Résolution numérique de l'équation de la diffusion thermique (aussi appelée « équation de la chaleur »).

Travail inspiré fortement, à l'origine, de python-prepa.github.io/edp_chaleur.html.

- On se limite à un problème 2D : coordonnées spatiales (x,y). Le matériau a des propriétés supposées uniformes : masse volumique ρ, capacité thermique massique c, conductivité thermique λ, diffusivité thermique $K = \frac{\lambda}{\rho c}$. La conduction thermique est le seul mode de transfert thermique présent, par hypothèse.
- Le domaine spatial étudié sera rectangulaire : $x \in [0, LX]$, $y \in [0, LY]$.

L'équation différentielle aux dérivées partielles à résoudre.

L'équation différentielle aux dérivées partielles qui va être résolue est la suivante :

$$\frac{\partial T}{\partial t} = K\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) + S \quad \forall x, y, t \qquad (E)$$

- On recherche la température $T(x,y,t)$.
- $S(x,y,t)$ est l'éventuel terme de « source » : $S = \frac{P_V}{\rho c}$ avec $P_V(x,y,t)$ la puissance thermique volumique *algébrique* reçue par le milieu.

Conditions initiales et aux limites :

Il convient de fournir également :

- la situation thermique initiale $T(x, y, t=0) \quad \forall x, y$
- les conditions aux limites spatiales sur chacune des quatre frontières du domaine rectangulaire : $x = 0$, $x = LX$, $y = 0$, $y = LY$. Ces conditions peuvent être de trois types : température fixée, OU densité de flux thermique fixée (cas particulier : frontière thermiquement isolée), OU loi type loi de Newton pour un transfert conducto-convectif avec un fluide. Les conditions peuvent être différentes sur des frontières différentes.

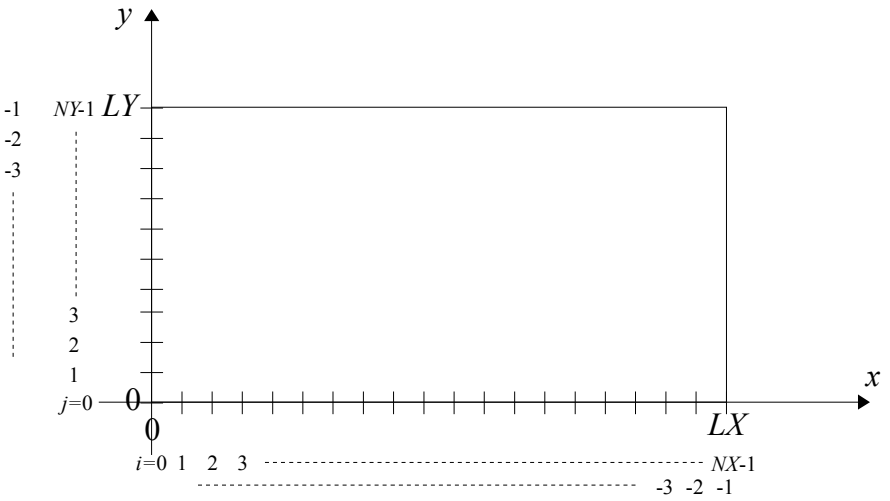
Discrétisation spatiale :

NX nombre de points dans la direction x : $i = 0, 1, 2, \ldots, NX-1$; pas $\mathrm{d}x = LX/(NX-1)$

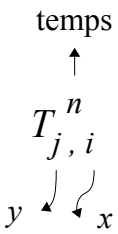
NY nombre de points dans la direction y : $j = 0, 1, 2, \ldots, NY-1$; pas $\mathrm{d}y = LY/(NY-1)$

Discrétisation temporelle :

on va résoudre pour t allant de 0 à *Time*, avec *NT* pas : $\mathrm{d}t = \frac{Time}{NT}$.



Résolution : on va résoudre par un schéma explicite en temps, en discrétisant (E) : on note



On calcule, en tout point qui ne se trouve pas sur une frontière, T à la date n+1 en fonction de T au même point et aux points environnants à la date n (= schéma explicite en temps).

L'équation discrétisée s'écrit alors :

$$\frac{T_{j,i}^{n+1}-T_{j,i}^{n}}{\mathrm{d}t}=S_{j,i}^{n}+\mathrm{K}\left[\frac{\frac{T_{j,i+1}^{n}-T_{j,i}^{n}}{\mathrm{d}x}-\frac{T_{j,i}^{n}-T_{j,i-1}^{n}}{\mathrm{d}x}}{\mathrm{d}x}+\frac{\frac{T_{j+1,i}^{n}-T_{j,i}^{n}}{\mathrm{d}y}-\frac{T_{j,i}^{n}-T_{j-1,i}^{n}}{\mathrm{d}y}}{\mathrm{d}y}\right]$$

c'est-à-dire $$T_{j,i}^{n+1}=T_{j,i}^{n}+S_{j,i}^{n}\,\mathrm{d}t+\mathrm{K}\,\mathrm{d}t\left(\frac{T_{j,i+1}^{n}-2T_{j,i}^{n}+T_{j,i-1}^{n}}{(\mathrm{d}x)^2}+\frac{T_{j+1,i}^{n}-2T_{j,i}^{n}+T_{j-1,i}^{n}}{(\mathrm{d}y)^2}\right)$$

Ce n'est probablement pas ce qu'il y a de mieux en terme de performance, (et de stabilité numérique, voir le site internet cité en début de document), mais c'est facile à comprendre et très simple à programmer.

L'équation discrétisée écrite juste au-dessus ne peut bien évidemment être appliquée qu'aux points qui ne sont pas sur une des frontières du domaine ; il faudra traiter ces points séparément dans une étape ultérieure, en tenant compte des conditions aux limites choisies. On va donc l'appliquer pour $i=1,2,3,\ldots,NX-2$ et $j=1,2,3,\ldots,NY-2$ sous la forme très « compacte » :

```
RHS = S[1:-1,1:-1]*dt + K * dt * ( (T[1:-1,2:] - 2 * T[1:-1,1:-1] + T[1:-1,:-2]) / (dx ** 2) + (T[2:,1:-1] - 2 * T[1:-1,1:-1] + T[:-2,1:-1]) / (dy ** 2) )
```

et `T[1:-1,1:-1] += RHS` : on calcule en tous ces points la température au pas n+1 en fonction de la température au pas n.

(RHS pour « Right Hand Side », « terme de droite »).

Il reste alors à tenir compte des conditions aux limites choisies pour déterminer la température, à la date $n + 1$, sur tous les points situés aux frontières du domaine.

Traitement des conditions aux limites.

Elles peuvent donc être, sur chaque frontière, de trois types :

1) Température fixée.

C'est bien sûr la plus simple à programmer (d'un point de vue pratique, il convient de mettre le système en contact sur la frontière considérée avec un thermostat) : on connaît la température de la frontière. Par souci de simplicité, on se limitera, en plus, à des températures sur les frontières indépendantes du temps, et en fait il n'y a donc rien à programmer, puisque ces températures ont été rentrées avec les conditions initiales.

2) Densité de flux thermique fixée.

Traitons pour exemple le cas de la frontière gauche :

$\vec{j}_Q(x=0,y,t)=j_G\vec{e}_x \quad \forall y,t$ (on suppose, là aussi pour simplifier, un flux uniforme sur toute la frontière, et indépendant du temps) (en fait c'est $\vec{j}_Q\cdot\vec{n}$ qui est fixé)

Avec la loi de Fourier, ceci devient : $\frac{\partial T}{\partial x}(x=0,y,t)=-\frac{j_G}{\lambda} \quad \forall y,t$.

En version discrétisée : $T[:,1]-T[:,0]=-\frac{j_G}{\lambda}\mathrm{d}x$; en fait c'est $T[:,0]$ que l'on cherche, et l'on utilisera donc : $T[:,0]=T[:,1]+\frac{j_G}{\lambda}\mathrm{d}x$.

On aura ainsi, pour les quatre frontières :

	Relation discrétisée	Remarque (définition du flux, et de son sens)
bord gauche	$T[:,0]=T[:,1]+\frac{j_G}{\lambda}\mathrm{d}x$	$\vec{j}_Q(x=0,y,t)=j_G\vec{e}_x \quad \forall y,t$
bord droit	$T[:,-1]=T[:,-2]-\frac{j_D}{\lambda}\mathrm{d}x$	$\vec{j}_Q(x=LX,y,t)=j_D\vec{e}_x \quad \forall y,t$
bord bas	$T[0,:]=T[1,:]+\frac{j_B}{\lambda}\mathrm{d}y$	$\vec{j}_Q(x,y=0,t)=j_B\vec{e}_y \quad \forall x,t$
bord haut	$T[-1,:]=T[-2,:]-\frac{j_H}{\lambda}\mathrm{d}y$	$\vec{j}_Q(x,y=LY,t)=j_H\vec{e}_y \quad \forall x,t$

3) Loi type loi de Newton.

...à faire...

```
"""Équation de la diffusion thermique à deux dimensions
Resolution numérique utilisant une méthode de différences finies.
Le domaine spatial d'intégration est rectangulaire."""

import numpy as np
import matplotlib.pyplot as plt

# de vieux "trucs" qui ne fonctionnent plus....
#if 'qt' in plt.get_backend().lower():
#    try:
#        from PyQt4 import QtGui
#    except ImportError:
#       from PySide import QtGui

from PyQt5 import QtWidgets

# Paramètres physiques
K = 0.5    # Diffusivité thermique
lambdat =0.1 # Conductivité thermique
Lx = 1.0   # Dimension du domaine en x
Ly = 2.0   # Dimension du domaine en y
Time = 0.4 # Durée totale d'intégration

# Paramètres numériques

NT = 8000      #Nombre de pas temporels
NX = 100       #Nombre de points dans la direction x
NY = 100       #Nombre de points dans la direction y
dt = Time/NT   #Pas temporel élémentaire (temps)
dx = Lx/(NX-1) #Pas de la grille dans la direction x
dy = Ly/(NY-1) #Pas de la grille dans la direction y

xx = np.linspace(0,Lx,NX)
yy = np.linspace(0,Ly,NY)

plt.ion()
plt.figure()

### MAIN PROGRAM ###

# Terme de source ; ici il est mis à zéro
S = np.zeros((NY,NX))

# Température initiale
T = np.zeros((NY,NX))
# RHS = Right Hand Side = Membre de droite de l'équation
# RHS = np.zeros((NX,NY))

# Initialisation du tableau jQ ; on l'initialise à zéro.
# En fait seuls les "bords" de ce tableau vont servir -> à améliorer
jQ = np.zeros((NY,NX))

# Conditions aux limites :
# elles peuvent être, sur chaque frontière
# (et même en chaque point de chaque frontière, mais on ne poussera pas ici le raffinement
# à ce point), de deux types différents. Soit on fixe la
# température, soit on fixe la densité de flux thermique.

# On initialise des grandeurs pour les deux conditions (conditions
# bien évidemment incompatibles). On n'utilisera pour chaque frontière qu'une
# seule des deux conditions ; le choix sera fait plus loin dans la boucle principale.
```

```
# Ici nous allons fixer la température T sur les bords ; ces valeurs ne seront pas modifiées par la suite, ce qui correspond
# à une condition de température fixée et constante sur les frontières (contact avec un thermostat)

# bord gauche
T[:,0] = np.zeros((NY))
# bord haut
T[-1,:] = np.zeros((1,NX))
# bord droit
T[:,-1] = np.zeros((NY))
# bord bas
T[0,:] = np.zeros((1,NX))

# Ici nous allons fixer la densité de flux thermique jQ sur les bords
# ces valeurs ne seront pas modifiées par la suite, ce qui correspond
# à un flux fixé et constant sur les frontières. Les valeurs ont déjà été fixées à zéro
# plus haut (ce qui correspond à une face isolée thermiquement) ; ces lignes ne sont
# donc vraiment utiles que si on fixe le flux à une valeur différente de zéro.

# bord gauche
jQ[:,0] = np.zeros((NY))
# bord haut
jQ[-1,:] = np.zeros((1,NX))
# bord droit
jQ[:,-1] = np.zeros((NY))
# bord bas
jQ[0,:] = np.ones((1,NX))

# On peut ici éventuellement rajouter une zone "source" (siège de réactions exo-
# ou endo-thermiques
# S[30:35,30:35] = 100*np.ones((5,5))

# Boucle principale
for n in range(0,NT):
  RHS = S[1:-1,1:-1]*dt + K*dt*( (T[2:,1:-1]-2*T[1:-1,1:-1]+T[:-2,1:-1])/(dy**2)  \
                + (T[1:-1,2:]-2*T[1:-1,1:-1]+T[1:-1,:-2])/(dx**2) )

  T[1:-1,1:-1] += RHS

# C'est dans les quatre lignes qui suivent que l'on choisit la condition aux limites
# finalement retenue pour chacune des quatre frontières.
# Si la ligne correspondante est commentée, c'est la température qui est fixée sur la
# frontière.
# Si la ligne n'est pas commentée, c'est le flux thermique qui est fixé
# (et la température de la frontière est alors modifiée).

#   T[:,0] = T[:,1]+dx/lambdat*jQ[:,0] # bord gauche
  T[-1,:] = T[-2,:]-dy/lambdat*jQ[-1,:] # bord haut
  T[:,-1] = T[:,-2]-dx/lambdat*jQ[:,-1] # bord droit
  T[0,:] = T[1,:]+dy/lambdat*jQ[0,:] # bord bas

#Plot every 100 time steps
  if (n%100 == 0):
    plotlabel = "t = %1.2f" %(n * dt)
# vmin=0 et vamx=1 sont absolument nécessaires pour "calibrer" l'échelle des couleurs,
# sinon elle va de la température min à la température max,et ce, à chaque date t fixée :
# si par exemple, on fixe T initiale à 0 partout et les 4 bords à 1, on ne verra JAMAIS
# le centre se "réchauffer" si on ne met pas cette option.
# Par contre cette option poserait un  problème si on devait aboutir à des températures qui sortent
# de cet intervalle (ce qui est possible si on fixe non pas T sur un bord mais jQ) :
# à revoir donc, dans ce cas
    plt.pcolormesh(xx,yy,T, shading='gouraud',cmap='rainbow',vmin=0,vmax=1)
    plt.title(plotlabel)
    plt.axis('image')
    plt.draw()
    if 'qt' in plt.get_backend().lower():
      QtWidgets.QApplication.processEvents()

plt.show()
```