Magistère de mathématiques

TP 3 – Méthodes directes de résolution de $Ax = b$ et méthodes itératives.

Les résolutions de systèmes linéaires n'utiliseront pas la commande `np.linalg.solve` durant ce TP.

Théorème (Décomposition LU)**.** *Soit $A \in \mathcal{M}_n(\mathbb{K})$, dont tous les mineurs principaux sont non nuls, c'est-à-dire que pour tout $k \in \{1, \dots, n\}$ la sous-matrice extraite $(A_{i,j})_{1 \leq i,j \leq k}$ est inversible.*
Alors il existe un unique couple (L, U) de matrices carrées d'ordre n tel que :
- *L est triangulaire inférieure avec uniquement des 1 sur la diagonale ;*
- *U est triangulaire supérieure avec des coefficients diagonaux tous non nuls ;*
- *$A = LU$.*

Une fois cette décomposition connue, la résolution d'un système linéaire $Ax = b$ est équivalente à la résolution de deux systèmes linéaires triangulaires, élémentaires à résoudre :

$$(Ax = b) \iff (Ly = b \text{ et } Ux = y).$$

Exercice 1. *Résolution de systèmes triangulaires*

• Programmer les algorithmes de descente et de remontée permettant de résoudre des systèmes triangulaires inférieurs ($Ly = b$) et supérieurs ($Ux = y$) respectivement.

Exercice 2. *Algorithme de décomposition LU*

L'algorithme de construction des matrices L et U s'appuie sur les opérations d'élimination du pivot de Gauss. Pour réduire de moitié la complexité en espace (**forme compacte**), l'algorithme agit directement sur la matrice A, la matrice U étant construite au fil des itérations dans la partie supérieure de A, et la matrice L correspondant aux opérations du pivot étant stockée dans la partie inférieure de A. On a ainsi à l'issue de l'algorithme $U_{i,j} = A_{i,j}$ si $i \leq j$, et $L_{i,j} = A_{i,j}$ si $i > j$, les autres coefficients de L et U étant nuls ou égaux à 1.

Algorithme de la factorisation LU.
Forme compacte
Pour $k = 1, \dots, n-1$
  Pour $i = k+1, \dots, n$
    $A_{i,k} \leftarrow A_{i,k}/A_{k,k}$
    Pour $j = k+1, \dots, n$
      $A_{i,j} \leftarrow A_{i,j} - A_{i,k}A_{k,j}$
    Fin pour j
  Fin pour i
Fin pour k.

• Programmer cet algorithme (on essayera si possible d'obtenir une programmation concise, idéalement avec seulement une boucle sur k) et le tester sur la matrice de Pascal suivante

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}.$$

Le but de la suite de ce TP est d'introduire et de comparer quelques méthodes itératives pour résoudre le système $Ax = b$.

Principe : La matrice A est décomposée sous la forme $A = M - N$ avec M facile et peu coûteuse à inverser et la solution x est approchée par une suite $(x_k)_{k \in \mathbb{N}}$ d'éléments de $\mathbb{R}^n$:

$$\begin{aligned} &x_0 \in \mathbb{R}^n \qquad \text{donné,} \\ &Mx_{k+1} = Nx_k + b. \end{aligned}$$

La matrice $M^{-1}N$ est fondamentale dans l'analyse de convergence. En effet la solution x est l'unique point fixe d'une application affine sur $\mathbb{R}^n$ dont la différentielle (constante) a pour matrice $M^{-1}N$. En particulier la convergence itérative de la méthode, requise pour tout choix de l'initialisation x_0, est équivalente à la condition suivante sur le rayon spectral :

$$\rho(M^{-1}N) < 1.$$

Quelques possibilités pour construire M et N : écrivons $A = D - E - F$ avec D la partie diagonale de A, $-E$ sa partie triangulaire inférieure stricte et $-F$ sa partie triangulaire supérieure stricte. Autrement dit

$$\begin{aligned} &D_{ii} = A_{ii},\ i = 1, \dots, n, \\ &E_{ij} = -A_{ij},\ 1 \leq j < i \leq n, \\ &F_{ij} = -A_{ij},\ 1 \leq i < j \leq n, \end{aligned}$$

tous les autres termes étant nuls. Pour la construction de ces trois parties utilisera la commande `np.diag`.

Méthode de Jacobi : $M = D$ et $N = E + F$.

Méthode de Gauss-Seidel : $M = D - E$ et $N = F$.

Méthode de relaxation : $M = \frac{1}{\omega}D - E$ et $N = \frac{1-\omega}{\omega}D + F$ avec $\omega \in \mathbb{R}_*^+$.

Exercice 3. *Comparaison des méthodes de Jacobi, Gauss-Seidel et de relaxation*

Soit la matrice carrée A de taille n, tridiagonale, intervenant dans le TP précédent, définie par

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

a. Programmer les trois méthodes précedentes.

b. Application : utiliser ces trois méthodes sur la matrice A et le second membre $B = (1, 0, \dots, 0, 1)^T$, pour $n = 10$. On effectuera 100 itérations. Dans le cas de la méthode de relaxation, on choisira $\omega = \frac{3}{2}$.

c. Dans le cas où $n = 20$, déterminer le paramètre optimal dans la méthode de relaxation, en représentant le rayon spectral de la matrice d'itération obtenue en fonction de ω.

d. Pour différentes valeurs de n, comparer le nombre d'itérations nécessaires pour avoir une précision de 10^{-12} entre la solution approchée et la solution exacte. Quelle relation y a t-il entre le nombre d'itérations et le rayon spectral ?

Exercice 4. *Résolution d'un problème de Poisson 2d*

On souhaite résoudre le problème de Poisson en dimension 2 d'espace

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \in [0, 1]^2,$$

assorti des conditions de Dirichlet au bord

$$u(x, y) = 0, \quad (x, y) \in \partial[0, 1]^2.$$

Pour ce faire, on approche l'équation par différences finies. Soit N un entier naturel, on pose $h = 1/(N + 1)$ le pas de discrétisation pour chacune des directions. Pour tout $0 \le i, j \le N + 1$, $u_{i,j}$ désigne une approximation de la valeur $u(x_i, y_j)$, les points de la grille étant $x_i = ih$, $y_j = jh$. L'EDP est approchée par des différences centrées :

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = f(x_i, y_j), \quad 1 \le i, j \le N,$$

et les conditions de bord sont données : $u_{i,j} = 0$ lorsque $ij(N + 1 - i)(N + 1 - j) = 0$.

a. Construire la matrice A de taille $N^2 \times N^2$ permettant de définir le vecteur des inconnues $U = (u_{i,j})_{1 \le i,j \le N}$ de $\mathbb{R}^{N \times N}$ comme solution du problème

$$AU = F.$$

b. En utilisant les méthodes itératives précédemment mises en œuvre, résoudre ce problème pour le second membre f donné dans l'exemple de code ci-dessous.

c. Représenter la solution. On pourra adapter l'exemple suivant pour tracer une surface en 3 dimensions :

```
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection = '3d')
n = 20
X = np.linspace(0,1,n+2)
Y = X
X, Y = np.meshgrid(X, Y)
def f(x,y):
    return -(x-0.5)**2 - (y-0.5)**2
Z = f(X,Y)
surf = ax.plot_surface(X, Y, Z,cmap = cm.coolwarm,linewidth = 0,
     antialiased = False)
plt.title("La surface f(x,y)=-(x-0.5)**2-(y-0.5)**2")
plt.show()
```

où `X` et `Y` sont les vecteurs avec les coordonnées des points du plan, et `Z` est la matrice contenant les valeurs `f(X,Y)`.