MNI-TP3-2015

April 2, 2015

TP 3 – Méthodes Numériques pour l'Ingénieur CM3 – Mars 2015

1 Systèmes linéaires

Le texte de cette session de travaux pratiques est également disponible ici
http://nbviewer.ipython.org/github/ecalzavarini/numerical-methods-at-polytech-lille/blob/master/MNI-TP3-2015.ipynb

1.0.1 Instructions pour ce TP

Pendant ce TP vous aurez à écrire plusieurs scripts (nous vous suggérons de les nommer script1.py , script2.py ,. . .)

Les scripts doivent être accompagnés par un document descriptif unique (README.txt). Dans ce fichier, vous devrez décrire le mode de fonctionnement des scripts et, si besoin, mettre vos commentaires. Merci d'y écrires aussi vos nomes et prénoms complets.

Tous les fichiers doivent etre mis dans un dossier appelé TP1-nom1-nom2 et ensuite être compressés dans un fichier TP1-nom1-nom2.tgz .

Enfin vous allez envoyer ce fichier par email à l'enseignant :
soit Enrico (enrico.calzavarini@polytech-lille.fr) ou Stefano (stefano.berti@polytech-lille.fr)

Vous avez une semaine de temps pour compléter le TP, c'est-à-dire que la date limite pour envoyer vos travaux est 7 jours après la date du TP courant.

1.1 Objectif

On se propose de résoudre le système linéaire $A\ \vec{x} = \vec{b}$ (où A représente une matrice régulière d'ordre n, $\vec{x}$ le vecteur inconnu et $\vec{b}$ le second membre) par deux méthodes :

a) la méthode directe de **Gauss** (en appliquant la stratégie du pivot partiel)

b) la méthode itérative de **Jacobi** (avec le critère d'arrêt $||\vec{x}_k - \vec{x}_{k+1}|| < \epsilon$ où k est l'indice d'itération).

Pour cela, on écrira un script python (un pour chaque méthode). Au bout de pouvoir utiliser le même script plusieurs fois, il est pratique de faire en sorte que la matrice A et le vecteur $\vec{b}$ puissent être fournis par l'utilisateur.

1.2 Programmation et validation

On testera le programme sur les systèmes ci-dessous :

$$A = \begin{pmatrix} 6 & 2 & 2 & 4 \\ 2 & 8 & 2 & 1 \\ 4 & 2 & 16 & 8 \\ 2 & 4 & 1 & 9 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 1 \\ 2 \\ 4 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 19 & 5 & 7 \\ 2 & 7 & 2 \\ 1 & 6 & 11 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 8 \\ 2 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 7 & 0 & 1 & -1 \\ 3 & 10 & 5 & 0 \\ 0 & 4 & 9 & 2 \\ 1 & -5 & 3 & 15 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 4 \\ 3 \\ -3 \\ 12 \end{pmatrix}$$

Dans l'application de la méthode de Jacobi, on étudiera l'influence de la précision ϵ sur la solution (et le nombre d'itérations nécessaires pour l'obtenir).

Remarque : Avant d'appliquer la méthode de Jacobi, on fera le test de la diagonale dominante de la matrice A. Le vecteur initial sera entré par l'utilisateur (nous vous suggérons d'essayer le vecteur où toutes les composantes sont égales à l'unité : $[1, 1, 1, \ldots]$).

Formuler une conclusion en comparant les résultats obtenus par les deux méthodes.

1.3 Application

On dispose de $n = 7$ points expérimentaux de coordonnées (x_i, y_i) $(i = 1, ..., n)$ suivantes:

$$\begin{array}{c} (-1.5, 0.9) \\ (-1.0, 1.2) \\ (-0.5, -.08) \\ (0.0, -2.0) \\ (0.5, -1.3) \\ (1.0, -0.5) \\ (1.3, 0.5) \end{array}$$

En utilisant la méthode de Gauss, trouver les paramètres $a_k (k = 0, 1, 2)$ du modèle parabolique $y(x) = a_0 + a_1 x + a_2 x^2$ approximant au mieux les données au sens des moindres carrés. Les valeurs des paramètres seront déterminées en résolvant le système linéaire suivant :

$$\begin{pmatrix} n & \Sigma_{i=1}^n x_i & \Sigma_{i=1}^n x_i^2 \\ \Sigma_{i=1}^n x_i & \Sigma_{i=1}^n x_i^2 & \Sigma_{i=1}^n x_i^3 \\ \Sigma_{i=1}^n x_i^2 & \Sigma_{i=1}^n x_i^3 & \Sigma_{i=1}^n x_i^4 \end{pmatrix} \quad \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \Sigma_{i=1}^n y_i \\ \Sigma_{i=1}^n x_i y_i \\ \Sigma_{i=1}^n x_i^2 y_i \end{pmatrix}$$

Les points et la courbe d'approximation obtenue seront visualisés sur un graphique.

Rappel de Python Pour définir une matrice ou un vecteur :

```
In [3]: import numpy as np
        A = np.array([[3,1],[-1,2]] ,float)
        b = np.array([2,-2],float)
```

Pour entrer des matrices et des vecteurs, ainsi que pour les combiner dans une seule matrice:

```
In []: import numpy as np

       N=input("N=?") # taille du probleme

       A=input("A=?") # lecture de la matrice A, p.ex. [[1,2],[3,4]]
       b=input("b=?") # lecture du vecteur b, p.ex. [0,1]

       # on définit A,b comme des arrays numpy
       A=np.asarray(A,float)
       b=np.asarray(b,float)

       # on ajoute b comme dernière colonne à A pour obtenir la matrice augmentée Ab
       Ab=np.column_stack((A,b))
```

```
        # on verifie
        print("A=")
        print(A)
        print("b=")
        print(b)
        print("Ab=")
        print(Ab)
```

Vous pouvez vérifier la solution du système linéaire $A\vec{x} = \vec{b}$ en utilisant la fonction **linalg.solve** de la bibliothèque Numpy :

```
In []: x = np.linalg.solve(A, b)
        print(x)
```

autres fonctions utiles de Numpy :

```
In []: np.dot(A,b) # produit scalaire matrice vecteur ( c'est différent de A*b! )

        np.sum(b)   # sommation des composantes du vecteur b
```