Duchanoy Colin
Lancery Hugo
Quin Hugo
Soucourre Vincent

IPSA Aéro 3

BE MA323

Méthode des différences finies

M.Couffignal & M.El Mahbouby
Fevrier 2022

Table des matières

I. Partie 1 : l'équation de la chaleur sur une barre

Le but de cet exercice est de modéliser le phénomène de conduction thermique dans une barre à partir d'une discrétisation de l'équation de la chaleur. Soient L et T deux réels strictement positifs. On fixe f $\in C^{\infty}$ ([0, L] × [0, T], ℝ), $u_0 \in C^{\infty}$([0, L], ℝ) et α, β $\in C^{\infty}$ ([0, T], ℝ) et γ un réel fixé strictement positif.

On considère le problème aux limites suivant : trouver une fonction $u = u(x,t)$représentant la température au point (x,t) avec $x \in [0, L]$ et $t \in [0, T]$, telle que :

$$\begin{cases} \frac{du}{dt} - \gamma \frac{d^2u}{dx^2} = f \quad dans\ [0,L] \times [0,T] \\ u(0,t) = \alpha(t) \quad \forall t \in [0,T] \\ u(L,t) = \beta(t) \quad \forall t \in [0,T] \\ u(x,0) = u_0(x) \ \forall x \in [0,L] \end{cases}$$

On fixe M et N deux entiers et on discrétise [0, L] × [0, T] en introduisant les points $x_j = j\ \Delta x$ pour j ∈ [[0, M + 1]] et les instants $t^n = n\ \Delta t$ pour $n \in [[0, N]]$. On cherche alors ${u_j}^n \approx u(x_j, t^n)$ pour tout $(j, n) \in [[1, M]] \times [[1, N]]$.

Nous allons appliquer le schéma numérique suivant

$$\frac{1}{\Delta t}\left(\frac{3}{2}{u_j}^{n+1} - 2{u_j}^n + \frac{1}{2}{u_j}^{n-1}\right) - \frac{\gamma}{(\Delta x)^2}\left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right) = f_j^{n+1}$$

1. Etude du schéma numérique

a. Type de l'EDP

On étudie l'EDP suivante : $\frac{du}{dt} - \gamma \frac{d^2u}{dx^2} = f$

La matrice associée à cette EDP est la matrice A=$\begin{pmatrix} \gamma & 0 \\ 0 & 0 \end{pmatrix}$

On obtient *det*(A)=0

Donc cette EDP est de type parabolique.

De plus, cette EDP est non homogène linéaire d'ordre 2.

b. Ordre du schéma

Le premier membre, celui de gauche est le terme selon le temps :

$$\frac{1}{\Delta t}\left(\frac{3}{2}U_j^{n+1} - 2U_j^n + \frac{1}{2}U_{ij}^{n-1}\right)$$

Le second terme de gauche est lui selon l'espace :

$$-\frac{\gamma}{(\Delta x)^2}\left(U_{j+A}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}\right)$$

Ce schéma est donc d'ordre 2 en temps et d'ordre 2 en espace.

c. Etude de la stabilité

Nous allons chercher à démontrer que le schéma est inconditionnellement stable au sens de Von Neumann-Fourier, c'est-à-dire $|\xi| < 1$

Tout d'abord on pose $u_j^n = \xi^n e^{ik\pi\ j}$ pour k fixé et $\lambda = \gamma\frac{\Delta t}{(\Delta x)^2}$

En remplaçant dans le schéma numérique on a donc :

$$\left(\frac{3}{2}\xi^{n+1}e^{ik\pi x_j} - 2\xi^n e^{ik\pi x_j} + \frac{1}{2}\xi^{n-1}e^{ik\pi x_j}\right) - \lambda\left(\xi^{n+1}e^{ik\pi x_{j+1}} - 2\xi^{n+1}e^{ik\pi x_j} + \xi^{n+1}e^{ik\pi x_{j-1}}\right) = 0$$

$$\Leftrightarrow e^{ik\pi x_j}\left(\frac{3}{2}\xi^{n+1} - 2\xi^n + \frac{1}{2}\xi^{n-1}\right) - \lambda\xi^{n+1}\left(e^{ik\pi x_{j+1}} - 2e^{ik\pi x_j} + e^{ik\pi x_{j-1}}\right) = 0$$

$$\Leftrightarrow e^{ik\pi x_j}\left(\frac{3}{2}\xi^{n+1} - 2\xi^n + \frac{1}{2}\xi^{n-1}\right) - \lambda\xi^{n+1}e^{ik\pi x_j}\left(e^{ik\pi\Delta x} - 2e^{ik\pi x_j} + e^{-ik}\right) = 0$$

$$\Leftrightarrow \frac{3}{2}\xi^2 - 2\xi + \frac{1}{2} - \lambda\xi^2(2\cos(k\pi\Delta x) - 2) = 0$$

$$\Leftrightarrow \frac{3}{2}\xi^2 - 2\xi + \frac{1}{2} - 2\lambda\xi^2(\cos(k\pi\Delta x) - 1) = 0$$

$$\Leftrightarrow \left(\frac{3}{2} - 2\lambda(\cos(k\pi\Delta x) - 1)\right)\xi^2 - 2\xi + \frac{1}{2} = 0 \quad (*)$$

Pour respecter la condition il faut que le det(*)≤0

On a donc :

$$2^2 - 4\left(\frac{3}{2} - 2\lambda(\cos(k\pi\Delta x) - 1)\right)\frac{1}{2} \leq 0$$

$$\Leftrightarrow 4 - 2\left(\frac{3}{2} - 2\lambda(\cos(k\pi\Delta x) - 1)\right) \leq 0$$

$$\Leftrightarrow 1 + 4\lambda(\cos(k\pi\Delta x) - 1) \leq 0$$

$$\Leftrightarrow 4\lambda(\cos(k\pi\Delta x) - 1) \leq -1$$

$$\Leftrightarrow \lambda \geq \frac{-1}{4(\cos(k\pi\Delta x) - 1)}$$

On peut minorer $(\cos(k\pi\Delta x) - 1)$ par -2.

Ce qui nous donne :

$$\Leftrightarrow \lambda \geq \frac{1}{8}$$

λ est donc nécessairement positif

d. Facteur d'amplification

Nous avons l'équation suivante que nous avons déterminé à la question d'avant :

$$\Leftrightarrow (\frac{3}{2} - 2\lambda(\cos(k\pi\Delta x) - 1))\xi^2 - 2\xi + \frac{1}{2} = 0$$

$$\Leftrightarrow \frac{3}{2}\xi^2 - 2\xi + \frac{1}{2} - 2\lambda\xi^2(\cos(k\pi\Delta x) - 1) = 0$$

$$\Leftrightarrow 2\lambda\xi^2(\cos(k\pi\Delta x) - 1) = \frac{3}{2}\xi^2 - 2\xi + \frac{1}{2}$$

$$\Leftrightarrow \lambda = \frac{\frac{3}{2}\xi^2 - 2\xi + \frac{1}{2}}{2\xi^2(\cos(k\pi\Delta x) - 1)} = \frac{\frac{3}{2} - \frac{2}{\xi} + \frac{1}{2\xi^2}}{2(\cos(k\pi\Delta x) - 1)}$$

$$\Leftrightarrow \lambda = \frac{\frac{3\xi^2 - 4\xi + 1}{2\xi^2}}{2(\cos(k\pi\Delta x) - 1)} = \frac{3\xi^2 - 4\xi + 1}{4\xi^2(\cos(k\pi\Delta x) - 1)}$$

Si $\xi \to 0$ alors $lim\ \lambda = \frac{1}{0} = +\infty$

e. Convergence du schéma

Le schéma est **consistant** et **stable** dans le cas où ξ tend vers 0 la solution approximée devient égale à la solution exacte et n'oscille plus autour de cette dernière.

D'après le théorème de Lax le schéma est convergent s'il est consistant et stable.

Notre objectif est donc d'avoir un λ grand afin de trouver une solution la plus proche possible de la solution exacte. Notre schéma augmente donc en stabilité quand ξ tend vers 0.

2. Schéma numérique sous la forme d'une suite vectorielle

On pose $u^n = (u_1^n, u_2^n, \dots, u_M^n)^T$

Nous avons le schéma numérique suivant :

$$\Leftrightarrow \left(\frac{3}{2}{u_j}^{n+1} - 2{u_j}^n + \frac{1}{2}{u_j}^{n-1}\right) - \lambda\left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right) = \Delta t f_j^{n+1}$$

$$\Leftrightarrow \left(\frac{3}{2} + 2\lambda\right)u_j^{n+1} - \lambda u_{j+1}^{n+1} - \lambda u_{j-1}^{n+1} - 2{u_j}^n + \frac{1}{2}{u_j}^{n-1} = \Delta t f_j^{n+1}$$

$$\Leftrightarrow \begin{pmatrix} \frac{3}{2}+2\lambda & -\lambda & 0 & \dots & 0 \\ -\lambda & \frac{3}{2}+2\lambda & -\lambda & & \vdots \\ 0 & -\lambda & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\lambda \\ 0 & \dots & 0 & -\lambda & \frac{3}{2}+2\lambda \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ \vdots \\ u_M^{n+1} \end{pmatrix} - \lambda \begin{pmatrix} u_0^{n+1} \\ 0 \\ \vdots \\ 0 \\ u_{M+1}^{n+1} \end{pmatrix} - 2{u_j}^n + \tfrac{1}{2}{u_j}^{n-1} = \Delta t f_j^{n+1} \quad (1)$$

On posera B=$\begin{pmatrix} \frac{3}{2}+2\lambda & -\lambda & 0 & \dots & 0 \\ -\lambda & \frac{3}{2}+2\lambda & -\lambda & & \vdots \\ 0 & -\lambda & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\lambda \\ 0 & \dots & 0 & -\lambda & \frac{3}{2}+2\lambda \end{pmatrix}$

On cherche u_j^1

On effectue son développement de Taylor

$$u_j^1 = u(x_j, \Delta t) = u(x_j, 0) + \Delta t \frac{\partial u}{\partial t}(x_j, 0)$$

$$\Leftrightarrow u_j^1 = u(x_j, 0) + \left(f^1 + \gamma \frac{\partial^2 u}{\partial x^2}(x_j, 0)\right)$$

$$\Leftrightarrow u_j^1 = u(x_j, 0) + \left(f^1 + \gamma\left(\frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{(\Delta x)^2}\right)\right)$$

On a donc $u_j^1 = u_0(x) + \Delta t(f^1 + \frac{\gamma}{(\Delta x)^2}\left(A * u_0(x) + \begin{pmatrix} \alpha(0) \\ 0 \\ \vdots \\ 0 \\ \beta(0) \end{pmatrix}\right)$

Avec A la matrice Laplacienne, tel que A=$\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & & \ddots & & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$

On a :

$$(1) \Leftrightarrow BU^{n+1} = \Delta t * f^{n+1} + 2I_M U^n - \frac{1}{2} I_M U^{n-1} + \lambda U^0$$

On pose :

$$F^{n+1} = \Delta t * f^{n+1} \ , \ C = 2I_M U^n \ , \ D = -\frac{1}{2} I_M U^{n-1} \ ,$$

On a alors $\begin{cases} u^{n+1} = B^{-1}F^{n+1} + B^{-1}CU^n - B^{-1}DU^{n-1} + \lambda U^0 \\ u_0, u_1 \epsilon \mathbb{R}^M \end{cases}$

3. Schéma numérique sous la forme matricielle

On passe maintenant le schéma numérique sous forme matricielle

On pose $u = \begin{pmatrix} u^1 \\ \vdots \\ \vdots \\ \vdots \\ u^N \end{pmatrix}$

On obtient donc le système suivant

$$\begin{pmatrix} -B^{-1}C & I_M & 0 & \dots & 0 \\ -C & -B^{-1}C & I_M & & \vdots \\ 0 & -C & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & I_M \\ 0 & \dots & 0 & -C & -B^{-1}C \end{pmatrix} \begin{pmatrix} u^1 \\ \vdots \\ \vdots \\ \vdots \\ u^N \end{pmatrix} + \lambda \begin{pmatrix} B^{-1}C * u_0^0 \\ 0 \\ \vdots \\ \vdots \\ I_M * u_M^{N+1} \end{pmatrix} = \begin{pmatrix} B^{-1}F^1 \\ 0 \\ \vdots \\ \vdots \\ B^{-1}F^{n+1} \end{pmatrix}$$

$$\Leftrightarrow Au = b$$

Avec A= $\begin{pmatrix} -B^{-1}C & I_M & 0 & \dots & 0 \\ -C & -B^{-1}C & I_M & & \vdots \\ 0 & -C & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & I_M \\ 0 & \dots & 0 & -C & -B^{-1}C \end{pmatrix}$ de taille (M×N, M×N)

$u = \begin{pmatrix} u^1 \\ \vdots \\ \vdots \\ \vdots \\ u^N \end{pmatrix}$ de taille (M×N,1) et $b = \begin{pmatrix} B^{-1}F^1 \\ 0 \\ \vdots \\ \vdots \\ B^{-1}F^{n+1} \end{pmatrix} - \begin{pmatrix} B^{-1}C * u_0^0 \\ 0 \\ \vdots \\ \vdots \\ I_M * u_M^{N+1} \end{pmatrix}$ de taille (M×N,1)

4. Comparaison des approches

En utilisant les mêmes conditions initiales que l'approche vectorielle pour l'approche matricielle,

5. Programmation sous python

On modélise maintenant à l'aide de Python le schéma numérique sous forme vectorielle

On entre d'abord les trois différents systèmes de conditions sur γ, α, β, L et $u_0(x)$

```python
#%%
#Partie 1
#%% Conditions initiales 1
M=20
N=20
L=10
T=100
gamma=4
def alpha(t):
    return 0
def beta(t):
    return 0
def f(x,t) :
    return np.zeros((len(x),1))
def u0(x,L):
    uo=np.zeros((len(x),1))
    for i in range (len(x)):
        uo[i,0]=x[i]*(L-x[i])
    return uo
```

```python
#%% Conditions initiales 2
M=50
N=20
L=10
T=100
gamma=3
def alpha(t):
    return 10*(1-np.cos(15*np.pi*t/T))
def beta(t):
    return 10*(1-np.cos(15*np.pi*t/T))
def f(x,t) :
    return np.zeros((len(x),1))
def u0(x,L):
    return np.zeros((len(x),1))
```

```python
#%% Conditions initiales 3 "Le premier envol"
M=80
N=100
L=20
T=50
gamma=100
def alpha(t):
    return 10*(1-np.cos(15*np.pi*t/T))
def beta(t):
    return 10*(1-np.cos(15*np.pi*t/T))
def f(x,t) :
    fo=np.zeros((len(x),1))
    for i in range(len(x)):
        fo[i,0] = np.sin(x[i])**2
    return fo
def u0(x,L):
    uo=np.zeros((len(x),1))
    for i in range (len(x)):
        uo[i,0]=x[i]*(L-x[i])
    return uo
```

N'étant pas parvenu à trouver un schéma numérique cohérent, il était compliqué de trouver des nouvelles conditions initiales intéressantes.

On crée ensuite une fonction *resolvect* qui dépend des paramètres de discrétisation. C'est cette fonction qui permet de trouver numériquement u et de tracer la solution en 2D sous forme d'une animation

```python
#%%
#Résolution numérique vectorielle

def resolvect(M,N):
    """

    Parameters
    ----------
    M,N : Paramètres de discrétisation.

    Returns
    -------
    Affichage 2D de la solution numérique par une approche en suite vectorielle

    """
```

On définit d'abord les paramètres de discrétisation ainsi que les conditions aux limites.

```
# Initialisation des constantes dépendant des paramètres d'échantillonnage
x=np.linspace(0,L,M)
t=np.linspace(0,T,N)
dx=L/(M-1)
dt=T/(N-1)
lbd=gamma*(dt/(dx**2))

# Initialisation des conditions limites : cl est un vecteur colonne de taille M
cl = np.zeros((M,1))
cl[0,0]=alpha(0)
cl[-1,0]=beta(0)
cl=lbd*cl
```

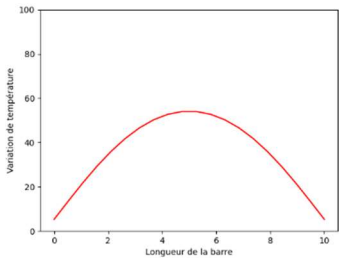
Puis on rentre le schéma numérique sous forme matricielle, tel que nous l'avons défini dans la partie théorique. Enfin on affiche la solution en 2D sous forme d'une animation.

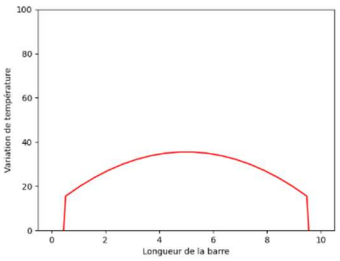
L'affichage se fait en même temp que la création de la matrice solution *u*

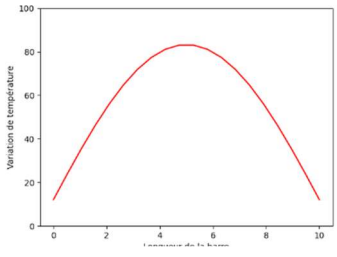
```
# Création de la matrice Laplacienne
Lap=np.eye(M)-np.diag(np.ones(M-1),1)
Lap=Lap.T+Lap

# U0 et U1 sont les conditions initiales de notre suite vectorielle, elles représentent respectivement
# les deux premières colonnes de notre matrice U.
U0 = u0(x,L)
U1 = U0+dt*(f(x,0)+(gamma/(dx**2))*(Lap @ U0 + cl))

U=np.zeros((M,N+2))
U[:,0]=U0.reshape(M)
U[:,1]=U1.reshape(M)

# Ici nous définissons les matrices B, C, D, et F qui nous permettent d'arriver à la forme de la suite matricielle
# obtenue dans la partie théorique.
B=(((3/4)+lbd))*np.eye(M)-lbd*np.diag(np.ones(M-1),1)
B=B.T+B
Binv=np.linalg.inv(B)
C=2*np.eye(M)
D=(-1/2)*np.eye(M)
F=f(x,t)*dt
```

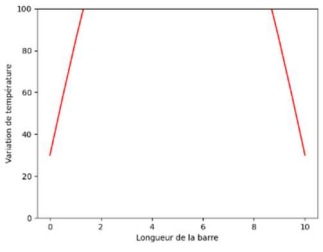
```
# Ici nous définissons les matrices B, C, D, et F qui nous permettent d'arriver à la forme de la suite matricielle
# obtenue dans la partie théorique.
B=(((3/4)+lbd))*np.eye(M)-lbd*np.diag(np.ones(M-1),1)
B=B.T+B
Binv=np.linalg.inv(B)
C=2*np.eye(M)
D=(-1/2)*np.eye(M)
F=f(x,t)*dt
# à chaque itération de cette boucle, la colonne suivante de la matrice U est calculée à partir des deux précédentes.
# On plot cette par la même occasion ce qui nous permet d'observer une évolution de notre matrie solution U.
for n in range(N):
    plt.clf()
    plt.xlabel("Longueur de la barre")
    plt.ylabel("Variation de température")
    plt.ylim(0,100)
    cl[0,0]=alpha(n)
    cl[-1,0]=beta(n)
    Unplus2 = Binv @ F + (Binv @ C) @ U[:,n+1].reshape((M,1)) - (Binv @ D) @U[:,n].reshape((M,1))+ dt*cl
    U[:,n+2] =  Unplus2.reshape(M)
    plt.plot(x,U[:,n],'r')
    plt.pause(2)
```

Le résultat obtenu est incohérent aux limites pour u^1, les valeurs prises sont trop élevées. En observant la matrice solution *u* on peut remarquer que cette incohérence n'est présente que pour u^1.







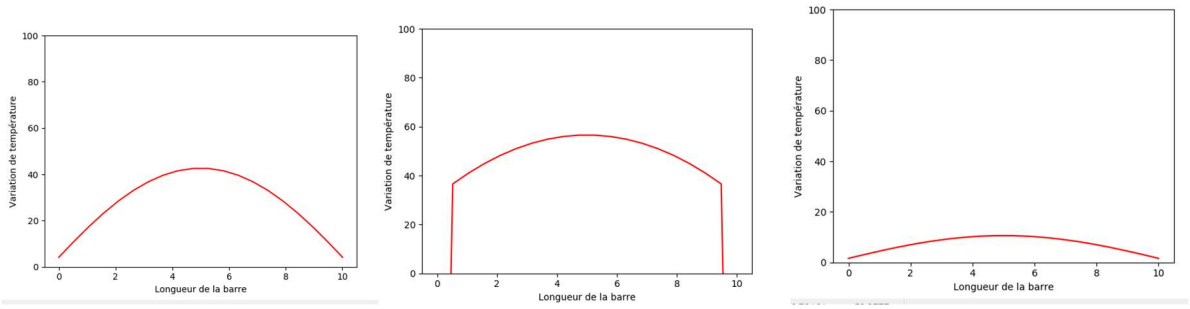


Afin de pouvoir comparer la méthode vectorielle avec la méthode matricielle nous avons implémenté cette dernière. Nous ne sommes cependant pas parvenus à obtenir un résultat cohérent.

Nous pouvons cependant donner les différences entre ces deux méthodes. La méthode matricielle fait entrer en jeu des matrices de très grandes tailles qui limitent rapidement le calcul si l'on ne dispose pas de suffisamment d'espace de mémoire vive. Elle permet cependant d'effectuer les opérations plus rapidement car tous les calculs se font d'un seul coup, à l'inverse de la méthode vectorielle qui calcul un à un les différents vecteurs colonnes de la matrice solution u

6. Modification et interprétation des conditions initiales

On modifie maintenant les conditions initiales afin d'interpréter le terme γ et l'influence de f sur l'évolution de la température dans la barre.
En testant le premier système de conditions pour un γ plus grand on se rend compte qu'à partir de $\gamma = 3$ le résultat converge et semble cohérent. Ce dernier divergeait pour un $\gamma = 1$.



Pour le second système avec $\gamma = 20$ on obtient une faible oscillation mais les conditions aux bords sont en rupture.

De manière générale le schéma vectoriel auquel nous sommes parvenus parait incorrecte. Lors de son codage sur Python nous ne parvenons pas à obtenir des résultats cohérents avec une étude de l'équation de la chaleur sur une barre.

II. Partie 2 : l'équation de la chaleur sur une surface

Le but de cet exercice est de modéliser le phénomène de conduction thermique dans une surface carrée à partir d'une discrétisation de l'équation de la chaleur.

On conservera les notations de l'exercice précédent. On considère le problème aux limites suivant : trouver une fonction $u = u(x, y, t)$ représentant la température au point (x, y, t) avec $(x, y) \in [0, L]^2$ et $t \in [0, T]$, telle que :

$$\begin{cases} \dfrac{du}{dt} - \gamma\left(\dfrac{d^2u}{dx^2} + \dfrac{d^2u}{dy^2}\right) = f \quad dans[0, L] \times [0, T] \\ u = 0 \quad sur\ \partial\ \in [0, \mathrm{L}]^2 \\ u(x, y, 0) = u_0(x, y)\ \forall (x, y)\ \in [0, \mathrm{L}]^2 \end{cases}$$

On fixe M et N deux entiers et on discrétise $[0, L]^2 \times [0, T]$ en introduisant les points $x_i = j\,\Delta x$ et $y_j = i\,\Delta y$ pour $(i,j) \in [[0, M+1]]^2$ et les instants $t^n = n\,\Delta t$ pour $n \in [[0, N]]$. On cherche alors $u_{ij}{}^n \approx u(x_i, y_j t^n)$ pour tout $(j, n) \in [[1, M]]^2 \times [[1, N]]$.

$$\frac{1}{\Delta t}\left(\frac{3}{2}u_{ij}^{n+1} - 2u_{ij}^{n} + \frac{1}{2}u_{ij}^{n-1}\right) - \frac{\gamma}{(\Delta x)^2}\left(\delta_x^2 u_{ij}^{n+1} + \delta_y^2 u_{ij}^{n+1}\right) = f_{ij}^{n+1}$$

1. Ordre du schéma

Le premier terme, celui de gauche est selon le temps.

$$\frac{1}{\Delta t}\left(\frac{3}{2}u_{ij}^{n+1} - 2u_{ij}^{n} + \frac{1}{2}u_{ij}^{n-1}\right)$$

Le second terme, celui de droite est selon l'espace.

$$\frac{\gamma}{(\Delta x)^2}\left(\delta_x^2 u_{ij}^{n+1} + \delta_y^2 u_{ij}^{n+1}\right)$$

Ce schéma est donc d'ordre 2 en temps et en espace.

2. Schéma numérique sous la forme matricielle

On cherche à étudier le schéma numérique suivant :

$$\frac{1}{\Delta t}\left(\frac{3}{2}u_{ij}^{n+1} - 2u_{ij}^{n} + \frac{1}{2}u_{ij}^{n-1}\right) - \frac{\gamma}{(\Delta x)^2}\left(\delta_x^2 u_{ij}^{n+1} + \delta_y^2 u_{ij}^{n+1}\right) = f_{ij}^{n+1}$$

Avec

$$\begin{cases} \delta_x^2 \emptyset_{ij}^n = \emptyset_{i+1j}^n - 2\emptyset_{ij}^n + \emptyset_{i-1j}^n \\ \delta_y^2 \emptyset_{ij}^n = \emptyset_{ij+1}^n - 2\emptyset_{ij}^n + \emptyset_{ij-1}^n \end{cases}$$

Pour simplifier les calculs on pose : $\alpha = \frac{\gamma \Delta t}{(\Delta x)^2}$

$$\Leftrightarrow (4\alpha + \frac{3}{2})u_{ij}^{n+1} - 2u_{ij}^{n} + \frac{1}{2}u_{ij}^{n-1} - \alpha\left(u_{i+1j}^{n+1} + u_{i-1j}^{n+1} + u_{ij+1}^{n+1} + u_{ij-1}^{n+1}\right) = \Delta t f_{ij}^{n+1}$$

$$\Leftrightarrow (4\alpha + \frac{3}{2})u_{ij}^{n+1} - \alpha u_{i+1j}^{n+1} - \alpha u_{i-1j}^{n+1} - \alpha u_{ij+1}^{n+1} - \alpha u_{ij-1}^{n+1} - 2u_{ij}^{n} + \frac{1}{2}u_{ij}^{n-1} = \Delta t f_{ij}^{n+1}$$

Dans un premier temps on détermine la matrice A suivante

$$A = \begin{pmatrix} 4\alpha + \frac{3}{2} & -\alpha & 0 & \dots & 0 \\ -\alpha & 4\alpha + \frac{3}{2} & -\alpha & & \vdots \\ 0 & -\alpha & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -\alpha \\ 0 & \dots & 0 & -\alpha & 4\alpha + \frac{3}{2} \end{pmatrix}$$

On écrit de nouveau le schéma

$$\Leftrightarrow Au^{n+1} + \begin{pmatrix} -\alpha u^0 \\ 0 \\ \vdots \\ 0 \\ -\alpha u^{N+1} \end{pmatrix} - \alpha u_{i-1j}^{n+1} - \alpha u_{ij-1}^{n+1} - 2u^n + \frac{1}{2}u^{n-1} = \Delta t f^{n+1}$$

On fait apparaitre la matrice P permettant de lier les termes en u^{n+1})

$$P = \begin{pmatrix} -\alpha & 0 & \cdots & \cdots & 0 \\ 0 & -\alpha & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & -\alpha \end{pmatrix};$$

Ainsi on peut écrire une nouvelle matrice faisant intervenir avec le produit de Kronecker les matrices P et A.

$$\Leftrightarrow \begin{pmatrix} A & P & \cdots & \cdots & 0 \\ P & A & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & P \\ 0 & \cdots & \cdots & P & A \end{pmatrix} u^{n+1} - 2u^n + \frac{1}{2}u^{n-1} = f^{n+1}\Delta t + C.L$$

$$B = \begin{pmatrix} A & P & \cdots & \cdots & 0 \\ P & A & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & P \\ 0 & \cdots & \cdots & P & A \end{pmatrix}$$

Le schéma numérique peut donc se simplifier sous la forme suivante :

$$\Leftrightarrow Bu^{n+1} - 2u^n + \frac{1}{2}u^{n-1} = f^{n+1}\Delta t + C.L$$

$$\Leftrightarrow u^{n+1} = B^{-1}(f^{n+1}\Delta t + 2\ u^n - \frac{1}{2}u^{n-1} + C.L)$$

Avec C.L une matrice de taille $(m^2 m^2)$ contenant les conditions aux limites.

Il nous reste juste à déterminer u^1 ainsi on pourra déterminer par récurrence sur python les u^n

On effectue son développement de Taylor

$$u_j^1 = u(x_i, y_j, \Delta t) = u(x_i, y_j, 0) + \Delta t \frac{\partial u}{\partial t}(x_i, y_j, 0)$$

On peut remplacer $\frac{\partial u}{\partial t}(x_i, y_j, 0)$ par les conditions que l'on a :

$$\Leftrightarrow u_j^1 = u(x_i, y_j, 0) + \left(f^1 + \gamma\left(\frac{d^2u(x_i, y_j, 0)}{dx^2} + \frac{d^2u(x_i, y_j, 0)}{dy^2}\right)\right)$$

$$\Leftrightarrow u_j^1 = u(x_i, y_j, 0) + \left(f^1 + \gamma\left(\frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{(\Delta x)^2} + \frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{(\Delta y)^2}\right)\right)$$

On a donc $u_j^1 = u_0(x, y) + \Delta t(f^1 + \gamma\left(\frac{C * u_0(x,y)}{(\Delta x)^2} + \frac{C * u_0(x,y)}{(\Delta y)^2}\right)$

Avec C la matrice Laplacienne, tel que C=$\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & & \ddots & & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$

Ainsi nous avons le schéma numérique complet que nous pouvons programmer sur python.

3. Programmation sous python

a. Détail du programme

Dans un premier temps nous allons définir toutes les conditions nécessaires à l'implémentation.

Nous allons créer un fonction *Matrix_Laplace* qui nous permettra de définir plus facilement les matrices nous permettant de mettre des coefficients sur les diagonales supérieurs et inférieurs de nos matrices.

```
def Matrix_Laplace(N=10,A=4,B=-1,C=-1):
    M=B*np.eye(N)+A*np.diag(np.ones(N-1),1)+C*np.diag(np.ones(N-1),-1)
    return M
```

Maintenant nous pouvons définir les matrices *A*, *P* et *B* en utilisant les produits de *Kronecker* ainsi que *Matrix_Laplace.*

```
A=Matrix_Laplace(M,-alpha,4*alpha+(3/2),-alpha)-alpha*np.eye(M,k=1)-
alpha*np.eye(M,k=-1)
P=-alpha*np.eye(M)
B=np.kron(np.eye(M),A)+np.kron(np.eye(M,k=1),P)+np.kron(np.eye(M,k=-
1),P)
CL=np.zeros((M*M,1))
B_inv=np.linalg.inv(B)
```

On obtient donc A de taille (M, M), P de taille (M,M) et par produit de *kronecker* on aura donc un matrice B de taille (M*M,M*M). On définit aussi une matrice CL contenant nos conditions limites. *B_inv* correspond à l'inverse de B qui nous servira pour calculer nos u^1

On peut maintenant implémenter nos vecteurs u^1.

```
Un0=np.zeros((M,M))
Un1=np.zeros((M,M))

#for i in range(0,M) :
#    for j in range(0,M) :
#        Un0[i,j]=4*np.abs(np.sin(2*dx*(i +2)*np.pi/L)+np.sin(2*dx*(j
+2)*np.pi/L)+np.sin(2*dx*(i*j +2)*np.pi/L))

for i in range(0,M):
    for j in range(0,M) :
        Un0[i,j]=np.abs(i*dx*(L-i*dx)*j*dx*(L-j*dx))/(4*L)
```

On définit tous d'abord deux matrices vides de taille (M, M) qui contiendront les données de u^0et u^1 .

A partir d'une boucle et d'un u^0 qu'on a choisi, on calcule cheque coefficient de u^0.

```
Un1=np.reshape(Un0+dt*(f-
gamma*(C@(Un0/(dx**2))+C@(Un0/(dy**2)))),(M*M,1))
Un0=np.reshape(Un0,(M*M,1))
```

Grace au développement de Taylor qu'on a effectué dans la partie précédente on détermine u^1 avec f , une fonction d'excitation qu'on aura précédemment choisie. Le C correspond à la matrice de Laplace comme vu dans la partie précédente.

```
for n in range (N):
    Un2=dt*B_inv@ f+2*B_inv@Un1-0.5*B_inv@Un0
    Un2.reshape(M,M)[0,:],Un2.reshape(M,M)[:,0],Un2.reshape(M,M)[M-
1,:],Un2.reshape(M,M)[:,M-1]=0,0,0,0
    Un0=Un1
```

On effectue une boucle permettant de calculer les u^n au rang supérieur, qu'on reshape en taille (M,M).

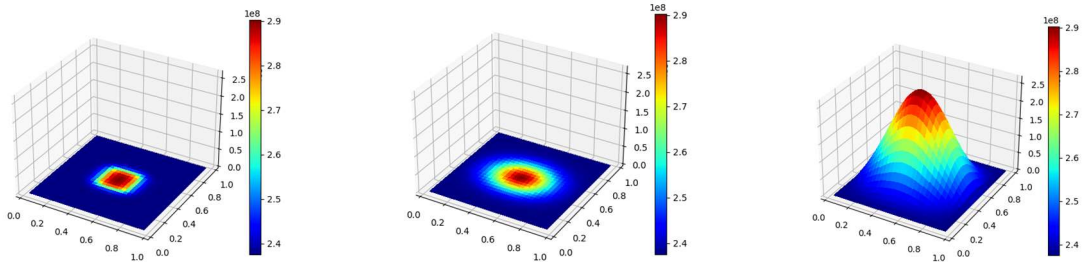
En implémentant tout cela dans une boucle, on pourra *plot* chaque u^n.

```
ax.set_xlim(dx,L-dx)
ax.set_ylim(dx,L-dx)
ax.set_zlim(0,1)
ax.set_xlabel('x', fontsize = 16)
ax.set_ylabel('y', fontsize = 16)
ax.set_zlabel('z', fontsize = 16)
plt.pause(0.1)
ax.cla()
Zn=Un2.reshape(M,M)
ax.plot_surface(X,Y,Zn,cmap=cm.coolwarm)
plt.show()
```

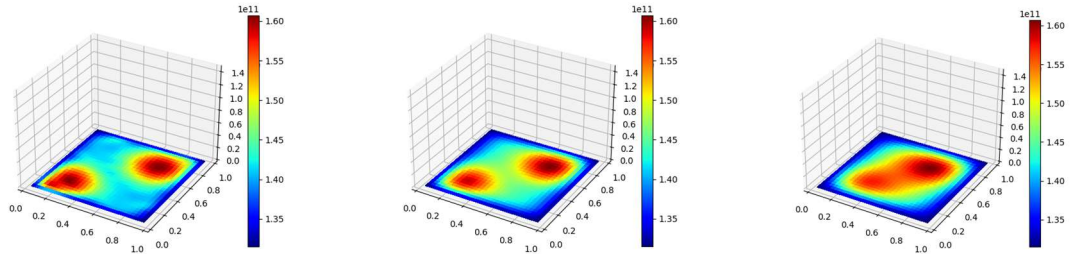
b. Rendu graphique

Dans cette partie nous allons présenter les rendus graphiques pour des matrices u obtenus. Nous avons fait 2 simulations avec des fonctions d'excitations différentes.

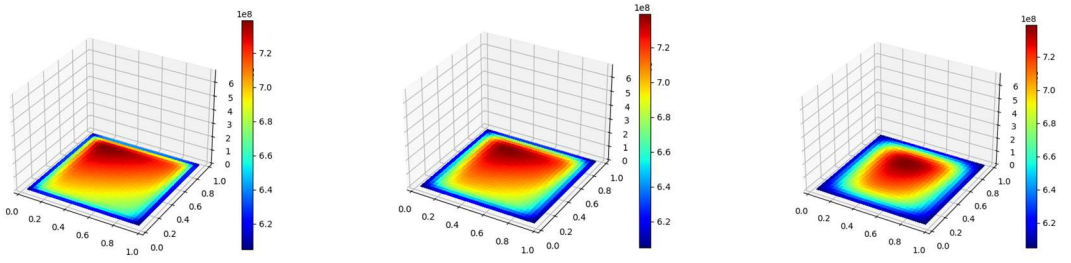
Fonction d'excitation 1(Avec CI= np.zeros((M,M))) :



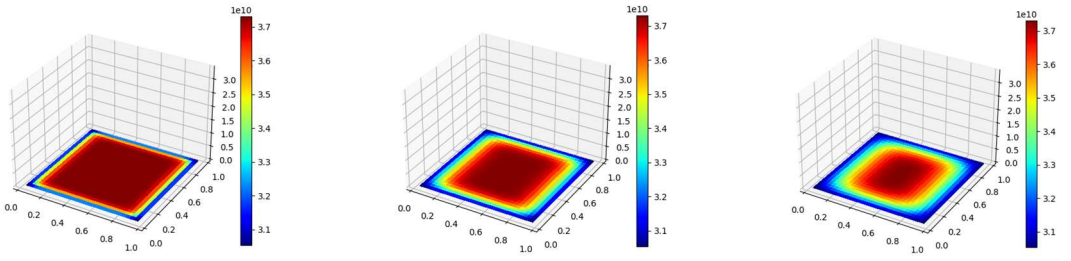
Fonction d'initialisation (Avec CI= np.zeros((M,M))) :



Fonction d'excitation 2 (Avec CI= np.zeros((M,M))) :



Fonction d'initialisation 2 (Avec CI=np.ones((M, M))) :



On observe ainsi que nos certaines valeurs de nos u^n sont beaucoup trop grandes, notamment avec la fonction d'excitation 1. Les valeurs ont tendance à croitre très rapidement. Cela peut être notamment dû à notre implémentation des conditions limites.