MNI-TP4-2016

May 19, 2016

TP 4 – Méthodes Numériques pour l'Ingénieur CM3

1 TP de controle

1.0.1 Instructions pour ce TP

Pendant ce TP vous aurez à écrire deux 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 votre nom et prénom complet.

Tous les fichiers doivent être mis dans un dossier appelé TP4-prenom-NOM et ensuite être compressés dans un fichier TP4-prenom-NOM.tgz .

Enfin vous allez envoyer ce fichier par email à l'enseignant :

soit Enrico (enrico.calzavarini@polytech-lille.fr) soit Stefano (stefano.berti@polytech-lille.fr)

Vous avez deux heures de temps pour compléter le TP.

1.1 Exercice 1: Equations différentielles ordinaires

Ecrire un script Python permettant de résoudre une équation différentielle ordinaire du second ordre $y'' = f(x, y(x))$ avec les conditions initiales $y(x_0) = y_0$ et $y'(x_0) = v_0$ par la méthode suivante:

$$y_{i+1} = y_i + v_i h + \frac{1}{2} a_i h^2$$

$$v_{i+1} = v_i + \frac{1}{2}(a_i + a_{i+1})h$$

$$i = 0, 1, 2, \ldots$$

Il s'agit d'une méthode d'ordre 2 appelée $leapfrog$, dans laquelle $v_i = y'(x_i)$ et $a_i = f(x_i, y_i)$.

Faire en sorte que le pas d'intégration h puisse être rentré par l'utilisateur. La fonction $f(x, y)$ pourra être définie à l'aide d'une $function$ dans le script.

a) Tester le script pour le cas oscillatoire $f(x, y) = -ky$ ayant pour solution $y(x) = y_0 \cos(\sqrt{k}x)$; prendre $k = 16$, $x_0 = 0$, $y_0 = 3$, $v_0 = 0$, $h = 0.01$. Faire varier x variant de 0 à 10.

b) Utiliser maintenant le script validé au point a) pour résoudre l'équation $y'' = f(x, y(x))$ avec la fonction $f(x, y)$ et les conditions initiales $y(0) = y_0$, $y'(x_0) = v_0$ fournies par votre enseignant, pour x variant de 0 à 10.

Faire les calculs en faisant varier le pas: $h = 0.01, 0.1, 1$ et comparer la valeur de y au point final $x = 10$.

Les valeurs de $y(x = 10)$ obtenues numériquement doivent être affichées de façon claire sur l'écran par le script et dans le compte-rendu dans un tableau.

c) Pour chaque valeur du pas h, représenter sur un graphique les courbes correspondantes aux solutions numériques y_i et v_i obtenues, sur l'interval $x_i \in [0, 10]$.

d) Représenter sur un autre graphique la courbe correspondante à v_i fonction de y_i en utilisant les valeurs numériques de y_i et v_i obtenues.

e) Comparer la solution numérique obtenue au point b), avec celle qui peut être obtenue avec la méthode (d'ordre 1) d'Euler $\vec{y}_{i+1} = \vec{y}_i + h\,\vec{f}(x_i, \vec{y}_i)$, avec $\vec{y}_i = (y(x_i), y'(x_i))$, pour $h = 0.01$.

Que peut-on conclure de cette comparaison?

Bonus

f) Comparer la solution numérique obtenue au point b), avec celle qui peut être obtenue avec la méthode du point au milieu $\vec{y}_{i+1} = \vec{y}_i + h\,\vec{f}(x_i + \frac{h}{2}, \vec{y}_i + \frac{h}{2}\,\vec{f}(x_i, \vec{y}_i))$, avec $\vec{y}_i = (y(x_i), y'(x_i))$, pour $h = 0.01$.

Que peut-on conclure de cette comparaison?

1.2 Exercice 2: Valeurs et vecteurs propres

On souhaite écrire un script Phyton permettant de trouver toutes les valeurs propres, λ_i, d'une matrice A de rang n.

On utilisera ici la méthode de déflation, qui est une méthode itérative qui se base sur la méthode de la puissance itérée :

a) Algorithme pour la méthode de la puissance iterée

On choisit un vecteur initial $\vec{x}^{\,(0)}$ et on le normalise afin de générer $\vec{q}^{\,(0)} = \frac{\vec{x}^{\,(0)}}{||\vec{x}^{\,(0)}||}$, où $||\dots||$ est la norme euclidienne.

Ensuite, à chaque itération ($k = 1, 2, 3, \dots$) on calcule :

$$\vec{x}^{\,(k)} = A\vec{q}^{\,(k-1)}$$

$$\lambda^{\,(k)} = \frac{\vec{q}^{\,(k)} \cdot A\vec{q}^{\,(k)}}{\vec{q}^{\,(k)} \cdot \vec{q}^{\,(k)}}$$

$$\vec{q}^{\,(k)} = \frac{\vec{x}^{\,(k)}}{||\vec{x}^{\,(k)}||}$$

où le point entre deux vecteurs indique un produit scalaire.

Le critère d'arrêt est $|\lambda^{(k+1)} - \lambda^{(k)}| < \epsilon$ où ϵ est la précision souhaitée.

Dans la limite de $k \to +\infty$ on obtiendra alors $\lim_{k\to+\infty} \lambda^{(k)} = \lambda_1$ et $\lim_{k\to+\infty} \vec{x}^{\,(k)} = \vec{x}_1$ où λ_1 est la valeur propre de A de plus grand module et $\vec{x}_1$ le vecteur propre associé à λ_1.

Comme vecteur initial il sera possible de considérer un vecteur où toutes les composantes sont égales à l'unité : $\vec{x}_0 = [1, 1, 1, \ldots]$.

b) Algorithme pour la méthode de déflation

Considérons la matrice A utilisée dans la partie précédente. On applique une première fois la méthode de la puissance itérée sur cette matrice. Ainsi on obtient la valeur propre la plus grande en module. On appelera λ_1 cette valeur propre et $\vec{x}_1$ son vecteur propre.

Maintenant on veut trouver les valeurs propres restantes, c'est-à-dire : $\lambda_2, \ldots, \lambda_n$.

L'idée consiste à créer une matrice B_1 qui a pour valeurs propres : $0, \lambda_2, ..., \lambda_n$. Soit :

$$B_1 = A - \lambda_1 \frac{\vec{x}_1\, \vec{x}_1^T}{\vec{x}_1^T \cdot \vec{x}_1}$$

Attention: ici $\vec{x}_1 \vec{x}_1^T$ (au numérateur) est le produit d'un vecteur colonne fois un vecteur ligne; par contre $\vec{x}_1^T \cdot \vec{x}_1$ (au dénominateur) est un produit scalaire.

Si on applique la méthode de la puissance itérée à cette nouvelle matrice, on obtient la deuxième plus grande valeur propre de A, soit λ_2, et son vecteur propre associé $\vec{x}_2$.

Pour obtenir toutes les autres valeurs propres de A il suffira de réitérer cette procédure en créant à chaque fois la matrice B_i selon la règle itérative :

$$B_i = B_{i-1} - \lambda_i \frac{\vec{x}_i\, \vec{x}_i^T}{\vec{x}_i^T \cdot \vec{x}_i}$$

a) Calculer les λ_i avec $i = 1, \ldots, n$ pour la matrice A donnée par votre enseignant.

Pour ce calcul prendre la valeur de la précision $\epsilon = 10^{-4}$. Affichez les résultats de façon claire sur l'écran.

b) Comparer les résultats numeriques obtenus pour λ_i ($i = 1, \ldots, n$) avec la solution fournie par la fonction **linalg.eig()** de la bibliothèque Numpy (voir ci-dessous).

Rappels de Python concernant les vecteurs et les matrices

Les matrices et les vecteurs peuvent être definis commes des arrays Numpy dans la manière suivante:

```
In [1]: import numpy as np

        # definition d'une matrice M (4 x 4)
        M = np.array([[6,2,2,4],[2,8,2,1],[4,2,16,8],[2,4,1,9]] ,float)
        # definition d'un vecteur v (avec 4 elements)
```

```
        v = np.array([1,2,4,1],float)

        # affichage de M et v pour controle
        print("M=")
        print(M)
        print("v=")
        print(v)
```

```
M=
[[  6.   2.   2.   4.]
 [  2.   8.   2.   1.]
 [  4.   2.  16.   8.]
 [  2.   4.   1.   9.]]
v=
[ 1.  2.  4.  1.]
```

Les matrices et les vecteurs peuvent également être rentrés par un utilisateur à travers des opérations de lecture comme dans l'exemple suivant :

```
In [2]: import numpy as np

        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)

        # affichage de A et b pour controle
        print("A=")
        print(A)
        print("b=")
        print(b)
```

```
A=?[[1,2],[3,4]]
b=?[0,1]
A=
[[ 1.  2.]
 [ 3.  4.]]
b=
[ 0.  1.]
```

Vous pouvez vérifier la solution du problème aux valeurs propres $A\vec{x} = \lambda\vec{x}$ en utilisant la fonction **linalg.eig**() de la bibliothèque Numpy :

```
In [3]: val , x = np.linalg.eig(A) # calcul des valeurs et des vecteurs propres
```

```
        # pour afficher les valeurs propres
        print(val)
```

```
[-0.37228132  5.37228132]
```

```
In [4]: # pour afficher les vecteurs propres
        print(x)
```

```
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Produit matrice-vecteur avec Numpy :

```
In [5]: np.dot(A,b) # produit matrice A - vecteur b ( c'est différent de A*b! )
```

```
Out[5]: array([ 2.,  4.])
```

Produit scalaire entre deux vecteurs avec Numpy :

```
In [6]: x=np.array([1,1]) # definition du vecteur x
        y=np.array([2,3]) # definition du vecteur y
        np.dot(x,y) # produit scalaire entre le vecteur x et le vecteur y
```

```
Out[6]: 5
```

Produit entre un vecteur colonne et un vecteur ligne avec Numpy :

```
In [7]: x=np.array([1,2]) # definition du vecteur x
        y=np.array([2,3]) # definition du vecteur y
        np.outer(x,y) # produit entre le vecteur colonne x et le vecteur ligne y
```

```
Out[7]: array([[2, 3],
               [4, 6]])
```