TP d'analyse numérique n° 2 : système linéaire

Introduction

Pour réaliser ce TP, des codes à compléter sont à votre disposition à l'adresse suivante :

`https://github.com/LucasMorlet/AnalyseNumerique/`

Pour programmer du Python, vous pouvez utilisez une installation sur votre machine personnelle, Spyder sur les machines de l'école, ou même travailler directement en ligne avec `https://www.onlinegdb.com/online_python_compiler`

A la fin du TP, vous devrez rendre un compte-rendu. Dans ce compte-rendu vous mettrez les réponses aux questions posées ainsi que les codes sources que vous avez produits. Ne mettez que les fonctions demandées, les fonctions de tests ne seront pas utiles au correcteur. Pour faciliter la correction, essayez de respecter la structure de ce document dans votre compte-rendu (même numéro de section/sous-section). Vous rendrez votre compte-rendu sous format PDF à l'adresse suivante : `lucas.morlet@eseo.fr` ou sur la plateforme dédiée (campus numérique).

Le barème est donné à titre indicatif ci-dessous :

Partie 1 : calcul de l'antécédent (4 points)

— Matrices diagonales
— Matrices triangulaires
— Pivot de Gauss
— Résolution Lower-Upper

Partie 2 : inversion de matrices (3 points)

— Matrices diagonales
— Matrices triangulaires
— Pivot de Gauss

Partie 3 : méthodes itératives (3 points)

— Résolution générique
— Méthode de Jacobi
— Méthode de Gauss-Siedel

1 Calcul de l'antécédent

Soit une matrice carrée A et deux vecteurs-colonnes x et b tels que $Ax = b$. L'objectif de cet exercice est de trouver x pour A et b donnés. Vous allez programmez différentes méthodes pour trouver la solution de cette équation selon les propriétés de A (diagonale, triangulaire..)

1.1 Matrices diagonales

Pour commencez, programmez la fonction *resolution_diagonale* qui renvoie l'antécédent pour une matrice diagonale A et une image b. **Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.**

1.2 Matrices triangulaires

Ensuite, programmez la fonction *resolution_inferieure* qui renvoie l'antécédent pour une matrice triangulaire supérieure A et une image b. Pour résoudre cette équation, on "descend" le triangle en faisant les opérations suivantes sur chaque ligne i :

— On calcule la "partie-gauche" de la ligne : $G_i = \sum_{j=0}^{i-1} A_{i,j}x_j$ (par définition $G_0 = 0$)

— On calcule $x_i = (b_i - G_i)/A_{i,i}$

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

Copiez-coller le code de *resolution_inferieure* vers *resolution_superieure*. Vous devriez pouvoir adapter la fonction en changeant uniquement les bornes des boucles de sorte à "remonter" le triangle. **Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.**

1.3 Pivot de Gauss

Programmez la fonction *resolution_gauss* qui résout l'équation pour n'importe qu'elle matrice carrée inversible par la méthode du pivot de Gauss. Le principe du pivot de Gauss est que l'on va rajouter des zéros dans la matrice jusqu'à obtenir une matrice diagonale. Comme vous avez déjà programmé la fonction *resolution_superieure*, vous vous contenterez de rajouter des zéros sous la diagonale. Pour cela vous programmerez les étapes suivantes pour chaque ligne i de la matrice A :

— choix du pivot : $p = A_{i,i}$ (par convention le pivot est choisi sur la diagonale)

— pour chaque ligne k en dessous de la ligne i :

  — calcul du coefficient de la ligne : $c = A_{k,i}/p$

  — modifier la ligne k telle que $L_k \leftarrow L_k - cL_i$ (ainsi $A_{k,i} = 0$)

  — diminuer b_k de cb_i

Une fois ces étapes finies, il ne vous reste plus qu'une matrice triangulaire supérieure, dont vous avez déjà programmé la résolution.

NB : chaque modification effectuée sur une ligne k de la matrice doit être également effectuée sur le coefficient b_k de l'image pour trouver la solution.

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

1.4 Méthode Lower-Upper

Programmez la fonction *resolution_LU* qui résout l'équation pour n'importe qu'elle matrice carrée inversible par la méthode Lower-Upper (abrégée LU). Cette méthode propose de décomposer la matrice A en deux matrices L et U, respectivement triangulaire inférieure et supérieure telles que $A = LU$. Ainsi résoudre $Ax = b$ revient à résoudre $(LU)x = b \Leftrightarrow L(Ux) = b$. En posant $y = Ux$ on obtient deux résolutions consécutives qui nécessitent uniquement de savoir résoudre pour une matrice triangulaire inférieure et supérieure : d'abord trouver y tel que $Ly = b$ puis résoudre $Ux = y$.

Pour décomposer une matrice quelconque en une matrice triangulaire inférieure L et une matrice triangulaire supérieure U, vous effectuerez les étapes suivantes :

— initialisation : $U = A$ et $L = Id$

— appliquer le pivot de Gauss sur la matrice U jusqu'à avoir une matrice triangulaire supérieure.

— chaque fois que vous calculez un coefficient $c = U_{k,i}/p$, vous le placerez sur la matrice L à la position $L_{k,i}$ (chaque fois que vous ajoutez un 0 à u, vous en enlevez un de L)

Une fois ces étapes réalisées vous pouvez utiliser les résolutions que vous avez déjà programmées pour conclure.

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

2 Inversion de matrices

Soit une matrice carrée A à inverser. Vous allez programmez différentes méthodes pour inverser cette matrice selon ces propriétés (diagonale, triangulaire..).

2.1 Matrices diagonales

Pour commencez, programmez la fonction *inversion_diagonale* qui renvoie l'inverse de la matrice diagonale A. Pour rappel, l'inverse d'une matrice diagonale est la matrice diagonale dont les coefficients sont l'inverse de ceux de la première.

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

2.2 Matrices triangulaires

Programmez la fonction *inversion_inferieure* qui renvoie l'inverse de la matrice triangulaire inférieure A passée en paramètre. Pour ce faire, la méthode consiste à transformer progressivement la matrice en la matrice identité. En appliquant exactement les mêmes transformations sur la matrice identité, on obtient l'inverse de la matrice de départn.

Dans le cas d'une matrice triangulaire inférieure, les étapes à appliquer sur chaque ligne i (de haut en bas) sont les suivantes :

- Diviser la ligne par l'élément sur la diagonale $L_i \leftarrow L_i / A_{i,i}$
- Faire de même pour la matrice inverse
- Pour chaque ligne k en dessous de la ligne i :
  - calcul du coefficient de la ligne : $c = A_{k,i}$
  - modifier la ligne k telle que $L_k \leftarrow L_k - cL_i$ (ainsi $A_{k,i} = 0$)
  - Faire la même modification sur la matrice inverse

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

Copiez-coller le code de *inversion_inferieure* vers *inversion_superieure*. Vous devriez pouvoir adapter la fonction en changeant uniquement les bornes des boucles de sorte à "remonter" le triangle. **Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.**

2.3 Matrices quelconques

Pour une matrice inversible n'ayant aucune propriété particulière, il suffit de faire les opérations pour une matrice triangulaire inférieure et pour une triangulaire supérieure l'une à la suite de l'autre. Il s'agit de la méthode du pivot de Gauss pour inverser les matrices.

Programmez la fonction *inversion_gauss* qui renvoie l'inverse d'une matrice quelconque A passée en paramètre. **Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.**

3 Méthodes itératives

Soit une matrice carrée A et deux vecteurs-colonnes x et b tels que $Ax = b$. L'objectif de cet exercice est de trouver x pour A et b donnés. Vous allez programmez deux méthodes itératives pour trouver la solution de cette équation.

Les méthodes itératives consistent à proposer une solution (fausse) et à corriger cette solution de proche en proche. Si la matrice d'itération possède certaines propriétés, on est assurés que la solution converge vers un point-fixe qui est la solution de l'équation. Dans le cas présent, on s'assure de la convergence en étudiant une matrice dont la diagonale est strictement dominante.

3.1 Résolution générique

Résoudre $Ax = b$ revient à résoudre $(M - N)x = b$ avec $A = M - N$.

$$\begin{aligned} (M - N)x &= b \\ Mx - Nx &= b \\ Mx &= Nx + b \\ x &= M^{-1}(Nx + b) \end{aligned}$$

x est donc le point fixe de la transformation $\Phi(u) = M^{-1}(Nu + b)$. Si la transformation est contractante, on peut construire une suite $u_{n+1} = \Phi(u_n)$ qui converge vers x. L'avantage de cette méthode est qu'elle est applicable sur n'importe quelle matrice, il suffit de choisir M et N (pour rappel $A = M - N$) telles que M est facilement inversible (*e.g.* diagonale ou triangulaire).

Programmez la fonction *resolution_iterative* qui résout par itération l'équation et renvoie le point-fixe. Cette fonction prend en paramètre M^{-1}, N et b. Pour l'instant, vous ne pouvez pas tester cette fonction, car il faut encore définir M et N. Pour cela, les deux méthodes suivantes sont à implémenter.

3.2 Méthode de Jacobi

La méthode de Jacobi est une méthode itérative. Elle décompose la matrice A en deux matrices :

— M qui est la diagonale de A (donc facilement inversible)

— N qui est tous les autres coefficients passés en négatif

Programmez la fonction *resolution_jacobi* qui décompose A en M et N, inverse M, puis appelle la fonction *resolution_iterative*. Pour inverser M, vous importerez la fonction *inversion_diagonale* que vous avez implémenté dans l'exercice précédent.

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.

3.3 Méthode de Gauss-Siedel

La méthode de Gauss-Siedel est aussi une méthode itérative. Elle décompose la matrice A en deux matrices :

— M qui est la triangulaire inférieure de A (donc facilement inversible)

— N qui est la triangulaire strictement supérieure de A passée en négatif

Programmez la fonction *resolution_gauss_siedel* qui décompose A en M et N, inverse M, puis appelle la fonction *resolution_iterative*. Pour inverser M, vous importerez la fonction *inversion_inferieure* que vous avez implémenté dans l'exercice précédent.

Faites une capture d'écran de votre code et de vos résultats et ajoutez-la à votre rapport.