AIX MARSEILLE UNIVERSITE

Licence 3 Mathématiques

Analyse numérique et Optimisation

Analyse Numérique - Projet

GEYER Rayane
KEBIR Younes
MECHRAOUI Ahmed

February 26, 2024

Contents

1 Introduction

Dans ce projet, on résout un modèle de diffusion thermique avec rayonnement (dans un matériau comme le verre, par exemple) avec une discrétisation par différences finies et une méthode de monotonie. Pour simplifier, on se limite à considérer le problème avec une seule dimension spatiale (un problème plus réel demande à prendre la variable spatiale dans $\mathbb{R}^3$).

Le modèle consiste à chercher la fonction u de $[0,1]$ à valeurs dans $\mathbb{R}$ solution du problème suivant :

$$-\frac{1}{10}u(x)^{''} + \int_0^1 \frac{1}{|x-y|}(u^4(x) - u^4(y))dy = 0 \quad x \in]0,1[$$

$$u(0) = 1 \quad u(1) = 2$$

On admet que cette solution existe (en particulier, on admet qu'elle est bien continue sur $[0,1]$ et deux fois dérivable sur $]0,1[$. On va calculer (de manière approchée) cette solution en utilisant une discrétisation par différences finies. Pour $n \geq 1$, on pose $h = \frac{1}{n+1}$. La discrétisation par différences finies de $(1)-(2)$ consiste à chercher le vecteur u de $\mathbb{R}^n$ solution de :

$$Au + R(u) = b$$

On défini $A \in M_n(\mathbb{R})$ telle que :

$A[i,i] = \frac{2}{10h^2}$; $A[i,j] = -\frac{1}{10h^2}$ si $|i-j| = 1$; $A[i,j] = 0$ si $|i-j| > 1$ avec $i,j \in \{1,...,n\}$.

On défini $R(u) \in \mathbb{R}^n$ tel que :

$$R(u)_i = \sum_{j \in \{1,..,n\}, j \neq i} \frac{\sqrt{h}}{\sqrt{|i-j|}}(u_i^4 - u_j^4) + \frac{\sqrt{h}}{2\sqrt{i}}(u_i^4 - 1) + \frac{\sqrt{h}}{2\sqrt{n+1-i}}(u_i^4 - 16)$$

On défini enfin $b \in \mathbb{R}^n$ tel que :

$$b_1 = \frac{1}{10h^2}; \quad b_n = \frac{2}{10h^2}; \quad b_i = 0 \quad 1 < i < n$$

2 Discrétisation

Pour trouver une solution au système on se donne $\beta \geq 0$ et on utilise la méthode itérative suivante :

Initialisation : $u^{(0)} \in \mathbb{R}^n, u_i^{(0)} = 1$ pour tout $i \in \{1,...,n\}$.
Itérations : Pour $k \geq 0$, $Au^{(k+1)} + \beta u^{(k+1)} = -R(u^{(k)}) + \beta u^{(k)} + b$.

La méthode utilisée consiste en une discrétisation de la première expression du modèle. En effet, en effectuant un développement limité au second ordre de notre première équation on obtient

$$DL_2\big[-\frac{1}{10}u(x)^{''} + \int_0^1 \frac{1}{|x-y|}(u^4(x) - u^4(y)dy\big] = \frac{1}{10h^2}(2u_i - u_{i-1} - u_{i+1}) + R(u)_i$$

Avec $R(u)_i$ désignant le reste du développement limité $i \in \mathbb{N}^*$ et u un vecteur de $\mathbb{R}^n$. On obtient ainsi $Au + R(u) = b$.

Dans un premier temps nous allons programmer la méthode itérative indiquée avec un test d'arrêt des itérations utilisant que la norme infinie de $(u(k+1)-u(k))$ est inférieure à une valeur donnée ϵ et un nombre maximal d'itérations.

La fonction R_i permet de calculer la composante i-ème du vecteur R tandis que la fonction R utilise cette dernière afin de déterminer chacune de ses composantes.

```
def R_i(u,i,n):
    s=0
    for j in range(n):
        if i!=j:
            s=s+(np.sqrt(h)/np.sqrt(np.abs(i-j)))*(u[i]**4-u[j]**4)
        else:
            s=s
    R_i=s+(np.sqrt(h)/2*np.sqrt(i))*(u[i]**4-1)+(np.sqrt(h)/2*np.sqrt(n+1-i))*(u[i]**2-16)
    return R_i

def R(u,n):
    R=np.zeros(n)
    for i in range(n):
        R[i]=R_i(u,i,n)
    return R

def Methode_Iterative(n,u0,A,b,beta,epsilon):
    compt=0
    I=np.identity(n)
    un=np.dot(alg.inv(A+beta*I),-R(u0,n)+beta*u0+b)
    while alg.norm(u0-un)> epsilon and compt<1000:
        u0=un
        un=np.dot(alg.inv(A+beta*I),-R(u0,n)+beta*u0+b)
        compt=compt+1
    return [un, compt]
```

Pour $\epsilon = 10^{-7}$ et plusieurs valeurs de n, on s'intéresse au comportement de l'algorithme en fonction des valeurs de β.

On constate tout d'abord que l'algorithme ne converge pas pour $\beta = 0$. En effet le code ci-dessous permet de le prouver.

```
L=[5,10,100]
for n in L:
    h=1/(n+1)

    I=np.identity(n)

    A=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if np.abs(i-j)==0:
                A[i,i]=2/(10*h*h)
            if np.abs(i-j)==1:
                A[i,j]=-1/(10*h*h)
            if np.abs(i-j)>1:
                A[i,j]=0
    b=np.zeros(n)
    b[0]=1/(10*h*h)
    b[n-1]=2/(10*h*h)

    print(Methode_Iterative(n, np.ones(n), A, b, 0, 0.00000001))
```

L'interpréteur nous renvoie le résultat suivant pour n=5 et n=10. Il en est de même pour $n = 200, 500$. On ne note pas leur résultat au vu de leur grandeur.

```
[array([nan, nan, nan, nan, nan]), 5]
[array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]), 5]
[array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
       nan, nan, nan, nan, nan, nan, nan, nan, nan]), 5]
```

En effectuant l'algorithme pour différentes valeurs de β, on constate qu'il existe $\beta_n > 0$ pour lequel, si $\beta \geq \beta_n$, la suite (u_k) converge et que cette convergence est monotone.

Déterminons ce β_n dans le cas $n = 10$. On utilise le même code que précédemment avec quelques modifications.

```
L=[10]
for n in L:
    h=1/(n+1)

    I=np.identity(n)

    A=np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if np.abs(i-j)==0:
                A[i,i]=2/(10*h*h)
            if np.abs(i-j)==1:
                A[i,j]=-1/(10*h*h)
            if np.abs(i-j)>1:
                A[i,j]=0
    b=np.zeros(n)
    b[0]=1/(10*h*h)
    b[n-1]=2/(10*h*h)

    for beta in range(15,20):
        print(beta)
        print(Methode_Iterative(n, np.ones(n), A, b, beta, 0.00000001))
```

Avec les sorties suivantes, on trouve que $\beta_n = 17$ pour $n = 10$.

```
15
[array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]), 18]
16
[array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan]), 36]
17
[array([1.33468567, 1.19345095, 1.42958085, 1.45888502, 1.67972955,
       1.89959836, 1.90920062, 1.74605332, 1.71785401, 1.70817919]), 1000]
18
[array([1.73851765, 1.57532094, 1.42284444, 1.66012129, 2.15857785,
       2.26232038, 2.21289288, 2.0442699 , 1.70597918, 1.62993956]), 1000]
19
[array([1.73587493, 1.26953463, 1.01633318, 1.44604537, 2.28324785,
       2.35039004, 2.3209229 , 2.21946868, 1.68972473, 1.49376367]), 1000]
```

Nous allons maintenant déterminer si dans le cas où la suite (u_k) converge, la solution approchée obtenue dépend de β. Pour ce faire on se place dans le cadre où $n = 10$ puis on détermine la solution au problème pour des valeurs succinctes de β comme suit.

```
n=10
h=1/(n+1)

I=np.identity(n)
A1=np.zeros((n,n))
for i in range(n):
    for j in range(n):
        if np.abs(i-j)==0:
            A1[i,i]=2/(10*h*h)
        if np.abs(i-j)==1:
            A1[i,j]=-1/(10*h*h)
        if np.abs(i-j)>1:
            A1[i,j]=0

b1=np.zeros(n)
b1[0]=1/(10*h*h)
b1[n-1]=2/(10*h*h)

for beta in range (72, 76):
    print(beta)
    print(Methode_Iterative(n, np.ones(n), A1, b1, beta, 0.00000001))
```

On obtient les résultats suivants.

```
72
[array([1.86651335, 1.97270888, 1.97291344, 1.95956255, 1.94475379,
       1.92990244, 1.91501492, 1.90010389, 1.88674561, 1.88813226]), 106]
73
[array([1.86651335, 1.97270887, 1.97291344, 1.95956254, 1.94475379,
       1.92990244, 1.91501492, 1.90010389, 1.88674561, 1.88813225]), 107]
74
[array([1.86651335, 1.97270887, 1.97291344, 1.95956255, 1.94475379,
       1.92990244, 1.91501492, 1.90010389, 1.88674561, 1.88813226]), 109]
75
[array([1.86651335, 1.97270887, 1.97291344, 1.95956254, 1.94475378,
       1.92990244, 1.91501492, 1.90010389, 1.88674561, 1.88813225]), 110]
```

On constate que le choix de β n'a pas d'influence sur le résultat approché obtenu.

On constate de plus que le nombre d'itérations nécessaires pour obtenir la solution approchée augmente lorsque β augmente. Par exemple, pour $\beta = 72$ il faut effectuer 106 itérations, $\beta = 73$ il faut en effectuer 107..

3 Nouvelle discrétisation

En effectuant une discrétisation différente avec :

$$R(u)_i = \sum_{j\in\{1,..,n\} j\neq i} \frac{2\sqrt{h}}{\sqrt{i-j}+0.5+\sqrt{|i-1|-0.5}}(u_i^4 - u_j^4) + \frac{\sqrt{h}}{\sqrt{i}+\sqrt{i-0.5}}(u_i^4 - 1)$$

$$+\frac{\sqrt{h}}{\sqrt{n+1-i}+\sqrt{n+0.5-i}}(u_i^4 - 16)$$

Avec cette discrétisation et des tests similaires à précédemment on constate que les valeurs de β_n ne changent pas par rapport à l'autre discrétisation. La variation du nombre d'itérations en fonction de β ne varie pas non plus.

```
def R_i(u,i,n):
    s=0
    for j in range(n):
        if i!=j:
            s= s + 2*np.sqrt(h)*(u[i]**4 - u[j]**4)/(np.sqrt(np.abs(i-j)+ 1/2) +
            np.sqrt(np.abs(i-j)- 1/2))
        else:
            None
    R_i= s + np.sqrt(h)*(u[i]**4 - 1)/( np.sqrt(i) + np.sqrt(i-0.5)) +
    np.sqrt(h)*(u[i]**4 - 16)/( np.sqrt(n+1-i) + np.sqrt(n+1-i-0.5))

    return R_i

def R(u,n):
    R=np.zeros(n)
    for i in range(n):
        R[i]=R_i(u,i,n)
    return R
```