TP 1 : Équation de Laplace

Cours de modélisation numérique

24 février 2023

Introduction

Le but de cette série est d'introduire l'utilisation de `NumPy` avec l'équation de Laplace en simulant l'équation de la chaleur sur un domaine carré. Les librairies `NumPy` (calcul) et `Matplotlib` (graphiques) doivent être installées.

Equation de la chaleur

L'équation de la chaleur décrit l'évolution spatiale et temporelle de la température et prend la forme :

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T, \tag{1}$$

Avec T une grandeur scalaire (par exemple la température) et α le coefficient de diffusion. Dans le cas où la température aux bords est imposée constante, la chaleur dans le domaine tend vers un état stationnaire. Dans un tel cas, la dérivée temporelle est nécessairement nulle, et on résoud donc l'équation dite de Laplace (dans le cas bidimensionnel) :

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0. \tag{2}$$

Nous proposons de résoudre l'équation 2 ci-dessus numériquement, en discrétisant le domaine en un nombre donné de cellules avec $T(i, j)$ la température sur la cellule en coordonnée (i, j), espacées dans les deux directions d'un pas Δx :

$$\frac{T(i+1,j) - 2T(i,j) + T(i-1,j)}{\Delta x^2} + \frac{T(i,j+1) - 2T(i,j) + T(i,j+1)}{\Delta x^2} = 0, \tag{3}$$

en appliquant un développement de Taylor.

Cela nous donne le schéma numérique suivant :

$$\boxed{T(i,j)_{k+1} = \frac{1}{4}\left[T(i+1,j)_k + T(i-1,j)_k + T(i,j+1)_k + T(i,j-1)_k\right]}, \tag{4}$$

où k est le numéro de l'étape ou *itération*. Après un nombre suffisamment grand d'itérations, si la différence de valeur entre $T_{k+1} = T_k$ est suffisamment faible ($|T_{k+1} - T_k| < \epsilon$), on dit que le schéma *a convergé*. Si le système est bien défini (équations correctes) et résolu (maillage spatial fin), le schéma numérique converge vers l'expression analytique.

Utilisation des matrices dans `NumPy`

Afin de pouvoir résoudre l'équation de la chaleur sans utiliser explicitement de boucle `for`, nous vous présentons quelques opérations de base qu'il est possible de faire sur les matrices avec `NumPy`. Les habitués de Matlab trouverons sur
`https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html` une introduction

adaptée à NumPy, qui présente des similarités avec Matlab. Voici quelques exemples que vous pouvez tester vous-même :

```
import numpy as np

tableau_1d = np.array([2., 3., 5., 7]) #  crée un tableau numpy array contenant 4 éléments

tableau_2d = np.array([[1,2,3],[4,5,6]], dtype=float) # matrice de 2 lignes et 3 colonnes
# (on a spécifié le type float pour les valeurs de tableau_2d)
print(tableau_2d.shape) # écrit la dimension de la matrice : (2,3)
print(tableau_2d.size) # écrit le nombre d'éléments de la matrice : 6

my_matrix = np.zeros((12,32)) # crée une matrice de taille 12x32 remplie de zeros
other_one = np.ones((2,3,2)) # crée une matrice de taille 2x3x2 remplie de 1.
random_matrix = np.random.random((3,3)) # remplit de valeurs aléatoires entre 0 et 1
```

NumPy permet un formalisme proche de la notation algébrique à laquelle nous sommes habitués. Les opérations d'additions et de multiplication, par exemple, se font ainsi :

```
import numpy as np

M = np.ones((3,3)) # crée une matrice de taille 3x3 remplie de 1.
N = M * 12 # N est une matrice 3x3 remplie de 12...
N /= 3 # ...et maintenant remplie de 4...
N += 8.5 #  ...et maintenant chacun de ses éléments est incrémenté de 8.5

# si une opération se fait entre deux matrices, alors elle est faite élément par élément:
A = np.array([1,2,3])
B = np.array([3,4,5])
print(A+B) # [4,6,8]
print(A*B) # [3,8,15]
```

Enfin, on peut accéder aux éléments d'une matrice via des "slices" :

```
import numpy as np
a = np.array([[1,2,3,4],[6,6,7,8]])
b = a[1:,2:] # on garde tout à partir de la ligne 1 et colonne 2 de a
c = a[0:2,1:3] # on ne garde que les lignes 0,1, et les colonnes 1,2 de a
```

NumPy est une librairie largement utilisée et contenant de nombreuses autres fonctions.

Implémentation

Pour l'implémentation, nous considèrerons un domaine de taille quelconque WxH, avec les conditions aux bords $T(0,j) = T(i,0) = 1 \; \forall \; i,j$ et $T(W,j) = T(i,H) = 0 \; \forall \; i,j$. Le domaine sera représenté numériquement, vous l'aurez compris, par une matrice de taille WxH, qui contiendra la valeur de la température (à une échelle arbitraire).

Pour que le code soit performant, il est toujours bon d'éviter de faire des boucles spatiales. Ainsi, pour accéder aux voisins d'une cellule, il est préférable d'utiliser la fonction `roll` de NumPy :

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a)
# decalle periodiquement les éléments de 1 indice selon l'axe 0 (lignes)
print(np.roll(a,1,axis=0))
# decalle periodiquement les éléments de 1 indice selon l'axe 1 (colonnes)
print(np.roll(a,1,axis=1))
```

Pour terminer, vous pouvez vous servir de `Matplotlib` pour visualiser la matrice à n'importe quel moment :

```
import matplotlib.pyplot as plt
W,H = 100,100
a = np.zeros((W,H))
# ici, pour des raisons didactiques, nous remplissons une matrice "a la main"
for i in range(W):
    for j in range(H):
        a[i,j] = i**2 + j**2
plt.clf() # clear figure
plt.imshow(a) # dessine a
plt.show() # ouvre une fenetre pour montrer le graph courant
plt.savefig("my_graph.png") # sauve la figure sur le disque

plt.clf()
plt.plot(a[:,H//2]) # dessine la courbe de a, pour tous les i, et pour j=H/2
plt.show()
```

Voici un début de code sur lequel vous pouvez baser votre programme :

```
import os
import numpy as np
import matplotlib.pyplot as plt


#  applique les conditions de bord
#  TODO Imposer les bonnes valeurs de temperature aux bords du domaine
def apply_boundaries(d):
    #  bord gauche
    #  bord droit
    #  bord haut
    #  bord bas

#  calcule la moyenne des voisins
#  TODO fonction d'evolution à implementer
def mean_neighbors(d):
        #  à implémenter vous-mêmes, en utilisant le schéma numérique ci-dessus

if __name__ == "__main__":

    W, H = 250, 250     #  domaine carre
    iter_max = 50000+1   #  nombre d'iterations maximal pour la simulation

    domain = np.ones((W, H)) * 0.5  #  cree une matrice remplie de 0.5
    apply_boundaries(domain)

    #  cree le dossier qui va contenir les fichiers de sortie
    os.system('mkdir -p ./domains_Laplace')

    # boucle de resolution
    for it in range(iter_max):
        domain = mean_neighbors(domain)
        apply_boundaries(domain)
        if it % 500 == 0:
```

```
        print("Iteration : "+str(it))
        plt.imshow(domain, origin='lower')
        plt.savefig('domains_Laplace/domain_'+str(it).zfill(4)+'.png')


#  TODO dessiner la courbe de temperature le long de la largeur du domaine,
#  a la mi-hauteur
#  plt.savefig('domains_Laplace/temperature.png')
```