Volumes Finis (MVF) est une technique numérique largement utilisée pour résoudre les Équations aux Dérivées Partielles (EDP), en particulier dans les problèmes de conservation comme la dynamique des fluides et le transfert de chaleur. La MVF divise le domaine en petits volumes (cellules) et applique un bilan de conservation à chaque volume, ce qui assure une conservation stricte des quantités telles que la masse, la chaleur ou la quantité de mouvement. Cet essai explore en détail la MVF et propose des exercices pratiques avec des programmes MATLAB pour illustrer son application.

La MVF repose sur le principe de conservation appliqué à des sous-domaines (volumes) du domaine de calcul. En appliquant des bilans de flux aux frontières de chaque volume, la méthode garantit que ce qui entre dans un volume sort dans les volumes voisins, assurant ainsi la conservation des quantités physiques.

Bilan de Conservation

Pour une équation de conservation générale sous la forme :

$$\frac{\partial \phi}{\partial t} + \nabla \cdot \mathbf{F} = 0$$

où ϕ représente la quantité conservée (par exemple, la densité ou l'énergie), et $\mathbf{F}$ est le flux, la MVF consiste à intégrer cette équation sur un volume de contrôle V avec une surface S :

$$\frac{d}{dt}\int_V \phi \, dV + \int_S \mathbf{F} \cdot d\mathbf{A} = 0$$

Cette équation assure que la variation de ϕ dans un volume est compensée par les flux à travers les surfaces du volume.

2.1. Applications de la Méthode des Volumes Finis

Exemple 1 : Équation de la Chaleur en 1D

Considérons l'équation de la chaleur en 1D, qui modélise la diffusion de la chaleur dans une barre :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

où $u(x,t)$ est la température, et α est le coefficient de diffusion thermique.

Discrétisation et Formulation Numérique

1. **Division du domaine en volumes finis** : Supposons que la barre de longueur L soit divisée en N volumes de contrôle égaux de largeur $\Delta x = \frac{L}{N}$. Le point médian de chaque volume est utilisé pour évaluer les quantités.
2. **Application du bilan de conservation** : Pour chaque volume de contrôle i, le flux de chaleur entrant et sortant est calculé. La différence de flux détermine la variation de température dans le volume.

   La forme discrétisée de l'équation est :

$$\frac{du_i}{dt} = \alpha \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

Programme MATLAB pour l'Équation de la Chaleur

Paramètres : Choisissons $L = 1$, $\alpha = 1$, $\Delta x = 0.1$, $\Delta t = 0.005$. Les conditions initiales sont $u(x,0) = \sin(\pi x)$.

```
% Paramètres

L = 1.0;          % Longueur de la barre
alpha = 1.0;      % Coefficient de diffusion thermique
dx = 0.1;         % Espacement spatial
dt = 0.005;       % Pas de temps
x = 0:dx:L;       % Grille spatiale
u = sin(pi * x);  % Condition initiale : température sinusoïdale
u(1) = 0;         % Condition aux limites gauche
u(end) = 0;       % Condition aux limites droite

% Boucle temporelle
n_steps = 100;    % Nombre d'étapes temporelles
for n = 1:n_steps
    u_new = u;    % Stocker les nouvelles valeurs de u
    for i = 2:length(x)-1
        u_new(i) = u(i) + alpha * dt / dx^2 * (u(i+1) - 2*u(i) + u(i-1));
    end
    u = u_new;    % Mettre à jour u
end
```

```
% Affichage des résultats
plot(x, u, 'b-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Distribution de température après 100 itérations');
grid on;
```

Exemple 2 : Équation de la Convection-Diffusion en 1D

Considérons une équation de convection-diffusion, où $u(x,t)$ représente une concentration transportée par un flux et diffusée :

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \alpha\frac{\partial^2 u}{\partial x^2}$$

Discrétisation et Formulation Numérique

1. **Division du domaine en volumes finis** : Comme dans l'exemple précédent, divisons le domaine $[0, L]$ en N volumes.
2. **Application du bilan de conservation** : Le flux total $\mathbf{F} = \mathbf{F}_{convection} + \mathbf{F}_{diffusion}$ est calculé à chaque frontière de volume.

   La forme discrétisée est :

$$\frac{du_i}{dt} + v\frac{u_{i+1} - u_{i-1}}{2\Delta x} = \alpha\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

Programme MATLAB pour l'Équation de Convection-Diffusion

Paramètres : Choisissons $L = 1$, $v = 1$, $\alpha = 0.01$, $\Delta x = 0.1$, $\Delta t = 0.005$. Les conditions initiales sont $u(x,0) = \exp(-100(x-0.5)^2)$.

```
% Paramètres
L = 1.0;          % Longueur du domaine
v = 1.0;          % Vitesse de convection
alpha = 0.01;     % Coefficient de diffusion
dx = 0.1;         % Espacement spatial
dt = 0.005;       % Pas de temps
x = 0:dx:L;       % Grille spatiale
u = exp(-100 * (x - 0.5).^2);  % Condition initiale : pic gaussien
u(1) = 0;         % Condition aux limites gauche
u(end) = 0;       % Condition aux limites droite
```

```
% Boucle temporelle
n_steps = 100;      % Nombre d'étapes temporelles
for n = 1:n_steps
   u_new = u;      % Stocker les nouvelles valeurs de u
   for i = 2:length(x)-1
      convection = -v * (u(i+1) - u(i-1)) / (2*dx);
      diffusion = alpha * (u(i+1) - 2*u(i) + u(i-1)) / dx^2;
      u_new(i) = u(i) + dt * (convection + diffusion);
   end
   u = u_new;      % Mettre à jour u
end

% Affichage des résultats
plot(x, u, 'r-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Solution de l''équation de convection-diffusion après 100 itérations');
grid on;
```

Exemple 3 : Écoulement Compressible en 2D

L'équation de continuité pour un fluide compressible est donnée par :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

où ρ est la densité du fluide, et $\mathbf{u}$ est la vitesse du fluide.

Discrétisation et Formulation Numérique

1. **Division du domaine en volumes finis** : Divisons le domaine en une grille régulière de volumes de contrôle.

2. **Application du bilan de conservation** : Pour chaque volume, le flux de masse est calculé aux interfaces.

   La forme discrétisée est :

$$\frac{d\rho_i}{dt} + \frac{(\rho \mathbf{u})_{i+1/2} - (\rho \mathbf{u})_{i-1/2}}{\Delta x} = 0$$

Programme MATLAB pour l'Écoulement Compressible

Paramètres : Choisissons un domaine $[0, 1] \times [0, 1]$ avec $\rho = 1 + 0.5 \sin(2\pi x)$ et $\mathbf{u} = (1, 0)$.

```
% Paramètres
L = 1.0;          % Taille du domaine
dx = 0.1;         % Espacement spatial
```

```
x = 0:dx:L;        % Grille spatiale en x
y = 0:dx:L;        % Grille spatiale en y
[X, Y] = meshgrid(x, y);
rho = 1 + 0.5 * sin(2 * pi * X);  % Densité initiale
u = ones(size(rho));              % Vitesse en x (constante)
v = zeros(size(rho));             % Vitesse en y (constante)

% Boucle temporelle
dt = 0.01;         % Pas de temps
n_steps = 100;     % Nombre d'étapes temporelles
for n = 1:n_steps
   rho_new = rho;  % Stocker les nouvelles valeurs de densité
   for i = 2:length(x)-1
      for j = 2:length(y)-1
         flux_in = rho(i-1, j) * u(i-1, j) + rho(i, j-1) * v(i, j-1);
         flux_out = rho(i, j) * u(i, j) + rho(i, j) * v(i, j);
         rho_new(i, j) = rho(i, j) - dt * (flux_out - flux_in) / dx;
      end
   end
   rho = rho_new;  % Mettre à jour rho
end

% Affichage des résultats
imagesc(x, y, rho);
colorbar;
title('Distribution de densité après 100 itérations');
xlabel('x');
ylabel('y');
```

En conclusion, la Méthode des Volumes Finis (MVF) est une technique numérique puissante pour résoudre les Équations aux Dérivées Partielles (EDP) dans des systèmes complexes, notamment ceux impliquant des principes de conservation. En appliquant des bilans de flux aux frontières de chaque volume de contrôle, la MVF assure une conservation stricte des quantités physiques et est particulièrement efficace pour des problèmes tels que le transfert de chaleur, la convection-diffusion et les écoulements de fluides compressibles. Les exemples illustrés par des programmes MATLAB montrent comment implémenter la MVF pour divers types d'EDP, fournissant une base solide pour résoudre des problèmes de conservation dans divers contextes scientifiques et d'ingénierie.

Chapitre II : Equations aux dérivées partielles (EDP)

C : La Méthode des Éléments Finis (MEF)

3.1. Introduction à la Méthode des Éléments Finis (MEF)

La Méthode des Éléments Finis (MEF) est une technique numérique puissante et flexible largement utilisée pour résoudre les Équations aux Dérivées Partielles (EDP). Elle est particulièrement efficace pour des problèmes avec des géométries complexes et des conditions aux limites variées. La MEF divise le domaine en petits sous-domaines appelés éléments, utilise des fonctions de forme pour approximer la solution et transforme les EDP en systèmes d'équations algébriques.

Cet essai explore la MEF en détail et propose des exercices pratiques avec des programmes MATLAB pour illustrer son application à différents types d'EDP.La MEF repose sur l'approximation de la solution d'une EDP sur un domaine complexe par des fonctions plus simples définies localement sur des sous-domaines (éléments) du domaine. Cette approche conduit à un système d'équations linéaires qui peut être résolu numériquement.

Principe de la MEF

1. **Discrétisation du Domaine** : Le domaine est divisé en un ensemble d'éléments finis. Chaque élément est associé à un certain nombre de nœuds, qui sont les points de calcul.
2. **Fonctions de Forme** : Les solutions sont approximées par des fonctions de forme définies localement sur chaque élément. Les fonctions de forme interpolent les valeurs aux nœuds de chaque élément.
3. **Formulation Variationnelle** : La MEF utilise une formulation varitionnelle ou faible de l'EDP. Cette approche consiste à intégrer le produit de l'EDP et d'une fonction de test sur le domaine.
4. **Assemblage du Système** : Les contributions locales de chaque élément sont assemblées pour former un système global d'équations linéaires.

Formulation Variationnelle

Soit une EDP sous la forme générale :

Soit une EDP sous la forme générale :

$$-\nabla \cdot (\kappa \nabla u) = f \text{ dans } \Omega$$

avec des conditions aux limites de Dirichlet $u = 0$ sur $\partial\Omega$. La formulation variationnelle est :

$$\int_{\Omega} \kappa \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} f v \, d\Omega$$

pour toute fonction de test v.

3.1. Applications de la Méthode des Éléments Finis

Exemple 1 : Résolution de l'Équation de Poisson en 1D

Exemple 1 : Résolution de l'Équation de Poisson en 1D

L'équation de Poisson en 1D est une EDP fondamentale utilisée pour modéliser des phénomènes tels que le potentiel électrostatique et la diffusion de chaleur stationnaire :

$$-\frac{d^2u}{dx^2} = f(x)$$

où $f(x)$ est une source de chaleur, et $u(x)$ est la température ou le potentiel.

Discrétisation et Formulation Numérique

1. **Discrétisation du domaine** : Divisons l'intervalle $[0, 1]$ en N éléments égaux. Chaque élément a deux nœuds.
2. **Fonctions de forme** : Utilisons des fonctions de forme linéaires sur chaque élément. Pour un élément e, les fonctions de forme sont définies comme :

   $$\phi_1(x) = 1 - \frac{x - x_1}{h}, \quad \phi_2(x) = \frac{x - x_1}{h}$$

   où x_1 et x_2 sont les nœuds de l'élément, et h est la longueur de l'élément.
3. **Formulation variationnelle** : La formulation variationnelle de l'équation de Poisson devient :

   $$\int_0^1 \frac{du}{dx}\frac{dv}{dx} \, dx = \int_0^1 f v \, dx$$

   Discrétisée en :

   $$KU = F$$

   où K est la matrice de rigidité et F est le vecteur de charge.

Programme MATLAB pour l'Équation de Poisson en 1D

Paramètres : Choisissons $N = 10$ éléments, $f(x) = 1$.

```
% Paramètres
N = 10;          % Nombre d'éléments
L = 1.0;         % Longueur du domaine
h = L / N;       % Taille de chaque élément
x = linspace(0, L, N+1); % Coordonnées des nœuds
```

```
f = ones(N+1, 1);  % Source de chaleur (constante)

% Matrice de rigidité et vecteur de charge
K = zeros(N+1, N+1);
F = zeros(N+1, 1);

% Assemblage du système
for i = 1:N
    K(i,i) = K(i,i) + 1/h;
    K(i,i+1) = K(i,i+1) - 1/h;
    K(i+1,i) = K(i+1,i) - 1/h;
    K(i+1,i+1) = K(i+1,i+1) + 1/h;

    F(i) = F(i) + f(i) * h / 2;
    F(i+1) = F(i+1) + f(i+1) * h / 2;
end

% Appliquer les conditions aux limites
K(1,:) = 0; K(1,1) = 1; F(1) = 0;
K(end,:) = 0; K(end,end) = 1; F(end) = 0;

% Résolution du système
U = K\F;

% Affichage des résultats
plot(x, U, 'b-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Solution de l''équation de Poisson en 1D');
grid on;
```

Exemple 2 : Résolution de l'Équation de Laplace en 2D

Chapitre II : Equations aux dérivées partielles (EDP)

L'équation de Laplace en 2D est utilisée pour modéliser des phénomènes tels que le potentiel

$$-\Delta u = 0$$

Discrétisation et Formulation Numérique

1. **Discrétisation du domaine** : Divisons un carré unité $[0, 1] \times [0, 1]$ en une grille régulière de $N \times N$ éléments.
2. **Fonctions de forme** : Utilisons des fonctions de forme bilinéaires sur chaque élément rectangulaire.
3. **Formulation variationnelle** : La formulation variationnelle devient :

$$\int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = 0$$

Discrétisée en :

$$KU = F$$

Programme MATLAB pour l'Équation de Laplace en 2D

Paramètres : Choisissons une grille de 10×10 éléments, $u = 0$ sur le bord.

```
.% Paramètres

N = 10;          % Nombre d'éléments par dimension
L = 1.0;         % Longueur du domaine
h = L / N;       % Taille de chaque élément
x = linspace(0, L, N+1); % Coordonnées des nœuds en x
y = linspace(0, L, N+1); % Coordonnées des nœuds en y
[X, Y] = meshgrid(x, y);

% Matrice de rigidité et vecteur de charge
K = zeros((N+1)^2, (N+1)^2);
F = zeros((N+1)^2, 1);

% Assemblage du système
for i = 1:N
   for j = 1:N
      node = @(i, j) (i-1)*(N+1) + j; % Numérotation des nœuds
      nodes = [node(i, j), node(i, j+1), node(i+1, j+1), node(i+1, j)]; % Nœuds de l'élément

      Ke = (1/h) * [1 -1 0 0; -1 1 0 0; 0 0 1 -1; 0 0 -1 1]; % Matrice locale
      for a = 1:4
         for b = 1:4
            K(nodes(a), nodes(b)) = K(nodes(a), nodes(b)) + Ke(a, b);
         end
      end
   end
```

```
end

% Appliquer les conditions aux limites
for i = 1:N+1
    for j = 1:N+1
        if i == 1 || j == 1 || i == N+1 || j == N+1
            idx = (i-1)*(N+1) + j;
            K(idx, :) = 0;
            K(idx, idx) = 1;
            F(idx) = 0;
        end
    end
end

% Résolution du système
U = K\F;

% Affichage des résultats
U_grid = reshape(U, [N+1, N+1]);
surf(X, Y, U_grid);
xlabel('x');
ylabel('y');
zlabel('u');
title('Solution de l''équation de Laplace en 2D');
```

Exemple 3 : Équation de la Chaleur en 2D

L'équation de la chaleur en 2D modélise la diffusion de chaleur dans un plan :

$$\frac{\partial u}{\partial t} = \alpha \Delta u$$

où α est le coefficient de diffusion.

Discrétisation et Formulation Numérique

1. **Discrétisation du domaine et du temps** : Divisons le carré unité $[0, 1] \times [0, 1]$ en une grille régulière et utilisons un pas de temps Δt.
2. **Fonctions de forme** : Utilisons des fonctions de forme bilinéaires.
3. **Formulation variationnelle** : La formulation variationnelle discrète est :

$$M \frac{dU}{dt} + KU = F$$

où M est la matrice de masse et K est la matrice de rigidité.

Programme MATLAB pour l'Équation de la Chaleur en 2D

Paramètres : Choisissons une grille de 10×10 éléments, $\alpha = 0.01$, $\Delta t = 0.01$.

```
% Paramètres
N = 10;          % Nombre d'éléments par dimension
```

```
L = 1.0;          % Longueur du domaine
h = L / N;        % Taille de chaque élément
alpha = 0.01;     % Coefficient de diffusion
dt = 0.01;        % Pas de temps
x = linspace(0, L, N+1); % Coordonnées des nœuds en x
y = linspace(0, L, N+1); % Coordonnées des nœuds en y
[X, Y] = meshgrid(x, y);

% Initialisation des matrices et vecteurs
K = zeros((N+1)^2, (N+1)^2);
M = zeros((N+1)^2, (N+1)^2);
F = zeros((N+1)^2, 1);

% Assemblage du système (similaire à l'équation de Laplace)

% Appliquer les conditions aux limites (similaire à l'équation de Laplace)

% Boucle temporelle
n_steps = 100;    % Nombre d'étapes temporelles
U = zeros((N+1)^2, 1); % Température initiale
for n = 1:n_steps
    U = (M + dt*K) \ (M*U + dt*F); % Mise à jour de la solution
end

% Affichage des résultats
U_grid = reshape(U, [N+1, N+1]);
surf(X, Y, U_grid);
xlabel('x');
ylabel('y');
zlabel('u');
title('Distribution de température après 100 itérations');
```

En conlusion, la Méthode des Éléments Finis (MEF) est une approche flexible et puissante pour résoudre les Équations aux Dérivées Partielles (EDP), particulièrement adaptée aux problèmes avec des géométries complexes et des conditions aux limites variées. En utilisant des exemples concrets et des programmes MATLAB, nous avons démontré comment la MEF peut être appliquée pour résoudre les équations de Poisson, de Laplace, et de la chaleur. La MEF offre une méthode robuste pour traiter des problèmes scientifiques et d'ingénierie dans divers domaines, fournissant une approximation numérique précise des solutions aux EDP complexes.

Travaux dirigés

A :Méthode des Différences Finies (MDF)

La MDF repose sur l'approximation des dérivées par des différences finies. Pour une fonctionu(x), les dérivées peuvent être approximées comme suit :

- **Différence avant (Forward Difference)** pour la dérivée première :

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x) - u(x)}{\Delta x}$$

- **Différence arrière (Backward Difference)** pour la dérivée première :

$$\frac{\partial u}{\partial x} \approx \frac{u(x) - u(x-\Delta x)}{\Delta x}$$

- **Différence centrée (Central Difference)** pour la dérivée première :

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x) - u(x-\Delta x)}{2\Delta x}$$

- **Différence centrée pour la dérivée seconde** :

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2}$$

Exercice 4: Équation de Convection-Diffusion en 1D

Problème: Résoudre $\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \alpha\frac{\partial^2 u}{\partial x^2}$ sur $[0, 1]$ avec $u(0, t) = 0$, $u(1, t) = 0$ et $u(x, 0) = \sin(\pi x)$.

Solution:

1. **Discrétisation** : $v = 1, \alpha = 0.01, N = 20$.
2. **Approximation** : Utiliser des différences centrées.

Programme MATLAB:

```
L = 1; v = 1; alpha = 0.01; N = 20; dx = L / N; dt = 0.005;
x = linspace(0, L, N+1); u = sin(pi * x); u(1) = 0; u(end) = 0;

for n = 1:100
   u_new = u;
   for i = 2:N
      convection = -v * (u(i+1) - u(i-1)) / (2*dx);
      diffusion = alpha * (u(i+1) - 2*u(i) + u(i-1)) / dx^2;
      u_new(i) = u(i) + dt * (convection + diffusion);
```

```
    end
    u = u_new;
end

plot(x, u, 'LineWidth', 2); title('Équation de Convection-Diffusion 1D');
```

Exercice 5: Équation de la Chaleur en 2D

Problème: Résoudre $\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$ sur $[0, 1] \times [0, 1]$.

Solution:

1. **Discrétisation** : Utiliser une grille 10×10, $\alpha = 0.01$.
2. **Approximation** : Différences centrées pour les dérivées spatiales.

Programme MATLAB:

```
L = 1; N = 10; dx = L / N; dt = 0.005; alpha = 0.01;
x = linspace(0, L, N+1); [X, Y] = meshgrid(x, x);
u = sin(pi * X) .* sin(pi * Y); u(:,1) = 0; u(:,end) = 0; u(1,:) = 0; u(end,:) = 0;

for k = 1:100
    u_new = u;
    for i = 2:N
        for j = 2:N
            u_new(i, j) = u(i, j) + alpha * dt * ...
                ((u(i+1, j) - 2*u(i, j) + u(i-1, j)) / dx^2 + ...
                 (u(i, j+1) - 2*u(i, j) + u(i, j-1)) / dx^2);
        end
    end
    u = u_new;
end

surf(X, Y, u); title('Équation de la Chaleur 2D');
```

Exercice 6: Équation d'Advection en 1D

Problème: Résoudre $\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0$ sur $[0, 1]$.

Solution:

1. **Discrétisation** : $v = 1$, $N = 50$.
2. **Approximation** : Différences centrées pour les dérivées spatiales.

Programme MATLAB:

```
L = 1; v = 1; N = 50; dx = L / N; dt = 0.01;
x = linspace(0, L, N+1); u = sin(2 * pi * x); u(1) = 0; u(end) = 0;

for n = 1:100
    u_new = u;
    for i = 2:N
        advection = -v * (u(i+1) - u(i-1)) / (2*dx);
        u_new(i) = u(i) + dt * advection;
    end
    u = u_new;
end

plot(x, u, 'LineWidth', 2); title('Équation d''Advection 1D');
```

Exercice 7: Équation de Poisson en 2D

Problème: Résoudre $-\Delta u = f(x, y)$ sur $[0, 1] \times [0, 1]$ avec $f(x, y) = 1$.

Solution:

1. Discrétisation : Utiliser une grille 10×10.
2. Approximation : Différences centrées pour les dérivées.

Programme MATLAB:

```
L = 1; N = 10; dx = L / N;
x = linspace(0, L, N+1); [X, Y] = meshgrid(x, x);
u = zeros(N+1); f = ones(N+1);

for k = 1:500
    u_new = u;
    for i = 2:N
        for j = 2:N
            u_new(i, j) = (u(i+1, j) + u(i-1, j) + u(i, j+1) + u(i, j-1) - dx^2 * f(i, j)) / 4;
        end
    end
    u = u_new;
end

surf(X, Y, u); title('Équation de Poisson 2D');
```

Exercice 8: Équation de Diffusion en 1D

Problème: Résoudre $\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$ sur $[0, 1]$ avec $D = 0.1$.

Solution:

1. **Discrétisation** : $N = 20$, $D = 0.1$.
2. **Approximation** : Différences centrées pour les dérivées.

Programme MATLAB:

```
L = 1; D = 0.1; N = 20; dx = L / N; dt = 0.01;
x = linspace(0, L, N+1); u = exp(-100 * (x - 0.5).^2); u(1) = 0; u(end) = 0;

for n = 1:100
    u_new = u;
    for i = 2:N
        u_new(i) = u(i) + D * dt / dx^2 * (u(i+1) - 2*u(i) + u(i-1));
    end
    u = u_new;
end

plot(x, u, 'LineWidth', 2); title('Équation de Diffusion 1D');
```

Exercice 9: Équation d'Advection-Diffusion en 1D

Problème: Résoudre $\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = D\frac{\partial^2 u}{\partial x^2}$ sur $[0, 1]$.

Solution:

1. **Discrétisation** : $v = 1$, $D = 0.1$, $N = 20$.
2. **Approximation** : Différences centrées pour les dérivées.

Programme MATLAB:

```
L = 1; v = 1; D = 0.1; N = 20; dx = L / N; dt = 0.005;
x = linspace(0, L, N+1); u = sin(pi * x); u(1) = 0; u(end) = 0;

for n = 1:100
    u_new = u;
    for i = 2:N
        convection = -v * (u(i+1) - u(i-1)) / (2*dx);
        diffusion = D * (u(i+1) - 2*u(i) + u(i-1)) / dx^2;
        u_new(i) = u(i) + dt * (convection + diffusion);
    end
    u = u_new;
end
```

```
plot(x, u, 'LineWidth', 2); title('Équation d''Advection-Diffusion 1D');
```

Exercice 10: Équation de Schrodinger en 1D

Problème: Résoudre l'équation de Schrodinger sur $[0, 1]$.

$$ih\frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2 \psi}{\partial x^2}$$

Solution:

1. **Discrétisation** : Utiliser des différences centrées pour l'espace et le temps.
2. **Approximation** : Approximations pour la dérivée seconde.

Programme MATLAB:

```
L = 1; N = 20; dx = L / N; dt = 0.01; hbar = 1; m = 1;
x = linspace(0, L, N+1); psi = exp(-100 * (x - 0.5).^2); psi(1) = 0; psi(end) = 0;

for n = 1:100
    psi_new = psi;
    for i = 2:N
        psi_new(i) = psi(i) + i*hbar*dt/(2*m*dx^2) * (psi(i+1) - 2*psi(i) + psi(i-1));
    end
    psi = psi_new;
end

plot(x, real(psi), 'LineWidth', 2); title('Équation de Schrodinger 1D');
```

B : Exercices Pratiques avec MATLAB : Méthode MVF

Exercice 4: Résolution de l'Équation d'Advection en 1D

Problème: Considérons une équation d'advection simple en 1D :

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0$$

Conditions:

- $v = 1$
- $u(0,t) = 0, u(1,t) = 0$ (conditions aux limites)
- $u(x,0) = \sin(2\pi x)$ (condition initiale)

Solution par MVF:

1. **Discrétisation de l'espace**: Utilisons $N = 50$ volumes.
2. **Bilan de conservation** pour chaque volume :

$$\frac{du_i}{dt} + v\frac{u_{i+1} - u_{i-1}}{2\Delta x} = 0$$

Programme MATLAB:

```
% Paramètres
L = 1.0;          % Longueur du domaine
v = 1.0;          % Vitesse d'advection
N = 50;           % Nombre de volumes de contrôle
dx = L / N;       % Taille de chaque volume
dt = 0.001;       % Pas de temps
x = linspace(0, L, N+1); % Coordonnées des centres des volumes
u = sin(2 * pi * x); % Condition initiale : onde sinusoïdale
u(1) = 0;         % Condition aux limites gauche
u(end) = 0;       % Condition aux limites droite

% Boucle temporelle
n_steps = 200;    % Nombre d'étapes temporelles
for n = 1:n_steps
   u_new = u;     % Stocker les nouvelles valeurs de u
   for i = 2:N
      advection = -v * (u(i+1) - u(i-1)) / (2*dx);
      u_new(i) = u(i) + dt * advection;
   end
   u = u_new;     % Mettre à jour u
end

% Affichage des résultats
plot(x, u, 'g-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Solution de l''équation d''advection après 200 itérations');
grid on;
```

Exercice 5: Résolution de l'Équation de Continuité en 2D

Problème: Considérons l'équation de continuité pour un fluide incompressible en 2D :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

Conditions:

- $\rho = 1 + 0.5 \sin(2\pi x)$
- $\mathbf{u} = (1, 0)$

Solution par MVF:

1. **Discrétisation de l'espace**: Utilisons une grille 10×10.
2. **Bilan de conservation** pour chaque volume :

$$\frac{d\rho_{i,j}}{dt} + \frac{(\rho u)_{i+1/2,j} - (\rho u)_{i-1/2,j}}{\Delta x} + \frac{(\rho v)_{i,j+1/2} - (\rho v)_{i,j-1/2}}{\Delta y} = 0$$

Programme MATLAB:

```
% Paramètres
L = 1.0;          % Taille du domaine
N = 10;           % Nombre de volumes de contrôle par dimension
dx = L / N;       % Taille de chaque volume
dy = dx;          % Taille de chaque volume (carré)
dt = 0.01;        % Pas de temps
x = linspace(0, L, N+1); % Coordonnées des centres des volumes en x
y = linspace(0, L, N+1); % Coordonnées des centres des volumes en y
[X, Y] = meshgrid(x, y);
rho = 1 + 0.5 * sin(2 * pi * X); % Densité initiale
u = ones(size(rho));          % Vitesse en x (constante)
v = zeros(size(rho));         % Vitesse en y (constante)

% Boucle temporelle
n_steps = 100;      % Nombre d'étapes temporelles
for n = 1:n_steps
   rho_new = rho;  % Stocker les nouvelles valeurs de densité
   for i = 2:N
      for j = 2:N
         flux_in_x = rho(i-1, j) * u(i-1, j);
         flux_out_x = rho(i, j) * u(i, j);
         flux_in_y = rho(i, j-1) * v(i, j-1);
         flux_out_y = rho(i, j) * v(i, j);
         rho_new(i, j) = rho(i, j) - dt * ((flux_out_x - flux_in_x) / dx + (flux_out_y -
flux_in_y) / dy);
      end
   end
```

```
    rho = rho_new;  % Mettre à jour rho
end

% Affichage des résultats
imagesc(x, y, rho);
colorbar;
title('Distribution de densité après 100 itérations');
xlabel('x');
ylabel('y');
```

C : Exercices Pratiques avec MATLAB : Méthode MEF

Exercice 4: Résolution de l'Équation de l'onde en 1D

Problème: Considérons l'équation de la vague en 1D :

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

Conditions:

- Domaine $[0, 1]$
- $c = 1$
- $u(0, t) = 0, u(1, t) = 0$
- $u(x, 0) = \sin(\pi x), \frac{\partial u}{\partial t}(x, 0) = 0$

Solution par MEF:

1. **Discrétisation du Domaine**: Utilisons $N = 20$ éléments.
2. **Fonctions de Forme**: Utiliser des fonctions de forme linéaires.
3. **Formulation Variationnelle**:

$$M\frac{d^2U}{dt^2} + KU = 0$$

Programme MATLAB:

```
% Paramètres
N = 20;          % Nombre d'éléments
L = 1.0;         % Longueur du domaine
h = L / N;       % Taille de chaque élément
c = 1.0;         % Vitesse de la vague
dt = 0.01;       % Pas de temps
x = linspace(0, L, N+1); % Coordonnées des nœuds

% Initialisation des matrices et vecteurs
K = zeros(N+1, N+1);
M = zeros(N+1, N+1);
```

```
F = zeros(N+1, 1);

% Assemblage du système (similaire aux exercices précédents)

% Conditions initiales
U = sin(pi * x)'; % Déplacement initial
V = zeros(N+1, 1); % Vitesse initiale

% Boucle temporelle
n_steps = 100;    % Nombre d'étapes temporelles
for n = 1:n_steps
    A = M + dt^2 * K;
    B = 2*M - M*V*dt + dt^2*K;
    C = M*U;
    U_new = A \ (B*U - C);
    V = (U_new - U) / dt;
    U = U_new;
end

% Affichage des résultats
plot(x, U, 'b-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Solution de l''équation de la vague après 100 itérations');
grid on;
```

Exercice 5: Résolution de l'Équation de la Chaleur Transitoire en 1D

Problème: Considérons l'équation de la chaleur transitoire en 1D :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Conditions:

- Domaine $[0, 1]$
- $\alpha = 0.01$
- $u(0, t) = 0$, $u(1, t) = 1$ (conditions aux limites de Dirichlet)
- $u(x, 0) = 0$ (condition initiale)

Solution par MEF:

1. **Discrétisation du Domaine**: Utilisons $N = 10$ éléments.
2. **Fonctions de Forme**: Utiliser des fonctions de forme linéaires.
3. **Formulation Variationnelle**:

$$M\frac{dU}{dt} + KU = F$$

```
% Paramètres
N = 10;           % Nombre d'éléments
L = 1.0;          % Longueur du domaine
h = L / N;        % Taille de chaque élément
alpha = 0.01;     % Coefficient de diffusion
dt = 0.01;        % Pas de temps
x = linspace(0, L, N+1); % Coordonnées des nœuds

% Initialisation des matrices et vecteurs
K = zeros(N+1, N+1);
M = zeros(N+1, N+1);
F = zeros(N+1, 1);

% Assemblage du système (similaire aux exercices précédents)

% Conditions initiales
U = zeros(N+1, 1); % Température initiale

% Boucle temporelle
n_steps = 100;    % Nombre d'étapes temporelles
for n = 1:n_steps
    U = (M + dt*K) \ (M*U + dt*F); % Mise à jour de la solution
end

% Affichage des résultats
plot(x, U, 'r-', 'LineWidth', 2);
xlabel('x');
ylabel('u');
title('Distribution de température après 100 itérations');
grid on;
```

Chapitre III : Techniques d’optimisation

1. Définition et formulation d'un problème d'optimisation.

2. Optimisation unique et multiple avec ou sans contraintes.

3. Algorithmes d'optimisation sans contraintes (Méthodes déterministes, Méthodes stochastiques).

4. Traitement des contraintes (Méthodes de transformation, Méthodes directes).

Chapitre III : Techniques d'optimisation

Introduction aux Techniques d'Optimisation

L'optimisation est une discipline fondamentale dans les mathématiques appliquées et la recherche opérationnelle, axée sur la détermination de la meilleure solution possible pour un problème donné. Elle consiste généralement à maximiser ou minimiser une fonction objective, tout en respectant certaines contraintes. Ces techniques d'optimisation sont omniprésentes et trouvent leur application dans de nombreux domaines, notamment l'ingénierie, l'économie, la finance, la gestion, la science des données, et bien d'autres. Le cœur de l'optimisation réside dans la définition claire des objectifs et la capacité de naviguer dans un espace de solutions pour atteindre ces objectifs de manière optimale.

A : Définition et Formulation d'un Problème d'Optimisation

Pour comprendre et résoudre un problème d'optimisation, il est essentiel de le formuler correctement. La formulation consiste à identifier les éléments clés du problème, à savoir la fonction objective, les variables de décision, et les contraintes éventuelles.

Composants d'un Problème d'Optimisation

1. **Fonction Objectif**: La fonction objectif est la fonction mathématique que l'on cherche à optimiser. Elle représente généralement une mesure de performance, telle que le coût, le temps, l'efficacité, ou tout autre indicateur de performance pertinent. L'objectif peut être soit de maximiser (comme le profit, le rendement) soit de minimiser (comme le coût, le temps, les pertes) cette fonction.
    - **Exemple**: En finance, une fonction objectif peut être de maximiser le rendement d'un portefeuille d'investissements, en cherchant à augmenter la valeur globale des actifs sous gestion.
2. **Variables de Décision**: Ce sont les variables que le décideur peut contrôler et ajuster. Les valeurs de ces variables influencent directement la fonction objectif. Elles peuvent représenter des quantités comme le nombre d'unités à produire, la quantité de matière à utiliser, ou les heures de travail à allouer.
    - **Exemple**: Dans la gestion de la production, les variables de décision pourraient inclure le nombre d'unités de chaque produit à fabriquer pour atteindre un coût minimum tout en respectant les besoins du marché.
3. **Contraintes**: Les contraintes définissent les limites dans lesquelles les variables de décision doivent opérer. Elles peuvent être des limites physiques (comme les capacités de production), réglementaires (comme les normes de sécurité), économiques (comme les budgets limités), ou d'autres restrictions spécifiques au problème.
    - **Exemple**: Une usine peut avoir une contrainte sur la quantité maximale de matières premières disponibles ou sur le nombre d'heures de travail que les employés peuvent effectuer dans une semaine donnée.

Formulation Mathématique

Un problème d'optimisation peut généralement être formulé de manière formelle comme suit :

- **Objectif**: Minimiser ou maximiser une fonction objectif f(x), où x est le vecteur des variables de décision.
- **Sous contraintes** :

  - **Minimiser** ou **Maximiser** $f(x)$, où f est la fonction objectif et x est le vecteur des variables de décision.
  - **Sous contraintes** : $g_i(x) \leq 0$ pour $i = 1, 2, ..., m$ (contraintes d'inégalité), et $h_j(x) = 0$ pour $j = 1, 2, ..., p$ (contraintes d'égalité).

La solution optimale est un ensemble de valeurs des variables de décision qui minimisent ou maximisent la fonction objectif tout en satisfaisant toutes les contraintes.

Applications Pratiques

1. **Industrie Aéronautique**: Optimiser la forme des ailes pour minimiser la traînée et maximiser l'efficacité énergétique tout en respectant les contraintes de matériaux et de sécurité.
2. **Gestion de Portefeuille**: Maximiser le rendement d'un portefeuille tout en minimisant le risque, en respectant des contraintes de liquidité et de réglementation.
3. **Logistique**: Minimiser les coûts de transport tout en respectant les contraintes de capacité des véhicules et les délais de livraison.

Pour bien comprendre comment définir et formuler un problème d'optimisation, il est essentiel de pratiquer avec des exercices concrets. Ces exercices couvrent différents types de problèmes d'optimisation, montrant comment passer d'une description textuelle à une formulation mathématique précise. Chaque exemple détaille les étapes nécessaires pour atteindre une formulation claire et exploitable.

Exercice 1: Optimisation de Production dans une Usine

Problème: Une usine fabrique deux types de produits, A et B. Chaque produit A nécessite 2 heures de travail et chaque produit B nécessite 3 heures de travail. L'usine dispose de 120 heures de travail disponibles par semaine. Le profit pour chaque unité de A est de 50 unités monétaires, et le profit pour chaque unité de B est de 80 unités monétaires. Combien de chaque produit l'usine doit-elle produire pour maximiser le profit hebdomadaire.

Applications : Formulation des problèmes d'optimisation

Exercice 1: Optimisation de Production dans une Usine

Problème: Une usine fabrique deux types de produits, A et B. Chaque produit A nécessite 2 heures de travail et chaque produit B nécessite 3 heures de travail. L'usine dispose de 120 heures de travail disponibles par semaine. Le profit pour chaque unité de A est de 50 unités monétaires, et le profit pour chaque unité de B est de 80 unités monétaires. Combien de chaque produit l'usine doit-elle produire pour maximiser le profit hebdomadaire?

Formulation du problème:

1. **Définir les variables de décision**:
   - x: Nombre d'unités de produit A à produire par semaine.
   - y: Nombre d'unités de produit B à produire par semaine.
2. **Définir la fonction objectif**:
   - Le profit total est donné par : $P(x, y) = 50x + 80y$.
3. **Définir les contraintes**:
   - Chaque produit A utilise 2 heures de travail : donc $2x$ heures.
   - Chaque produit B utilise 3 heures de travail : donc $3y$ heures.
   - Le temps total de travail ne doit pas dépasser 120 heures : $2x + 3y \leq 120$.
   - Il n'est pas possible de produire un nombre négatif de produits : $x \geq 0, y \geq 0$.
4. **Formulation mathématique complète**:

$$\text{Maximiser } P(x, y) = 50x + 80y$$

$$\text{sous contraintes :}$$

$$2x + 3y \leq 120, \quad x \geq 0, \quad y \geq 0$$

Exercice 2: Optimisation de l'Allocation de Portefeuille

Problème: Un investisseur a un budget de 100,000 unités monétaires et souhaite l'investir dans deux actions, X et Y. L'action X a un rendement espéré de 8% par an, tandis que l'action Y a un rendement espéré de 10% par an. L'investisseur souhaite maximiser son rendement annuel tout en limitant le risque, en ne permettant pas plus de 60,000 unités monétaires dans une seule action.

Formulation du problème:

1. **Définir les variables de décision**:
   - x: Montant investi dans l'action X.
   - y: Montant investi dans l'action Y.

2. **Définir la fonction objectif**:
   - Le rendement total est donné par : $R(x, y) = 0.08x + 0.10y$.
3. **Définir les contraintes**:
   - Budget total : $x + y = 100,000$.
   - Limite de risque pour une seule action : $x \leq 60,000$ et $y \leq 60,000$.
   - Les investissements ne peuvent pas être négatifs : $x \geq 0, y \geq 0$.
4. **Formulation mathématique complète**:

$$\text{Maximiser } R(x, y) = 0.08x + 0.10y$$

$$\text{sous contraintes :}$$

$$x + y = 100,000, \quad x \leq 60,000, \quad y \leq 60,000, \quad x \geq 0, \quad y \geq 0$$

Exercice 3: Optimisation de la Conception d'un Réservoir

Problème: Un ingénieur doit concevoir un réservoir cylindrique sans couvercle pour minimiser le coût de fabrication. Le réservoir doit contenir un volume de 500 m³. Le coût de fabrication du fond du réservoir est de 50 unités monétaires par mètre carré, et celui des côtés est de 30 unités monétaires par mètre carré.

Formulation du problème:

1. **Définir les variables de décision**:
   - r: Rayon de la base du cylindre en mètres.
   - h: Hauteur du cylindre en mètres.
2. **Définir la fonction objectif**:
   - Le coût total de fabrication est donné par : $C(r, h) = \text{Coût du fond} + \text{Coût des côtés}$.
   - Coût du fond : $50\pi r^2$.
   - Coût des côtés : $30(2\pi rh)$.
3. **Définir les contraintes**:
   - Volume du cylindre : $\pi r^2 h = 500$.
   - Les dimensions ne peuvent pas être négatives : $r > 0, h > 0$.
4. **Formulation mathématique complète**:

$$\text{Minimiser } C(r, h) = 50\pi r^2 + 60\pi rh$$

$$\text{sous contraintes :}$$

$$\pi r^2 h = 500, \quad r > 0, \quad h > 0$$

Exercice 4: Optimisation du Chemin de Livraison

Problème: Une entreprise de livraison veut minimiser la distance parcourue lors de la livraison de colis à trois emplacements différents : A, B, et C. Le dépôt de l'entreprise est situé au point D. Les distances entre les points sont les suivantes : DA = 5 km, DB = 8 km, DC = 3 km, AB = 4 km, AC = 7 km, BC = 6 km. Quelle est la séquence de livraison qui minimise la distance totale parcourue?

Formulation du problème:

1. **Définir les variables de décision**:
   - Les variables peuvent être définies comme des permutations des points A, B, C, et D. Chaque permutation représente une séquence de livraison possible.
2. **Définir la fonction objectif**:
   - La fonction objectif est de minimiser la distance totale parcourue : $D_{\text{total}} =$ Distance totale en fonction de la séquence choisie.
3. **Définir les contraintes**:
   - Chaque point doit être visité exactement une fois.
   - La livraison commence et se termine au dépôt D.
4. **Formulation mathématique complète**:
   - Les permutations possibles sont : D → A → B → C → D, D → A → C → B → D, etc.
   - Calculer la distance totale pour chaque permutation et choisir la plus petite.

Exercice 5: Optimisation de la Répartition du Temps d'Étude

Problème: Un étudiant a 20 heures par semaine à consacrer à l'étude de deux matières, Mathématiques et Physique. Pour maximiser ses notes, l'étudiant doit répartir son temps entre ces deux matières de manière optimale. Le score attendu pour les Mathématiques est donné par $S_1(t_1) = 10t_1 - 0.5t_1^2$ et pour la Physique par $S_2(t_2) = 8t_2 - 0.2t_2^2$, où t_1 et t_2 sont les heures passées à étudier chaque matière.

Formulation du problème:

1. **Définir les variables de décision**:
   - t_1: Temps consacré aux Mathématiques.
   - t_2: Temps consacré à la Physique.
2. **Définir la fonction objectif**:
   - Le score total est donné par : $S(t_1, t_2) = S_1(t_1) + S_2(t_2) = 10t_1 - 0.5t_1^2 + 8t_2 - 0.2t_2^2$.
3. **Définir les contraintes**:
   - Temps total disponible : $t_1 + t_2 = 20$.
   - Temps ne peut pas être négatif : $t_1 \geq 0$, $t_2 \geq 0$.
4. **Formulation mathématique complète**:

$$\text{Maximiser } S(t_1, t_2) = 10t_1 - 0.5t_1^2 + 8t_2 - 0.2t_2^2$$

$$\text{sous contraintes :}$$

$$t_1 + t_2 = 20, \quad t_1 \geq 0, \quad t_2 \geq 0$$

Conclusion

Ces exercices montrent comment définir les variables de décision, la fonction objectif, et les contraintes pour divers types de problèmes d'optimisation. Chaque exercice présente un

contexte pratique, transforme les descriptions textuelles en formulations mathématiques, et illustre l'approche méthodique nécessaire pour aborder les problèmes d'optimisation de manière systématique. En pratiquant avec ces types d'exercices, on développe une compréhension approfondie de la manière de formuler et résoudre efficacement des problèmes d'optimisation

B : Optimisation Unique et Multiple avec ou sans Contraintes

Optimisation Unique

L'optimisation unique se concentre sur la maximisation ou la minimisation d'une seule fonction objectif. Ces problèmes sont souvent plus simples à formuler et à résoudre car ils impliquent un seul critère de performance.

1. **Exemple**: Une entreprise cherche à minimiser les coûts de production tout en respectant les niveaux de production nécessaires pour satisfaire la demande du marché. Ici, la fonction objectif est le coût total, et l'objectif est de le minimiser.
2. **Méthodes**: Les méthodes utilisées pour résoudre ces problèmes incluent les techniques de programmation linéaire, le gradient descendant, et la méthode de Newton, selon la nature de la fonction objectif et des contraintes.

Optimisation Multiple

L'optimisation multiple, ou multi-objectifs, implique la maximisation ou la minimisation simultanée de plusieurs fonctions objectifs. Cela nécessite souvent de trouver un compromis ou un équilibre entre des objectifs contradictoires. Ces problèmes sont plus complexes car il n'y a pas toujours une solution unique optimale, mais plutôt un ensemble de solutions Pareto-optimales, où aucune solution ne peut être améliorée sur un objectif sans détériorer au moins un autre objectif.

1. **Exemple**: Dans la conception automobile, les objectifs peuvent inclure la minimisation de la consommation de carburant tout en maximisant la performance du véhicule. Il s'agit d'un compromis entre efficacité énergétique et performance.
2. **Méthodes**: Les méthodes pour résoudre les problèmes multi-objectifs incluent les approches de pondération des objectifs, les méthodes de contraintes, et les algorithmes évolutionnaires multi-objectifs comme NSGA-II (Non-dominated Sorting Genetic Algorithm II).

Applications Pratiques

1. **Conception de Produits**: Trouver le meilleur compromis entre coût, qualité et durabilité d'un produit.

2. **Planification Énergétique**: Maximiser l'efficacité de production d'énergie tout en minimisant les impacts environnementaux.
3. **Gestion de la Chaîne d'Approvisionnement**: Minimiser les coûts de stockage et de transport tout en maximisant la satisfaction client.

Optimisation avec et sans Contraintes

Optimisation sans Contraintes

Les problèmes d'optimisation sans contraintes sont plus faciles à analyser et à résoudre car ils n'ont pas de restrictions autres que les limites intrinsèques des variables de décision.

1. **Exemple**: Optimiser le rendement d'un réseau neuronal dans un modèle d'apprentissage machine sans tenir compte des contraintes de calcul ou de temps.
2. **Méthodes**: Les méthodes d'optimisation sans contraintes incluent le gradient descendant, la méthode de Newton, et l'algorithme de Quasi-Newton. Ces méthodes s'appuient sur les dérivées pour trouver la direction d'amélioration.

Applications : Optimisation Unique et Multiple avec ou sans Contraintes

Exercices sur l'Optimisation Unique et Multiple avec ou sans Contraintes

Voici une série d'exercices progressifs et pratiques couvrant l'optimisation unique et multiple, avec et sans contraintes. Chaque exercice est détaillé pour faciliter la compréhension des concepts et des méthodes d'optimisation.

Optimisation Unique Sans Contrainte

Exercice 1: Optimisation de Fonction Quadratique Simple

Problème: Trouvez le minimum de la fonction suivante :

$$f(x) = 3x^2 - 12x + 7$$

Solution:

1. **Calcul de la dérivée première**: Pour trouver le point critique, on calcule la dérivée de $f(x)$:

$$f'(x) = 6x - 12$$

2. **Résolution de l'équation dérivée à zéro**: Pour trouver les points critiques :

$$6x - 12 = 0 \Rightarrow x = 2$$

3. **Calcul de la valeur de la fonction en ce point**:

$$f(2) = 3(2)^2 - 12(2) + 7 = 12 - 24 + 7 = -5$$

4. **Conclusion**: Le minimum de la fonction est -5 atteint à $x = 2$.

Exercice 2: Optimisation d'une Fonction Exponentielle

Problème: Trouvez le minimum de la fonction suivante :

$$f(x) = e^{2x} - 4x$$

Solution:

1. **Calcul de la dérivée première**:

$$f'(x) = 2e^{2x} - 4$$

2. **Résolution de l'équation dérivée à zéro**:

$$2e^{2x} - 4 = 0 \Rightarrow e^{2x} = 2 \Rightarrow x = \frac{\ln(2)}{2}$$

3. **Calcul de la valeur de la fonction en ce point**:

$$f\left(\frac{\ln(2)}{2}\right) = e^{\ln(2)} - 4\left(\frac{\ln(2)}{2}\right) = 2 - 2\ln(2)$$

4. **Conclusion**: Le minimum de la fonction est $2 - 2\ln(2)$ atteint à $x = \frac{\ln(2)}{2}$.

Exercice 3: Optimisation de Fonction de Coût dans la Production

Problème: Une entreprise fabrique des produits avec un coût total donné par $C(x) = x^2 - 10x + 25$. Trouvez le niveau de production qui minimise le coût total.

Solution:

1. **Calcul de la dérivée première de la fonction de coût**:
$$C'(x) = 2x - 10$$
2. **Résolution de l'équation dérivée à zéro**:
$$2x - 10 = 0 \Rightarrow x = 5$$
3. **Calcul de la valeur de la fonction de coût en ce point**:
$$C(5) = 5^2 - 10 \cdot 5 + 25 = 25 - 50 + 25 = 0$$
4. **Conclusion**: Le coût total minimum est atteint lorsque 5 unités sont produites, avec un coût total de 0.

Exercice 4: Optimisation d'une Fonction Trigonométrique

Problème: Trouvez le maximum de la fonction suivante :
$$f(x) = \sin(x) - \frac{x}{2}$$

Solution:

1. **Calcul de la dérivée première**:
$$f'(x) = \cos(x) - \frac{1}{2}$$
2. **Résolution de l'équation dérivée à zéro**:
$$\cos(x) = \frac{1}{2} \Rightarrow x = \frac{\pi}{3}, \frac{5\pi}{3} \text{ (dans } [0, 2\pi])$$
3. **Calcul de la valeur de la fonction en ces points**:
$$f(\frac{\pi}{3}) = \sin(\frac{\pi}{3}) - \frac{\pi}{6} = \frac{\sqrt{3}}{2} - \frac{\pi}{6}$$
$$f(\frac{5\pi}{3}) = \sin(\frac{5\pi}{3}) - \frac{5\pi}{6} = -\frac{\sqrt{3}}{2} - \frac{5\pi}{6}$$
4. **Conclusion**: Le maximum est atteint à $x = \frac{\pi}{3}$ avec une valeur de $\frac{\sqrt{3}}{2} - \frac{\pi}{6}$.

Optimisation Unique Avec Contrainte

Exercice 1: Optimisation avec Contrainte Linéaire

Problème: Maximisez la fonction $f(x, y) = 3x + 2y$ sous la contrainte $x + y = 10$.

Solution:

1. **Utiliser la contrainte pour exprimer une variable en fonction de l'autre:**

$$y = 10 - x$$

2. **Substituer dans la fonction objectif:**

$$f(x) = 3x + 2(10 - x) = 3x + 20 - 2x = x + 20$$

3. **Calculer le maximum:**
   - x peut prendre des valeurs entre 0 et 10.
   - $f(x) = x + 20$ augmente linéairement avec x, donc le maximum est atteint pour $x = 10$.
4. **Conclusion:** Le maximum est atteint pour $(x, y) = (10, 0)$ avec $f(10, 0) = 30$.

Exercice 2: Optimisation Quadratique avec Contrainte

Problème: Minimisez la fonction $f(x, y) = x^2 + y^2$ sous la contrainte $x + y = 4$.

Solution:

1. **Utiliser la contrainte pour exprimer une variable en fonction de l'autre:**

$$y = 4 - x$$

2. **Substituer dans la fonction objectif:**

$$f(x) = x^2 + (4 - x)^2 = x^2 + 16 - 8x + x^2 = 2x^2 - 8x + 16$$

3. **Trouver la dérivée première et la résoudre:**

$$f'(x) = 4x - 8 = 0 \Rightarrow x = 2$$

$$y = 4 - 2 = 2$$

4. **Calculer la valeur de la fonction en ce point:**

$$f(2, 2) = 2^2 + 2^2 = 8$$

5. **Conclusion:** Le minimum de la fonction est 8 atteint à $(2, 2)$.

Exercice 3: Optimisation de Fonction de Coût avec Contrainte

Problème: Une entreprise fabrique deux produits, A et B. Le coût total est donné par $C(x, y) = 4x^2 + 9y^2$. Ils doivent produire au moins 20 unités au total. Trouvez le niveau de production qui minimise le coût total.

Solution:

1. **Contrainte:** $x + y \geq 20$.
2. **Utiliser la méthode des multiplicateurs de Lagrange :**

$$L(x, y, \lambda) = 4x^2 + 9y^2 + \lambda(20 - x - y)$$

3. **Calcul des dérivées:**

$$\frac{\partial L}{\partial x} = 8x - \lambda = 0 \Rightarrow \lambda = 8x$$

$$\frac{\partial L}{\partial y} = 18y - \lambda = 0 \Rightarrow \lambda = 18y$$

4. Équilibrer les équations:

$$8x = 18y \Rightarrow x = \frac{9}{4}y$$

5. Utiliser la contrainte:

$$\frac{9}{4}y + y = 20 \Rightarrow \frac{13y}{4} = 20 \Rightarrow y = \frac{80}{13}, x = \frac{180}{13}$$

6. Calculer le coût:

$$C\left(\frac{180}{13}, \frac{80}{13}\right) = 4\left(\frac{180}{13}\right)^2 + 9\left(\frac{80}{13}\right)^2$$

7. Conclusion: Le coût minimum est atteint pour $x = \frac{180}{13} \approx 13.85$ et $y = \frac{80}{13} \approx 6.15$.

Exercice 4: Optimisation avec Contrainte d'Inégalité

Problème: Maximisez la fonction $f(x, y) = 3x + 4y$ sous les contraintes $x \geq 0, y \geq 0$, et $x^2 + y^2 \leq 25$.

Solution:

1. Représenter la contrainte comme un cercle: $x^2 + y^2 = 25$ est un cercle de rayon 5.
2. Utiliser la méthode des multiplicateurs de Lagrange:

$$L(x, y, \lambda) = 3x + 4y + \lambda(25 - x^2 - y^2)$$

3. Calculer les dérivées:

$$\frac{\partial L}{\partial x} = 3 - 2\lambda x = 0 \Rightarrow \lambda = \frac{3}{2x}$$

$$\frac{\partial L}{\partial y} = 4 - 2\lambda y = 0 \Rightarrow \lambda = \frac{2}{y}$$

4. Résoudre le système:

$$\frac{3}{2x} = \frac{2}{y} \Rightarrow 3y = 4x$$

$$x^2 + \left(\frac{3}{4}x\right)^2 = 25$$

5. Calculer les points:

$$x = 4, y = 3$$

6. Calculer la valeur maximale:

$$f(4,3) = 3(4) + 4(3) = 24$$

7. **Conclusion**: Le maximum est atteint pour $(x, y) = (4, 3)$ avec $f(4, 3) = 24$.

Optimisation Multiple Sans Contrainte

Exercice 1: Optimisation de Deux Fonctions Objectif Simples

Problème: Maximisez $f_1(x, y) = x^2 + y^2$ et minimisez $f_2(x, y) = (x-2)^2 + (y-2)^2$.

Solution:

1. **Identifier les conflits**: f_1 est maximisé par des valeurs élevées de x et y, f_2 est minimisé près de $(2, 2)$.
2. **Utiliser une pondération**:

   $$L(x, y) = w_1 f_1(x, y) + w_2 f_2(x, y)$$

   Choisir $w_1 = 0.5$, $w_2 = 0.5$.
3. **Formuler le problème**:

   $$L(x, y) = 0.5(x^2 + y^2) + 0.5[(x-2)^2 + (y-2)^2]$$
4. **Trouver les dérivées**:

   $$\frac{\partial L}{\partial x} = x + (x-2)$$

   $$\frac{\partial L}{\partial y} = y + (y-2)$$
5. **Trouver les points critiques**:

   $$2x - 2 = 0 \Rightarrow x = 1$$

   $$2y - 2 = 0 \Rightarrow y = 1$$
6. **Conclusion**: Le compromis optimal est atteint à $(x, y) = (1, 1)$.

Exercice 2: Optimisation d'Investissements avec Profits Multiples

Problème: Une entreprise veut maximiser les profits de deux projets. Les profits sont donnés par $P_1 = 4x$ et $P_2 = 6y$. Le budget total est de 100 unités.

Solution:

1. **Définir la fonction objectif multi-objectifs**: $f_1 = 4x, f_2 = 6y$.
2. **Utiliser la méthode de pondération**: maximiser $L(x, y) = 0.5f_1 + 0.5f_2$.
3. **Sous contrainte de budget**: $x + y = 100$.
4. **Utiliser la substitution**:

   $$y = 100 - x$$

4. **Trouver les points critiques avec des dérivées partielles et les résoudre.**
5. **Conclusion**: Les valeurs optimales sont atteintes pour un compromis équilibré entre coût et durabilité sous contrainte de ressources, typiquement $x = 8, y = 12$.

Optimisation Multiple Avec Contrainte

Exercice 1: Conception de Produit sous Contraintes

Problème: Optimisez la conception d'un produit pour maximiser la durabilité $D(x, y) = 3x + 4y$ et minimiser le coût $C(x, y) = x^2 + y^2$, sous la contrainte de ressources $x + 2y \leq 12$.

Solution:

1. **Formuler le problème avec des fonctions d'objectif conflictuelles et des contraintes.**
2. **Utiliser la méthode des multiplicateurs de Lagrange**:

$$\mathrm{L}(x, y, \lambda) = 3x + 4y - \lambda(x + 2y - 12)$$

3. **Calcul des dérivées**:

$$\frac{\partial \mathrm{L}}{\partial x} = 3 - \lambda = 0 \Rightarrow \lambda = 3$$

$$\frac{\partial \mathrm{L}}{\partial y} = 4 - 2\lambda = 0 \Rightarrow \lambda = 2$$

4. **Résoudre le système**:

$$x + 2y = 12$$

5. **Calculer les points**:

$$x = 4, y = 4$$

6. **Conclusion**: Le compromis optimal entre durabilité et coût est atteint pour $x = 4$ et $y = 4$.

Exercice 2: Optimisation de Production avec Contrainte

Problème: Maximisez la production de deux usines avec $P_1 = 5x + 3y$ et $P_2 = 2x + 7y$ sous la contrainte $x + y \leq 30$.

Solution:

1. **Identifier les fonctions et les contraintes conflictuelles.**
2. **Utiliser la méthode des multiplicateurs de Lagrange**:

$$\mathrm{L}(x, y, \lambda) = 5x + 3y + 2x + 7y - \lambda(x + y - 30)$$

3. **Calculer les dérivées partielles et les résoudre.**
4. **Utiliser les conditions de Kuhn-Tucker pour maximiser.**
5. **Conclusion**: Les valeurs optimales sont $x = 15, y = 15$.

Exercice 3: Gestion de Projet sous Contraintes

Problème: Minimisez le temps de réalisation de deux projets $T_1 = x + 2y$ et $T_2 = 3x + y$ sous la contrainte de budget $4x + 2y \leq 60$.

Solution:

1. **Utiliser une approche pondérée**:

$$L(x, y) = T_1 + T_2 - \lambda(4x + 2y - 60)$$

2. **Résoudre les dérivées partielles**.
3. **Utiliser la contrainte pour trouver les valeurs critiques**.
4. **Conclusion**: Le temps optimal est atteint pour des valeurs de x et y qui respectent la contrainte budgétaire tout en minimisant les temps de projet.

Exercice 4: Optimisation de Réseaux de Transport sous Contraintes

Problème: Maximisez la couverture de transport $C(x, y) = 6x + 5y$ et minimisez le coût $M(x, y) = 2x^2 + 3y^2$ sous la contrainte de capacité $x + 2y \leq 18$.

Solution:

1. **Utiliser la méthode des multiplicateurs de Lagrange**:

$$\mathrm{L}(x, y, \lambda) = 6x + 5y - \lambda(x + 2y - 18)$$

2. **Calculer les dérivées partielles**.
3. **Résoudre le système avec les contraintes de Kuhn-Tucker**.
4. **Conclusion**: La solution optimale est atteinte pour des valeurs équilibrées de x et y, typiquement $x = 6, y = 6$.

Ces exercices offrent une gamme complète de scénarios d'optimisation, mettant en évidence les différentes approches et techniques utilisées pour résoudre les problèmes d'optimisation dans des contextes réels, avec ou sans contraintes.

C : Algorithmes d'Optimisation Sans Contraintes

Les algorithmes d'optimisation sans contraintes sont utilisés pour résoudre des problèmes où les variables de décision ne sont pas limitées par des contraintes explicites. Ces algorithmes peuvent être catégorisés en deux types principaux : déterministes et stochastiques.

Méthodes Déterministes

Les méthodes déterministes suivent des règles fixes et utilisent des dérivées pour guider le processus de recherche vers l'optimum. Elles sont efficaces et prévisibles, mais peuvent être limitées par leur tendance à se bloquer dans des optima locaux.

1. **Méthode du Gradient**: Cette méthode utilise les dérivées premières de la fonction objectif pour déterminer la direction dans laquelle se déplacer pour atteindre l'optimum. C'est l'une des méthodes les plus simples et les plus utilisées.
    - **Application**: Ajustement des poids dans un modèle de réseau de neurones pour minimiser la fonction de coût pendant l'apprentissage supervisé.
2. **Méthode de Newton**: Basée sur le calcul de la Hessienne (matrice des secondes dérivées), cette méthode ajuste non seulement la direction mais aussi la taille des pas, permettant une convergence plus rapide.
    - **Application**: Estimation des paramètres dans des modèles de régression pour des données complexes où la fonction objectif est non linéaire.
3. **Méthode des Régions de Confiance**: Elle consiste à définir une région autour de la solution courante et à minimiser une approximation de la fonction objectif dans cette région. Cette méthode est plus robuste et peut gérer des fonctions non linéaires complexes.
    - **Application**: Optimisation de la forme des structures en ingénierie pour minimiser le poids tout en respectant les contraintes de résistance et de durabilité.

Méthodes Stochastiques

Les méthodes stochastiques utilisent des éléments aléatoires pour explorer l'espace de recherche. Elles sont particulièrement utiles pour les problèmes avec de nombreux optima locaux, où les méthodes déterministes peuvent échouer.

1. **Algorithmes Génétiques**: Ces algorithmes sont inspirés par la théorie de l'évolution et utilisent des concepts de mutation, croisement, et sélection pour trouver des solutions optimales. Ils sont capables de gérer des espaces de solutions très larges et complexes.
    - **Application**: Optimisation de la conception de réseaux de communication pour minimiser les interférences tout en maximisant la couverture et la fiabilité.

2. **Recuit Simulé**: Basé sur le processus de refroidissement des métaux, cet algorithme permet d'explorer l'espace de recherche de manière aléatoire. Il accepte parfois des solutions moins bonnes pour éviter de rester coincé dans des optima locaux.
    - **Application**: Planification des horaires de travail pour minimiser les coûts de main-d'œuvre tout en respectant les contraintes de disponibilité des employés et des horaires de travail.
3. **Optimisation par Essaim de Particules (PSO)**: Inspirée par les comportements sociaux des animaux, cette méthode utilise une population de solutions candidates qui se déplacent dans l'espace de recherche en suivant à la fois leur propre expérience et celle des autres particules.
    - **Application**: Optimisation des paramètres dans les modèles de prévision météorologique pour minimiser l'erreur de prévision.

Traitement des Contraintes

Les contraintes sont des restrictions qui limitent les solutions possibles d'un problème d'optimisation. Le traitement des contraintes est un aspect crucial de l'optimisation, surtout dans des applications pratiques où les contraintes sont souvent inévitables.

Application : Méthodes déterministes et stochastiques

Méthodes Déterministes

Les méthodes déterministes d'optimisation sans contraintes utilisent des règles fixes et des calculs analytiques pour trouver les points critiques et optimiser les fonctions. Ces méthodes incluent la méthode du gradient, la méthode de Newton, la méthode de la région de confiance, et d'autres.

Exercice 1: Méthode du Gradient

Problème: Trouvez le minimum de la fonction suivante :

$$f(x, y) = x^2 + y^2 + 2x + 4y + 5$$

Solution pas à pas:

1. **Calculer les dérivées partielles:**

$$\frac{\partial f}{\partial x} = 2x + 2 \quad \text{et} \quad \frac{\partial f}{\partial y} = 2y + 4$$

2. **Définir la direction de descente:**
    - Direction de descente $d = -\nabla f = -(2x + 2, 2y + 4)$.
3. **Choisir un point initial et un pas:**
    - Supposons $(x_0, y_0) = (0, 0)$.
    - Pas $\alpha = 0.1$.
4. **Calculer les itérations:**
    - Itération 1 : $(x_1, y_1) = (0, 0) + 0.1(-2, -4) = (-0.2, -0.4)$.
    - Itération 2 : Calculer les nouvelles dérivées et répéter.

5. **Convergence**:
   - Continuer jusqu'à ce que $\|\nabla f\|$ soit proche de zéro.
6. **Conclusion**: Le minimum est atteint à $x = -1, y = -2$ avec une valeur de $f(-1,-2) = 0$.

Exercice 2: Méthode de Newton

Problème: Trouvez le minimum de la fonction suivante :

$$f(x) = x^4 - 4x^3 + 6x^2$$

Solution pas à pas:

1. **Calculer la dérivée première et la dérivée seconde**:

$$f'(x) = 4x^3 - 12x^2 + 12x$$

$$f''(x) = 12x^2 - 24x + 12$$

2. **Utiliser la méthode de Newton**:
   - Point initial : $x_0 = 2$.
   - Formule de Newton : $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$.

3. **Itération 1**:

$$x_1 = 2 - \frac{4(2)^3 - 12(2)^2 + 12(2)}{12(2)^2 - 24(2) + 12} = 1$$

4. **Continuer les itérations**:
   - Répéter jusqu'à convergence.
5. **Conclusion**: Le minimum est atteint à $x = 1$ avec $f(1) = 1$.

Exercice 3: Méthode de la Région de Confiance

Problème: Trouvez le minimum de la fonction suivante :

$$f(x,y) = x^2 + 2y^2 - 4x + 4y$$

Solution pas à pas:

1. **Calculer les dérivées partielles et l'Hessienne**:

$$\frac{\partial f}{\partial x} = 2x - 4, \quad \frac{\partial f}{\partial y} = 4y + 4$$

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

2. **Choisir un point initial et définir une région de confiance**:
   - Point initial $(x_0, y_0) = (0, 0)$.
   - Région de confiance $\Delta = 1$.
3. **Minimiser l'approximation quadratique**:
   - Approximation $m(x, y) = f(x_0, y_0) + \nabla f \cdot (x, y) + \frac{1}{2}(x, y)H(x, y)^T$.
4. **Trouver le minimum de $m(x, y)$ dans la région de confiance**:
   - Minimiser $m(x, y)$ pour $\|(x, y)\| \leq \Delta$.
5. **Conclusion**: Le minimum est atteint pour $x = 2, y = -1$ avec une valeur de -3.

Exercice 4: Méthode de Gradient Conjugué

Problème: Trouvez le minimum de la fonction quadratique suivante :

$$f(x) = 2x_1^2 + 3x_2^2 + 2x_1x_2 - 4x_1 - 5x_2$$

Solution pas à pas:

1. **Définir le gradient**:

$$\nabla f(x) = \begin{bmatrix} 4x_1 + 2x_2 - 4 \\ 6x_2 + 2x_1 - 5 \end{bmatrix}$$

2. **Choisir un point initial et une direction initiale**:
   - Point initial $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.
   - Direction initiale $d_0 = -\nabla f(x_0)$.
3. **Calculer les itérations**:
   - Trouver le pas α pour minimiser $f(x + \alpha d)$.
   - Calculer $x_1 = x_0 + \alpha d_0$.
   - Mettre à jour la direction avec $d_1 = -\nabla f(x_1) + \beta d_0$.
4. **Convergence**:
   - Continuer jusqu'à ce que $\|\nabla f\|$ soit proche de zéro.
5. **Conclusion**: Le minimum est atteint pour une valeur spécifique de x.

Exercice 5: Méthode de la Pente la Plus Raide

Problème: Trouvez le minimum de la fonction suivante :

$$f(x, y) = x^2 + y^2 - 3x - 3y$$

Solution pas à pas:

1. **Calculer le gradient**:

$$\nabla f(x, y) = \begin{bmatrix} 2x - 3 \\ 2y - 3 \end{bmatrix}$$

2. **Choisir un point initial**:
   - Supposons $(x_0, y_0) = (0, 0)$.
3. **Calculer les directions de descente**:
   - Itération 1 : $(x_1, y_1) = (0, 0) - \alpha(2x_0 - 3, 2y_0 - 3)$.
4. **Choisir le pas α et itérer**:
   - Continuer jusqu'à convergence.
5. **Conclusion**: Le minimum est atteint pour $x = 1.5, y = 1.5$ avec $f(1.5, 1.5) = -4.5$.

Méthodes Stochastiques

Les méthodes stochastiques utilisent des processus aléatoires pour explorer l'espace de recherche. Elles sont utiles pour les problèmes complexes avec de nombreux optima locaux, comme les algorithmes génétiques, le recuit simulé, et les essaims de particules.

Exercice 1: Recuit Simulé

Problème: Trouvez le minimum de la fonction suivante :

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x$$

Solution pas à pas:

1. **Choisir un point initial et une température**:
   - Point initial $x_0 = 0$.
   - Température initiale $T = 100$.
2. **Générer un voisin aléatoire**:
   - $x_1 = x_0 + \text{aléa}$.
3. **Calculer la différence de coût** $\Delta f = f(x_1) - f(x_0)$.
4. **Accepter le mouvement**:
   - Si $\Delta f < 0$, accepter.
   - Si $\Delta f > 0$, accepter avec probabilité $e^{-\Delta f/T}$.
5. **Répéter en diminuant la température**:
   - Réduire T, générer un nouveau voisin, et appliquer les règles de décision.
6. **Conclusion**: Le recuit simulé converge vers un minimum global ou proche d'un minimum local.

Exercice 2: Algorithme Génétique

Problème: Trouvez le maximum de la fonction suivante :

$$f(x) = \sin(x) + \cos(2x)$$

Solution pas à pas:

1. **Initialiser une population de chromosomes**:
   - Supposons une population de 10 individus avec des valeurs de x choisies aléatoirement.
2. **Évaluer la fonction objectif pour chaque individu**:
   - Calculer $f(x)$ pour chaque individu.
3. **Sélectionner les meilleurs individus pour la reproduction**:
   - Sélectionner les meilleurs 50% selon $f(x)$.
4. **Croisement et mutation**:
   - Croisement : Combiner les valeurs de x des parents pour créer de nouveaux enfants.
   - Mutation : Modifier légèrement certaines valeurs de x.
5. **Créer une nouvelle génération**:
   - Répéter les étapes de sélection, croisement, et mutation pour plusieurs générations.
6. **Conclusion**: Le meilleur individu de la population finale donne une approximation du maximum de la fonction.

Exercice 3: Optimisation par Essaim de Particules (PSO)

Problème: Trouvez le minimum de la fonction suivante :

$$f(x, y) = (x - 2)^2 + (y + 3)^2$$

Solution pas à pas:

1. **Initialiser une population de particules**: Supposons 5 particules avec des positions initiales aléatoires.
2. **Définir les vitesses et positions initiales**:
   - Position : (x_i, y_i) pour chaque particule.
   - Vitesse : (v_{x_i}, v_{y_i}).
3. **Évaluer la fonction objectif pour chaque particule**:
   - Calculer $f(x_i, y_i)$.
4. **Mettre à jour les positions et les vitesses**:
   - Mettre à jour la vitesse en fonction de la meilleure position trouvée par la particule et par l'ensemble de l'essaim.
   - Mettre à jour la position.
5. **Répéter jusqu'à convergence**:
   - Continuer jusqu'à ce que les changements deviennent insignifiants.
6. **Conclusion**: L'algorithme converge vers le minimum global à $(2, -3)$ avec $f(2, -3) = 0$.

Exercice 4: Méthode de Recherche Aléatoire

Problème: Trouvez le maximum de la fonction suivante :

$$f(x, y) = x\sin(4x) + y\cos(3y)$$

Solution pas à pas:

1. **Choisir un point initial aléatoire**: Supposons $(x_0, y_0) = (1, 1)$.
2. **Générer des points aléatoires autour du point actuel**:
   - Générer (x_1, y_1) près de (x_0, y_0).
3. **Évaluer la fonction en ces points**:
   - Calculer $f(x_1, y_1)$ et comparer avec $f(x_0, y_0)$.
4. **Déplacer vers le point qui maximise la fonction**:
   - Si $f(x_1, y_1) > f(x_0, y_0)$, déplacer vers (x_1, y_1).
   - Sinon, générer un autre point.
5. **Répéter le processus**:
   - Continuer jusqu'à ce que les changements deviennent insignifiants.
6. **Conclusion**: La méthode converge vers un maximum local ou global selon le nombre de points générés et l'exploration.

Exercice 5: Recherche Tabou

Problème: Trouvez le minimum de la fonction suivante :

$$f(x, y) = x^2 + y^2 - \sin(x)\cos(y)$$

Solution pas à pas:

1. **Initialiser une solution de départ**: Supposons $(x_0, y_0) = (2, 2)$.
2. **Définir une liste tabou**: Liste pour stocker les solutions récentes.
3. **Générer des solutions voisines**:
   - Calculer $f(x, y)$ pour des points voisins de (x_0, y_0).
4. **Choisir la meilleure solution voisine non tabou**:
   - Vérifier si la solution voisine est dans la liste tabou.
   - Si non, comparer les valeurs et choisir la meilleure.
5. **Mettre à jour la solution courante et la liste tabou**:
   - Déplacer vers la meilleure solution voisine.
   - Ajouter la solution courante à la liste tabou.
6. **Répéter jusqu'à convergence**:
   - Continuer jusqu'à ce que aucune amélioration ne soit trouvée.
7. **Conclusion**: La recherche tabou permet de trouver le minimum local en évitant les cycles.

Ces exercices illustrent comment utiliser les méthodes déterministes et stochastiques pour résoudre des problèmes d'optimisation sans contraintes. En suivant les étapes détaillées, on peut comprendre les différences entre ces approches et apprendre à choisir la méthode la plus adaptée à un problème spécifique.

D : Méthodes de Transformation

Les méthodes de transformation convertissent un problème avec contraintes en un problème sans contraintes en intégrant les contraintes dans la fonction objectif. Ces méthodes sont particulièrement utiles pour simplifier le traitement des contraintes et appliquer des techniques d'optimisation sans contraintes.

1. **Méthode des Pénalités**: Cette méthode ajoute un terme de pénalité à la fonction objectif pour chaque contrainte violée. Plus la violation est grande, plus la pénalité est élevée, rendant les solutions qui violent les contraintes moins attractives.
2. **Formulation**:

- Formulation: $L(x) = f(x) + P \sum_{i=1}^{m} \max(0, g_i(x))^2$, où P est un paramètre de pénalité.

Application: Conception de composants électroniques où la violation des contraintes de fabrication entraîne des coûts de réparation et de re-conception. La fonction de pénalité aide à minimiser ces coûts.

3. **Méthode des Barrières**: Contrairement à la méthode des pénalités, la méthode des barrières empêche complètement d'approcher les limites des contraintes en ajoutant un terme de barrière qui devient infini lorsque la solution s'approche de la frontière de la contrainte.

- Formulation: $L(x) = f(x) - \frac{1}{g(x)}$, où $g(x)$ représente la contrainte.

Application: Optimisation de la gestion de l'eau dans les systèmes agricoles où les limites d'eau doivent être strictement respectées pour éviter des dommages environnementaux.

Méthodes Directes

Les méthodes directes traitent les contraintes explicitement sans les transformer. Elles sont plus intuitives et garantissent que les solutions restent toujours dans le domaine admissible.

1. **Algorithmes de Projection**: Ces algorithmes ajustent les solutions candidates pour qu'elles respectent les contraintes en les projetant sur le domaine admissible. C'est une méthode couramment utilisée dans les problèmes où les contraintes sont simples et linéaires.

- **Application**: Planification de la production dans des usines où les ressources telles que la main-d'œuvre et les matériaux doivent être strictement respectées pour éviter les pénuries et les retards.

2. **Méthode des Multiplicateurs de Lagrange**: Cette méthode convertit le problème avec contraintes en un ensemble de conditions de premier ordre en introduisant des multiplicateurs de Lagrange qui pondèrent les contraintes.

- **Formulation**: Le problème est transformé en maximisation de la fonction de Lagrange $L(x, \lambda) = f(x) + \sum_{j=1}^{p} \lambda_j h_j(x)$.

Application: Optimisation des stratégies de placement de capteurs dans les réseaux de surveillance pour maximiser la couverture tout en minimisant les coûts d'installation et d'entretien.

3. **Méthode de Points Intérieurs**: Cette méthode commence par une solution initiale qui respecte toutes les contraintes d'inégalité strictement et se déplace vers l'optimum en restant à l'intérieur de la région admissible.
   - **Application**: Conception de systèmes de transport urbain où les objectifs incluent la minimisation des temps de trajet et des coûts d'exploitation tout en respectant les contraintes de capacité et de budget.

Applications : Méthodes de Transformation

Les contraintes jouent un rôle crucial dans de nombreux problèmes d'optimisation. Le traitement des contraintes se divise généralement en deux approches : les méthodes de transformation et les méthodes directes. Les méthodes de transformation intègrent les contraintes dans la fonction objectif, transformant ainsi le problème en une optimisation sans contraintes. Les méthodes directes gèrent les contraintes de manière explicite, souvent en ajustant les solutions candidates pour respecter les contraintes. Voici cinq exercices détaillés pour bien comprendre ces méthodes.

Méthodes de Transformation

Exercice 1: Méthode des Pénalités pour une Fonction Simple

Problème: Minimisez la fonction suivante sous la contrainte $x^2 \leq 4$:

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

Solution pas à pas:

1. **Formuler la contrainte en tant que pénalité**:
   - Contrainte : $g(x) = x^2 - 4 \leq 0$.
   - Fonction de pénalité : $P(x) = \max(0, x^2 - 4)^2$.
2. **Définir la fonction objectif pénalisée**:

$$L(x) = f(x) + \rho P(x) = (x-2)^2 + \rho \max(0, x^2 - 4)^2$$

3. **Choisir un paramètre de pénalité** ρ : Supposons $\rho = 100$.
4. **Minimiser la fonction pénalisée:**
   - Pour $x \in [-2, 2]$, $P(x) = 0$, donc $L(x) = (x-2)^2$.
   - Pour $x > 2$ ou $x < -2$, $P(x) > 0$, ce qui augmente $L(x)$.
5. **Résoudre:**
   - Le minimum de $L(x)$ se trouve à $x = 2$, car ici $P(x) = 0$ et $f(x) = 0$.
6. **Conclusion:** La solution optimale est $x = 2$ avec $f(2) = 0$, satisfaisant la contrainte.

Exercice 2: Méthode des Barrières Logarithmiques

Problème: Minimisez la fonction suivante sous la contrainte $x > 0$:

$$f(x) = x^2 + 1$$

Solution pas à pas:

1. **Formuler la contrainte en tant que barrière:**
   - Contrainte : $g(x) = -x \leq 0$.
   - Fonction de barrière : $B(x) = -\ln(-g(x)) = -\ln(x)$.
2. **Définir la fonction objectif avec barrière:**

$$L(x) = f(x) + \mu B(x) = x^2 + 1 - \mu \ln(x)$$

3. **Choisir un paramètre de barrière** μ : Supposons $\mu = 0.5$.
4. **Minimiser la fonction avec barrière:**
   - Calculer la dérivée : $L'(x) = 2x - \frac{\mu}{x}$.
   - Résoudre $L'(x) = 0$: $2x - \frac{0.5}{x} = 0 \Rightarrow 2x^2 = 0.5 \Rightarrow x = \sqrt{0.25} = 0.5$.
5. **Calculer la valeur de la fonction en ce point:**

$$L(0.5) = 0.5^2 + 1 - 0.5\ln(0.5) = 1.25 + 0.3466 \approx 1.5966$$

6. **Conclusion:** Le minimum de la fonction est atteint à $x = 0.5$.

Exercice 3: Méthode des Pénalités Quadratiques

Problème: Minimisez la fonction suivante sous la contrainte $x^2 + y^2 \leq 1$:

$$f(x, y) = x^2 + y^2$$

Solution pas à pas:

1. **Formuler la contrainte en tant que pénalité:**
   - Contrainte : $g(x, y) = x^2 + y^2 - 1 \leq 0$.
   - Fonction de pénalité : $P(x, y) = \max(0, x^2 + y^2 - 1)^2$.

2. **Définir la fonction objectif pénalisée**:

$$L(x,y) = f(x,y) + \rho P(x,y) = x^2 + y^2 + \rho \max(0, x^2 + y^2 - 1)^2$$

3. **Choisir un paramètre de pénalité** ρ : Supposons $\rho = 10$.
4. **Minimiser la fonction pénalisée**:
   - Pour $x^2 + y^2 \leq 1$, $P(x,y) = 0$, donc $L(x,y) = x^2 + y^2$.
   - Le minimum est donc sur le bord $x^2 + y^2 = 1$.
5. **Trouver le point optimal sur la contrainte**:
   - Pour $x^2 + y^2 = 1$, minimum de $L(x,y)$ est à $(x,y) = (0,1)$ ou $(1,0)$.
6. **Conclusion**: La solution optimale est sur le cercle unité, avec une valeur minimale de 1.

Exercice 4: Méthode de Pénalité Douce

Problème: Maximisez la fonction suivante sous la contrainte $x + y \geq 2$:

$$f(x,y) = x + y$$

Solution pas à pas:

1. **Formuler la contrainte en tant que pénalité douce**:
   - Contrainte : $g(x,y) = 2 - x - y \geq 0$.
   - Fonction de pénalité douce : $P(x,y) = -\min(0, 2 - x - y)$.
2. **Définir la fonction objectif pénalisée**:

$$L(x,y) = f(x,y) - \rho P(x,y) = x + y - \rho \min(0, 2 - x - y)$$

3. **Choisir un paramètre de pénalité** ρ : Supposons $\rho = 1$.
4. **Maximiser la fonction pénalisée**:
   - Si $x + y \geq 2$, $P(x,y) = 0$ donc maximiser $L(x,y) = x + y$.
5. **Trouver le point optimal**:
   - Maximum est sur la ligne $x + y = 2$, choisir $(1,1)$.
6. **Conclusion**: La solution optimale est $(x,y) = (1,1)$ avec une valeur maximale de 2.

Exercice 5: Méthode de Barrière Inversée

Problème: Minimisez la fonction suivante sous les contraintes $x > 0$ et $y > 0$:

$$f(x,y) = x^2 + y^2$$

Solution pas à pas:

1. **Formuler les contraintes en tant que barrières inversées:**
   - Contraintes : $g_1(x) = -x < 0, g_2(y) = -y < 0.$
   - Fonction de barrière inversée : $B(x,y) = -\ln(x) - \ln(y).$
2. **Définir la fonction objectif avec barrière:**

$$L(x,y) = f(x,y) + \mu B(x,y) = x^2 + y^2 - \mu(\ln(x) + \ln(y))$$

3. **Choisir un paramètre de barrière μ :** Supposons $\mu = 0.1.$
4. **Minimiser la fonction avec barrière:**
   - Calculer les dérivées : $\frac{\partial L}{\partial x} = 2x - \frac{\mu}{x}, \frac{\partial L}{\partial y} = 2y - \frac{\mu}{y}.$
   - Résoudre : $2x - \frac{0.1}{x} = 0 \Rightarrow x = \sqrt{0.05} \approx 0.22.$
5. **Trouver le point optimal:**
   - Même calcul pour y, donc $y = \sqrt{0.05} \approx 0.22.$
6. **Conclusion:** Le minimum est approximativement atteint à $(0.22, 0.22).$

Méthodes Directes

Exercice 1: Méthode des Multiplicateurs de Lagrange

Problème: Trouvez le maximum de $f(x,y) = xy$ sous la contrainte $x + y = 10.$

Solution pas à pas:

1. **Définir la fonction de Lagrange:**

$$\mathrm{L}(x,y,\lambda) = xy + \lambda(10 - x - y)$$

2. **Calculer les dérivées partielles et les équilibrer à zéro:**

$$\frac{\partial \mathrm{L}}{\partial x} = y - \lambda = 0$$

$$\frac{\partial \mathrm{L}}{\partial y} = x - \lambda = 0$$

$$\frac{\partial \mathrm{L}}{\partial \lambda} = 10 - x - y = 0$$

3. **Résoudre le système d'équations:**
   - $x = \lambda, y = \lambda,$ et $x + y = 10$ donnent $2\lambda = 10 \Rightarrow \lambda = 5.$
4. **Conclusion:** $x = 5, y = 5$ avec un maximum de 25.

$$\frac{\partial L}{\partial x} = y - \lambda = 0$$
$$\frac{\partial L}{\partial y} = x - \lambda = 0$$
$$\frac{\partial L}{\partial \lambda} = 10 - x - y = 0$$

Problème: Minimisez $f(x, y, z) = x^2 + y^2 + z^2$ sous la contrainte $x + y + z = 1$.

Solution pas à pas:

1. **Réduire la dimension**:
   - Utiliser la contrainte pour exprimer z : $z = 1 - x - y$.
2. **Substituer dans la fonction**:
$$f(x, y) = x^2 + y^2 + (1 - x - y)^2$$
3. **Minimiser la nouvelle fonction**:
   - Calculer les dérivées partielles par rapport à x et y, les équilibrer à zéro.
4. **Trouver le minimum**:
   - $x = y = \frac{1}{3}, z = \frac{1}{3}$.
5. **Conclusion**: Minimum atteint avec une valeur de $\frac{1}{3}$.

Exercice 5: Méthode des Sous-problèmes

Problème: Maximisez $f(x, y) = x + 2y$ sous les contraintes $x^2 + y^2 \leq 1$ et $y \geq 0$.

Solution pas à pas:

1. **Décomposer en sous-problèmes**:
   - Résoudre pour $x^2 + y^2 = 1$ et $y = 0$.
2. **Trouver les solutions des sous-problèmes**:
   - Sur le cercle unité : $x = 1, y = 0$.
   - $f(1, 0) = 1$ et pour $y = 1, x = 0$ donne $f(0, 1) = 2$.
3. **Combiner les solutions**:
   - Comparer les solutions des sous-problèmes pour maximiser f.
4. **Conclusion**: Maximum atteint pour $(x, y) = (0, 1)$ avec une valeur de 2.

Ces exemples illustrent comment utiliser les méthodes de transformation et les méthodes directes pour traiter les contraintes dans les problèmes d'optimisation. Chaque méthode a ses propres avantages et est plus adaptée à certains types de problèmes. Les méthodes de transformation sont souvent plus faciles à mettre en œuvre, mais les méthodes directes peuvent offrir une meilleure précision pour les problèmes complexes.

Exercice 5: Méthode des Sous-problèmes

Problème

Maximiser $f(x, y) = xy$ sous les contraintes $x^2 + y^2 \leq 1$ et $x, y \geq 0$.

Solution pas à pas

1. **Décomposer en sous-problèmes**:
   - Résoudre pour $x^2 + y^2 = 1$ et $x, y \geq 0$.
2. **Trouver les solutions des sous-problèmes**:
   - Sur le cercle unité : $x^2 + y^2 = 1$.
   - Si $x = \cos(\theta)$ et $y = \sin(\theta)$, maximiser $\cos(\theta)\sin(\theta)$.
3. **Combiner les solutions**:
   - Comparer les solutions des sous-problèmes pour maximiser $f(x, y)$.
4. **Conclusion**: Maximum atteint pour $x = y = \frac{1}{\sqrt{2}}$ avec une valeur de 0.5.

Conclusion

Les techniques d'optimisation sont essentielles pour résoudre une large gamme de problèmes complexes rencontrés dans les domaines de l'industrie, de l'économie, de la gestion, et bien d'autres. La définition précise du problème, la compréhension des différentes méthodes d'optimisation, et le traitement efficace des contraintes sont des éléments cruciaux pour le succès de l'optimisation. Grâce à ces techniques, il est possible de prendre des décisions plus éclairées, de maximiser les performances, de minimiser les coûts et de trouver des solutions durables et efficaces dans des environnements dynamiques et complexes. Les algorithmes d'optimisation, qu'ils soient déterministes ou stochastiques, offrent une boîte à outils puissante pour aborder les défis modernes de l'optimisation.

Projet 1

Optimisation Numérique d'un Avion : Modélisation, Conception, et Optimisation Multi-Objectif

Introduction

L'optimisation de la performance des avions est un sujet crucial en ingénierie aéronautique, visant à améliorer l'efficacité énergétique, réduire les coûts d'exploitation, minimiser l'impact environnemental, et assurer la sécurité et la fiabilité. Ce document explore en profondeur la résolution numérique de problèmes d'optimisation avec contraintes à plusieurs variables pour un avion modélisé avec des variables d'état élevées. Nous explorerons plusieurs objectifs d'optimisation, tels que la consommation de carburant, la géométrie de l'aile, et d'autres facteurs influençant les performances. Des comparaisons entre différentes méthodes d'optimisation seront faites pour illustrer les résultats variés.

Objectifs et Méthodologie

Objectifs de l'optimisation :

1. **Minimisation de la consommation de carburant** : Réduire la consommation de carburant pour des coûts d'exploitation plus faibles et un impact environnemental réduit.
2. **Optimisation de la géométrie de l'aile** : Modifier la conception de l'aile pour maximiser l'efficacité aérodynamique.
3. **Optimisation multi-objectif** : Combiner plusieurs objectifs pour obtenir un design optimal qui équilibre les différents critères de performance.

Méthodologie :

1. **Définition du Modèle** : Construction d'un modèle d'avion réaliste avec des variables d'état clés.
2. **Formulation du Problème d'Optimisation** : Définition des fonctions objectifs et des contraintes.
3. **Méthodes d'Optimisation** : Application de diverses méthodes d'optimisation numérique pour résoudre le problème, telles que la méthode de Lagrange, les algorithmes génétiques, et les méthodes de programmation quadratique.

Modèle d'Avion

Pour cet exemple, nous utilisons un avion de transport commercial standard. Les variables d'état incluent :

1. W : Poids de l'avion (N)
2. S : Surface alaire (m^2)
3. C_L : Coefficient de portance
4. C_D : Coefficient de traînée
5. v : Vitesse de croisière (m/s)
6. ρ : Densité de l'air (kg/m^3)
7. T : Poussée des moteurs (N)
8. C_f : Consommation spécifique de carburant (kg/N.s)
9. Λ : Angle de flèche de l'aile (degrés)
10. b : Envergure de l'aile (m)
11. AR : Allongement de l'aile

1. Optimisation de la Consommation de Carburant

L'un des principaux objectifs d'optimisation est de minimiser la consommation de carburant, qui est directement liée à la traînée et à l'efficacité des moteurs. La consommation de carburant est une fonction de la traînée totale et de la consommation spécifique de carburant des moteurs.

Fonction Objectif

Pour minimiser la consommation de carburant F, nous utilisons l'équation suivante :

$$F = C_f \cdot T$$

où C_f est la consommation spécifique de carburant (kg/N.s), et T est la poussée requise pour vaincre la traînée totale.

Contraintes

1. **Équilibre de la Portance** : Pour un vol en palier, la portance L doit équilibrer le poids W :

$$L = W = \frac{1}{2}\rho v^2 S C_L$$

2. **Équilibre de la Traînée** : La poussée T doit être suffisante pour vaincre la traînée :

$$T = D = \frac{1}{2}\rho v^2 S C_D$$

3. **Contrainte sur la Surface Ailaire** :

$$S_{min} \leq S \leq S_{max}$$

4. **Contrainte sur le Coefficient de Portance** :

$$C_{L_{min}} \leq C_L \leq C_{L_{max}}$$

2. Optimisation de la Géométrie de l'Aile

L'optimisation de la géométrie de l'aile implique de maximiser l'efficacité aérodynamique, mesurée par le rapport portance/traînée L/D. Cela inclut l'optimisation de la surface alaire, de l'angle de flèche, et de l'envergure.

Fonction Objectif

Maximiser L/D, où :

$$\frac{L}{D} = \frac{C_L}{C_D}$$

Contraintes

1. **Contrainte de Portance** : La portance doit équilibrer le poids pour un vol en palier.

2. **Contrainte de Traînée** : La traînée doit être minimale pour réduire la consommation de carburant.
3. **Contrainte Structurelle** : L'aile doit pouvoir supporter les charges aérodynamiques.

3. Optimisation Multi-Objectif

Combiner les deux objectifs mentionnés ci-dessus pour minimiser la consommation de carburant tout en maximisant L/D. Cela peut être abordé par des techniques comme les algorithmes génétiques multi-objectifs.

Méthodes d'Optimisation

1. Méthode de Lagrange pour l'Optimisation avec Contraintes

Cette méthode introduit des multiplicateurs de Lagrange pour transformer un problème d'optimisation contraint en un problème non contraint.

Fonction de Lagrange :

$$\mathrm{L}(S, C_L, C_D, \lambda_1, \lambda_2) = C_f \cdot T + \lambda_1 \left(\frac{1}{2}\rho v^2 S C_L - W\right) + \lambda_2 \left(\frac{1}{2}\rho v^2 S C_D - T\right)$$

où λ_1 et λ_2 sont les multiplicateurs de Lagrange.

2. Programmation Quadratique

La programmation quadratique est utilisée pour les problèmes d'optimisation où la fonction objectif est quadratique et les contraintes sont linéaires.

Fonction Objectif Quadratique :

$$\min\left(\frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}\right)$$

Contraintes Linéaires :

$$A\mathbf{x} \leq \mathbf{b}$$

3. Algorithmes Génétiques

Les algorithmes génétiques sont des méthodes d'optimisation basées sur les processus de sélection naturelle. Ils sont particulièrement efficaces pour les problèmes d'optimisation multi-objectifs.

Implémentation en Matlab

Exemple : Optimisation de la Consommation de Carburant

Étape 1 : Définir les Paramètres et les Équations

```
% Définir les paramètres
rho = 1.225; % Densité de l'air (kg/m^3)
v = 250; % Vitesse de croisière (m/s)
W = 700000; % Poids de l'avion (N)
Cf = 0.0008; % Consommation spécifique de carburant (kg/N.s)
S_min = 100; % Surface alaire minimale (m^2)
S_max = 500; % Surface alaire maximale (m^2)
CL_min = 0.5; % Coefficient de portance minimal
CL_max = 1.5; % Coefficient de portance maximal

% Définir les fonctions de portance et de traînée
L = @(S, CL) 0.5 * rho * v^2 * S * CL; % Portance
D = @(S, CD) 0.5 * rho * v^2 * S * CD; % Traînée

% Définir la consommation de carburant
Fuel = @(T) Cf * T;
```

Étape 2 : Formuler la Fonction de Lagrange

```
% Définir la fonction de Lagrange
Lagrange = @(S, CL, CD, lambda1, lambda2) Fuel(D(S, CD)) + ...
        lambda1 * (L(S, CL) - W) + ...
```

```
    lambda2 * (D(S, CD) - T);
```

Étape 3 : Calculer le Gradient de Lagrange

```
% Définir les dérivées partielles
grad_S = @(S, CL, CD, lambda1, lambda2) (Cf * 0.5 * rho * v^2 * CD) + lambda1 * (0.5 *
rho * v^2 * CL) - lambda2 * (0.5 * rho * v^2 * CD);
grad_CL = @(S, CL, lambda1) lambda1 * (0.5 * rho * v^2 * S);
grad_CD = @(S, CD, lambda2) S - lambda2 * (0.5 * rho * v^2 * S);
grad_lambda1 = @(S, CL) L(S, CL) - W;
grad_lambda2 = @(S, CD) D(S, CD) - T;
```

Étape 4 : Résoudre le Système d'Équations avec fsolve

```
% Utiliser fsolve pour trouver les valeurs optimales
options = optimoptions('fsolve', 'Display', 'iter');
fun = @(x) [grad_S(x(1), x(2), x(3), x(4), x(5));
        grad_CL(x(1), x(2), x(4));
        grad_CD(x(1), x(3), x(5));
        grad_lambda1(x(1), x(2));
        grad_lambda2(x(1), x(3))];

x0 = [200; 1; 0.02; 1; 1]; % Initial guess [S, CL, CD, lambda1, lambda2]
sol = fsolve(fun, x0, options);

S_opt = sol(1);
CL_opt = sol(2);
CD_opt = sol(3);

disp(['Surface alaire optimale : ', num2str(S_opt), ' m^2']);
disp(['Coefficient de portance optimal : ', num2str(CL_opt)]);
disp(['Coefficient de traînée optimal : ', num2str(CD_opt)]);
```

Étape 5 : Validation des Résultats

1. **Vérification des Contraintes** : Vérifier que la portance égale le poids et que la traînée est minimale.
2. **Analyse des Résultats** : Comparer les résultats obtenus avec les performances théoriques et expérimentales pour valider le modèle.

Optimisation de la Géométrie de l'Aile

Étape 1 : Définir les Paramètres et les Relations Empiriques

```
% Définir les paramètres de l'aile
```

```
b = 35; % Envergure de l'aile (m)
AR = b^2 / S_opt; % Allongement de l'aile
e = 0.8; % Facteur d'efficacité de l'aile
CL_max = 1.4; % Coefficient de portance maximum

% Définir la relation pour la traînée induite
CDi = @(CL, S, AR, e) CL^2 / (pi * AR * e);

% Définir la relation pour la traînée parasite
CDp = @(Lambda) 0.02 + 0.04 * (cosd(Lambda))^2; % Fonction simplifiée

% Définir la fonction de coût pour l'optimisation de la flèche
cost_function = @(Lambda) CDp(Lambda) + CDi(CL_opt, S_opt, AR, e);
```

Étape 2 : Utiliser une Méthode Numérique pour Minimiser la Traînée

```
% Utiliser fmincon pour optimiser l'angle de flèche
Lambda0 = 15; % Angle de flèche initial (degrés)
lb = 0; % Borne inférieure
ub = 45; % Borne supérieure

options = optimoptions('fmincon', 'Display', 'iter');
Lambda_opt = fmincon(cost_function, Lambda0, [], [], [], [], lb, ub, [], options);

disp(['Angle de flèche optimal : ', num2str(Lambda_opt), ' degrés']);
```

Comparaison des Résultats avec des Méthodes d'Optimisation Différentes

1. **Algorithmes Génétiques** : Utilisation pour explorer un espace de solution plus large et potentiellement trouver des optima globaux.
2. **Programmation Quadratique** : Appropriée pour les problèmes où la fonction objectif est quadratique et les contraintes sont linéaires.
3. **Optimisation Multi-Objectif** : Utilisation de l'algorithme génétique multi-objectif pour équilibrer la consommation de carburant et la performance aérodynamique.

Conclusion

L'optimisation des avions est un domaine complexe qui nécessite des méthodes numériques robustes pour résoudre des problèmes avec plusieurs variables d'état et contraintes. En utilisant des méthodes comme la méthode de Lagrange, la programmation quadratique, et les algorithmes génétiques, nous pouvons optimiser efficacement la consommation de carburant, la géométrie de l'aile, et d'autres facteurs de performance des avions. Ces techniques permettent d'améliorer l'efficacité énergétique, de réduire les coûts, et de concevoir des avions plus performants et respectueux de l'environnement. L'implémentation en Matlab offre une

plateforme puissante pour modéliser et optimiser ces systèmes complexes, fournissant aux ingénieurs des outils essentiels pour le développement de la prochaine génération d'avions.

Project : 2

Aspects théoriques et pratiques de l'optimisation stochastique (Using Matlab)

Pour formuler un problème d'optimisation stochastique avec contraintes pour un système non linéaire multivariable, tout en fournissant des valeurs numériques et du code Matlab, est une tâche très ambitieuse et complexe. Ce document couvrira les, incluant la modélisation du système, la définition de la fonction objectif et des contraintes, l'application de techniques d'optimisation stochastique, et l'implémentation avec Matlab. Nous inclurons des exemples concrets avec des valeurs numériques pour illustrer chaque étape du processus.

Introduction

Les systèmes non linéaires multivariables sont omniprésents dans les domaines de l'ingénierie, des sciences appliquées, et de l'économie. Ces systèmes, caractérisés par des relations complexes et non linéaires entre leurs variables, posent des défis importants en matière de modélisation et d'optimisation. Optimiser de tels systèmes est crucial pour améliorer les performances, l'efficacité énergétique, et la robustesse des systèmes industriels, des dispositifs électroniques, et des processus biologiques.

Les méthodes d'optimisation stochastique, telles que les algorithmes génétiques (GA), l'optimisation par essaims particulaires (PSO), et les stratégies évolutionnaires, sont particulièrement bien adaptées aux systèmes non linéaires multivariables en raison de leur capacité à gérer des fonctions objectif non convexes et des espaces de recherche complexes.

Chapitre III : Techniques d'optimisation

Ces méthodes exploitent des processus aléatoires pour explorer efficacement l'espace de solutions, évitant ainsi les minima locaux et trouvant des solutions optimales globales.

1. Description du Système Non Linéaire Multivariable

Nous considérons un système non linéaire multivariable décrit par un ensemble d'équations différentielles non linéaires. Soit un système dynamique modélisé par les équations suivantes :

$$\frac{dx_1}{dt} = f_1(x_1, x_2, u_1, u_2) = x_1^2 - x_2 + u_1$$

$$\frac{dx_2}{dt} = f_2(x_1, x_2, u_1, u_2) = x_1 \cdot x_2 + u_2$$

$$y = h(x_1, x_2) = x_1^2 + x_2^2$$

Où :

- x_1 et x_2 sont les variables d'état du système.
- u_1 et u_2 sont les variables de contrôle ou d'entrée.
- y est la sortie du système, qui est une fonction non linéaire des états du système.
- f_1 et f_2 sont des fonctions non linéaires décrivant la dynamique du système.

2. Définition de la Fonction Objectif

L'objectif est de minimiser une fonction coût qui combine plusieurs aspects du système, y compris l'énergie consommée, les pertes et la déviation par rapport à une sortie désirée. La fonction objectif est définie comme suit :

$$J = \alpha \cdot E(x_1, x_2, u_1, u_2) + \beta \cdot P(x_1, x_2, u_1, u_2) + \gamma \cdot \frac{1}{T_{\text{response}}} + \delta \cdot (y - y_d)^2$$

Où :

- $E(x_1, x_2, u_1, u_2)$ est l'énergie totale consommée par le système.
- $P(x_1, x_2, u_1, u_2)$ représente les pertes.
- T_{response} est le temps de réponse du système.
- y est la sortie réelle du système.
- y_d est la sortie désirée (ici supposée être une constante, $y_d = 5$).
- $\alpha, \beta, \gamma, \delta$ sont des coefficients de pondération qui équilibrent l'importance de chaque terme.

3. Contraintes

Le problème d'optimisation doit respecter plusieurs contraintes :

1. **Limites sur les Variables de Contrôle** : Les entrées u_1 et u_2 doivent rester dans des limites spécifiques pour des raisons de sécurité et de performance.

$$u_1 \in [0, 10]$$

$$u_2 \in [0, 10]$$

2. **Limites sur les Variables d'État** : Les états du système doivent également être contrôlés pour éviter des comportements instables ou dangereux.

$$x_1 \in [-10, 10]$$

$$x_2 \in [-10, 10]$$

3. **Limitation de Courant** : Si les variables de contrôle représentent des courants, elles doivent rester en dessous d'une valeur maximale, par exemple, 10 unités.

4. Méthodes d'Optimisation Stochastique

4.1 Algorithmes Génétiques (GA)

Les algorithmes génétiques sont des méthodes d'optimisation basées sur les principes de la sélection naturelle et de la génétique. Voici les étapes clés de l'algorithme :

- **Initialisation** : Créer une population initiale de solutions aléatoires.
- **Évaluation** : Calculer la valeur de la fonction objectif pour chaque individu de la population.
- **Sélection** : Choisir les meilleurs individus en fonction de leur valeur de la fonction objectif.
- **Croisement** : Combiner les solutions des individus sélectionnés pour créer une nouvelle génération.
- **Mutation** : Introduire des modifications aléatoires pour explorer de nouvelles solutions.
- **Remplacement** : Remplacer les individus les moins performants par les nouveaux.

4.2 Optimisation par Essaims Particulaires (PSO)

L'optimisation par essaims particulaires s'inspire du comportement des groupes d'animaux. Chaque particule de l'essaim représente une solution potentielle et se déplace dans l'espace de recherche en suivant les meilleures solutions trouvées par elle-même et par ses voisins.

5. Implémentation en Matlab

5.1 Définition des Paramètres et des Fonctions

Avant d'implémenter l'algorithme, nous devons définir les paramètres et les fonctions du système :

```
% Paramètres du système
alpha = 0.5;
```

```
beta = 0.3;
gamma = 0.2;
delta = 0.1;
y_d = 5; % Sortie désirée

% Bornes des variables de contrôle
lb = [0, 0]; % Limite inférieure pour u1 et u2
ub = [10, 10]; % Limite supérieure pour u1 et u2

% Fonction objectif
function J = objective(params)
    u1 = params(1);
    u2 = params(2);
    % Appeler la fonction de simulation pour obtenir E, P, T_response, et y
    [E, P, T_response, y] = simulate_system(u1, u2);
    J = alpha * E + beta * P + gamma / T_response + delta * (y - y_d)^2;
end

% Contraintes
function [c, ceq] = constraints(params)
    ceq = []; % Pas de contraintes d'égalité
    u1 = params(1);
    u2 = params(2);
    [T, y] = simulate_constraints(u1, u2); % Simulation pour contraintes
    c = [u1 - 10; u2 - 10; -u1; -u2; T - 80]; % Contrainte d'inégalité
end
```

5.2 Implémentation de l'Optimisation avec GA

Le code suivant utilise l'algorithme génétique pour résoudre le problème d'optimisation.

```
% Options pour l'optimisation GA
options = optimoptions('ga', 'Display', 'iter', 'PopulationSize', 50, 'MaxGenerations', 100);

% Exécution de l'optimisation GA
params_opt = ga(@objective, 2, [], [], [], [], lb, ub, @constraints, options);

disp('Paramètres optimisés:');
disp(params_opt);
```

5.3 Fonction de Simulation

Les fonctions simulate_system et simulate_constraints sont des fonctions personnalisées que vous devez définir pour simuler la dynamique du système et vérifier les contraintes. Un exemple simplifié de ce que ces fonctions pourraient ressembler :

```
function [E, P, T_response, y] = simulate_system(u1, u2)
    % Simuler la dynamique du système avec les valeurs de u1 et u2
    % Exemple simplifié : calculer l'énergie, les pertes, et le temps de réponse
    E = u1^2 + u2^2; % Énergie consommée
    P = u1 + u2; % Pertes (simplifiées)
    T_response = 1 / (1 + exp(-u1 - u2)); % Temps de réponse (non linéaire)
    y = u1^2 + u2^2; % Sortie (simplifiée)
end

function [T, y] = simulate_constraints(u1, u2)
    % Simuler pour vérifier les contraintes
    T = 60 + 0.5 * (u1 + u2); % Température (exemple)
    y = u1^2 + u2^2; % Sortie (comme dans la simulation principale)
end
```

6. Analyse des Résultats

Après avoir exécuté l'optimisation, il est essentiel d'analyser les résultats pour vérifier que les solutions trouvées respectent les contraintes et optimisent effectivement la fonction objectif. Voici quelques approches pour analyser les résultats :

- **Vérification des contraintes** : Assurez-vous que les valeurs optimisées de u1u_1u1 et u2u_2u2 respectent toutes les contraintes définies, telles que les limites de courant et de température.
- **Comparaison des performances** : Comparez les valeurs de la fonction objectif avant et après optimisation pour évaluer l'amélioration.
- **Robustesse** : Testez la robustesse des solutions optimisées sous différentes conditions de simulation pour vérifier leur efficacité dans des scénarios variés.

Conclusion

L'optimisation stochastique avec contraintes pour des systèmes non linéaires multivariables est une approche puissante pour améliorer les performances des systèmes complexes. En utilisant des algorithmes tels que les GA, il est possible de trouver des solutions optimales qui respectent un ensemble complexe de contraintes et minimisent une fonction objectif multivariable. Les exemples fournis montrent comment cette approche peut être mise en œuvre dans Matlab, avec des valeurs numériques et des simulations spécifiques pour guider le processus.

Project 3

Optimisation Dynamique d'un Moteur Électrique DC Non Linéaire :

Introduction

Les moteurs électriques sont des éléments essentiels dans de nombreux systèmes industriels et de consommation. L'optimisation de leur performance est cruciale pour améliorer l'efficacité énergétique, minimiser les pertes et maximiser la durabilité. Les moteurs électriques présentent des comportements dynamiques complexes et souvent non linéaires. Pour cette raison, une modélisation précise et l'utilisation de méthodes de résolution numérique sont nécessaires pour analyser et optimiser leur performance.

Ce document se concentre sur l'optimisation dynamique d'un moteur électrique de nature non linéaire en utilisant des méthodes de résolution numérique. Nous détaillons la formulation des modèles dynamiques, l'utilisation de méthodes numériques pour simuler ces modèles, et l'application de techniques d'optimisation avec contraintes pour améliorer des aspects tels que la consommation d'énergie et la réduction des pertes.

Modèle Dynamique du Moteur Électrique

Un moteur électrique typique, tel qu'un moteur à courant continu (DC) ou un moteur à courant alternatif (AC), peut être modélisé par un ensemble d'équations différentielles non linéaires qui décrivent la relation entre les tensions, les courants, les flux magnétiques, et les couples.

Modèle de Base d'un Moteur à Courant Continu

Les équations suivantes représentent un modèle de base pour un moteur à courant continu (DC) :

1. **Équation de tension** : Cette équation lie la tension d'entrée V, la résistance R, l'inductance L, le courant I, et la vitesse angulaire ω.

$$V = L\frac{dI}{dt} + RI + K_e\omega$$

où K_e est la constante de force électromotrice (FEM) du moteur.

2. **Équation de couple** : Relie le couple T, la constante de couple K_t, le courant I, le moment d'inertie J, la friction B, et la vitesse angulaire ω.

$$T = J\frac{d\omega}{dt} + B\omega$$

$$T = K_t I$$

Modèle Dynamique Non Linéaire

Pour rendre le modèle plus réaliste, nous introduisons des non-linéarités dues aux caractéristiques de saturation magnétique et aux pertes fer et cuivre. Les équations non linéaires peuvent être représentées comme suit :

1. **Équation de tension non linéaire** :

$$V = L(I)\frac{dI}{dt} + R(I)I + K_e\omega$$

où $L(I)$ et $R(I)$ sont des fonctions non linéaires de courant.

2. **Équation de couple non linéaire** :

$$T = J\frac{d\omega}{dt} + B\omega + f(\omega, I)$$

où $f(\omega, I)$ représente les effets non linéaires dus aux pertes.

Objectifs d'Optimisation

1. **Minimisation des pertes d'énergie** : Réduire les pertes énergétiques en optimisant les paramètres du moteur.
2. **Optimisation de la géométrie** : Ajuster la conception géométrique pour améliorer l'efficacité et les performances du moteur.
3. **Maximisation de l'efficacité** : Optimiser la conversion de l'énergie électrique en énergie mécanique.

Méthodologie

1. **Modélisation Dynamique** : Formuler les équations de mouvement non linéaires du moteur.

2. **Simulation Numérique** : Utiliser des méthodes d'intégration numérique pour simuler le comportement du moteur.
3. **Optimisation avec Contraintes** : Appliquer des techniques d'optimisation pour ajuster les paramètres du moteur tout en respectant les contraintes.

1. Modélisation Dynamique

Nous modélisons un moteur DC avec des caractéristiques de non-linéarité, y compris la saturation magnétique et les pertes résistives non linéaires.

Paramètres du Modèle

- Résistance $R(I)$: $R(I) = R_0 + R_1 I$, où R_0 est la résistance nominale et R_1 est un coefficient de perte non linéaire.
- Inductance $L(I)$: $L(I) = L_0(1 - \alpha I^2)$, où α représente la saturation magnétique.
- Constante de couple K_t : Fixée à une valeur nominale.

.

2. Simulation Numérique

Les équations non linéaires du moteur sont résolues numériquement en utilisant la méthode de Runge-Kutta d'ordre 4 (RK4). Cette méthode permet une approximation précise des équations différentielles ordinaires, idéale pour simuler les dynamiques complexes d'un moteur électrique.

Implémentation en Matlab

Étape 1 : Définir les Paramètres et les Équations

```
% Paramètres du moteur
R0 = 1; % Résistance nominale (ohms)
R1 = 0.01; % Coefficient de perte non linéaire (ohms/A)
L0 = 0.5; % Inductance nominale (H)
alpha = 0.001; % Coefficient de saturation magnétique
Ke = 0.1; % Constante de FEM (V.s/rad)
Kt = 0.1; % Constante de couple (Nm/A)
J = 0.01; % Moment d'inertie (kg.m^2)
B = 0.001; % Coefficient de friction (Nm.s/rad)

% Conditions initiales
I0 = 0; % Courant initial (A)
omega0 = 0; % Vitesse angulaire initiale (rad/s)
V = 24; % Tension d'entrée (V)
```

```
% Temps de simulation
t_end = 10; % Temps total de simulation (s)
dt = 0.01; % Pas de temps
N = t_end / dt; % Nombre de pas de temps

% Initialisation des variables
I = zeros(1, N);
omega = zeros(1, N);
I(1) = I0;
omega(1) = omega0;
```

Étape 2 : Implémenter la Méthode de Runge-Kutta d'Ordre 4

```
% Boucle de simulation utilisant Runge-Kutta d'ordre 4
for k = 1:N-1
    % Calculer les coefficients non linéaires
    R = R0 + R1 * I(k);
    L = L0 * (1 - alpha * I(k)^2);

    % Calculer les k1
    k1_I = (V - R * I(k) - Ke * omega(k)) / L;
    k1_omega = (Kt * I(k) - B * omega(k)) / J;

    % Calculer les k2
    I_half = I(k) + dt/2 * k1_I;
    omega_half = omega(k) + dt/2 * k1_omega;
    R_half = R0 + R1 * I_half;
    L_half = L0 * (1 - alpha * I_half^2);
    k2_I = (V - R_half * I_half - Ke * omega_half) / L_half;
    k2_omega = (Kt * I_half - B * omega_half) / J;

    % Calculer les k3
    I_half = I(k) + dt/2 * k2_I;
    omega_half = omega(k) + dt/2 * k2_omega;
    R_half = R0 + R1 * I_half;
    L_half = L0 * (1 - alpha * I_half^2);
    k3_I = (V - R_half * I_half - Ke * omega_half) / L_half;
    k3_omega = (Kt * I_half - B * omega_half) / J;

    % Calculer les k4
    I_full = I(k) + dt * k3_I;
    omega_full = omega(k) + dt * k3_omega;
    R_full = R0 + R1 * I_full;
```

```
    L_full = L0 * (1 - alpha * I_full^2);
    k4_I = (V - R_full * I_full - Ke * omega_full) / L_full;
    k4_omega = (Kt * I_full - B * omega_full) / J;

    % Mise à jour des variables d'état
    I(k+1) = I(k) + dt/6 * (k1_I + 2*k2_I + 2*k3_I + k4_I);
    omega(k+1) = omega(k) + dt/6 * (k1_omega + 2*k2_omega + 2*k3_omega + k4_omega);
end

% Afficher les résultats
t = 0:dt:t_end-dt;
figure;
subplot(2,1,1);
plot(t, I, 'r');
xlabel('Temps (s)');
ylabel('Courant (A)');
title('Évolution du Courant');

subplot(2,1,2);
plot(t, omega, 'b');
xlabel('Temps (s)');
ylabel('Vitesse angulaire (rad/s)');
title('Évolution de la Vitesse Angulaire');
```

3. Optimisation avec Contraintes

L'optimisation du moteur électrique implique l'ajustement des paramètres pour minimiser les pertes énergétiques tout en maintenant une performance adéquate. Nous utiliserons la méthode des multiplicateurs de Lagrange pour gérer les contraintes et optimiser les paramètres du moteur.

Définir le Problème d'Optimisation

1. **Fonction Objectif** : Minimiser les pertes de puissance, proportionnelles à $I^2R(I)$.
2. **Contraintes** :
   - Vitesse angulaire cible ω_{cible}.
   - Courant maximum $I_{\max}$.

Formulation de la Fonction Lagrangienne

La fonction Lagrangienne L inclut la fonction objectif et les contraintes :

$$L = \int_0^{t_{\text{fin}}} I^2 R(I) dt + \lambda_1(\omega(t_{\text{fin}}) - \omega_{\text{cible}}) + \lambda_2(I_{\max} - I(t))$$

où λ_1 et λ_2 sont les multiplicateurs de Lagrange.

.Implémentation en Matlab de l'Optimisation avec Contraintes

```
% Définir les cibles
omega_target = 100; % Vitesse angulaire cible (rad/s)
I_max = 10; % Courant maximum (A)

% Initialiser les multiplicateurs de Lagrange
lambda1 = 1;
lambda2 = 1;

% Méthode de descente de gradient pour l'optimisation
for iter = 1:100
    % Calculer les trajectoires actuelles avec les paramètres actuels
    % (Utilisation du code de simulation ci-dessus)

    % Calculer les dérivées des multiplicateurs de Lagrange
    dL_dI = 2 * I .* R + I.^2 * R1;
    dL_domega = omega(end) - omega_target;

    % Mise à jour des multiplicateurs de Lagrange
    lambda1 = lambda1 - 0.01 * dL_domega;
    lambda2 = lambda2 - 0.01 * max(0, I - I_max);

    % Ajuster les paramètres pour minimiser L
    I = I - 0.01 * (dL_dI + lambda1 * dL_domega + lambda2 * max(0, I - I_max));
end

% Afficher les résultats optimisés
disp(['Vitesse angulaire finale : ', num2str(omega(end)), ' rad/s']);
disp(['Courant maximum : ', num2str(max(I)), ' A']);
```

Conclusion

L'optimisation des moteurs électriques non linéaires nécessite une approche analytique et numérique robuste, impliquant des modèles dynamiques précis et des techniques d'optimisation avancées. En utilisant des méthodes telles que la méthode de Runge-Kutta pour la simulation et la méthode des multiplicateurs de Lagrange pour l'optimisation, il est possible d'améliorer significativement l'efficacité énergétique, de minimiser les pertes, et d'atteindre

les objectifs de performance souhaités.Cette approche fournit une base solide pour le développement et l'amélioration des moteurs électriques modernes, ouvrant la voie à de nouvelles innovations dans les domaines de l'efficacité énergétique et de la durabilité industrielle. L'intégration de la simulation avec des techniques d'optimisation permet non seulement de modéliser le comportement dynamique du moteur mais aussi de le modifier pour atteindre des performances optimales, ce qui est essentiel pour répondre aux exigences croissantes en matière de consommation d'énergie et de réduction des coûts dans les systèmes industriels et commerciaux.

Project 4 :

Les algorithmes génétiques (GA) et l'optimisation par essaims particulaires (PSO) à un système non linéaire

Appliquer des méthodes d'optimisation stochastique telles que les algorithmes génétiques (GA) et l'optimisation par essaims particulaires (PSO) à un système non linéaire multivariable avec des contraintes est une tâche complexe et rigoureuse. Ce document fournira une formulation complète du problème d'optimisation, expliquera les concepts théoriques sous-jacents, et offrira un exemple de code Matlab pour implémenter ces techniques d'optimisation avec des valeurs numériques spécifiques. L'objectif est de fournir un guide détaillé et pratique pour l'optimisation stochastique des systèmes non linéaires multivariables.

Introduction

Les systèmes non linéaires multivariables sont courants dans de nombreux domaines de l'ingénierie et des sciences appliquées, tels que les systèmes mécaniques, électriques, biologiques, et économiques. Ces systèmes sont caractérisés par des relations complexes entre plusieurs variables d'état et de contrôle, rendant leur optimisation difficile. Les méthodes d'optimisation stochastique, telles que les algorithmes génétiques (GA) et l'optimisation par essaims particulaires (PSO), offrent des solutions robustes pour ces types de problèmes en explorant efficacement l'espace de recherche et en évitant les minima locaux.

Ce document se concentrera sur la formulation d'un problème d'optimisation stochastique pour un système non linéaire multivariable, en appliquant des contraintes spécifiques et en utilisant des valeurs numériques pour illustrer les concepts. Nous fournirons également un exemple de code Matlab pour implémenter ces méthodes d'optimisation, permettant ainsi de résoudre le problème de manière pratique.

1. Description du Système Non Linéaire Multivariable

1.1 Modèle Mathématique du Système

Considérons un système dynamique non linéaire décrit par un ensemble d'équations différentielles. Le système est caractérisé par deux variables d'état x_1 et x_2, et deux variables de contrôle u_1 et u_2. Les équations de ce système sont les suivantes :

$$\frac{dx_1}{dt} = x_1^2 - x_2 + u_1$$

$$\frac{dx_2}{dt} = x_1 \cdot x_2 + u_2$$

$$y = x_1^2 + x_2^2$$

Où :

- x_1 et x_2 sont les variables d'état du système.
- u_1 et u_2 sont les variables de contrôle.
- y est la sortie du système, représentant une fonction non linéaire des états.

.

1.2 Objectifs du Système

L'objectif principal est de minimiser une fonction coût qui inclut l'énergie consommée, les pertes, et la déviation par rapport à une sortie désirée ydy_dyd. La fonction objectif est formulée comme suit :

$$J = \alpha \cdot E(x_1, x_2, u_1, u_2) + \beta \cdot P(x_1, x_2, u_1, u_2) + \gamma \cdot \frac{1}{T_{\text{response}}} + \delta \cdot (y - y_d)^2$$

Où :

- $E(x_1, x_2, u_1, u_2)$ est l'énergie totale consommée par le système.
- $P(x_1, x_2, u_1, u_2)$ représente les pertes.
- T_{response} est le temps de réponse du système.
- y est la sortie réelle du système.
- y_d est la sortie désirée (fixée à $y_d = 5$ pour cet exemple).
- $\alpha, \beta, \gamma, \delta$ sont des coefficients de pondération.

1.3 Contraintes

Le problème d'optimisation est soumis à plusieurs contraintes :

1. **Limites sur les Variables de Contrôle** :

$$0 \leq u_1 \leq 10$$

$$0 \leq u_2 \leq 10$$

2. **Limites sur les Variables d'État** :

$$-10 \leq x_1 \leq 10$$

$$-10 \leq x_2 \leq 10$$

3. **Limitation de Courant** (si applicable aux variables de contrôle représentant des courants) :

$$u_1, u_2 \leq 10$$

4. **Limitation de Température** (si applicable) :

$$T(x_1, x_2) \leq 80$$

2. Méthodes d'Optimisation Stochastique

2.1 Algorithmes Génétiques (GA)

Les algorithmes génétiques sont une méthode d'optimisation inspirée par la sélection naturelle. Ils sont bien adaptés aux problèmes d'optimisation non convexes où les méthodes classiques peuvent échouer à trouver l'optimum global. Le processus GA comprend les étapes suivantes :

1. **Initialisation** : Génération d'une population initiale de solutions aléatoires.
2. **Évaluation** : Calcul des valeurs de la fonction objectif pour chaque individu de la population.
3. **Sélection** : Choisir les individus les plus performants pour la reproduction.
4. **Croisement** : Combiner des solutions pour créer une nouvelle génération.
5. **Mutation** : Introduire des variations aléatoires pour maintenir la diversité.
6. **Remplacement** : Remplacer les moins performants par les nouveaux individus.

2.2 Optimisation par Essaims Particulaires (PSO)

L'optimisation par essaims particulaires est une technique qui simule le comportement collectif d'un groupe d'individus, ou particules, se déplaçant dans l'espace de recherche. Chaque particule ajuste sa position en fonction de sa propre expérience et de celle de ses voisins. Les étapes clés de PSO incluent :

1. **Initialisation** : Positionnement des particules de manière aléatoire dans l'espace de recherche.
2. **Mise à jour des vitesses et positions** : Ajustement en fonction de la meilleure position trouvée par chaque particule et par l'ensemble de l'essaim.
3. **Évaluation** : Calcul de la valeur de la fonction objectif pour chaque particule.
4. **Mise à jour des meilleures positions** : Ajuster les meilleures positions individuelles et globales en fonction des nouvelles évaluations.

3. Implémentation en Matlab

3.1 Configuration du Modèle et des Paramètres

Avant de mettre en œuvre les algorithmes GA et PSO, définissons les paramètres et les fonctions nécessaires pour modéliser le système. Les valeurs numériques seront utilisées pour fournir des ***résultats concrets.***

```
% Définir les paramètres de pondération
alpha = 0.5; % Poids pour l'énergie
beta = 0.3; % Poids pour les pertes
gamma = 0.2; % Poids pour la rapidité
delta = 0.1; % Poids pour l'écart par rapport à la sortie désirée
y_d = 5; % Sortie désirée

% Bornes des variables de contrôle
lb = [0, 0]; % Limites inférieures pour u1 et u2
ub = [10, 10]; % Limites supérieures pour u1 et u2
```

3.2 Fonction Objectif

La fonction objectif doit être définie de manière à inclure toutes les composantes importantes du système, telles que l'énergie consommée, les pertes et la rapidité.

```
% Fonction objectif
function J = objective(params)
    u1 = params(1);
    u2 = params(2);
    % Simuler le système pour obtenir les valeurs d'E, P, T_response, et y
    [E, P, T_response, y] = simulate_system(u1, u2);
    J = alpha * E + beta * P + gamma / T_response + delta * (y - y_d)^2;
end
```

3.3 Fonctions de Simulation et de Contraintes

Ces fonctions sont essentielles pour modéliser la dynamique du système et pour vérifier que les contraintes sont respectées.

```
% Simulation des dynamiques du système
function [E, P, T_response, y] = simulate_system(u1, u2)
    % Exemples simplifiés de calculs de performance
    E = u1^2 + u2^2; % Énergie consommée
    P = u1 + u2; % Pertes (simplifiées)
```

```
    T_response = 1 / (1 + exp(-u1 - u2)); % Temps de réponse (non linéaire)
    y = u1^2 + u2^2; % Sortie (simplifiée)
end

% Contraintes
function [c, ceq] = constraints(params)
    ceq = []; % Pas de contraintes d'égalité
    u1 = params(1);
    u2 = params(2);
    [T, y] = simulate_constraints(u1, u2); % Simulation pour contraintes
    c = [u1 - 10; u2 - 10; -u1; -u2; T - 80]; % Contraintes d'inégalité
end

% Fonction pour simuler les contraintes
function [T, y] = simulate_constraints(u1, u2)
    T = 60 + 0.5 * (u1 + u2); % Température (exemple)
    y = u1^2 + u2^2; % Sortie (comme dans la simulation principale)
end
```

3.4 Implémentation de l'Optimisation avec GA et PSO

Nous allons maintenant implémenter les algorithmes GA et PSO en utilisant les fonctions définies ci-dessus.

Algorithme Génétique (GA) :

```
% Options pour l'optimisation GA
options_ga = optimoptions('ga', 'Display', 'iter', 'PopulationSize', 50, 'MaxGenerations', 100);

% Exécution de l'optimisation GA
params_opt_ga = ga(@objective, 2, [], [], [], [], lb, ub, @constraints, options_ga);

disp('Paramètres optimisés avec GA:');
disp(params_opt_ga);
```

Optimisation par Essaims Particulaires (PSO) :

```
% Options pour l'optimisation PSO
options_pso = optimoptions('particleswarm', 'Display', 'iter', 'SwarmSize', 50, 'MaxIterations', 100);

% Exécution de l'optimisation PSO
params_opt_pso = particleswarm(@objective, 2, lb, ub, options_pso);
```

```
disp('Paramètres optimisés avec PSO:');
disp(params_opt_pso);
```

4. Analyse des Résultats

Après l'exécution des algorithmes GA et PSO, il est essentiel d'analyser les résultats pour s'assurer que les contraintes sont respectées et que la fonction objectif est minimisée. Voici quelques approches pour analyser les résultats :

- **Vérification des contraintes** : S'assurer que les valeurs optimisées de u1u_1u1 et u2u_2u2 respectent les contraintes de courant et de température.
- **Comparaison des performances** : Comparer les valeurs de la fonction objectif avant et après optimisation pour évaluer l'amélioration.
- **Comparaison des méthodes** : Comparer les résultats obtenus avec GA et PSO pour déterminer quelle méthode est plus efficace pour ce problème spécifique.

Conclusion

L'optimisation stochastique avec contraintes pour des systèmes non linéaires multivariables est une approche puissante pour résoudre des problèmes complexes d'optimisation. Les algorithmes génétiques et l'optimisation par essaims particulaires sont particulièrement adaptés pour gérer des espaces de recherche non convexes et des fonctions objectifs complexes. Les exemples fournis montrent comment ces techniques peuvent être mises en œuvre dans Matlab, offrant des solutions optimales tout en respectant les contraintes définies.

Références

1. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
2. Kennedy, J., & Eberhart, R. C. (1995). *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks.
3. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
4. Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.