Topographie électrostatique

Capacités exigibles

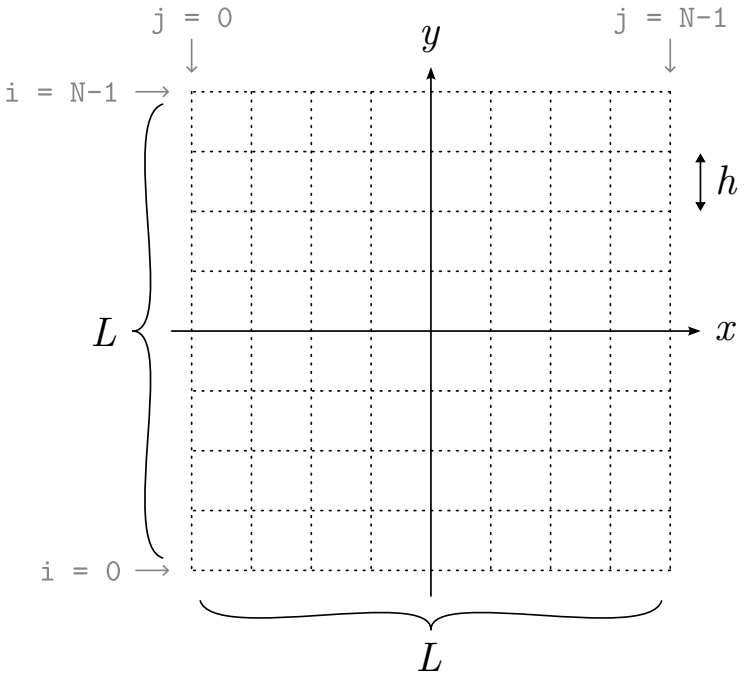
- À l'aide d'un langage de programmation, tracer l'allure du potentiel électrostatique, étant donnée une distribution de charge.
- Faire le lien entre équipotentielles et champ électrique.

I Documents

Document 1 : Grille numérique

Pour mener des calculs de champ numériquement, il est usuel de décomposer l'espace en une grille de taille $N \times N$. Dans ce TP, le lien entre les coordonnées physiques (x, y) et les indices numériques `(i, j)` se résume ainsi :

$$x \in \left[-\frac{L}{2}; \frac{L}{2}\right] \quad \leftrightarrow \texttt{j} \in \texttt{[0, N]}$$
$$y \in \left[-\frac{L}{2}; \frac{L}{2}\right] \quad \leftrightarrow \texttt{i} \in \texttt{[0, N]}$$



⚠ Avec numpy, le premier indice `i` correspond au numéro de la ligne ; et le second `j`, à celui de la colonne !

Ainsi, un champ scalaire $A(x, y)$ se modélise simplement par une matrice de taille `N` $\times$ `N`.
On notera h le pas spatial de la grille.

Document 2 : Gradient numérique

Le gradient d'un champ scalaire A est définit en coordonnées cartésiennes comme :

$$\vec{u} = \overrightarrow{\text{grad}}\, A = \frac{\partial A}{\partial x}\vec{e_x} + \frac{\partial A}{\partial y}\vec{e_y} \qquad \text{(en 2D)}$$

Pour le déterminer en un nœud `(i, j)` d'une grille, on calcule la pente moyenne du champ A entre les nœuds adjacents :

$$\begin{cases} (u_x)_{i,j} = \dfrac{A_{i,j+1} - A_{i,j-1}}{2h} \\ (u_y)_{i,j} = \dfrac{A_{i+1,j} - A_{i-1,j}}{2h} \end{cases}$$

Document 3 : Loi de Poisson

La relation liant le champ électrostatique V à une distribution de charge ρ est

$$\Delta V + \frac{\rho}{\epsilon_0} = 0$$

Avec Δ symbolisant l'opérateur laplacien :

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

Dans tout ce TP, on prendra $\epsilon_0 = 1$ pour alléger les notations.

II Énoncé

A Charges ponctuelles

Ici, le but est de tracer le champ électrique dans une configuration simple en présence uniquement de charges ponctuelles.

1. Introduisez les paramètres de la simulation : $L = 10$ et $N = 1000$. Définir ensuite h en fonction de ces grandeurs.
2. À l'aide des fonctions `np.linspace` et `np.meshgrid`, créez deux matrices `x` et `y` de taille $N \times N$, contenant les valeurs des coordonnées x et y en chaque nœud de la grille.
3. Codez une fonction `gradient(A)` prenant comme entrée une matrice `A` modélisant un champ scalaire, et renvoyant deux matrices `ux` et `uy`, correspondant aux projections du gradient de A selon $\overrightarrow{e_x}$ et $\overrightarrow{e_y}$.

   Aide : Il pourra être malin de se servir de la fonction `np.roll`.

   NB : Les valeurs aux limites seront fixées nulles.
4. Rappeler la forme du potentiel électrostatique créé par une charge q placée à l'origine du repère.

5. Dans cette situation, créez une matrice `V` contenant les valeurs du champ électrostatique V en tout point de la grille.
6. Afficher les équipotentielles et superposer le champ vectoriel électrique $\overrightarrow{E}$.

   NB : La décroissance de V en fonction de la distance peut être trop élevée, ce qui a pour effet de concentrer les contours autour de la charge. Pour les redistribuer de manière plus homogène, on pourra calculer les iso-courbes du logarithme de la valeur absolue de V :

```
>>> plt.contour(x, y, np.log(np.abs(V)), 50, linestyles='dashed', cmap='Greys')
```

7. Tracez les champs V et $\overrightarrow{E}$ dans le cas de deux charges opposées très proches l'une de l'autre.

B Charges étendues

À présent, on souhaite étudier la création d'un champ électrique à partir d'une distribution de charge quelconque.

(8) On va avoir besoin de résoudre la loi de Poisson (cf. document 3), donc de coder le laplacien numériquement. Pour cela, répondez aux questions ci-dessous :

- Écrire les développements limités d'une fonction $x \to f(x)$ en $x+h$ et $x-h$ à l'ordre 2 lorsque $h \to 0$.
- Sommer ces développements afin d'exprimer $f''(x)$.
- Écrire une version discrétisée de cette expression : x est représenté par l'indice `j`, et f par une liste $(f_j)_{j\in[0,N]}$.
- Généraliser ce développement pour un laplacien en 2D :

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- En déduire que l'on peut exprimer le potentiel en un nœud `(i, j)` en fonction de celui de ses voisins :

$$(1) \qquad V_{i,j} = \frac{1}{4}(\rho_{i,j}\, h^2 + V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1})$$

Avec $\rho_{i,j}$ la valeur de ρ sur le nœud considéré.

L'idée pour construire le champ V, est de partir d'une matrice nulle et de modifier petit à petit ses valeurs, jusqu'à on nombre d'itérations suffisant, pour que la simulation tende vers la solution exacte.

(9) Créez un nouveau fichier en copiant vos réponses aux trois premières questions.

(10) Initialisez une matrice `rho = np.zeros((N, N))` modélisant la distribution de charge. Modifiez ensuite cette matrice afin de simuler l'intérieur d'un condensateur : une ligne sera partiellement remplie de 1 (charge positive), et sa symétrique par rapport à l'origine de -1 (charge négative).

(11) Codez une boucle de 500 itérations, recalculant la matrice de V à partir de la précédente selon l'équation (1).
Aide : Utilisez la fonction `np.roll` !

(12) De même que dans votre précédent document, affichez les équipotentielles et les lignes de champ de $\vec{E}$.

III Annexe

`np.linspace(xi, xf, N)`

Créer un tableau 1D allant de `xi` à `xf` en `N` points.

Exemple :

```
>>> np.linspace(-1, 1, 5)
array([-1.0 , -0.5, 0.0, 0.5, 1.0 ])
```

`np.meshgrid(xx, yy)`

Créer deux matrices renvoyant les coordonnées des abscisses et ordonnées de chaque nœud.

Exemple :

```
>>> xx = np.linspace(-1, 1, 5)
>>> x, y = np.meshgrid(xx, xx)
>>> x
array([[-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ]])
>>> y
array([[-1. , -1. , -1. , -1. , -1. ],
       [-0.5, -0.5, -0.5, -0.5, -0.5],
       [ 0. ,  0. ,  0. ,  0. ,  0. ],
       [ 0.5,  0.5,  0.5,  0.5,  0.5],
       [ 1. ,  1. ,  1. ,  1. ,  1. ]])
```

`np.roll(A, k, axis=?)`

On a deux modes d'utilisation :

`axis=0` Fait "rouler" une matrice `A` en déplaçant toutes ses lignes d'une quantité `k`.

`axis=1` Même chose selon les colonnes.

Exemple :

```
>>> A                          # Matrice de départ
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

>>> np.roll(A, 1, axis=0)     # On pousse les lignes d'une case vers les bas
array([[12, 13, 14, 15],      # La dernière ligne se retrouve au début !
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> np.roll(A, -1, axis=1)   # On tire les colonnes d'une case vers la gauche
array([[ 1,  2,  3,  0],      # La première colonne se retrouve à la fin
       [ 5,  6,  7,  4],
       [ 9, 10, 11,  8],
       [13, 14, 15, 12]])
```

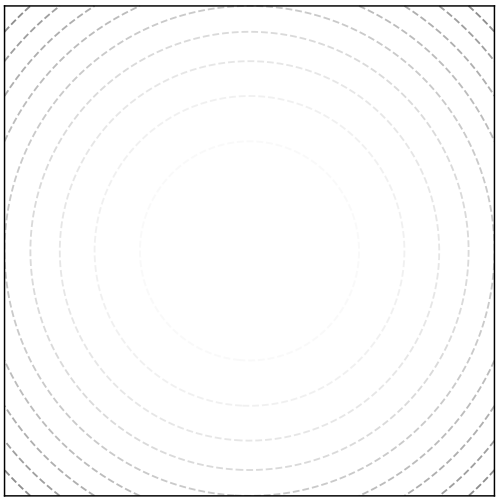
`plt.contour(x, y, A, n)`

Trace des courbes isométriques du champ scalaire A.

- `x` et `y` sont les matrices des coordonnées des nœuds (générées avec `np.meshgrid`) ;
- `A` est la matruce modélisant le champ scalaire en question ;
- `n` est le nombre de courbes à dessiner ;

Exemple :

```
>>> plt.contour(x, y, x*x + y*y, 50, linestyles='dashed', cmap='Greys') # Les derniers arguments sont purement esthétiques
>>> plt.gca().set_aspect('equal')   # Permet d'afficher un graphique en repère orthonormé
>>> plt.show()
```



`plt.streamplot(x, y, ux, uy, density)`

Dessine les lignes du champ vectoriel $\overrightarrow{u} = (u_x, u_y)$, dans la grille de coordonnées (x, y). Le paramètre `density` permet de multiplier le nombre de lignes tracées.

Exemple :

```
>>> ux = np.ones((N, N))
>>> uy = np.zeros((N, N))
>>> plt.streamplot(x, y, ux, uy, .5, color='#d62626')
>>> plt.gca().set_aspect('equal')
>>> plt.show()
```