Synthèse TP Poisson

Justin Kim et Raphaelle Ezeghian

February 4, 2023

1 Equation de Poisson et maillage de l'espace

Au cours de ce TP, on cherche à résoudre l'équation de Poisson :

$$\frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y) = f(x,y)$$

avec f une fonction source.

Pour cela, on résoud par la méthode des différences finies, en définissant un maillage de l'espace. On limite ce maillage à une portion du plan xy. On limite le plan entre 0 et I selon x et entre 0 et J selon y.

$$\begin{cases} x_i = x_0 + i\Delta x & 0 \leq x < I \\ y_i = y_0 + j\Delta x & 0 \leq x < J \end{cases}$$

Dans notre cas, nous allons nous intérésser à une fonction de Poisson en particulier : celle d'un potentiel électrostatique en présence de charge.

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = \frac{\rho(x,y)}{\epsilon_0}$$

ρ = densité volumique de charges
ϵ_0 = permitivité relative du vide
On choisit $I = J = 1$ et on fait un maillage à 65 points.

2 Développement de Taylor de l'équation

On développe à l'ordre 2, l'équation de Poisson :

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+\Delta x,y)+u(x-\Delta x,y)-2u(x,y)}{\Delta x^2} \\ \frac{\partial^2 u}{\partial y^2} \approx \frac{u(x,y+\Delta y)+u(x,y-\Delta y)-2u(x,y)}{\Delta y^2} \end{cases}$$

En notation discrète l'équation devient :

$$\frac{1}{\Delta x^2}(u_{i+1,j} + u_{i-1,j} - 2u_{i,j}) + \frac{1}{\Delta y^2}(u_{i,j+1} + u_{i,j-1} - 2u_{i,j}) = \rho_{i,j}$$

On réindèxe la matrice pour avoir une matrice $1D$

$$\begin{cases} 0 \leq x < I \\ 0 \leq x < J \end{cases}$$

$$\Longrightarrow k = iJ + j,\ 0 \leq k \leq IJ$$

On a alors une nouvelle fomulation de l'équation de Poisson :

$$\frac{1}{\Delta x^2}(u_{k+J+1} + u_{k-J-1}) + \frac{1}{\Delta y^2}(u_{k+1} + u_{k-1}) - 2(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2})u_k = \rho_k$$

3 Conditions aux limites de Dirichlet

Pour résoudre l'équation, on introduit des conditions aux limites. Dans notre cas, on impose des conditions de Dirichlet. La solution est nulle en dehors du domaine.

- Sur le côté gauche ($i = 0$):

$$\frac{1}{\Delta x^2}(u_{1,j} - 2u_{0,j}) + \frac{1}{\Delta y^2}(u_{0,j+1} + u_{0,j-1} - 2u_{0,j}) = \rho_{0,j}$$

- Sur le côté droit ($i = I - 1$):

$$\frac{1}{\Delta x^2}(u_{I-2,j} - 2u_{I-1,j}) + \frac{1}{\Delta y^2}(u_{I-1,j+1} + u_{I-1,j-1} - 2u_{iI-1j}) = \rho_{I-1,j}$$

- De même en bas ($j = 0$) et en haut ($j = J - 1$)

Ici, on choisit d'imposer les conditions de Dirichlet pour fixer les conditions au bord. On aurait pu imposer les conditions de Neumann, mais celles-ci sont généralement imposées lorsque l'on a des situations avec des conditions de bords dites "infinis" ce qui n'est pas le cas dans l'exemple ici

4 Lignes de charge

La densité volumique de charges dépend des coordonées. On a :

- Une ligne de charge positive pour $y = 0.4$ et $0.25 \leq x \leq 0.75$ - Une ligne de charge négative pour $y = 0.6$ et $0.25 \leq x \leq 0.75$

5 Méthodes de résolution de l'équation

Dans ce TP, nous avons décidé de tester et comparer 4 méthodes différentes de résolutions de systèmes linéaires : la méthode du Pivot de Gauss, la méthode utilisant la fonction linalg du module numpy, la méthode utilisant des matrices quasi creuses utilisant le module Sparse et la méthode de Gauss Seidel sous forme de sur-relaxation.

Dans le code, nous dédions tout d'abord une partie à l'implémentation du pivot de Gauss (qui est assez longue) qui se fonde sur des opérations linéaires appliqués à la matrice représentant le système d'équations, permettant de résoudre ce dit système. Puis, on utilise la fonction linalg du module numpy dont on ne connaît pas vraiment la façon dont elle a été codée mais permet de résoudre des systèmes d'équations.

On implémente également la méthode de sur relaxation. Nous avons choisi de ne pas faire Gauss Seidel (cas où le w = 1) mais directement implémenter le cas général qui est la méthode de sur relaxation. On utilise ici un critère de convergence relatif qui va nous servir comme condition d'arrêt de la boucle d'itération par rapport à une précision souhaitée. On utilise des suites pour résoudre des systèmes linéaires de types Ax=b en écrivant A = D -E-F avec D une matrice diagonale, E une matrice triangulaire supérieur et F inférieur. On aboutit après calcul de Ax = b à la formule donnée par le pdf du github en injectant le paramètre w dans l'équation. Pour l'équation de Poisson avec les conditions de Dirichlet, il existe un omega optimal mathématique qui est :

$$\omega_{opti} = \frac{2}{1 + sin(\pi * h)} \tag{1}$$

Finalement, on implémente la méthode utilisant le module sparse qui va être très pratique dans notre cas. Dans un premier temps, on travaille sur une seule dimension et ensuite on l'étend en 2D au maillage. Ici comme on manipule des matrices pleines de 0 (matrices quasi creuses), on utilise la méthode sparse qui consiste à remplir les matrices avec seulement les endroits où l'élement est non nul. Attention, si il y a pas beaucoup de zero cette méthode est très longue car on remplit un par un la matrice mais dans notre cas on manipule des matrices creuses (la matrice source est quasimment vide etc...) donc on peut utiliser cette méthode.
Dans le code, nous réimplémentons toutes les matrices sous le format sparse, notamment celle du système. Nous avons décidé de revoir la façon dont on fabrique le système d'équations en implémentant via la méthode sparse les différentes dérivées partielles.

6 Analyse et comparaison des résultats

Le temps de calcul le plus court est celui de la méthode Sparse.

On se rends compte ici que l'utilisation de matrices Sparse est ici un choix judicieux car on travaille avec des matrices quasi creuses qui sont adaptées au problème de l'équation de Poisson (source seulement réparti dans un petit coin de l'espace). Cette méthode est quasi 10 fois plus performante que l'utilisation classique de la résolution par linalg.solve et bien plus que la relaxation, même je ne pense pas avoir utilisé les paramètres les plus optimisés pour la relax-

ation. Je ne parle même pas de la méthode de Gauss qui est juste impossible à utiliser avec du N =65 (à part sur un supercalculateur ce que je ne possède malheureusement pas...)

On obtient des résultats qui correspondent bien aux lignes de courant que l'on connaît bien d'un condensateur néanmoins il y a quelque chose que je n'arrive pas à expliquer, c'est la différence entre les valeurs obtenues analytiquement avec la méthode Sparse et celle avec np.linalg.solve et la relaxation. J'ai regardé comment la fonction linalg.solve était construite mais je n'ai pas compris d'où venait cette différence. Cependant, la méthode de relaxation donne les mêmes valeurs donc je me suis peut être trompé dans la méthode sparse en construisant la matrice sparse (erreur de coefficient éventuellement?).

Par ailleurs, les valeurs en ordre de grandeur du potentiel ne sont pas bonnes car on a pris e = 1 pour simplifier les calculs.

En Conclusion, nous proposons la méthode Sparse pour résoudre ce problème car elle semble plus simple et plus efficace, et plus juste au niveau du graphique car au centre on ne voit pas vraiment le caractère uniforme du champ pour les méthodes linalg et relaxation alors que c'est bien le cas pour la méthode sparse, même si cette méthode renvoie des valeurs différentes des deux autres méthodes. On supposera que cela vient éventuellement d'une erreur dans les coefficients des matrices mais nous n'avons pas su identifier l'erreur. Pour ce qui est de la précision, hormis les écarts de valeur, les méthodes sparse et linalg.solve semblent fournir des résultats identiques donc on ne préférera pas une méthode à une autre pour ce qui est de la précision.