cours - lineaire2-Copy1

September 24, 2023

1 Méthodes itératives pour systèmes linéaires et calcul de valeurs propres

Objectifs :

- Méthode de Jacobi
- Exemple de système pour lequel la méthode de Jacobi ne converge pas
- Méthode de Gauss-Seidel
- Méthode de la puissance itérée

2 Méthode de Jacobi

2.1 L'algorithme

Pour résoudre le système linéaire

$$(1) \qquad Ax = b,$$

on commence par décomposer $A = D - E - F$, où D est la diagonale de A, $-E$ représente la partie sous-diagonale de A, et $-F$ la partie sur-diagonale. Si D est inversible, résoudre (1) est donc équivalent à résoudre

$$(2) \qquad x = D^{-1}(E + F)x + y$$

où $y = D^{-1}b$. La fonction `DEF(A)` ci-dessous renvoie le triplet (D, E, F). On s'aide de la fonction `numpy.diag` qui permet soit d'extraire la diagonale d'une matrice, soit de créer une matrice diagonale, voir `help(numpy.diag)`

[2]:
```
import numpy as np
def DEF(A):
    D=np.diag(np.diag(A))
    n=len(A)
    E=np.zeros((n,n))
    F=np.zeros((n,n))
    for i in range(1,n):
        for j in range(0,i):
            E[i,j]=-A[i,j]
    for j in range(1,n):
        for i in range(0,j):
            F[i,j]=-A[i,j]
```

```
    return(D,E,F)
```

On la teste sur l'exemple de la matrice

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

[3]:
```
DEF(np.array([[1,2],[3,4]]))
```

[3]:
```
(array([[1, 0],
        [0, 4]]),
 array([[ 0.,  0.],
        [-3.,  0.]]),
 array([[ 0., -2.],
        [ 0.,  0.]]))
```

La méthode Jacobi consiste, pour résoudre (1) de manière approchée, à considérer (2) comme une équation de point fixe pour l'application $x \mapsto D^{-1}(E+F)x + y$. En supposant que cette application soit une contraction (ce qui est équivalent au fait que le rayon spectral de $D^{-1}(E+F)$ soit strictement inférieur à 1, cf votre cours), le théorème du point fixe de Banach assure que si

- Un point $x_0 \in \mathbb{R}^n$ est choisi, quelconque,
- On calcule $x_1 = D^{-1}(E+F)x_0 + y$ puis itérés successifs $x_2, x_3, ...$ par $x_k = D^{-1}(E+F)x_{k-1} + y$,

alors $x_n \to x$ l'unique solution de (1) et (2). La taille de l'erreur résiduelle $|Ax_k - b|$ tend alors vers 0. La méthode de Jacobi consiste à choisir un seuil d'erreur $\epsilon > 0$, à calculer ces itérés successifs, et à s'arrêter lorsque la taille de l'erreur résiduelle rapportée à la taille du vecteur à inverser est plus petite que le seuil :

$$\frac{|Ax_k - b|}{|b|} < \epsilon.$$

La fonction suivante `Jacobi(A,b,e)` code l'algorithme de Jacobi pour trouver une solution approchée x_{ap} de $Ax = b$ avec l'estimation sur le résidu $\frac{|Ax_{ap} - b|}{|b|} \leq e$. Pour calculer la norme euclidienne, on s'aide de la fonction `numpy.linalg.norm` qui permet de calculer de nombreuses normes vectorielles (et par défaut, sans précision de ses paramètres, calcule la norme euclidienne).

[4]:
```
def Jacobi(A,b,e):
    n=len(b)
    normb=np.linalg.norm(b)
    x=np.reshape(np.zeros(n),(n,1))
    r=2*e
    (D,E,F)=DEF(A)
    Dinv=np.diag(np.ones(n)/np.diag(A))
    y=Dinv@b
    while r>e:
        x=Dinv@((E+F)@ x)+y
        r=np.linalg.norm(A@x-b)/normb
    return(x,r)
```

On teste cet algorithme pour la matrice à diagonale strictement dominante et le vecteur suivants :

$$A = \begin{pmatrix} 10 & 1 & 2 & 4 \\ -5 & 9 & 0 & 2 \\ 1 & -2 & 6 & 1 \\ -1 & -3 & 0 & 8 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

[5]:
```
A=np.array([[10 , 1 , 2 ,4],[ -5 , 9 ,0 ,2],[ 1 ,-2 ,6, 1],[ -1 , -3 , 0 ,8]] )
b=np.array([[1],[2],[3],[4]])
Jacobi(A,b,0.0001)
```

[5]:
```
(array([[-0.18480248],
        [ 0.01255567],
        [ 0.45471116],
        [ 0.48159242]]),
 7.353405474067329e-05)
```

2.2 Convergence de l'algorithme et Exercice

On rappelle qu'une condition suffisante pour que la méthode Jacobi converge est que la matrice A soit à diagonale strictement dominante, cf cours.

Exercice 1.

Cet exercice vise à déterminer lorsque la méthode de Jacobi converge ou non pour la matrice suivante :

$$A = A[a] = \begin{pmatrix} 1 & a & a \\ a & 1 & a \\ a & a & 1 \end{pmatrix}, \qquad a \in \mathbb{R}.$$

1. Montrer que les valeurs propres de A sont $1+2a$ et $1-a$. Donner un vecteur propre e_1 associé à $1+2a$, et deux vecteurs propres non colinéaires e_2 et e_3 associés à $1-a$. Vérifier vos calculs numériquement sur un exemple en choisissant une valeur de a en vous aidant de `numpy.linalg.eig`.

2. On a la décomposition :

$$A = D - E - F, \qquad D = Id, \qquad E = \begin{pmatrix} 0 & 0 & 0 \\ -a & 0 & 0 \\ -a & -a & 0 \end{pmatrix}, \qquad F = \begin{pmatrix} 0 & -a & -a \\ 0 & 0 & -a \\ 0 & 0 & 0 \end{pmatrix}.$$

   La suite des itérés de l'algorithme de Jacobi est donc donnée par la relation de récurrence

$$x^{(k+1)} = D^{-1}(E+F)x^{(k)} + D^{-1}b = (E+F)x^{(k)} + b.$$

   On calcule $x^{(1)} = (E+F)x^{(0)} + b$, puis $x^{(2)} = (E+F)x^{(1)} + b = (E+F)^2 x^{(0)} + (D-A)b + b$, et ainsi de suite, soit:

$$x^{(k)} = (E+F)^k x^{(0)} + \sum_{i=0}^{k-1} (E+F)^i b$$

On décompose $x^{(0)}$ et b dans la base $\{e_1, e_2, e_3\}$ trouvée à la question 1. : $x^{(0)} = \sum_1^3 x_i^{(0)} e_i$ et $b = \sum_1^3 b_i e_i$.

Montrer que si $a \notin \{-\frac{1}{2}, 1\}$ alors $x^{(k)}$ s'écrit $x^{(k)} = x_1^{(k)} e_1 + x_2^{(k)} e_2 + x_3^{(k)} e_3$ avec :

$$(3) \qquad x_1^{(k)} = (-2a)^k x_1^{(0)} + \frac{1-(-2a)^k}{1+2a} b_1, \quad x_2^{(k)} = a^k x_2^{(0)} + \frac{1-a^k}{1-a} b_2, \quad x_3^{(k)} = a^k x_3^{(0)} + \frac{1-a^k}{1-a} b_3.$$

3. Déduire de (3) que dès que $|a| > 1/2$ et que $x^{(0)} + \frac{b_1}{1-2a} \neq 0$ alors $\lim_{k\to\infty} |x^{(k)}| = \infty$.

Interprétation : Puisque $x^{(0)}$ est fixé quelconque dans la méthode de Jacobi, il est très probable que $x^{(0)} - \frac{b_1}{1-2a} \neq 0$. Dès que $|a| > 1/2$ il est donc très probable que la méthode de Jacobi ne converge pas. En revanche, lorsque $|a| < 1/2$, alors la matrice A est à diagonale strictement dominante, et, c.f. notes de cours, la méthode de Jacobi converge.

4. Choisissez $|a| > 1/2$, b et $x^{(0)}$ selon votre choix, et vérifier numériquement, en représentant graphiquement les premiers termes de la suite $(x^{(k)})$, que celle-ci diverge.

3 Méthode de Gauss-Seidel

3.1 L'algorithme

Soit A une matrice. On écrit A sous la forme $A = L - F$, où L est triangulaire inférieure, et F est triangulaire supérieure avec une diagonale nulle. Résoudre le système linéaire

$$Ax = b$$

est donc équivalent à résoudre

$$(4) \qquad Lx = Fx + b.$$

La méthode de Gauss-Seidel consiste donc : - À choisir un vecteur de départ $x_0 \in \mathbb{R}^n$, - Puis calculer la solution x_1 de $Lx_1 = x_0 + b$. Attention ! Puisque L est triangulaire inférieure, on trouve x_1 en appliquant la méthode du pivot, pas en inversant L ce qui serait trop couteux en temps. Puis les itérés successifs $x_2, x_3, \ldots$ par $Lx_k = Fx_{k-1} + b$. Si par exemple A est symmétrique définie positive, ou si elle est à diagonale dominante, la suite $(x_k)_{k\in\mathbb{N}}$ converge vers l'unique solution x de (4). Nous vous renvoyons à vos notes de cours pour plus de détails sur la convergence.

La fonction `solgauss(L,b)` ci-dessous donne, étant donnée une matrice L triangulaire inférieure de diagonale non nulle, la solution de $Lx = b$ calculée par l'algorithme du pivot de Gauss. Nous renvoyons à la feuille de TP concernant la méthode du pivot pour plus de détails.

[6]:
```
def solgauss(L,b):
    n=len(b)
    x=np.zeros(n)
    for i in range(n):
        x[i]=(b[i]-np.sum(x[0:i]*L[i,0:i]))/L[i,i]
    return(x)
```

Étant donné un seuil d'erreur $\epsilon > 0$, on souhaite arrêter les itérations lorsqu'on a l'estimation sur le résidu $\frac{|Ax_k - b|}{|b|} \leq e$. La fonction `gaussseidel(A,b,e)` code l'algorithme de Gauss-Seidel pour trouver une solution approchée x_{ap} de $Ax = b$ avec une telle estimation sur le résidu.

[7]:
```
def gaussseidel(A,b,e):
    n=len(b)
    normb=np.linalg.norm(b)
    (D,E,F)=DEF(A)
    L=D-E
    r=2*e
    x=np.zeros(n)
    while r>e:
        x=solgauss(L,F@x+b)
        r=np.linalg.norm(A@x-b)/normb
    return(x,r)
```

On teste cet algorithme pour la matrice symétrique et définie positive suivante, et le vecteur suivant :

$$A = \begin{pmatrix} 10 & 1 & 2 & 3 \\ 1 & 9 & 4 & 2 \\ 2 & 4 & 11 & 0 \\ 3 & 2 & 0 & 8 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

[8]:
```
A=np.array([[10 , 1 , 2 ,3],[ 1 , 9 ,4 ,2],[ 2 ,4 ,11, 0],[ 3 , 2 , 0 ,8]] )
b=np.array([1,1,1,1])
b
```

[8]:
```
array([1, 1, 1, 1])
```

[9]:
```
u=gaussseidel(A,b,0.0000001)
u[0]
```

[9]:
```
array([0.05535477, 0.05858856, 0.05953966, 0.08959482])
```

[10]:
```
A@np.reshape(u[0],(4,1))
```

[10]:
```
array([[1.00000003],
       [1.00000007],
       [1.        ],
       [1.        ]])
```

3.2 Exercice

Exercice 2

On suppose que l'on a une borne $\|L^{-1}F\| \leq \kappa$ avec $0 < \kappa < 1$, et que $x_0 = 0$ par simplicité. On fixe $\epsilon > 0$. Si $L \in \mathbb{R}^{n\times n}$, combien d'opérations sont effectuées par la méthode de Gauss-Seidel ? Formuler votre réponse sous la forme $O(n^\alpha)$ où α est à déterminer. Comparez avec le nombre d'opérations pour résoudre $Ax - b$ avec la méthode du pivot (voir feuille de TP précédente). Quelle méthode vous semble la plus efficace ?

4 Méthode de la puissance itérée

4.1 L'algorithme

Le but est de déterminer le vecteur propre d'une matrice associé à la valeur propre de plus grand module, en supposant que celle-ci est unique. Étant donné une matrice réelle A de taille $n \times n$ et un vecteur $x_0 \in \mathbb{R}^n$, la suite de vecteurs $(x_k)_{k\in\mathbb{N}}$ est définie par :

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|}\,.$$

En supposant par exemple que la matrice A soit diagonalisable, alors il est possible de montrer que pour un x_0 générique, alors $(x_k)_{k\in\mathbb{N}}$ converge à un signe près vers un vecteur propre unitaire associé à la plus grande valeur propre λ_1 de A en valeur absolue. C'est-à-dire, que l'on a soit que x_k converge si signe$(\lambda_1) = +1$ ou que $(-1)^k x_k$ converge si signe$(\lambda_1) = -1$, et en notant v le vecteur limite vers les deux cas : $Av = \lambda_1 v$.

En effet :

$$x_k = \frac{Ax_{k-1}}{\|Ax_{k-1}\|} = \frac{A^2x_{k-2}}{\|A^2x_{k-2}\|} = \dots = \frac{A^kx_0}{\|A^kx_0\|}\,.$$

Vu que A est diagonalisable, soit $(v_1, v_2, \dots, v_n)$ une base de vecteurs propres unitaires de A associés aux valeurs propres $\lambda_1, \lambda_2, \dots, \lambda_n$. Sans perte de généralité, on suppose que λ_1 est la valeur propre de plus grand module, c'est-à-dire $|\lambda_1| > \max(|\lambda_2|, |\lambda_3|, \dots, |\lambda_n|)$. À noter que cela implique que λ_1 est réelle. Le vecteur x_0 se décompose dans la base:

$$x_0 = \sum_{i=1}^{n} c_i v_i\,,$$

ainsi en supposant que $c_1 \neq 0$:

$$A^kx_0 = \sum_{i=1}^{n} c_i\lambda_i^k v_i = \lambda_1^k \left(c_1v_1 + \sum_{i=2}^{n} c_i \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i \right).$$

Par conséquent puisque v_1 est unitaire et que $(\lambda_i/\lambda_1)^k \to 0$ lorsque $k \to \infty$ pour tout $i \geq 2$:

$$\|A^kx_0\| \sim |\lambda_1|^k|c_1| \qquad \text{lorsque } k \to \infty.$$

On obtient donc:

$$x_k = \frac{\lambda_1^k}{|\lambda_1^k|} \underbrace{\frac{|\lambda_1^k||c_1|\|v_1\|}{\|A^kx_0\|} \left(\frac{c_1}{|c_1|}v_1 + \sum_{i=2}^{n} \frac{c_i}{|c_1|} \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i \right)}_{\longrightarrow \text{signe}(c_1)v_1 \quad \text{lorsque } k\to\infty}$$

et, ci-dessus, ou bien $\frac{\lambda_1^k}{|\lambda_1^k|} = 1$ si signe$(\lambda_1) = +1$ ou bien $\frac{\lambda_1^k}{|\lambda_1^k|} = (-1)^k$ si signe$(\lambda_1) = -1$

En choisissant x_0 aléatoirement, alors avec presque sûrement $c_1 \neq 0$ et donc la suite $(x_k)_{k\in\mathbb{N}}$ converge à un signe près vers un vecteur propre normalisé associé à la valeur propre de plus grand module.

De la discussion ci-dessus on déduit que $\frac{A^{2^k}x_0}{\|A^{2^k}x_0\|}$ converge vers un vecteur propre unitaire associé à la plus grande valeur propre de A. On définit l'erreur dans la convergence vers la limite pour $k \geq 1$ par:

$$e_k = \left\| \frac{A^{2^k}x_0}{\|A^{2^k}x_0\|} - \frac{A^{2^{k-2}}x_0}{\|A^{2^{k-2}}x_0\|} \right\|.$$

4.2 Exercice

Exercice 3.

1. Ecrire un programme `pgvectprop(A,e)` qui prend en entrée une matrice A et un seuil $\epsilon > 0$, et calcule un vecteur propre unitaire associé à la valeur propre la plus grande de A par la méthode que l'on vient de présenter et s'arrête lorsque l'erreur est $e_k < \epsilon$.
2. Calculer numériquement un vecteur propre unitaire associé à la plus grande valeur propre de la matrice :
$$A = \begin{pmatrix} 0 & -1 \\ 2 & 3 \end{pmatrix}.$$
3. Calculer la valeur propre correspondante
4. Ecrire une fonction qui calcule la plus grande valeur propre d'une matrice avec une certaine précision. Comparez son temps d'exécution avec la routine `numpy.linalg.eig`
5. Si on prend la matrice
$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
qui n'est pas diagonalisable. Est-ce que x_k admet une limite lorsque $k \to \infty$? En vous rappelant la décomposition de Jordan de toute matrice, en déduire que dès lors qu'une matrice A admet une unique valeur propre de plus grand module, alors l'algorithme de la puissance itérée converge (pour un vecteur d'initialisation x_0 générique).

[ ]: