BLAISE PASCAL
PT 2021-2022

TP 6 – Informatique

Résolution numérique de l'équation de Laplace

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 vérifie $\Delta T = 0$ en régime permanent [1], mais on la rencontre également en gravitation, en mécanique des fluides, en mécanique quantique, pour l'étude des vibrations d'une peau de tambour, etc. 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 au passage d'observer les effets de bord dans un condensateur plan. Pour que le TP reste raisonnablement abordable dans une séance de deux heures, nous nous limiterons à deux dimensions, ce qui revient physiquement à supposer une invariance par translation dans la troisième dimension.

Vous disposez du script Python `tp06_eq-laplace_script-depart.py`, qui contient quelques fonctions qui nous serviront à l'initialisation des calculs et à l'affichage des résultats. Ouvrir ce fichier et l'enregistrer sous un nom différent (pour pouvoir retrouver lesdites fonctions en cas de fausse manœuvre).

I - Principe de la résolution par méthode de relaxation

I.A - Schéma général

Exactement comme pour les équations différentielles temporelles, il est 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 dimensions $N_x \times N_y$.

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, mais peuvent aussi inclure d'autres points à l'intérieur de celle-ci sur lesquels le potentiel serait imposé, par exemple les armatures d'un condensateur.

Les conditions aux limites étant fixées, on utilise ensuite une méthode de résolution par différences finies. L'idée est exactement la même que pour résoudre une équation différentielle avec le schéma 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 et « aléatoirement » en tout point de la grille), on améliore par récurrence la précision du résultat pour qu'il soit conforme à l'équation de Laplace.

I.B - Discrétisation de l'équation de Laplace

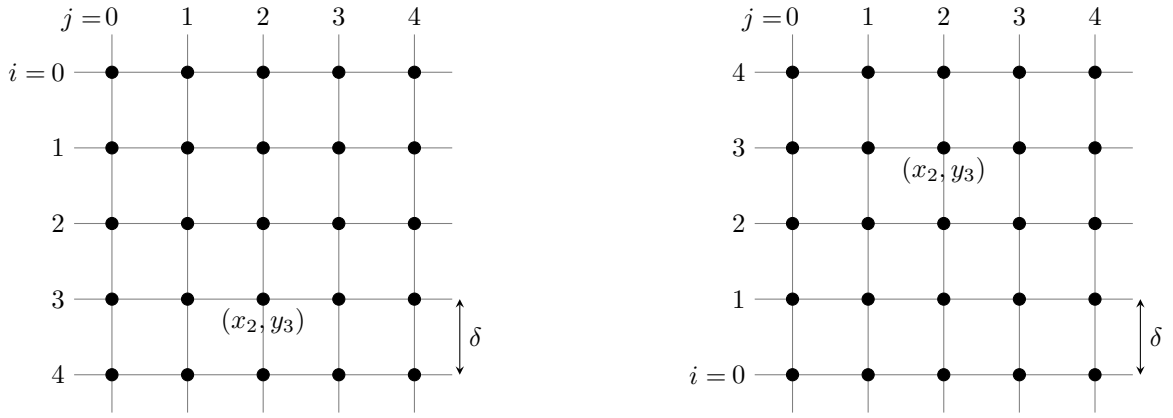
On rappelle qu'on se place sur une grille à deux dimensions de taille $N_x \times N_y$, schématisée figure 1, de pas spatial δ.

Attention ! Les tableaux `numpy` sont indicés avec la convention mathématique « ligne, colonne » plutôt qu'avec la convention physique « abscisse, ordonnée ». Cela ne changera pas grand chose pour l'écriture des fonctions, si ce n'est un peu de vigilance dans les limites des boucles ... mais cela génère aussi quelques complications dans les fonctions graphiques qui sont données en début de TP, ce qui peut conduire à des courts-circuits cérébraux chez celui qui les a écrites ☺. En particulier, l'utilisation de l'argument optionnel `origin='lower'` dans les fonctions de tracé permet de ne pas se préoccuper du fait que l'axe y est dirigé vers le bas.

⇝ avec ces conventions, le point d'indices (i, j) a pour coordonnées « physiques » $(j\delta, i\delta)$ alors que le point dont les coordonnées sont $(x_p = p\delta, y_q = q\delta)$ a pour indices (q, p).

1. Nous le démontrerons dans le cours sur la conduction thermique.

Étienne Thibierge, 3 décembre 2021, www.etienne-thibierge.fr



(a) Indices et coordonnées avec les conventions Python

(b) Indices et coordonnées avec les conventions Python et l'option de tracé `origin='lower'`

Figure 1 – Grille de discrétisation.

À deux dimensions cartésiennes, le laplacien s'écrit

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

Il faut donc discrétiser la dérivée seconde sous forme de différence finie, ce qui se fait comme toujours grâce à des développements limités. En transposant la formule de Taylor,

$$f(a+h) = f(a) + h\,f'(a) + \frac{h^2}{2} f''(a) + o(h^2)\,,$$

on peut écrire

$$V(x_j+\delta, y_i) = V(x_j, y_i) + \delta\,\frac{\partial V}{\partial x}(x_j, y_i) + \frac{\delta^2}{2}\,\frac{\partial^2 V}{\partial x^2}(x_j, y_i) + o(\delta^2)$$

et de même

$$V(x_j-\delta, y_i) = V(x_j, y_i) - \delta\,\frac{\partial V}{\partial x}(x_j, y_i) + \frac{\delta^2}{2}\,\frac{\partial^2 V}{\partial x^2}(x_j, y_i) + o(\delta^2)\,.$$

Le potentiel V sera stocké dans une variable globale sous forme d'un tableau `numpy` `V` tel que `V[i][j]` donne la valeur du potentiel $V(j\delta, i\delta)$. Par simplicité de notation, on notera donc dans la suite

$$V[i][j] = V(x_j, y_i)$$

On pourrait travailler de façon exactement identique avec des listes de listes, sans changer l'écriture des fonctions, mais travailler avec des tableaux `numpy` permet d'intialiser les tableaux et de faire les tracés plus aisément.

1 - Montrer que

$$\frac{\partial^2 V}{\partial x^2}[i][j] \simeq \frac{V[i][j+1] + V[i][j-1] - 2V[i][j]}{\delta^2}$$

puis que

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

Ainsi, l'équation de Laplace discrétisée s'écrit, pour tout point (i, j) du domaine de résolution,

$$\Delta V[i][j] = \frac{V[i+1][j] + V[i-1][j] + V[i][j+1] + V[i][j-1] - 4V[i][j]}{\delta^2} = 0$$

ce que l'on peut encore mettre sous la forme

$$\boxed{V[i][j] = \frac{V[i+1][j] + V[i-1][j] + V[i][j+1] + V[i][j-1]}{4}\,.} \tag{1}$$

Lorsqu'il vérifie l'équation de Laplace, le potentiel en un point est donc une moyenne des potentiels aux points voisins : c'est ainsi qu'il sera possible de construire par récurrence les valeurs de V en tout point de la grille en partant des bords où les conditions aux limites sont connues.

I.C - 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 peut également inclure d'autres points intérieurs.

Les points appartenant au bord du domaine sont stockés sous forme d'un tableau de booléens `B`, de taille $N_y \times N_x$. 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 doit vérifier l'équation (1).

Les tableaux décrivant le bord et le potentiel sont respectivement initialisés à des `False` et à des zéros ... ce qui revient sensiblement au même : rappelons qu'en Python les booléens sont équivalents à des 0 et des 1.

2 - Écrire une fonction `initialisation_contour(V0)` qui prend comme argument un flottant `V0`. Cette fonction initialise les points du contour de la grille : elle leur affecte la valeur `True` dans `B` et la valeur V_0 dans `V` sans toucher aux autres points. Les variables `B` et `V` étant globales, la fonction les modifie mais ne retourne rien.

Exécuter cette fonction et, pour vérifier son bon fonctionnement, afficher les bords du domaine grâce à la fonction `trace_bords` (dans la rubrique des outils graphiques, au début du script fourni), qui affiche en noir les points appartenant au bord du domaine et en blanc les autres. Choisir par exemple $N_x = N_y = 60$ pour obtenir un résultat visuel.

II - Algorithme de Jacobi

II.A - Principe

L'algorithme de Jacobi permet une résolution itérative de l'équation (1) : 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

$$\boxed{V_{n+1}[i][j] = \frac{V_n[i+1][j] + V_n[i-1][j] + V_n[i][j+1] + V_n[i][j-1]}{4}\,.} \tag{2}$$

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

$$e = \sqrt{\frac{1}{N_x N_y} \sum_{i,j} \left(V_n[i][j] - V_{n-1}[i][j]\right)^2} \tag{3}$$

qui représente l'écart entre la nouvelle et l'ancienne valeur du potentiel moyenné sur toute la grille. On choisit de stopper la simulation lorsque e devient inférieur à une valeur seuil ε définie au début de la simulation, en fonction d'un compromis entre précision des résultats et temps de calcul.

II.B - Mise en pratique

L'algorithme de Jacobi est le suivant :

- ▷ **Initialisation :**
  → créer le tableau `B` et le remplir avec des `True` et des `False` pour décrire les bords du domaine ;
  → créer le tableau `V` et l'initialiser avec les valeurs voulues sur les bords du domaine.
- ▷ **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 (2).
- ▷ **Terminaison :** calculer après chaque itération l'écart e à la précédente avec l'équation (3), et cesser la procédure lorsque e devient inférieur à ε.

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 (3).

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.

> ***Rappels utiles :***
>
> *Pour effectuer une copie d'un tableau `numpy` on utilise `V_copie = V.copy()` mais pas `V_copie = V`, sans quoi les deux sont liés et toute modification de `V` entraîne une modification de `V_copie`.*
>
> *Avoir choisi un tableau de booléens pour `B` permet d'écrire directement les tests sous la forme « `if B[i][j]:` », ce qui est exactement équivalent à écrire « `if B[i][j] == True:` », et même « `if B[i][j] == 1:` ».*

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, votre fonction doit renvoyer le nombre d'itérations. Pour suivre la convergence en temps réel, on pourra ajouter un `print(e)` à un endroit bien choisi.

II.C - Application au condensateur plan de taille fini

6 - Représenter les surfaces équipotentielles et lignes de champ électrostatique d'un condensateur plan de taille finie, en procédant de la façon suivante :

- ▷ Choisir une grille de taille 100×100 et créer les tableaux `B` et `V` dont toutes les valeurs sont nulles.
- ▷ Initialiser le contour de la grille à un potentiel nul, et ajouter avec la fonction `initialisation_condensateur` un condensateur plan de longueur 30 pixels et d'épaisseur 8 pixels, dont les armatures sont respectivement aux potentiels $\pm 5\,\mathrm{V}$.
- ▷ Résoudre l'équation de Laplace par l'algorithme de Jacobi en choisissant comme critère de terminaison $\varepsilon = 1 \cdot 10^{-3}$.
- ▷ Afficher les surfaces équipotentielles et les lignes de champ avec les fonctions `trace_equipot` et `trace_ldc`.

7 - Identifier les effets de bord et la distance sur laquelle ils se manifestent. Comme vous le savez (n'est-ce pas ?) les équipotentielles d'un condensateur plan infini sont équidistantes, parallèles entre elles, et parallèles aux armatures du condensateur ☺

III - Algorithme de Gauss-Seidel adaptatif

On peut montrer que 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.

III.A - Principe

La méthode consiste essentiellement à remplacer la relation de récurrence (2) par

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

Il y a donc 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 n, pondérée avec un poids w. On peut montrer que pour une grille carrée de taille $N \times N$ il existe une valeur optimale

$$w_{\text{opt}} = \frac{2}{1 + \dfrac{\pi}{N}}\,.$$

D'autre part, le calcul fait intervenir les valeurs du potentiel aux points voisins à l'itération précédente n 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, il ne faut plus utiliser un tableau copie mais directement itérer `V` avec l'équation (4).

8 - Écrire les fonctions `iteration_gauss_seidel` et `gauss_seidel` qui jouent un rôle analogue aux fonctions du même nom écrites pour l'algorithme de Jacobi.

9 - Tester ces fonctions sur l'exemple précédent du condensateur. Vérifier que le nombre d'itérations est inférieur. Pour que la comparaison ait un sens, penser à réinitialiser la matrice `V` entre les deux simulations.

III.B - Taille du domaine de calcul

La taille du domaine de calcul est un point crucial dans les simulations numériques de ce type : il est nécessaire pour la simulation de fixer la valeur du potentiel sur tout le contour de la grille, mais ce choix non-physique ne doit pas ou peu influer sur le résultat de la simulation dans la zone intéressante.

On étudie cet effet dans le cas de domaines carrés de taille $N \times N$, les différentes valeurs étudiées étant stockées dans une liste `lst_N`. En guise d'illustration, on étudie l'influence de la valeur de N sur le potentiel en un point de référence choisi arbitrairement, légèrement à l'extérieur du condensateur. Les valeurs du potentiel en ce point pour les différentes valeurs de N sont stockées dans une liste `Vref`. On ne pourra prendre que des valeurs $N \geq 40$ compte tenu du point choisi.

10 - Compléter la fonction `influence_taille_domaine(lst_N)`, qui renvoie la liste `Vref`. On notera que quelques précautions s'imposent avec les variables choisies comme étant globales dans les simulations précédentes et dont cette fonction modifie les valeurs.

11 - Représenter le potentiel au point de référence en fonction de la taille du domaine de calcul.

III.C - Application à l'effet de pointe

- **Introduction**

En l'absence de perturbations, le potentiel électrostatique augmente quasi-linéaire dans l'atmosphère entre le sol (qui est à un potentiel V_{sol} que l'on prendra égal à $0\,\text{V}$) et l'ionosphère. Dans une atmosphère non perturbée, par temps calme, le gradient de potentiel est de $100\,\text{V}\cdot\text{m}^{-1}$ 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 et de largeur D, dont la pointe est une demi-sphère de diamètre D, voir figure 2. Plus précisément, notre programme étant bidimensionnel, nous simulons en fait un plan de hauteur H, épaisseur D, de longueur très grande dans la direction perpendiculaire au plan de la simulation. Les temps de calculs seraient trop grands à trois dimensions sur les ordinateurs du lycée.

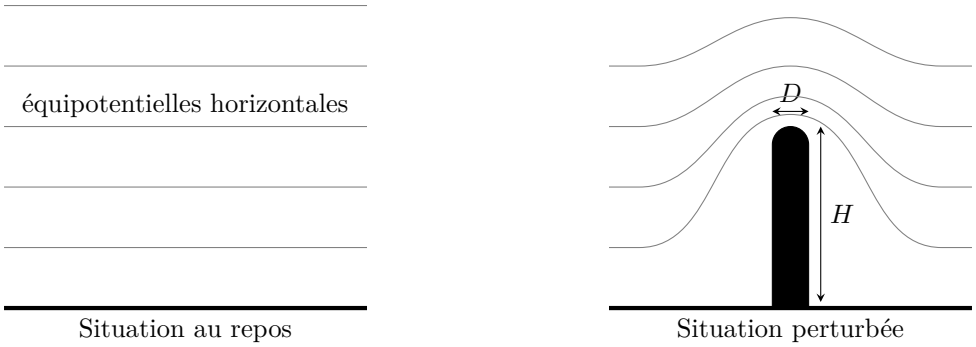


Figure 2 – Allure qualitative des équipotentielles en présence d'une pointe.

L'objectif est d'obtenir numériquement une relation entre la valeur E_{max} du champ électrique et le diamètre D de la pointe de la tige.

- **Calcul du potentiel**

La fonction `initialisation_effet_pointe(Vmin,Vmax,h,d)` fournie dans le script de départ permet d'initialiser un domaine avec

- ▷ un contour dont le bord bas est au potentiel V_{min}, le bord haut au potentiel V_{max} et où le potentiel augmente linéairement de V_{min} à V_{max} sur les bords droit et gauche ;
- ▷ une tige de hauteur h et épaisseur d (en pixels), au potentiel fixé V_{min}.

Pour choisir des valeurs ayant un sens physique, on raisonne sur un pas de grille $\delta = 3\,\text{cm}$, et on prendra

- ▷ $h = 50$ et $d = 13$, ce qui correspond à une poteau de hauteur $H = 1{,}5\,\text{m}$ et de largeur $D = d \times \delta = 30\,\text{cm}$;
- ▷ un domaine de taille 120×120 points, ce qui correspond à $3{,}6 \times 3{,}6\,\text{m}$;
- ▷ un potentiel $V_{\text{min}} = 0\,\text{V}$ au niveau du sol et $V_{\text{max}} = N_y \times \delta \times 100\,\text{V}$ en haut du cadre, ce qui donne bien un gradient au repos de $100\,\text{V}\cdot\text{m}^{-1}$.

12 - Initialiser le problème à l'aide de cette fonction. Vérifier que tout est correct à l'aide de `trace_bords`.

13 - Utiliser ensuite l'algorithme de Gauss-Seidel pour obtenir le potentiel. On prendra encore $\varepsilon = 10^{-3}$. Visualiser la solution à l'aide des fonctions `trace_equipot` et `trace_ldc`.

14 - Sur les équipotentielles précédentes, identifier la zone où le champ est le plus intense.

- **Calcul du champ électrique**

Pour calculer le champ électrique, on utilisera la fonction `gradient` de la bibliothèque `numpy` : `Ey,Ex = np.gradient(-V)/delta`, la division par δ étant nécessaire pour exprimer le champ en volts par mètre au lieu de volts par pixel.

15 - Écrire une fonction `champ_max(d)` qui prend comme argument la largeur de la tige, calcule le potentiel par la méthode de Gauss-Seidel puis le champ, et renvoie le maximum de la norme $||\vec{E}||$ du champ.

16 - Appliquer cette fonction dans le cas précédent : par combien la présence de l'obstacle multiplie-t-elle le champ ?

17 - Reproduire la procédure pour quelques valeurs de d (p.ex. 7, 13, 17, 23) et représenter le maximum de la norme de $||\vec{E}||$ en fonction de d. Ceci va-t-il dans le sens de l'effet de pointe mentionné précédemment ?