PROJET ALGORITHMIQUE 2 – LANGAGE PYTHON ET C

MATRICES TRIDIAGONALES, MÉTHODE DES DIFFÉRENCES FINIES EN 1D, CONVERGENCE EN MAILLAGE

VALENTINE CROIBIEN – MASTER 1 MATHÉMATIQUES ET APPLICATIONS
PARCOURS STATISTIQUES POUR L'ÉVALUATION ET LA PRÉVISION

INTRODUCTION

Un problème elliptique en une dimension est un problème du type :

$$\begin{cases} -\varepsilon u''(x) + u(x) = f(x) \text{ dans }]0, L[\\ u(0) = a \,;\, u(L) = b \end{cases}$$

Pour résoudre ce type de problème, nous pouvons utiliser la méthode des différences finies en une dimension. Pour cela, nous définissons un maillage de l'intervalle [0, L] à n points et de pas h = L/(n-1).

Le schéma aux différences finies classique correspondant au problème elliptique, où u_i approxime $u(x_i)$, est donné par :

$$\begin{cases} \frac{\varepsilon}{h^2}(-u_{i-1} + 2u_i - u_{i+1}) + u_i = f(x_i),\ \forall i = 1, ..., n-2 \\ u_0 = a \,;\, u_{n-1} = b \end{cases}$$

Dans ce projet, nous allons mettre ce problème en application pour :

$$\begin{cases} -10^{-5}\, u''(x) + u(x) = 0 \text{ dans }]0, 1[\\ u(0) = 1 \,;\, u(1) = 0 \end{cases}$$

Dont la solution analytique est : $u(x) = \left(\frac{1}{1 - e^{-2/\sqrt{\varepsilon}}}\right) e^{-x/\sqrt{\varepsilon}} + \left(\frac{1}{1 - e^{+2/\sqrt{\varepsilon}}}\right) e^{+x/\sqrt{\varepsilon}}$.

ÉTAPE N°1 : SOLVEUR LU POUR UN SYSTÈME LINÉAIRE TRIDIAGONAL

La première étape de ce projet était de programmer un solveur LU pour un système linéaire tridiagonal.

Un solveur LU permet de résoudre un système linéaire en utilisant la factorisation LU d'une matrice.

En effet, un système linéaire peut être écrit sous forme matricielle : $Ax = B$.

Réaliser une factorisation LU de la matrice A consiste à trouver une matrice L triangulaire inférieure dont la diagonale est remplie de 1 et trouver une matrice U triangulaire supérieure.

Dans le cas d'un système linéaire triadiagonale, la matrice carrée A de taille n est alors de la forme :

$$A = \begin{pmatrix} a_1 & b_1 & & \\ c_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & c_{n-1} & a_n \end{pmatrix}$$

La factorisation LU de la matrice A est donc composée des matrices L et U de la forme :

$$L = \begin{pmatrix} 1 & & & \\ f_1 & 1 & & \\ & \ddots & \ddots & \\ & & f_{n-1} & 1 \end{pmatrix} \qquad U = \begin{pmatrix} d_1 & e_1 & & \\ & \ddots & \ddots & \\ & & d_{n-1} & e_{n-1} \\ & & & d_n \end{pmatrix}$$

On peut définir les vecteurs d, e et f par :

Pour k allant de 1 à n-1 :

$f_k = c_k / a_k$
$d_k = a_k$
$e_k = b_k$
$a_k = a_k - d_k$
$a_{k+1} = a_{k+1} - f_k * e_k$
$b_k = b_k - f_k * d_k$
$c_k = c_k - e_k$

$d_n = a_n$

Finalement, pour la résolution de ce système linéaire nous utilisons les méthodes de remontée et descente de la manière suivante :

$$Ax = B$$
$$LUx = B$$

Nous remplaçons Ux par y :

$$Ly = B$$

L étant une matrice triangulaire inférieure, nous pouvons résoudre ce système par la méthode de descente. Nous avons donc maintenant :

$$Ux = y$$

U étant une matrice triangulaire supérieure, nous pouvons donc résoudre ce système par la méthode de remontée.

Afin de pouvoir définir un objet de type Vecteur en C, nous devons créer un type Vecteur et toutes les fonctions et procédures correspondantes à ce type d'objet :

- D'allouer de la mémoire pour un objet ;
- Libérer l'espace utilisé par un objet ;
- D'afficher un vecteur ;
- D'affecter un vecteur.

Le programme solveurLU.c est composé de ces fonctions et procédures mais également des procédures permettant de réaliser la factorisation LU d'une matrice tridiagonal définie par des vecteurs, la remontée, la descente et le solveur tridiagonal.

Dans les paramètres des procédures nous utilisons des pointeurs pour faire référence aux Vecteurs ciblés.

Nous validons ces fonctions et procédures dans la partie `int main()` du code solveur LU.

Les complexités algorithmiques des procédures FactoLU, remontee et descente sont toutes de BLAS (1) :

Complexité FactoLU : $\sum_{k=1}^{n-1} 7 = 7(n-1) = 7n - 7$

Complexité remontee : $1 + \sum_{k=1}^{n-1} 3 = 1 + 3(n-1) = 3n - 2$

Complexité descente : $\sum_{k=2}^{n} 2 = 2(n-1) = 2n - 2$

ÉTAPE N°2 : PROBLÈME ELLIPTIQUE 1D ET MÉTHODE DES DIFFÉRENCES FINIES

La deuxième étape de ce projet était de programmer un solveur MDF pour résoudre un problème elliptique.

Pour cela, nous avons besoin de la formulation matricielle du problème. Cette formulation s'écrit de la manière suivante :

$$AUh = B$$

avec $A = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ -\lambda & \beta & -\lambda & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -\lambda & \beta & -\lambda \\ 0 & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}$ une matrice tridiagonale de taille n,

$Uh = (u_i), \forall\, i \in [\![0, n-1]\!]$ et $B = \begin{pmatrix} a \\ f(x_i) \\ \vdots \\ f(x_{n-1}) \\ b \end{pmatrix}$, $\lambda = \frac{\varepsilon}{h^2}$ et $\beta = 2\lambda + 1$, où

a, b, ε et h définissent le schéma aux différences finies du problème elliptique.

La matrice A étant tridiagonale, nous pouvons utiliser la factorisation LU afin de résoudre le problème elliptique avec la méthode des différences finies.

Le programme solveurMDF.c est composé de toutes fonctions nécessaires pour réaliser la résolution de système linéaire tridiagonal par factorisation LU.

La procédure solveur_MDF_1D permet de définir le système linéaire tridiagonal correspondant au problème elliptique puis de le résoudre en utilisant la procédure solveurLU et enfin de calculer l'erreur entre la solution approchée par le solveurLU et la solution exacte.

La partie `int main()` du code résout le problème elliptique en dimension une pour différentes valeurs du nombre d'intervalles permettant de créer le maillage.

Dans ce programme, nous créons également des fichiers textes contenant les valeurs de X (le maillage), les valeurs de la solution exacte U, les valeurs approchées de la solution Uh et un fichier convergence.txt contentant le nombre d'intervalles demandé, le pas h du maillage et l'erreur associée.

ÉTAPE N°3 : ÉTUDE GRAPHIQUE DE LA CONVERGENCE EN MAILLAGES

La dernière étape de ce projet était d'étudier les résultats obtenus à partir de notre programme solveurMDF.c.

Pour cela, nous avons créé un programme main.py qui demande à l'utilisateur les nombres d'intervalles sur lesquels il souhaite travailler, puis lance le programme solveurMDF.c.

Une fois le programme en c exécuté, le programme en python récupère les informations des fichiers créés par le programme en c. À partir des données récupérées, nous pouvons à présent afficher la solution obtenue et la solution exacte pour comparer les résultats. Nous avons choisi d'afficher les résultats pour la première et la dernière valeur du nombre d'intervalles choisies par l'utilisateur.

Nous avons également dans un autre graphique l'erreur U-Uh pour ces deux nombres d'intervalles.

Finalement, nous utilisons les données du fichier Convergence.txt pour tenter de modéliser l'ordre de convergence p. Pour cela, nous utilisons la fonction stats.linregress() du package scipy. Sur un même graphique, nous affichons la droite de la régression linéaire obtenue auparavant, les points de coordonnées (-log(h), -log(einf)) et finalement l'ordre de convergence p.

L'ordre de convergence p correspond au coefficient directeur de la droite de la régression linéaire. Nous ne l'avons donc pas calculé graphiquement mais avons affiché la valeur de `lr.slope`.