Méthodes des différences finies
Régime stationnaire

6 mai 2019

Table des matières

Chapitre 1

Généralités sur les différences finies

1.1 Introduction

En physique et en ingénierie de nombreux problèmes peuvent être mis sous la forme d'une équation différentielle aux limites ou d'une équation différentielle partielle. La particularité de ces équations réside dans la formulation de conditions aux limites. Généralement, l'équation est exprimée sur un domaine Ω et les conditions aux limites aux bords de ce domaine $\partial\Omega$. Ce sont très souvent des problèmes de champ dont la solution dépend de la position du point considéré. On peut distinguer les problèmes stationnaires qui ne font pas intervenir le temps et expriment l'équilibre du système, et les problèmes transitoires qui eux dépendent du temps.

Malheureusement la recherche de solutions exactes pour ces équations se révèle complexe ou se réduit à des cas trop simples pour les domaines d'applications réelles. Il est nécessaire de mettre en place des méthodes de résolution numériques. Les différences finies sont une famille de méthodes possibles largement étudiées dans la littérature scientifique. Dans ce document nous verrons principalement les méthodes de résolution numérique pour les problèmes en une dimension en régime stationnaire et la mise en place de scripts sous python.

Ces équations se retrouvent dans une grande diversité des applications de l'ingénierie et de la physique.
Application 1 : diffusion
exemple :
Equation de la chaleur dans un solide homogène isotrope $\nabla(\lambda\nabla T)+q=0$, ou en 1D, $\frac{\partial}{\partial x}(\lambda\frac{\partial T}{\partial x})+q=0$. Cette équation décrit le champ de température dans le solide soumis à une source de chaleur volumique q.

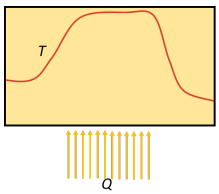

FIGURE 1.1 – Un problème de conduction

Application 2 : advection (transport)
exemple :
Traitement des fumées dans un incinérateur à cogénération. $\vec{u}_0\cdot\nabla C+k_rC^\beta=0$, ou en 1D,$u_0\frac{\partial C}{\partial x}+k_rC^\beta=0$. Cette équation décrit la variation de la concentration C d'un élément chimique soumis à une réaction le long d'un filtre.
Application 3 : équation de Burgers (advection non linéaire)
Cette équation est utilisée en mécanique des fluides. $u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2}$

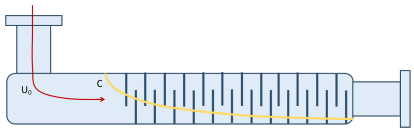

FIGURE 1.2 – Modélisation d'un filtre à fumée

1.2 Principe de la méthode

Objectif : transformer une équation continue $L(u, \partial_x u, \partial_x^2 u, ...)$ sur un domaine continu Ω en un système d'équations algébriques à N inconnues en s'appuyant sur une discrétisation de Ω appelée *maillage*.

$$L(u, \partial_x u, \partial_x^2 u, ...) + f = 0 \rightarrow [K]\{T\} = \{F\}$$

Méthode : discrétiser les dérivées du champ u dans l'équation d'équilibre, ainsi que les conditions aux limites, à l'aide d'un développement de Taylor au noeud i.

Exemples :
- $\frac{\partial u}{\partial x}(x_i) \simeq \frac{u_{i+1} - ui - 1}{2\delta x}$ dérivée première centrée d'ordre 2 ;
- $\frac{\partial u}{\partial x}(x_i) \simeq \frac{u_{i+1} - ui}{\delta x}$ dérivée première décentrée à droite (aval) d'ordre 1 ;
- $\frac{\partial u}{\partial x}(x_i) \simeq \frac{u_i - ui - 1}{\delta x}$ dérivée première décentrée à gauche (amont) d'ordre 1 ;
- $\frac{\partial^2 u}{\partial x^2}(x_i) \simeq \frac{u_{i+1} - 2u_i + ui - 1}{\delta x^2}$ dérivée seconde centrée d'ordre 2.

1.3 Avantages et Inconvénients

TABLE 1.1 – Avantages et inconvénients

Avantages	Inconvénients
Simplicité de la méthode, facilité d'implémentation ;	Expression des conditions aux limites ;
Discrétisation 1D immédiate ;	Impossibilité de travailler avec des maillages non réguliers ;
Ressources scientifiques importantes, grande diversité de schémas pour différents problèmes ;	- Difficultés pour les extensions 2D ou 3D ;
Structuration des données simple.	Impossibilité de résoudre certains problèmes en forme forte comme la mécanique.

En effet, la méthode des différences finies est très intuitive et est donc facile à coder comparativement aux éléments finis. En 1D on peut directement construire les matrices de manière globale, ce qui facilite le travail de codage. De plus, si votre équation adopte une forme un peu particulière, il y a énormément de schémas dans la littérature. Par exemple, le schéma de Lax-Wendroff pour l'équation d'advection. De plus, il n'y a pas besoin d'une grosse structuration de vos données car la géométrie est généralement simple et cela allège l'implémentation d'une solution.

Cependant, comme la géométrie est simple il est impossible de travailler sur des maillages non réguliers, ce qui limite la méthode. En outre, le passage à la 2D n'est pas aisé et cela même pour des géométries très simples. Aussi, cette méthode s'appuie sur une formulation « forte », et cela peut limiter les problèmes pour lesquels la méthode est applicable. Par exemple, un simple problème de poutre est résoluble par différences finies mais l'obtention de la forme forte de l'équation est ardue à obtenir et on perd vite l'intérêt de la méthode. Aussi, du fait de cette formulation forte les équations aux limites n'apparaissent pas naturellement contrairement à la méthode des éléments finis. Les conditions aux limites peuvent se montrer plus dures à exprimer, mais cela concerne essentiellement le régime transitoire ou non linéaire.

Chapitre 2

Equation de diffusion 1D : équation de la chaleur

2.1 Equation de la chaleur cas général

l'équation de la chaleur dans un solide est une équation de diffusion, la température se diffuse dans le matériau. Au même titre que la chaleur, d'autres phénomènes peuvent être modélisés par une équation de diffusion, comme la loi de Fick pour les espèces chimiques ou la diffusion de matière dans les milieux poreux ou liquide. Pour l'équation de la chaleur la loi de Fourier permet de retrouver l'expression du flux.

$$\Phi(x) = -k(x) \cdot \nabla T(x)$$

et l'équation d'évolution de la température est donnée par ; $\rho c_p \frac{\partial T}{\partial t} = \nabla \Phi(x) + q(x)$
Dans le cas du régime stationnaire sur un domaine 1D ;

$$k \frac{\partial^2 T}{\partial x^2} + q(x) = 0$$

où :

- k est la conductivité thermique $[W.K^{-1}.m^{-1}]$;
- *q(x)* est une source de chaleur volumique $[W.m^{-3}]$.

Discrétisation par différences finies.

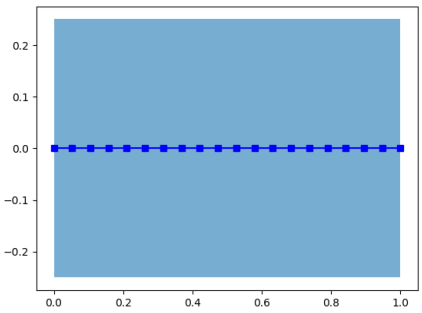

FIGURE 2.1 – Schéma de discrétisation de la géométrie

On discrétise le domaine en (N+1) points régulièrement espacés de Δx, on obtient l'équation algébrique par différences finies ;

$$k(\frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2}) + q_i = 0 \forall i \in [1; N-1]$$

2.2 Ecriture matricielle

On peut écrire sous forme de matrice le système suivant :

$$[K]\{T\} = \{F\}$$

$$[K] = \frac{k}{\Delta x^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \text{ et } F = -\begin{Bmatrix} q_0 \\ q_1 \\ \vdots \\ q_i \\ \vdots \\ q_{n-1} \\ q_n \end{Bmatrix}$$

2.3 Conditions aux limites

Il existe plusieurs méthodes pour mettre en place les conditions aux limites. Une première est d'écrire les conditions de manière discrète aux nœuds $i = 0$ et $i = N$ puis de *substituer* les équations obtenues dans la première et dernière ligne de la matrice $[K]$ et du vecteur $\{F\}$. Une autre méthode consiste à utiliser les équations aux limites discrétisées pour exprimer la valeur à la frontière et d'ajouter sa contribution dans la matrice $[K]$ et le vecteur $\{F\}$ après avoir réduit le nombre de degrés de liberté aux nœuds internes (*réduction*). On utilise alors une cellule "fantome", cette méthode est privilégiée pour les problèmes non linéaires ou instationnaires. La première méthode est très simple mais elle n'est pas utilisable pour le régime transitoire, ou pour les problèmes non linéaires.

Conditions aux limites à gauche

TABLE 2.1 – Conditions aux limites à gauche

Condition	**Equation**	**Discrétisation**	$[K]$	$\{F\}$
Dirichlet (température)	$T(x=0) = T_g$	$T_0 = T_g$	$k_{00} = 1$ $k_{01} = 0$	$f_0 = T_g$
Neumann (flux entrant)	$-k \cdot \partial_x T(x=0) = \Phi_g$	$\frac{-k}{\Delta x}(T_1 - T_0) = \Phi_g$	$k_{00} = k/\Delta x$ $k_{01} = -k/\Delta x$	$f_0 = \Phi_g$
Robin (convection)	$-k \cdot \partial_x T(x=0) = h(T_g - T_0)$	$\frac{-k}{\Delta x}(T_1 - T_0) = h(T_g - T_0)$	$k_{00} = k/\Delta x + h$ $k_{01} = -k/\Delta x$	$f_0 = hT_g$

Conditions aux limites à droite

TABLE 2.2 – Conditions aux limites à droite

Condition	**Equation**	**Discrétisation**	$[K]$	$\{F\}$
Dirichlet	$T(x=L) = T_d$	$T_0 = T_d$	$k_{nn} = 1$ $k_{nn-1} = 0$	$f_n = T_d$
Neumann	$k \cdot \partial_x T(x=L) = \Phi_d$	$\frac{k}{\Delta x}(T_n - T_{n-1}) = \Phi_d$	$k_{nn} = k/\Delta x$ $k_{nn-1} = -k/\Delta x$	$f_n = \Phi_d$
Robin	$-k \cdot \partial_x T(x=0) = h(T_n - T_d)$	$\frac{-k}{\Delta x}(T_n - T_{n-1}) = h(T_n - T_d)$	$k_{nn} = k/\Delta x + h$ $k_{nn-1} = -k/\Delta x$	$f_n = hT_d$

2.4 Problème 1 : Equation de la chaleur, conductivité uniforme

On s'intéresse à l'équation de la chaleur en 1D avec une conductivité uniforme sur le domaine. On se propose d'écrire une fonction **solveHeat1D** de 6 arguments. Cette fonction permettra de résoudre un problème quelconque de conduction 1D en fonction des données fournies en entrée.

- **k** la conductivité thermique (*float*) ;
- **q** la fonction de x qui donne la source volumique de chaleur ;
- **L** la longueur de domaine considéré (*float*) ;
- **n** le nombre de subdivisions du domaine (*int*) ;
- **left** un tuple qui donne la condition à gauche et ses paramètres (*tuple*) ;
- **right** un tuple qui donne la condition à droite et ses paramètres (*tuple*).

Cette fonction retourne le vecteur des positions des nœuds **x** et les solutions nodales de l'équation **T**. Ce problème vous propose de mettre en place un solveur de l'équation de la chaleur en régime stationnaire réutilisable dans votre domaine d'étude.

2.4.1 Solution proposée

Imports nécessaires ;

```
##Imports
from numpy import eye, linspace
from numpy.linalg import solve , norm
```

D'abord, il faut discrétiser le domaine continu Ω et construire la matrice $[K]$ et le vecteur $\{F\}$. Pour construire le vecteur **x** on peut utiliser *linspace* de *numpy*. Pour la matrice tridiagonale **K** on peut utiliser la fonction *eye(size,k=rang)* de *numpy*, où size est la taille de la matrice et k le rang de la diagonale voulue.

```
##Heat Equation Solver
def solveHeat1D(k,q,L,n,left,right):
    # Step 0 : discretization
    x = linspace(0,L,n+1)
    dx = L/n

    # step 1 : compute the matrices
    K = k/dx**2*(eye(n+1,k=-1)-2*eye(n+1)+eye(n+1,k=1))
    F = -q(x)
```

Les conditions aux limites sont variées, et à chaque frontière il y a trois possibilités ; température imposée, flux imposé, convection. La syntaxe que j'ai choisie est la suivante.

```
right/left = ('temperature',T)
right/left = ('flux',g)
right/left = ('convection',h,T)
```

Enfin, pour résoudre le système $[K] \cdot \{T\} = \{F\}$ j'utilise **solve(K,F)** de la bibliothèque *numpy.linalg*.

```
#step 4 : solve
u = solve(K,F)
```

2.4.2 Exemple de résolution

On se propose de résoudre le problème suivant :

$$k\frac{\partial^2 T}{\partial x^2} + q(x) = 0$$

avec ;
$x \in [0; L]$
CL :

$-k\partial_x T_{x=0} = h_g \cdot (T_g - Tx = 0)$ à gauche ;
$k\partial_x T_{x=L} = h_d \cdot (T_d - Tx = L)$ à droite.

$q(x) = \frac{500}{x_d\sqrt{\pi}} \exp(-\frac{(x-x_c)^2}{x_d^2})$

Données : $L = 0.01m; k = 1.5W.K^{-1}.m^{-1}; x_c = 0.006; xd = 0.0005; h_g = 5W.K^{-1}.m^{-2}; h_d = 12W.K^{-1}.m^{-2}$;
$T_g = 293.15K; T_d = 288.15K$

Voici les résultats obtenus et l'appel de la fonction **solveHeat1D**.

```
import numpy as np

k = 0.5
n = 150
L = 0.01

Td,hd = 20,10
Tg,hg = 18,10

xc,xd = 0.006 , 0.0005
source = lambda x : 500/(xd*np.sqrt(np.pi))*np.exp(-(x-xc)**2/xd**2)

left = ('convection',hg,Tg)
right = ('convection',hd,Td)


x,T = solveHeat1D(k,source,L,n,left,right)
```

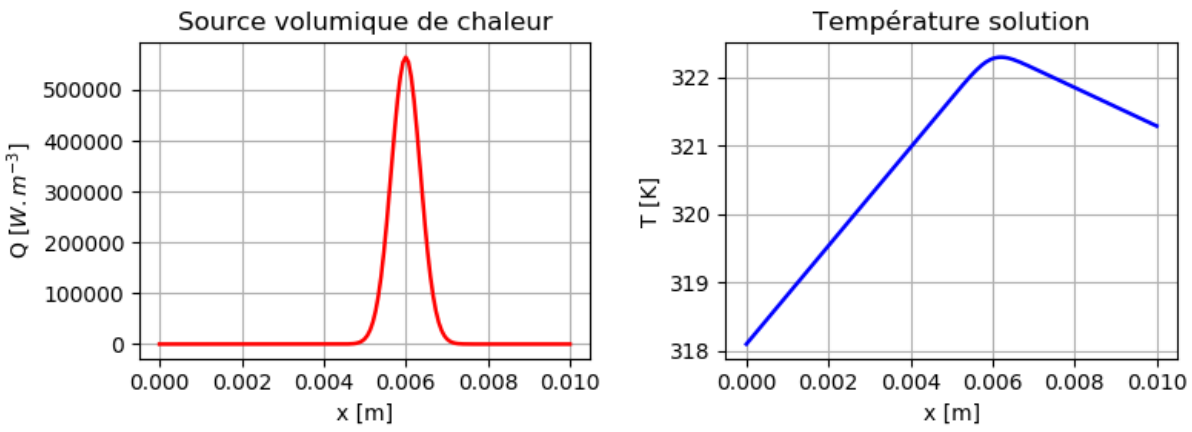

FIGURE 2.2 – Résolution problème 1

2.5 Discrétisation conservative

Pour certains problèmes, notamment lorsque la conductivité est variable ou non uniforme, il est nécessaire d'adopter une forme conservative.

forme conservative $\frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) \neq k\frac{\partial^2 T}{\partial x^2}$ forme non conservative

Il faut alors adopter une stratégie de discrétisation différente pour tenir compte de la variation de la conductivité en fonction de la position x.
Voici une solution possible : $\partial_x k \partial_x T_i \simeq k_{i+\frac{1}{2}} \frac{T_{i+1}-T_i}{\Delta x^2} - k_{i-\frac{1}{2}} \frac{T_i - T_{i-1}}{\Delta x^2}$
Il y a plusieurs méthodes pour évaluer les termes $k_{i-\frac{1}{2}}$ et $k_{i+\frac{1}{2}}$, on peut par exemple utiliser la moyenne des valeurs aux noeuds.
$k_{i+\frac{1}{2}} = 0.5(k_i + k_{i+1})$ et $k_{i-\frac{1}{2}} = 0.5(k_i + k_{i-1})$.

2.6 Problème 2 : Equation de la chaleur, conductivité variable

Cette fois, nous nous proposons de résoudre l'équation de la chaleur mais avec une conductivité variable sur l'épaisseur du domaine considéré.

Résoudre

$$\frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) = 0$$

avec

$$x \in [0, L]$$

$$T_{x=0} = T_g \text{ et } T_{x=L} = T_d$$

Pour la conductivité, on choisit une fonction constante par morceau qui permet de modéliser une problème multicouche.

$$k(x) = \begin{cases} 0.28 W.K^{-1}.m^{-1} & \text{si } x \leq 0.10m; \\ 0.84 W.K^{-1}.m^{-1} & \text{si } 0.10 < x \leq 0.35m; \\ 0.05 W.K^{-1}.m^{-1} & \text{si } 0.35 < x \leq 0.40m \end{cases}$$

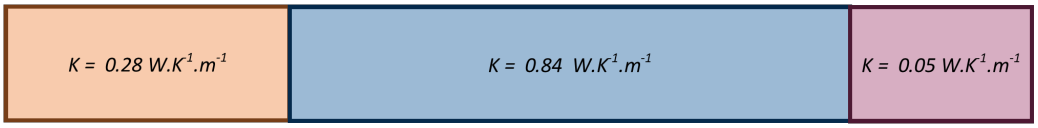

FIGURE 2.3 – Problème multicouche

2.6.1 Solution proposée

Avant tout, il faut construire la fonction de la conductivité suivant la position. Ensuite, pour construire la matrice il est toujours possible d'utiliser une méthode compacte avec la fonction *eye* de *numpy*. Mais par souci de clarté, je préfère utiliser une méthode itérative avec une boucle *for*. Dans cette boucle, je ne remplis que les ligne 1 à N-1, car les lignes 0 et N sont réservées aux conditions aux limites. La solution complète est disponible dans le fichier Heat1DConservative.py.

Imports et écriture de la fonction conductivité ;

```
## Imports
import numpy as np
from numpy.linalg import solve
import matplotlib.pyplot as plt

##Data
k1,x1 = 0.28,0.1
k2,x2 = 0.84,0.25
k3,x3 = 0.04,0.05
L = x1+x2+x3
k = lambda x : k1*(x<=x1) + k2*(x>x1)*(x<=x2+x1) + k3*(x>x1+x2)*(x<=x2+x1+x3)
```

Discrétisation et assemblage de la matrice $[\mathbf{K}]$;

```
##Discretization and matrix

N = 75
dx = L/N
x = np.linspace(0,L,N+1)
f = np.zeros_like(x)
K = np.zeros((N+1,N+1))

for i in range(1,N):
    ki = k((x[i-1]+x[i])/2)
    kj = k((x[i+1]+x[i])/2)
    K[i,i-1:i+2:] = [ ki/dx**2,-ki/dx**2-kj/dx**2,kj/dx**2 ]
```

Conditions aux limites et résolution

```
##Boundary conditions
K[0,0] = 1
K[-1,-1] = 1
f[0] = Tleft
f[-1] = Tright

##Solve
T = solve(K,f)
```

Enfin, voici les résultats obtenus.

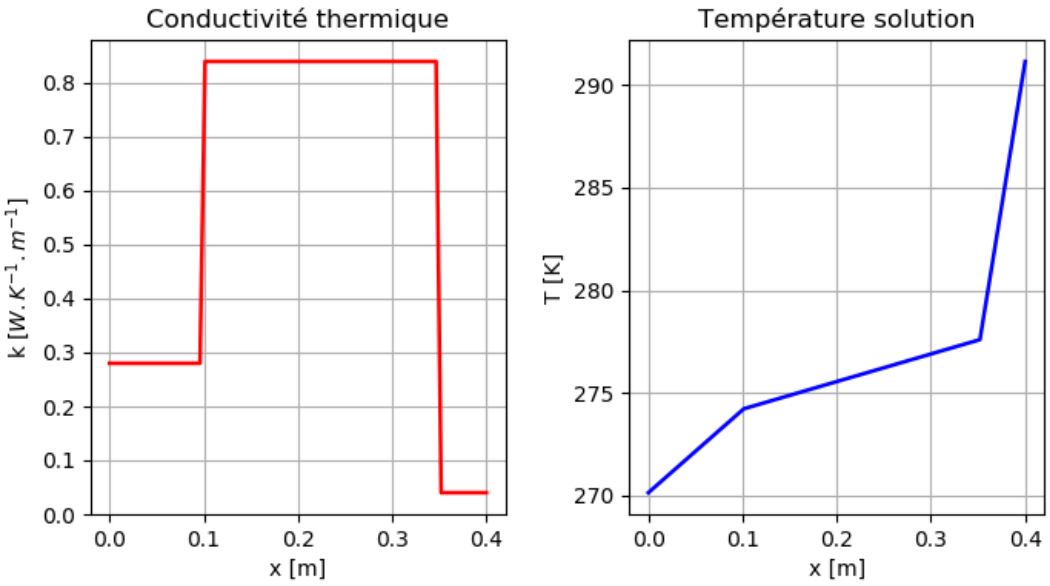

FIGURE 2.4 – Résolution problème 2

Chapitre 3

Equation d'advection (transport) : schéma Upwind

3.1 L'équation d'advection-diffusion

L'équation d'advection, de convection ou de transport, désigne le transport d'une quantité scalaire ou vectorielle par le mouvement du milieu environnant. Généralement cette équation est associée aux milieux fluides ou gazeux. On peut étudier le transport de la chaleur, de l'énergie, d'une concentration chimique, de charges électriques ou encore de la vitesse elle-même. L'advection de la vitesse elle-même correspond au transport de la quantité de mouvement.

$$\vec{c} \cdot \nabla \mathbf{u} = 0$$

L'équation d'advection pure n'a pas beaucoup d'intérêt à être résolue numériquement puisqu'elle traduit la conservation de la quantité u suivant la direction de la vitesse $\vec{c}$. Par ailleurs, on peut étudier le transport d'une quantité soumise à un effet de diffusion, à une source volumique, à des flux extérieurs. On étudiera alors plutôt l'équation d'advection-diffusion en régime stationnaire.

$$\vec{c}\nabla u = \nabla(D\nabla u) + f(\mathbf{r})$$

avec $\mathbf{r}$ le vecteur position.
Réduisons l'expression précédente au cas 1D.

$$c\frac{\partial u}{\partial x} = \frac{\partial}{\partial x}(D\frac{\partial u}{\partial x}) + f(x)$$

u est la quantité transportée, c la célérité ou vitesse suivant x, et D la diffusivité de la grandeur u dans le milieu considéré.

Exemple 1 : Equation de la chaleur dans un fluide
Le bilan de chaleur dans un fluide mène à l'équation suivante, sous réserve de k,ρ et c_p constants.

$$\rho c_p u_0 \frac{\partial T}{\partial x} = k\frac{\partial^2 T}{\partial x^2} + q(x)$$

Exemple 2 : Transport d'espèce chimique dans un réacteur
On peut modéliser un réacteur chimique de la façon suivante ; l'espèce chimique a, représentée par sa concentration, subit un transport, une diffusion et une réaction d'ordre β.

$$u_0\frac{\partial C_a}{\partial x} - D\frac{\partial^2 C_a}{\partial x^2} + k_r C_a^\beta = 0$$

3.2 Cas d'études et conditions aux limites

Avant d'aborder l'étape de discrétisation de l'équation, regardons les applications possibles dans le cas 1D. Cette équation concerne des quantités d'un milieu fluide animé d'un mouvement donc d'une vitesse d'écoulement. En 1D, les écoulements sont simples, de gauche à droite, ou de droite à gauche. Il n'y a que deux cas, qui ne varient que par l'orientation de la vitesse. Pour simplifier, nous ne considérerons que le cas où la vitesse u est positive, c'est à dire de gauche vers la droite. Aussi, les conditions aux limites sont moins variés, le plus souvent la valeur à l'entrée est fixée, alors qu'à la sortie on impose un gradient nul. On se limite donc à l'étude de l'évolution d'une quantité caractéristique d'un fluide le long d'une conduite.

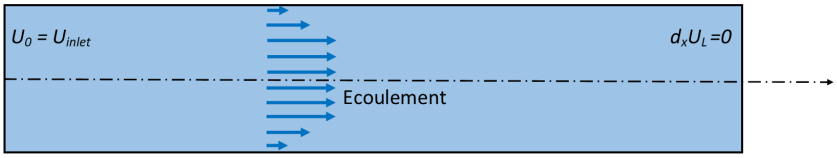

FIGURE 3.1 – Conditions aux limites pour un problème d'advection

3.3 Résolution d'un problème modèle

Précédemment nous avons vu comment discrétiser une équation aux limites par différences finies et nous avons appliqué la méthode à différents problèmes de diffusion. Nous allons maintenant regarder ce qu'il en est de l'équation d'advection-diffusion. Nous verrons qu'il est nécessaire de décentrer le schéma pour obtenir une solution satisfaisante du problème. Afin d'analyser les performances des schémas on utilisera un problème-modèle.

$$c\frac{\partial u}{\partial x} - d\frac{\partial^2 u}{\partial x^2} = 0$$
$$u(0) = 0$$
$$u(L) = 1$$

Ce problème n'a pas de sens physique, mais on connaît une solution analytique ce qui permet d'obtenir une référence. Cette solution fait apparaître le nombre de *Péclet*, $P_e = L\frac{c}{d}$.

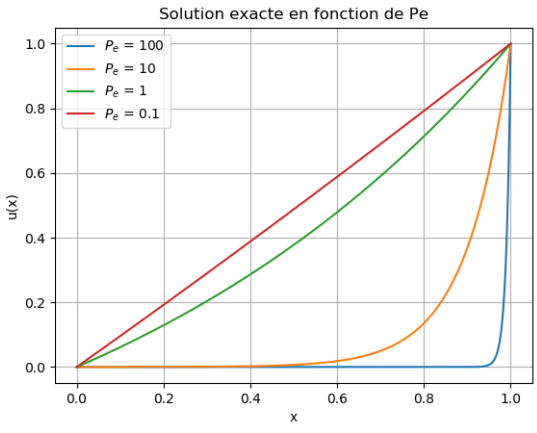

FIGURE 3.2 – Solution problème modèle

$$u(x) = \frac{\exp -\frac{P_e}{L}x - 1}{\exp -\frac{P_e}{L} - 1}$$

3.4 Discrétisation et décentrement

Schéma centré Pour discrétiser le problème nous serions tentés d'utiliser un schéma centré d'ordre 2. Ce schéma est bien connu pour fournir des solutions oscillantes et donc instables, néanmoins, avec un grand nombre de points on stabilise la méthode qui arrive à converger vers une solution exacte. La condition de stabilité de l'équation est donnée par une inégalité faisant apparaître le nombre de Péclet de maille $Pe_{\Delta x} = \Delta x \frac{c}{d} < 1$. En pratique cette condition est extrêmement restrictive. Par exemple, pour une fluide comme l'air la diffusivité thermique est de l'ordre de $1e-4m^2/s$, avec une échelle caractéristique $c = 1m/s$ et $L = 1m$ on obtient un nombre de Péclet de l'ordre de $1e+4$. Pour satisfaire la condition de stabilité il faudrait 10000 points pour un problème 1D. Cette solution n'est donc pas envisageable.

Schéma décentré amont ou Upwind Essayons maintenant un schéma décentré d'ordre 1 pour le terme advectif, le but est de décentré le schéma en remontant le courant ou la vitesse de transport. On appelle aussi ce schéma Upwind.

$$c\frac{u_i - u_{i-1}}{\Delta x} - d\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = 0$$

Cette discrétisation ne produira pas d'oscillation, mais l'ordre de convergence de la méthode est réduit. La simplicité et le faible coût de la méthode la rend toutefois très attractive. On peut écrire un schéma upwind généralisé qui s'adapte au signe de la vitesse c.

$$c\partial_x u \simeq \frac{c + |c|}{2} \cdot (\frac{u_i - u_{i-1}}{\Delta x}) + \frac{c - |c|}{2} \cdot (\frac{u_{i+1} - u_i}{\Delta x})$$

Si la vitesse est positive on garde la discrétisation de gauche et si la vitesse est négative la discrétisation de droite reste. On conserve la forme upwind.

Résultat numérique On observe bien l'oscillation de la solution pour de faibles nombres de nœuds avec le schéma centré. Puis, par la suite la solution tend progressivement vers la solution exacte. La schéma upwind lui n'oscille pas, néanmoins on observe une convergence plus lente vers la solution exacte. On voit clairement l'avantage de solution décentré lorsque le nombre de Péclet augmente, la solution avec schéma décentré garde une allure acceptable. En pratique on préférera les méthodes avec des contraintes de stabilité moins stricte.

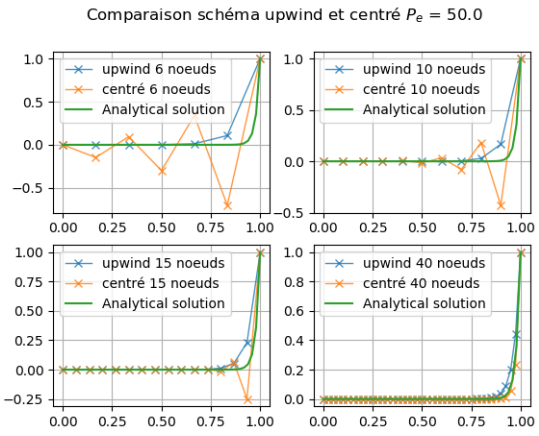

FIGURE 3.3 – Résultats numériques problème modèle

3.5 Problème 3 : Etude d'un filtre à manche

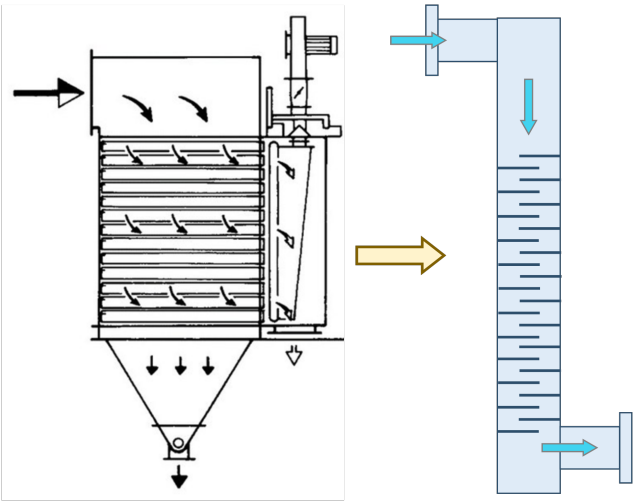

FIGURE 3.4 – Modélisation d'un filtre à fumées

On se propose d'étudier le filtrage de composants dangereux présent dans les fumées à la sortie d'un incinérateur à cogénération. Ces fumées entrent avec une concentration C_{inlet} dans le filtre. L'équation d'équilibre de la concentration est une équation d'advection-diffusion-réaction linéaire donnée par la loi de Fick. On utilisera la méthode Upwind pour résoudre le problème.

Résoudre

$$u_x \frac{\partial C}{\partial x} - \alpha \frac{\partial^2 C}{\partial x^2} + k_r C = 0$$

avec

$$x \in [0, L]$$

$$C_{x=0} = C_{inlet} \text{ et } \partial_x C = 0$$

Écrire une fonction **solveReactor** d'un argument n, le nombre de points pour la discrétisation, qui renvoi le vecteur **x** des positions des nœuds et C les solutions nodales de l'équation.
Données ; $L = 3.5m; k_r = 1.5; \alpha = 1.85e-5m^{-2}; u_{inlet} = 0.5m/s; C_{inlet} = 1.0;$

3.5.1 Solution proposée

Après une phase de discrétisation du domaine, il faut réécrire le problème sous forme matriciellle. Dans ce cas, j'ai fait le choix d'inclure la réaction dans la matrice et j'obtiens le système suivant ; $[A]\{X\} = \{b\}.[A]$ est la sommme des matrices de diffusion, advection et réaction. Si la réaction n'est pas linéaire on ne peut pas écrire ce système et la méthode de subsitution directe dans la matrice **A** ne fonctionne plus. Il faut adopter une méthode de réduction des *ddl*, comme la réduction de Guyan, pour prendre en compte les conditions aux limites. De plus, il faudrait utiliser un solveur non linéaire.

```
##Imports
import numpy as np
from numpy.linalg import solve
import matplotlib.pyplot as plt

##Data
```

```
u = 0.5 #inlet velocity
Cin = 1 #inelt concentration
alpha = 1.85e-5#chemical diffusivity
kr = 1.5 #reaction constant
L = 3.5 #lenght

##Upwind scheme

def solveReaction(n) :
    print("Pe = {:.5f}".format(u*L/alpha))
    #discretization
    dx = L/n
    x = np.linspace(0,L,n+1)
    print("Pe_dx = {:.5f}".format(u*dx/alpha))
    #Matrices
    Umat = u*( np.eye(n+1,k=0) - np.eye(n+1,k=-1) )/(dx) #advection
    Dmat = alpha/dx**2*( np.eye(n+1,k=1) + np.eye(n+1,k=-1) - 2*np.eye(n+1) ) #diffusi
    Rmat = kr*np.eye(n+1) #reaction
    #Write the system A.x = b
    b = np.zeros_like(x)
    A = Umat-Dmat + Rmat
    #Boundary condition
    A[0] = 0 ; A[0,0] = 1 ; b[0] = Cin
    A[-1,-2:] = [-1,1] #right zero gradient
    #Solve
    C = solve(A,b)
    return x,C
```

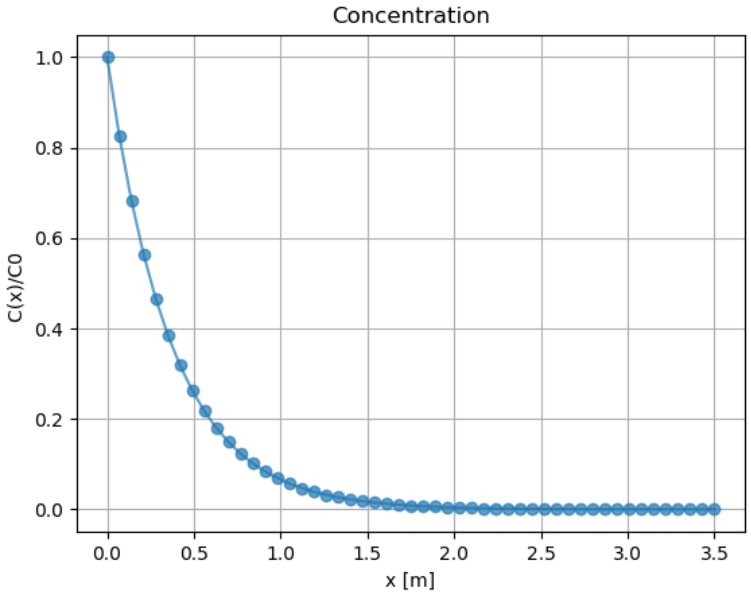

FIGURE 3.5 – Résultats réacteur chimique

Chapitre 4

Problèmes 2D à maillage régulier

4.1 Mise en équation et discrétisation

On étudie une équation de type diffusion pure avec terme source en coordonnées cartésiennes 2D. Pour l'équation de la chaleur on obtient l'équation suivante à résoudre avec ses conditions aux limites.

$$k\frac{\partial^2 T}{\partial x^2} + k\frac{\partial^2 T}{\partial y^2} + Q(x,y) = 0$$

On utilise la discrétisation différences finies pour résoudre le problème. La discrétisation spatiale donne deux pas spatiaux, un sur x l'autre sur y. Si on représente le champs de température sur un tableau de $N_y + 1$ par $N_x + 1$ cases, on obtient la discrétisation au point (i, j), où i est l'indice de la ligne, soit y, et j de la colonne qui donne x. Ce choix est arbitraire et n'a pas d'incidence sur la formulation du problème.

$$k(\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta y^2}) + k(\frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta x^2}) + Q(x_j, y_i) = 0$$

On peut formuler le problème de manière matricielle. Mais il faut d'abord réarranger le vecteur inconnu des températures. Voici une formulation possible ;

$$\{T\} = \begin{Bmatrix} T_{0,0} & T_{0,1} & \cdots & T_{0,n_x} & \cdots & T_{i,j} & \cdots & T_{n_y,n_x} \end{Bmatrix}^T$$

et la matrice de raideur ;

$$[K] = \begin{bmatrix} [A] & [B] & 0 & \cdots & 0 \\ [B] & [A] & [B] & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & [B] & [A] & [B] \\ 0 & \cdots & 0 & [B] & [A] \end{bmatrix}$$

avec $[A] = \frac{k}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} - \frac{2k}{\Delta y^2}[Id]$ et $[B] = \frac{k}{\Delta y^2}[Id]$

La matrice $[K]$ peut être obtenue avec un produit de *Kronecker* sur les martices $[A]$ et $[B]$.

Cette formulation est très lourde et conduit à des matrices très grandes qui peuvent très vite devenir trop grosses à manipuler. Il faut passer par des matrices creuses qui permettront de réduire fortement le coût de stockage. Avant de résoudre le problème, il faut prendre en compte les conditions aux limites. Pour cela, la méthode est la même que précédemment au niveau de la matrice [K]. Mais comme le problème est en 2D il faut particulièrement bien maîtriser les indices des bords.

Toutes ces problématiques rendent la méthode des différences finies contraignante et devient finalement plus complexe que la méthode des éléments finis pour un problème 2D à maillage régulier. Néanmoins, une autre solution est possible grâce à la formulation vectorielle du langage python, une méthode itérative qui ne fait plus intervenir de matrices.

4.2 Méthode de résolution itérative

A partir de la discrétisation formulée au point (i, j) à l'intérieur du domaine, on peut donner la valeur de la température à partir des valeurs qui entourent le point.

$$T_{i,j} = \frac{1}{2(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2})} \times (k(\frac{T_{i-1,j} + T_{i+1,j}}{\Delta y^2}) + k(\frac{T_{i,j-1} + T_{i,j+1}}{\Delta x^2}) + Q(x_j, y_i))$$

De plus, python et les objets *array* de *numpy* permettent une action nommée *slicing*. Cette fonctionnalité permet d'appliquer une opération algébrique sur l'ensemble, ou une partie, d'un vecteur ou d'un tableau. Donc, l'équation précédente est applicable directement sur le champ de température. On peut ensuite ajuster les conditions aux limites directement sur le champ. La méthode est répétée de manière itérative jusqu'à atteindre une condition de convergence. La condition de convergence est généralement définie par la norme de la différence entre le champ à l'itération n et la suivante $n + 1$.

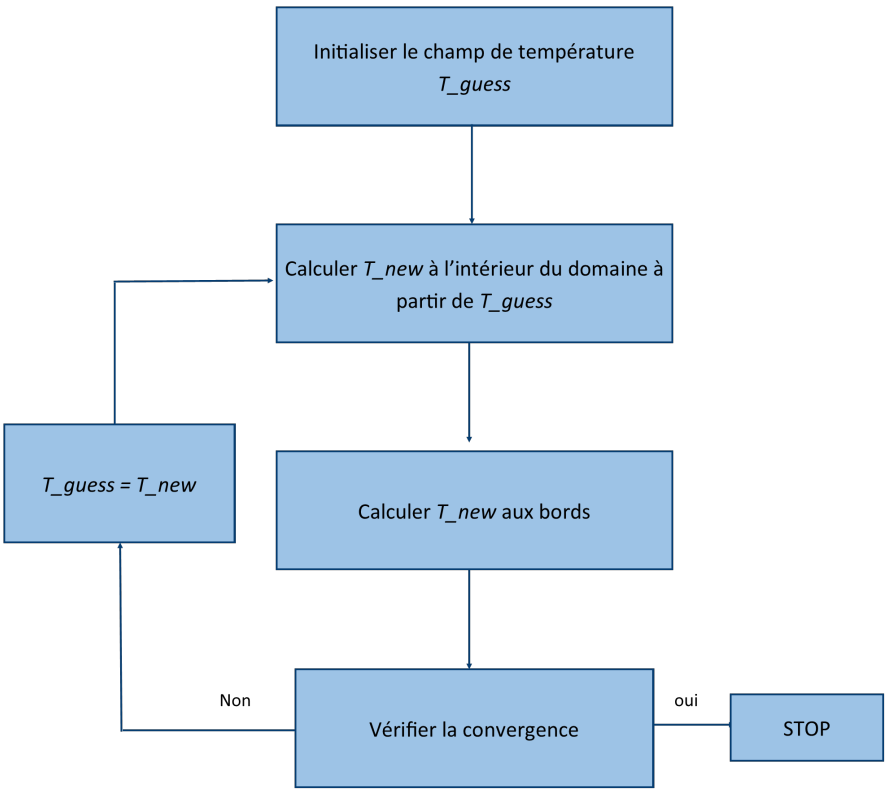

FIGURE 4.1 – Algorithme de résolution itérative

Néanmoins, je déconseille cette résolution dans un cadre professionnel. En effet, avec cette méthode nous ne sommes pas garantis de converger vers la bonne solution. Il suffit de regarder la dépendance à la tolérance. La solution évolue encore beaucoup même si la différence entre deux itérations et de l'ordre de 10e-3. Mettre une tolérance trop faible augmente la quantité d'itération nécessaire pour converger et on peut vite atteindre la limite d'itérations. Pour une tolérance de 10e-4 il a fallu 10000 itérations. Cette méthode est facile à coder et reste un outil pédagogique simple.

4.3 Méthode matricielle

Comme vu précédemment on peut écrire le problème sous forme matricielle. Pour cela il faut réarranger le champ de températures 2D dans un vecteur 1D. La solution précédemment proposée est "d'empiler les lignes les unes sur les autres". Il faut passer un peu de temps sur le papier avant de se lancer dans le codage. Il faut absolument préparer notre étude est bien maîtriser le passage d'un tableau de températures 2D au vecteur de températures 1D. Cette méthode nécessite de bien comprendre comment vont évoluer les indices entre les deux formes du champ de températures. N'hésitez pas à prendre du temps pour vérifier que vous accédez bien aux indices dans le vecteur en utilisant l'affichage de tableaux 2D dans *matplotlib* par exemple.

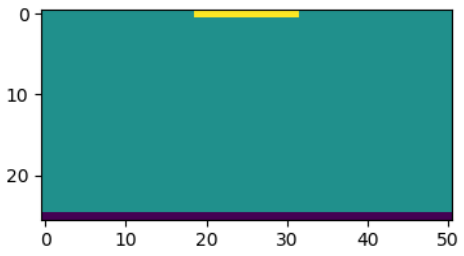

FIGURE 4.2 – Indexation des frontières

Si on prend les conditions aux limites du *problème 1.4* décrit dans la partie suivante, je vérifie que j'accède bien aux deux frontières où les températures sont imposées. La ligne violette pour la température la plus froide et la ligne jaune pour la température chaude. Cette vérification me permet de m'assurer d'accéder aux bons éléments du vecteur température.

Ensuite, la deuxième contrainte concerne la taille de la matrice de conduction $[K]$. En effet, celle-ci est de taille $(N_x + 1) \times (N_y + 1)$ ce qui peut vite conduire à des matrices qui contiennent plus de 100 000 éléments. Si les matrices sont denses, c'est-à-dire que tous les éléments sont chargés en mémoire, la limite est dépassée si la dimension est importante. Or comme la matrice $[K]$ est une matrice pleine de zéros on peut utiliser les matrices creuses ou *sparse*. Cette formulation particulière des matrices est très utile pour représenter de très grandes matrices qui contiennent beaucoup de zéros et soulage la charge en mémoire. Le module scipy de python possède un sous module *sparse* qui permet d'écrire des matrices creuses et possède tous les outils classiques d'algèbre.

Doc *scipy.sparse* : `https://docs.scipy.org/doc/scipy/reference/sparse.html`

Pour résoudre un problème 2d avec une formulation matricielle classique on peut adopter la structure suivante.

Pour l'étape 1, on forme les matrices $[A]$ et $[B]$, on peut utiliser la fonction *eye* de *numpy* ou de *scipy.sparse*, comme ces matrices sont relativement petites par rapport à $[K]$ cela n'a pas d'importance. Ensuite, on peut utiliser le produit de Kronecker de la bibliothèque *scipy.sparse*. Avant de pouvoir prendre en compte les conditions aux limites, il faut d'abords convertir la matrice $[K]$ en un autre type de matrice creuse. En effet, la fonction *kron* de *scipy* renvoie une matrice de type *csrmatrix*, ce type de matrice n'est pas optimale pour accéder à ces éléments par *slicing*, je conseille de la convertir en *lilmatrix*. Ensuite, on peut résoudre le système en utilisant le module *scipy.sparse.linalg* et la fonction *spsolve*.

Doc *scipy.sparse.linalg* :
`https://docs.scipy.org/doc/scipy/reference/sparse.linalg.html`
Enfin pour afficher les résultats je vous conseille d'utiliser *contour* et *contourf*.
Doc *matplotlib.pyplot.contourf* et *matplotlib.pyplot.contour* :
`https://matplotlib.org/api/_as_gen/matplotlib.pyplot.contourf.html`
`https://matplotlib.org/api/_as_gen/matplotlib.pyplot.contour.html`

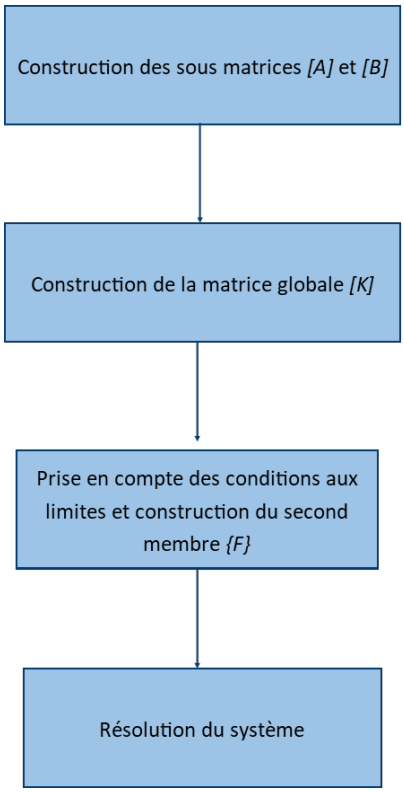

FIGURE 4.3 – Algorithme de résolution avec matrices

4.4 Mise en pratique résolution problème 2D

4.5 Problème 4 : méthode itérative

On se propose d'étudier un problème 2D de l'équation de la chaleur. On étudie une plaque rectangulaire où un côté est entièrement à la température de 283.15K et une partie du côté opposé est lui à 350.15K.

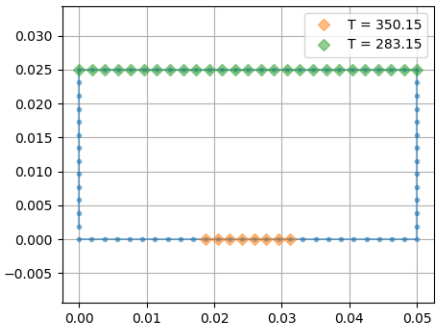

FIGURE 4.4 – Définition du problème 2D

Dans un premier temps on se propose de résoudre le problème avec une résolution itérative.
Données :
$L_x = 0.05m$; $L_y = 0.025m$;
Résoudre avec différents nombres de points ; $\{20, 50, 100\}$
Aussi, on prendra comme tolérance tol=1e-4 et un nombre d'itérations limité à 20 000.

4.6 Solution proposée

La première étape du script est d'initialiser la géométrie, la discrétisation, les paramètres de convergence de la boucle itérative, les conditions aux limites.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors

##DATA

#Discretization
Lx = 50e-3
Ly = Lx/2

Nx = 100
Ny = Nx//2

print('Number of degree of freedom : {}'.format((Nx+1)*(Ny+1)))
dx = Lx/Nx
dy = Ly/Ny

#Loop condition
tolerance = 1e-4
maxIteration  = 20000

#boundaries
hotbound =  [j for j in range(3*(Nx+1)//8,5*(Nx+1)//8+1)]
coldbound = [j for j in range(Nx+1)]

Tcold = 293.15
Thot = 350.15
```

Ensuite, il faut initialiser le tableaux T qui contiendra le champ des températures. Il faut deux tableaux *Tguess* une valeur initiale et *Tnew* une valeur calculée nouvelle ; cela permet de calculer la convergence de la résolution. Enfin, on utilise le *slicing* pour calculer la nouvelle valeur du champ de température à l'intérieur du domaine. Ensuite, il faut garantir le gradient nul sur les frontières et les températures imposées. Pour finir, il faut calculer la convergence de la méthode.

```
T_guess = np.ones((Ny+1,Nx+1))*Tcold
T_new = np.zeros_like(T_guess)

delta = np.linalg.norm((T_guess-T_new))
step = 0
while delta > tolerance and step < maxIteration :
```

```
    T_new[1:-1,1:-1] = 1/(2/dx**2+2/dy**2)*\
                       ((T_guess[1:-1,:-2] + T_guess[1:-1,2:])/dx**2  +\
                        (T_guess[:-2,1:-1] + T_guess[2:,1:-1])/dy**2 )

    #Zeros gradient boundaries
    T_new[0] = T_new[1]
    T_new[-1] = T_new[-2]
    T_new[:,0] = T_new[:,1]
    T_new[:,-1] = T_new[:,-2]

    #Imposed temperature on coldbound
    T_new[-1,coldbound] = Tcold
    #Imposed temperature on hotbound
    T_new[0,hotbound] = Thot

    #update delta and k
    step+=1
    delta = np.max(np.abs(T_guess-T_new))
    T_guess = T_new.copy()
```

Enfin, on peut afficher la solution. Afin que l'affichage des résultats ne soit pas un frein voici une solution possible. *[T]* doit être sous forme d'un tableau $Ny+1$ par $Nx+1$.

```
import matplotlib.pyplot as plt
from matplotlib import colors

x = np.linspace(0,Lx,Nx+1)
y = np.linspace(0,Ly,Ny+1)
X,Y = np.meshgrid(x,y)

fig = plt.figure(0)

Tmax = T_new.max()
Tmin = T_new.min()

ax = fig.add_subplot(111)

ax.contour(X,Y,T_new,
                colors = 'white',
                levels=np.linspace(Tmin,Tmax,20),
                norm=colors.Normalize(vmin=Tmin,vmax=Tmax))

ctf = ax.contourf(X,Y,T_new,
                cmap = 'jet',
                levels=np.linspace(Tmin,Tmax,500),
                norm=colors.Normalize(vmin=Tmin,vmax=Tmax))
fig.colorbar(ctf)
ax.axis('equal')
ax.set_xlim(0,Lx)

plt.show()
plt.grid()
```

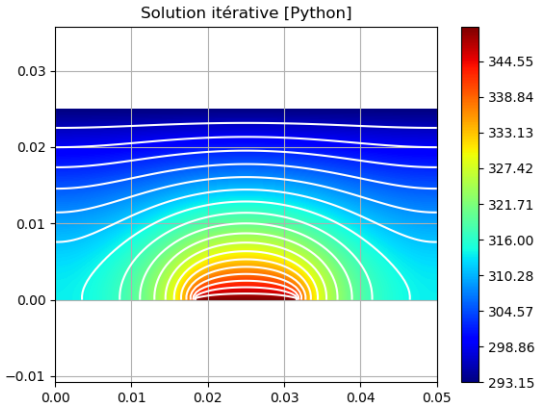

FIGURE 4.5 – Solution itérative 2D

4.7 Problème 5 : méthode matricielle

On se propose de résoudre le problème précédent avec cette fois-ci la méthode matricielle.

4.8 Solution proposée

La première étape concerne les imports qui sont plus importants car on utilise des matrices creuses.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from scipy.sparse import csc_matrix, lil_matrix, kron, eye
from scipy.sparse.linalg import spsolve
```

Ensuite, il faut initialiser la géométrie, la discrétisation ... Cette étape est très similaire à celle de la résolution précédente.

```
#Discretization
Lx = 50e-3
Ly = Lx/2

Nx = 100
Ny = Nx//2
ndof = (Nx+1)*(Ny+1)

print('Number of degree of freedom : {}'.format(ndof))
dx = Lx/Nx
dy = Ly/Ny

hotbound =  [j for j in range(3*(Nx+1)//8,5*(Nx+1)//8+1)]
coldbound = [j+Ny*(Nx+1) for j in range(Nx+1) ]

Tcold = 293.15
Thot = 350.15
```

Ensuite, il faut construire les matrices $[A]$ et $[B]$ puis $[K]$ et enfin le vecteur $\{F\}$.

```
##Compute matrices

A = 1/dx**2*(eye(Nx+1,k=1)+eye(Nx+1,k=-1)-2*eye(Nx+1)) - 2*1/dy**2*eye(Nx+1)
B = 1/dy**2*eye(Nx+1)

K = kron(eye(Ny+1),A) + kron(eye(Ny+1,k=1)+eye(Ny+1,k=-1),B)
K = lil_matrix(K) #lil matrix will be more efficient for slicing
F = np.zeros(ndof)
```

Enfin, il faut prendre en compte les conditions aux frontières.

```
##Boundary conditions

#Zeros gradient
#y = 0
index = np.setdiff1d([j for j in range(Nx+1)],hotbound)
K[index] = 0
K[index,index] = 1/dy
K[index,index+(Nx+1)] = -1/dy

#x = 0
index = np.array([(Nx+1)*i for i in range(Ny+1)])
K[index] = 0
K[index,index] = 1/dx
K[index,index+1] = -1/dx

#x = Lx
index = np.array([(Nx+1)*i + Nx for i in range(Ny+1)])

K[index] = 0
K[index,index] = 1/dx
K[index,index-1] = -1/dx

#imposed temperature
index = hotbound + coldbound
K[index] = 0
K[index,index] = 1
F[index] = [Thot]*len(hotbound) + [Tcold]*len(coldbound)
```

Pour finir, il faut résoudre le système. Pour accélérer la résolution, une conversion de [K] en matrice *cscmatrix* est nécessaire. On peut aussi déterminer la mémoire nécessaire pour stocker la matrice $[K]$. Puis, on résout le système avec la fonction *spsolve* et on restructure le vecteur température en un tableau 2D.
La résolution du système linéaire reste assez simple car le problème est 2D et la physique est simple, mais on peut utiliser d'autres solveurs itératifs plus souvent utilisés pour ces problèmes comme l'algorithme *GMRES*.

```
##Solve
#transform in csc_matrix
K = csc_matrix(K)

memory = K.indices.nbytes + K.indptr.nbytes + K.data.nbytes
print("Dimension of sparse matrix [K] :",K.shape,"; memory :",
      memory,"bytes")
```

```
T = spsolve(K,F)
T = T.reshape((Ny+1,Nx+1))
```

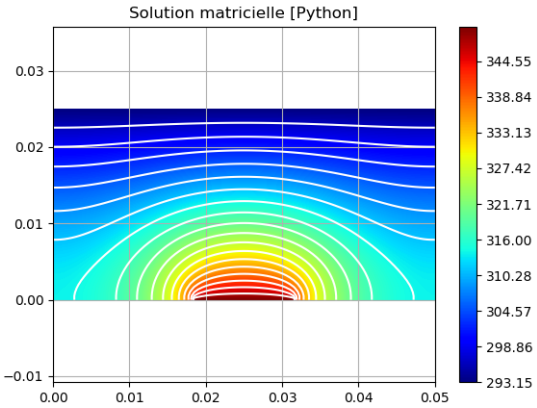

FIGURE 4.6 – Solution itérative 2D

Conclusion

La méthode des différences finies est une méthode simple de discrétisation des équations aux dérivées partielles et des équations aux limites. Elle s'appuie sur la formulation forte de l'équation à laquelle on vient approcher les termes de dérivées spatiales par un développement de Taylor. Cette méthode apparaît comme étant la plus simple à comprendre et à mettre en oeuvre car elle s'appuie directement sur l'équation d'équilibre du problème. A la différence des volumes finis ou des éléments finis, elle n'a pas besoin d'intégrer l'équation. En outre, les conditions aux limites peuvent être plus difficiles à exprimer surtout si le problème est non linéaire. Dans ce document seul des problèmes linéaires ont été abordés. Nous verrons comment imposer des conditions aux limites avec une autre méthode que celle abordée ici. La méthode des différences finies est extensible aux problèmes 2D et 3D, mais les géométries sont souvent trop restreintes. Les différences finies sont difficilement applicables avec des géométrie plus complexes. Mais la méthode est très efficace en 1D, elle permet d'aborder une grande quantité de problème avec une implémentation facilité. Ce document n'a pas abordé de problème très concret mais les différences finies ne se limitent pas à des applications académiques. Dans le cadre de l'application de conditions aux limites dans un problème non linéaire nous aborderons un sujet de thermique portant sur de le dimensionnement d'un groupe Evaporateur-Surchauffeur-Economiseur et la modélisation thermique d'un matériau non linéaire.

Table des figures

Liste des tableaux

Bibliographie

[1] Ch ampagnat Nicolas. *Différences finies et analyse numérique matricielle.* PhD thesis, 2010.

[2] Cappelaere Bernard Guinot Vincent. *Methodes Numeriques Appliquees Resolution numerique.* PhD thesis, Polytech'Montpellier, 2005.

[3] B. Lucquin. *Equations aux dérivées partielles et leurs approximations.* PhD thesis, Ellipse, 2004.

[4] S.Gounand. *Introduction à la méthode des éléments finis en mécanique des fluides incompressibles.* PhD thesis, Polytech'Montpellier, 2012.

[5] A. Oussalah T. Aubidert. *Informatique : Programmation et calcul scientifique en Python et Scilab.* Ellipse, 2014.