TP5 - Résolution numérique de l'équation de Laplace (d'après CCP TPC 2017)

Les équations de Maxwell permettent de montrer que, en régime stationnaire et dans le vide, le potentiel électrostatique vérifie l'équation de Laplace, c'est-à-dire $\Delta V = 0$, où Δ est l'opérateur laplacien. Cette équation se retrouve dans de nombreux domaines : ainsi, la température de tout système thermodynamique en régime stationnaire vérifie également $\Delta T = 0$. Des solutions analytiques existent dans des cas simples, mais dans le cas général, résoudre cette équation demande d'avoir recours à des méthodes numériques : c'est le but de ce TP, qui permettra d'observer les effets de bord dans un condensateur plan et d'illustrer l'effet de pointe. Pour simplifier, nous nous limiterons à deux dimensions, ce qui revient physiquement à supposer une invariance par translation dans la troisième dimension.

Objectifs

- Retravailler la manipulation des tableaux Numpy ;
- Résoudre d'autres équations différentielles que celles accessibles par la méthode d'Euler ;
- Tracer des équipotentielles et des lignes de champ.

1 Principe de la résolution par méthode de relaxation

1.1 Schéma général

Il est évidemment impossible de résoudre numériquement l'équation de Laplace en tout point de l'espace : on se restreint à un domaine $\mathcal{D}$ fini, discrétisé sous forme d'une grille de N_v *pixels* verticalement et N_h *pixels* horizontalement. On peut démontrer mathématiquement que l'équation de Laplace admet une unique solution dans un domaine fermé $\mathcal{D}$ si la valeur de V est fixée sur les bords $\mathcal{B}$ du domaine (on parle en mathématiques de *problème de Dirichlet*). Dans notre cas, les bords correspondent au contour de la grille, et d'autres points à l'intérieur de celle-ci sur lesquels le potentiel sera imposé, par exemple les armatures d'un condensateur ou une pointe. Ces conditions aux limites étant fixées, on utilise ensuite une méthode de résolution par différences finies. L'idée est la même que pour résoudre une équation différentielle par la méthode d'Euler : les dérivées spatiales du potentiel sont approximées par des différences entre les valeurs du potentiel entre deux points voisins de la grille. La nouveauté par rapport à la méthode d'Euler est la résolution itérative : partant d'une *condition initiale* arbitraire (valeur de V fixée arbitrairement en tout point de la grille), on améliore par récurrence la précision du résultat pour qu'il converge vers la solution de l'équation de Laplace.

1.2 Discrétisation de l'équation de Laplace

On rappelle qu'on se place sur une grille à deux dimensions de taille $N_v \times N_h$. Le pas spatial de la grille est noté δ, si bien que le point (i, j) de la grille a pour coordonnées $(i\delta, j\delta)$. Attention, i va de bas en haut, et j de gauche à droite.

À deux dimensions, le laplacien s'écrit en coordonnées cartésiennes :

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

Il faut donc discrétiser la dérivée seconde sous forme de différence finie, ce qui peut se faire grâce à des développements limités :

$$V(x+h,y) = V(x,y) + h\frac{\partial V}{\partial x}(x,y) + \frac{h^2}{2}\frac{\partial^2 V}{\partial x^2}(x,y) + o(h^3) \tag{1}$$

et

$$V(x,y+h) = V(x,y) + h\frac{\partial V}{\partial y}(x,y) + \frac{h^2}{2}\frac{\partial^2 V}{\partial y^2}(x,y) + o(h^3) \tag{2}$$

□ **Question 1.** En sommant les 4 équations obtenues en prenant $h = \pm\delta$ dans les équations précédentes, et en notant $V[i,j] = V(x,y) = V(i\delta, j\delta)$, montrer que :

$$V[i,j] \simeq \frac{V[i-1,j] + V[i,j-1] + V[i+1,j] + V[i,j+1]}{4} \tag{3}$$

1.3 Définition des conditions aux limites

Rappelons que l'on appelle ici *bord* du domaine de résolution l'ensemble des points où la valeur du potentiel est imposée. Pour que la méthode utilisée s'applique, le bord doit nécessairement inclure le contour géométrique de la grille mais intègre aussi d'autres points intérieurs. Les points appartenant au bord du domaine sont stockés sous forme d'un tableau de booléens `B`, de taille $N_v \times N_h$. Ce tableau est construit de la façon suivante :

- ⇝ si le point (i, j) appartient au bord, alors `B[i,j]` vaut `True` : le potentiel en ce point est imposé, sa valeur ne doit jamais être modifiée ;
- ⇝ si le point (i, j) n'appartient pas au bord, alors `B[i,j]` vaut `False` : le potentiel en ce point est inconnu a priori mais devra vérifier l'équation (3).

□ **Question 2.** Écrire une fonction `initialisation_domaine()`, qui initialise les tableaux `B` et `V`. Les variables `B` et `V` étant globales, la fonction les modifie mais ne retourne rien :

- ⇝ Le tableau `V` est mis à 0, *bord* compris.
- ⇝ Les première ligne et première colonne du tableau `B` ainsi que les dernière ligne et dernière colonne sont mises à `True`, et l'intérieur du domaine à `False`.

Exécuter cette fonction et, pour vérifier son bon fonctionnement, afficher les bords du domaine grâce à la fonction `graphe_bords` (dans la rubrique des outils graphiques), qui affiche en noir les points appartenant au bord du domaine et en blanc les autres.

2 Algorithme de Jacobi

2.1 Principe

L'algorithme de Jacobi permet une résolution itérative de l'équation (3) : la valeur du potentiel $V_{n+1}[i, j]$ au point (i, j) à l'itération $n + 1$ se déduit de la valeur des potentiels voisins à l'itération n par :

$$V_{n+1}[i, j] = \frac{V_n[i-1, j] + V_n[i, j-1] + V_n[i+1, j] + V_n[i, j+1]}{4}. \tag{4}$$

Ainsi, la valeur du potentiel en tout point est recalculée à chaque itération en fonction de ce qu'elle aurait dû être si l'itération précédente vérifiait l'équation de Laplace. La convergence de la méthode est démontrable mathématiquement : au bout d'un nombre « suffisant » d'itérations, on est assuré d'approcher de la solution exacte et les nouvelles itérations ne font *presque plus* évoluer le potentiel. Le nombre d'itérations dépend de la précision souhaitée. Dans ce TP, on utilise un critère basé sur l'écart quadratique moyen :

$$e = \sqrt{\frac{\sum_{i,j} (V_{n+1}[i, j] - V_n[i, j])^2}{N_v N_h}} \tag{5}$$

qui mesure l'évolution du potentiel entre 2 itérations. On choisit de stopper la simulation lorsque e devient inférieur à une valeur donnée ε, compromis entre précision des résultats et temps de calcul.

2.2 Mise en pratique

Algorithme de Jacobi :

- ▶ Initialisation :
  - ⇝ Initialiser le tableau `B` avec des `True` et des `False` pour décrire les bords du domaine ;
  - ⇝ Initialiser le tableau `V`.
- ▶ Itérations : actualiser le tableau `V` : pour tout point (i, j) n'appartenant pas au bord, la nouvelle valeur est calculée selon la relation de récurrence (4).
- ▶ Terminaison : calculer après chaque itération l'écart e avec l'équation (5), et cesser la procédure lorsque e devient inférieur à ε fixé. `V` vérifie alors approximativement $\Delta V = 0$.

Question 3. Écrire une fonction `ecart(V1,V2)` prenant en argument deux tableaux Numpy et renvoyant l'écart entre ces deux tableaux au sens de l'équation (5).

Rappels utiles :

- Pour effectuer une copie d'un tableau Numpy on utilise la syntaxe `V_copie = V.copy()`. Au contraire, l'instruction `V_copie = V` ne fait que donner 2 noms au même tableau.
- Avoir choisi un tableau de booléens pour `B` permet d'écrire directement les éventuels tests sous la forme « `if not B[i,j]:` », ce qui est exactement équivalent à écrire « `if B[i,j] == False:` ».

Question 4. Écrire une fonction `iteration_jacobi()` qui effectue une itération de l'algorithme ci-dessus. Cette fonction ne prend aucun argument et doit renvoyer l'écart e entre les potentiels avant et après itération.

Question 5. Écrire une fonction `jacobi(eps)` qui prend en argument un flottant `eps` et qui itère l'algorithme tant que l'écart est supérieur à `eps`. À des fins de comparaison avec la méthode suivante, la fonction renverra le nombre d'itérations.

2.3 Application au condensateur plan de taille fini

Question 6. Ecrire une fonction `initialisation_condensateur(Vhaut,Vbas,L,e)` qui *crée* un condensateur. Elle doit :

- initialiser le domaine grâce à `initialisation_domaine()` ;
- placer deux plaques horizontales de largeur `L` distantes de `e` au centre du domaine, de taille 1 *pixel* : on mettra `B` à `True` en ces points ;
- attribuer le potentiel `Vhaut` à la plaque supérieure (associée à la plus petite valeur de `i`), et le potentiel `Vbas` à la plaque inférieure.

Appeler votre fonction avec des potentiels de $\pm 10V$. Vérifier le résultat avec `graphe_bords()`.

Question 7. Résoudre l'équation de Laplace par l'algorithme de Jacobi en choisissant comme critère de terminaison $\varepsilon = 10^{-3}$. Puis, afficher les surfaces équipotentielles avec la fonction `graphe_equipot()`. Rajouter éventuellement les lignes de champ avec la fonction `ldc()`.

Question 8. Retrouver les propriétés du champ et des équipotentielles. Faire varier `L` et `e`. Commenter.

3 Algorithme de Gauss-Seidel adaptatif

On peut montrer que la complexité de l'algorithme de Jacobi pour une grille de taille $N \times N$ est pire que $\mathcal{O}(N^3)$, ce qui est néfaste pour les temps de calcul et devient rapidement gênant, même pour des géométries simples. L'algorithme de Gauss-Seidel adaptatif est une amélioration possible, mais moins immédiatement compréhensible.

3.1 Principe

La méthode consiste essentiellement à remplacer la relation de récurrence (4) par

$$V_{n+1}[i,j] = (1-w)V_n[i,j] + w\frac{V_{n+1}[i-1,j] + V_{n+1}[i,j-1] + V_n[i+1,j] + V_n[i,j+1]}{4} \tag{6}$$

Il y deux différences majeures par rapport à l'algorithme précédent. D'une part, on utilise la valeur $V_n[i,j]$ de `V` au point (i,j) à l'itération précédente, pondérée avec un poids $(1-w)$.

D'autre part, le calcul fait intervenir les valeurs du potentiel aux points voisins à l'itération précédente comme c'était déjà le cas, mais aussi certaines valeurs à l'itération $n+1$ elle-même. Ceci est rendu possible par le fait que la grille est parcourue à i et j croissants, ces valeurs ont donc déjà été actualisées lors de l'itération n : on comprend ainsi que cela améliore la vitesse de convergence. En pratique, on peut itérer directement sur `V` avec l'équation (6).

On peut montrer que pour une grille carrée de taille $N \times N$ il existe une valeur optimale pour w permettant la convergence avec un nombre d'itérations en $\mathcal{O}(N)$ pour une valeur de ε fixée :

$$w_{opt} = \frac{2}{1+\pi/N}$$

On prendra cette valeur dans la suite.

□ **Question 9.** Écrire les fonctions `iteration_gauss_seidel()` et `gauss_seidel()` qui jouent un rôle analogue aux fonctions correspondantes dans l'algorithme de Jacobi.

□ **Question 10.** Tester ces fonctions sur l'exemple précédent du condensateur. Vérifier que le nombre d'itérations est inférieur à celui pour la méthode de Jacobi. On n'oubliera pas de rappeler la fonction `initialisation_condensateur(Vhaut,Vbas,L,e)` entre les deux simulations !

3.2 Application à l'effet de pointe

En l'absence de perturbations, le potentiel électrostatique augmente quasi-linéairement dans l'atmosphère entre le sol et l'ionosphère. Dans une atmosphère non perturbée, par temps calme, le gradient de potentiel est de 100 $V.m^{-1}$ environ.

L'ajout d'une tige verticale conductrice, un paratonnerre par exemple, vient perturber cette situation : ses propriétés conductrices font qu'elle est au même potentiel en tout point, égal à celui du sol auquel elle est reliée. Ceci va déformer les surfaces équipotentielles, et donc modifier le champ électrique. On s'attend à ce que ce dernier augmente vers la pointe : c'est ce que l'on nomme l'effet de pointe, et qui explique que la foudre tombe préférentiellement sur les objets pointus.

On simule ici une tige verticale de hauteur h. La fonction `initialisation_effet_pointe(Vhaut,Vbas,h)` fournie permet d'initialiser un tel domaine avec un contour dont le bord bas est au potentiel `Vbas`, le bord haut au potentiel `Vhaut` et où le potentiel augmente linéairement de `Vbas` à `Vhaut` sur les bords droit et gauche ; une tige de hauteur h au potentiel fixé `Vbas` est rajoutée.

□ **Question 11.** Initialiser le problème à l'aide de cette fonction. Vérifier que tout est correct à l'aide de `graphe_bords.`

□ **Question 12.** Résoudre le problème (on prendra encore $\varepsilon = 10^{-3}$) et visualiser la solution. Commenter.

4 Vectorisation

Pour accélérer les calculs, il faut utiliser les tableaux Numpy sans faire de boucles, en utilisant uniquement les fonctions du module. Voici quelques propriétés (hors programme, mais quelques fois rappelées en annexe) qui permettent d'augmenter les performances.

- On peut multiplier/diviser tous les éléments d'un tableau par une valeur. Par exemple `V/4`.
- On peut additionner deux tableau de même taille. Par exemple `V[:,:-1] + V[:,1:]`.
- On peut faire de l'indexation booléenne : dans ce problème par exemple, `U = V[B].copy()` est une sauvegarde des valeurs de `V` aux points où `B[i,j]` vaut `True`. On peut ensuite reconstituer le bord par `V[B] = U`.

□ **Question 13.** Réécrire une version sans boucle de `iteration_jacobi()`. Résoudre un des problèmes précédents et apprécier le gain temporel.