Lycée Charles Coëffin — Sciences physique

Fiche de travaux pratiques — CPGE TSI2

TP 11 : **Électrostatique** **Durée :** 2 h

Objectifs

- À l'aide d'un langage de programmation, tracer quelques lignes de champ et lignes équipotentielles pour une distribution donnée.

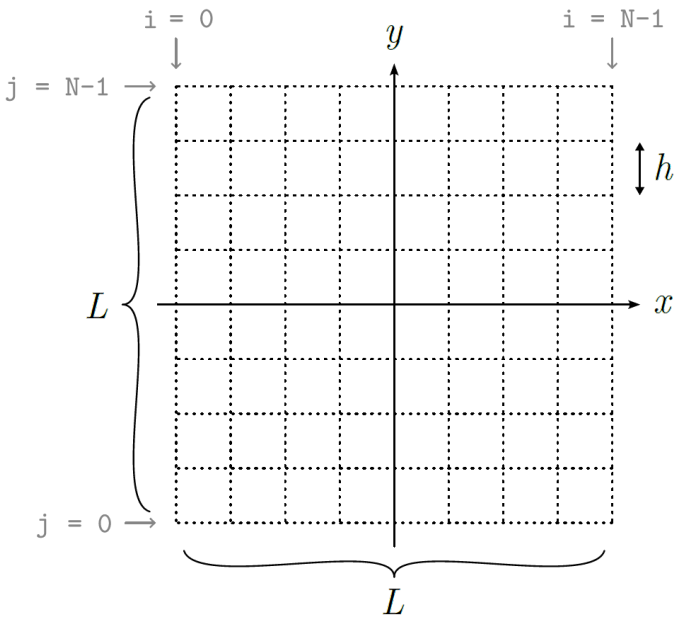
Pré-requis : réactions d'oxydo-réduction, nombre d'oxydation, potentiel d'électrode, formule de Nernst, diagrammes de prédominance ou d'existence.

Décomposition de l'espace en grille

Pour mener des calculs de champ numériquement, il est usuel de décomposer un espace à deux dimensions 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 :

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

⚠ 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.

Gradient et gradient numérique

En deux dimensions, le gradient d'un champ scalaire A est définit en coordonnées cartésiennes comme :

$$\vec{B} = \overrightarrow{\text{grad}}\, A = \frac{\partial A}{\partial x}\vec{u}_x + \frac{\partial A}{\partial y}\vec{u}_y.$$

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

$$\vec{B}[j,i] = B_x[j,i]\vec{u}_x + B_y[j,i]\vec{u}_y \quad \text{avec} \quad \begin{cases} B_x[j,i] = \dfrac{A[j,i+1] - A[j,i-1]}{2h} \\ B_y[j,i] = \dfrac{A[j+1,i] - A[j-1,i]}{2h} \end{cases}.$$

Loi de Poisson

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

$$\Delta V = -\frac{\rho}{\varepsilon_0}$$

avec Δ symbolisant l'opérateur laplacien qui, en deux dimensions, est tel que

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

Charges ponctuelles

Procédure

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

1. **Introduire** les paramètres de la simulation : $L = 10$ et $N = 100$. **Définir** ensuite h en fonction de ces grandeurs.
2. À l'aide des fonctions `np.linspace` et `np.meshgrid`, **créer** deux matrices `x` et `y` de taille $N \times N$, contenant les valeurs des coordonnées x et y en chaque noeud de la grille.
3. **Coder** une fonction `grad` prenant comme entrée une matrice `A` modélisant un champ scalaire, et renvoyant deux matrices `gradA_x` et `gradA_y`, correspondant aux projections du gradient de A selon $\vec{u}_x$ et $\vec{u}_y$.

   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, **créer** une matrice `V` contenant les valeurs du champ électrostatique V créer par une charge ponctuelle $Q = 10^5 e$, avec e la charge électrique élémentaire, en tout point de la grille.
6. **Afficher** les équipotentielles et superposer le champ vectoriel électrique $\vec{E}$.

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 ')
```

Charges étendues

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

Procédure

7. On exploite l'équation de Poisson. Pour cela on rappelle que les développements limités à l'ordre 2 d'une fonction $f(x)$ en $x-h$ et en $x+h$ sont

$$f(x-h) = f(x) - h\frac{\mathrm{d}f}{\mathrm{d}x} + \frac{h^2}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \quad \text{et} \quad f(x+h) = f(x) + h\frac{\mathrm{d}f}{\mathrm{d}x} + \frac{h^2}{2}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}.$$

   (a) **Sommer** ces développements afin d'exprimer $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$.

   (b) **Écrire** une version discrétisée de cette expression : x est représenté par l'indice `j`, et f par une liste $(f_j)_{j\in[0,N]}$.

   (c) **Généraliser** ce développement pour un laplacien en 2D :

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

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

$$V_{i,j} = \frac{1}{4}\left(\frac{\rho_{i,j}}{\varepsilon_0}h^2 + V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}\right) \quad (1)$$

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

On doit construire le champ V à partir d'une matrice nulle et on modifie petit à petit ses valeurs jusqu'à un nombre d'itérations suffisant pour que la simulation tende vers la solution exacte.

8. À partir de la une matrice `rho = np.zeros((N, N))` modélisant la distribution de charge, **modifier** ensuite cette matrice afin de simuler l'intérieur d'un condensateur : une colonne en $x = 0$ sera remplie d'une charge de $1\,\mathrm{C}$ par point, et une colonne en $x = 999$ sera remplie d'une charge de $-1\,\mathrm{C}$ par point.
9. **Créer** une fonction permettant de calculer la valeur du potentiel `V_etendu` de cette distribution de charge à partir de l'équation (1).
10. **Coder** une boucle de 500 itérations pour obtenir une valeur finale de `V_etendu` .
11. **Afficher** les équipotentielles et les lignes de champ de $\vec{E}$. On n'utilisera pas la fonction `np.log(np.abs(V))` mais tout simplement `V` .

Grille d'évaluation

Critères	Points
Présentation (propreté, orthographe, schéma)	4
Rigueur (résultats calculatoires, utilisations de lois et de théorèmes, ...)	6
Interprétation et validation	4
Implication (nombres de questions traitées)	6

Annexes

`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. ]])
```

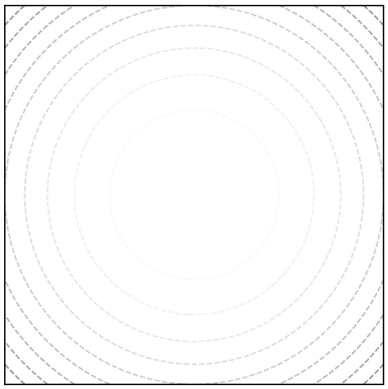
`plt.contour(x, y, A, n)`

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()
```