Résolution de l'équation de Laplace à 2 dimensions

Au programme de MPI : à l'aide d'un langage de programmation résoudre numériquement l'équation de Laplace à une ou deux dimensions, les conditions aux limites étant fixées.

Une fonction f suit l'équation de Laplace si son laplacien est nul, $\Delta f = 0$.

Cette équation est rencontrée dans de nombreux domaines de la physique : en électrostatique pour le potentiel dans une zone vide de charges, en conduction thermique pour la température en régime stationnaire, en gravitation pour le potentiel gravitationnel dans une zone vide de masses, mais aussi en hydrodynamique, diffusion de particules, déformations d'une membrane, etc.

I. Algorithme de résolution

L'équation de Laplace est une équation aux dérivées partielles dite elliptique. Ces équations servent à décrire un problème aux limites (« boundary value problem » en anglais) où les valeurs de la fonction dans le domaine doivent être déduites de celles sur les bords du domaine, sans évolution temporelle.

Nous nous intéressons ici à une méthode numérique par différences finies : la fonction f inconnue est discrétisée aux points (i,j) d'un maillage (cf. figure ci-dessous). Il faut alors également discrétiser l'équation à résoudre, ce qui mène à un système d'équations portant sur les $f(x_i, y_j)$ à résoudre.

Définition de la grille et du domaine

On se restreint au cas à deux dimensions[1]. L'espace étudié est alors un rectangle de dimensions $L_x \times L_y$, représenté par une grille de taille $N_x \times N_y$.

Pas spatial δ : Distance entre 2 points de la grille , soit

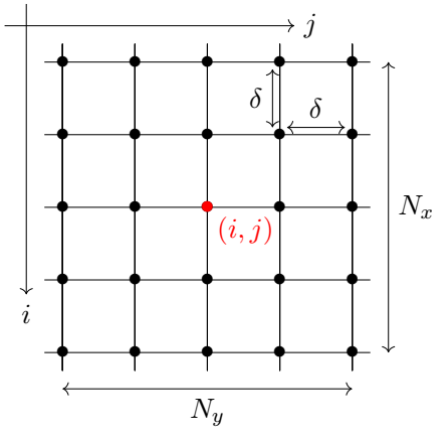
$$L_x = (N_x - 1)\delta \text{ et } L_y = (N_y - 1)\delta.$$

On note (x_i, y_j) les coordonnées du point (i,j) de la grille, avec :

$i \in [0\,;\, N_x - 1]$ et $j \in [0\,;\, N_y - 1]$.

Une cellule correspond à un carré de côté δ délimité par quatre points de la grille.

La grandeur physique f tq $\Delta f = 0$ est représentée par un tableau f avec f[i,j] donnant la valeur de $f(x_i, x_j)$ au point (i,j) de la grille.



L'équation de Laplace admet une unique solution dans un domaine $\mathcal{D}$ de l'espace si on connaît en tout point du bord de $\mathcal{D}$ la valeur de f (condition dite de Dirichlet).

Les bords du domaine peuvent simplement être le cadre de la grille, mais peuvent aussi inclure d'autres points de la grille pouvant représenter les armatures d'un condensateur, un conducteur métallique à un potentiel V_0 fixe, une surface à une température imposée, etc.

Afin de tenir compte de ces conditions aux limites, on utilise un tableau B de taille $N_x \times N_y$, tel que :

- ▷ Si B[i,j]=1, alors la case (i,j) de la grille appartient au bord du domaine, et la valeur f[i,j]est imposée par une condition de Dirichlet ;
- ▷ Si B[i,j]=0, alors la case (i,j) n'appartient pas au bord du domaine ; sa valeur est déterminée par résolution de $\Delta f = 0$ (voir suite).

II. Discrétisation de l'équation

La fonction f étant connue uniquement aux points de la grille, il est nécessaire de discrétiser l'opérateur laplacien, ici en coordonnées cartésiennes et à deux dimensions.

Comme dans l'équation de la diffusion thermique, on utilise une approximation des dérivées secondes selon :

[1] Le cas à 3D ne pose pas de difficultés supplémentaires, mais requiert des temps de calculs plus longs et des méthodes de visualisation des résultats plus complexes.

$$\left(\frac{\partial^2 f}{\partial x^2}\right)_y (x_i, y_i) \simeq \frac{f(x_i+\delta, y_i) + f(x_i-\delta, y_i) - 2f(x_i, y_i)}{\delta^2}$$

On procède de même dans la direction y, et on obtient ainsi, en exploitant les notations associées au tableau :

$$\Delta f[i,j] \simeq \frac{f[i+1,j] + f[i-1,j] + f[i,j+1] + f[i,j-1] - 4f[i,j]}{\delta^2}$$

Avec $\Delta f = 0$, on a alors $\forall\ (i,j)$ dans le domaine et à l'ordre deux en δ :

$$f[i,j] \simeq \frac{f[i+1,j] + f[i-1,j] + f[i,j+1] + f[i,j-1] - 4f[i,j]}{4} \qquad (1)$$

III. Méthode itérative de Jacobi

La méthode de Jacobi consiste à appliquer l'équation (1) pour construire par itérations les valeurs de f sur la grille, et ceci jusqu'à ce que les valeurs de f n'évoluent quasiment plus.

L'algorithme est le suivant :

- **Initialisation** :

On crée la matrice B et on l'initialise avec des 0 ou des 1 pour décrire les bords du domaine souhaité.

On crée la matrice f et on l'initialise avec les valeurs voulues sur les bords du domaine et avec la valeur initiale ailleurs.

- **Itérations** :

Une itération consiste à effectuer, pour tout point (i,j) qui n'appartient pas à un bord, le calcul d'une nouvelle valeur de f selon l'équation (1). Ainsi f à l'itération k est obtenue en fonction de f à l'itération $k-1$ via la relation

$$f_k[i,j] \simeq \frac{f_{k-1}[i+1,j] + f_{k-1}[i-1,j] + f_{k-1}[i,j+1] + f_{k-1}[i,j-1] - 4f_{k-1}[i,j]}{4}$$

En python ceci nécessite de faire une copie de la matrice f

- **Critère de terminaison** :

On stoppe la simulation lorsque les itérations ne font quasiment plus évoluer les valeurs de f.

On définit pour cela l'écart e entre la nouvelle valeur de f et l'ancienne en moyenne sur toute la grille (simple différence ou écart quadratique) :

$$e = \sqrt{\frac{1}{N_x N_y}\sum_{i,j}\left[f_k(x_i,y_i) - f_{k-1}(x_i,y_j)\right]^2} \qquad (2)$$

Les itérations sont arrêtées lorsque e devient inférieur à une valeur ε fixée à l'avance appelée critère de convergence.

```
def sol_iter_2D():
    V2D=np.zeros((N2Dx,N2Dy))       #  on définit un tableau de dimensions N2D sur N2D
    for i in range(1,N2Dx-1):
        V2D[i][N2Dy-1]=1     # pour tout i, les bords correspondant à j = N2D-1 sont dé
                             #(conditions aux limites)
    for i in range(1,N2Dx-1):
        for j in range(1,N2Dy-1):
            V2D[i][j]=np.random.rand()
  #on remplit le tableau V au hasard hormis pour les conditions limite,
  #ce qui permet (bizarrement ??) une convergence plus rapide

  #définition du critère de convergence
   erreur=1
   epsilon=1E-4      #écart seuil pour lequel on considère la convergence atteinte
   t0=time.time()
   while (erreur>epsilon):
        V2Dold=np.copy(V2D)       # crée une copie du tableau avec les valeurs avant la
        for i in range(1,N2Dx-1):
            for j in range(1,N2Dy-1):
   #equation discrète issue de Laplacien = 0
                V2D[i][j]=(V2Dold[i+1][j]+V2Dold[i-1][j]+V2Dold[i][j+1]+V2Dold[i][j-1])
  #calul de l'erreur :
   #renvoie la valeur max du tableau correspondant aux écarts de valeurs entre 2 itéra
        erreur=np.amax(np.abs(V2D-V2Dold))
```