Capacité numérique

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

Le but de cette activité est de tracer les lignes de champ électrique et les équipotentielles étant donnée une répartition de charges ponctuelles.
On étudiera en particulier le cas d'une charge ponctuelle puis d'un dipôle et enfin d'un condensateur plan, modélisé par une juxtaposition de charges ponctuelles.

Les lignes de champ seront représentées en 2 dimensions dans un plan xOy.

Les schémas numériques sont obtenus à l'aide de la méthode d'Euler explicite.

Le script `Python` utilisé est retranscrit dans l'annexe A du présent document. La version exécutable du script est également jointe à cette ressource.

A. Lignes de champ et équipotentielles autour d'une charge ponctuelle.

On souhaite, dans un premier temps, chercher le champ créé par un électron de charge $q = -1{,}6 \cdot 10^{-19}$ C placé en $O(x_O,\ y_O)$.

Une charge ponctuelle q placée en O de coordonnées $(x_O,\ y_O)$ crée au point M de coordonnées $(x_M,\ y_M)$, un potentiel électrique :

$$V(M) = \frac{q}{4\pi\varepsilon_0 OM} = \frac{q}{4\pi\varepsilon_0\sqrt{(x_M - x_O)^2 + (y_M - y_O)^2}}$$

On calculera le potentiel créé par la charge puis on en déduira le champ électrique.

A.1. Déclaration des charges

Pour pouvoir adapter le programme facilement au tracé de lignes de champ avec des répartitions de charges différentes, on va définir l'ensemble des objets chargés qui engendrent le champ en représentant chaque objet chargé par un triplet (x,y,q), où (x,y) sont les coordonnées de l'objet et q est la charge électrique de l'objet en Coulombs, positive ou négative.

L'ensemble des objets à considérer sera alors représenté par une liste de triplets *(x, y, q)*, mémorisée dans une variable objets.

On commence donc par créer une fonction `ajouter_objet(x,y,q)` qui permettra d'ajouter une charge q placée en *(x,y)* dans la variable objets.

Dans ce premier cas, la variable objets ne contiendra qu'un seul triplet *(0,0,q)*.

```
objets = []

def ajouter_objet(x,y,q):
    objets.append([x,y,q])

ajouter_objet(0,0, - q)
```

A.2. Maillage de l'espace

On veut tracer les équipotentielles et les lignes de champ dans un carré de côté L.
Pour cela, on va découper la surface sur laquelle on trace les lignes de champ en petits éléments de surface carrée de côté a. Cette étape s'appelle le maillage. Les grandeurs physiques *V(x,y)*, $E_x(x,y)$ et $E_y(x,y)$ seront alors représentées comme des matrices de taille $L/a \times L/a$ qui contiennent les valeurs des grandeurs en chaque point du maillage. Ceci peut être fait à l'aide de la fonction `meshgrid` de `numpy` qui crée une grille rectangulaire à partir de deux tableaux unidimensionnels.

```
x = np.linspace( - L/2, L/2, L/a)
y = np.linspace( - L/2, L/2, L/a)
X, Y = np.meshgrid(x, y)
```

On désire tracer les équipotentielles et les lignes de champ créées par l'électron à l'échelle d'un atome dans lequel l'électron se trouverait.
La taille L choisie pour le tracé sera donc la taille typique d'un atome $L = 10^{-10}$ m. Pour le pas a, on choisit un maillage contenant 100×100 points, soit $a = 10^{-12}$ m.

A.3. Calcul du potentiel

On écrit une fonction `calculV(xM, yM, q, xO, yO)` permettant de calculer au point M de coordonnées *(x_M, y_M)* le potentiel créé par la charge q placée au point O de coordonnées *(x_0, y_0)*.

Il faut traiter à part le cas où $x_M = x_0$ et $y_M = y_0$ pour lequel le potentiel n'est pas défini. On prendra *V(O) = 0*.

```
def calculV(xM,yM,q,xO,yO):
    # problème pour la position de coordonnées 0,0
    if xM == xO and yM == yO:
        return 0
    else:
        return q/(4*pi*e_0)*1/((xM-xO)**2+(yM-yO)**2)**(1/2)
```

Dans la suite, pour calculer le potentiel créé par la répartition de charges, il faudra faire appel à la fonction `calculV` pour chaque objet chargé contenu dans la liste objets et calculer le potentiel en chaque point du maillage. Grâce au principe de superposition, on somme les potentiels créés par chaque objet chargé.

```
nM = len(X)
no = len(objets)
V = np.zeros((nM, nM))

for i in range(0, nM) :
  for j in range (0, nM) :
    for k in range(0, no) :
      V[i,j] = V[i,j] + calculV(x[i], y[j], objets[k][2], objets[k][0], objets[k][1])
```

A.4. Calcul du champ électrique

On va maintenant calculer le champ électrique à partir du potentiel, en utilisant les deux relations :

$$E_x = -\frac{\partial V}{\partial x} \quad \text{et} \quad E_y = -\frac{\partial V}{\partial y}$$

La méthode d'Euler explicite consiste en l'approximation :

$$E_x(x,y) = -\frac{V(x+a,y) - V(x,y)}{a} \quad \text{et} \quad E_y(x,y) = -\frac{V(x,y+a) - V(x,y)}{a}$$

Les coefficients de la matrice donnant les composantes du champ électrique en chaque point du maillage sont donc obtenus à partir des coefficients de celle donnant le potentiel grâce au schéma numérique suivant :

```
Ex[i,j] = - (V[i,j+1] - V[i,j])/a
Ey[i,j] = - (V[i+1,j] - V[i,j])/a
```

A.5. Tracé des équipotentielles et lignes de champ

Pour tracer les équipotentielles, on peut utiliser la fonction `contour` de `matplotlib.pyplot` et pour tracer les lignes de champ, la fonction `streamplot` de `matplotlib.pyplot`.

Mais, avant de procéder au tracé, il faut ajuster les tailles de la matrice donnant les valeurs de V et de celle donnant les coordonnées des points du maillage à celles donnant E_x et E_y qui sont plus petites.

On obtient le résultat suivant :

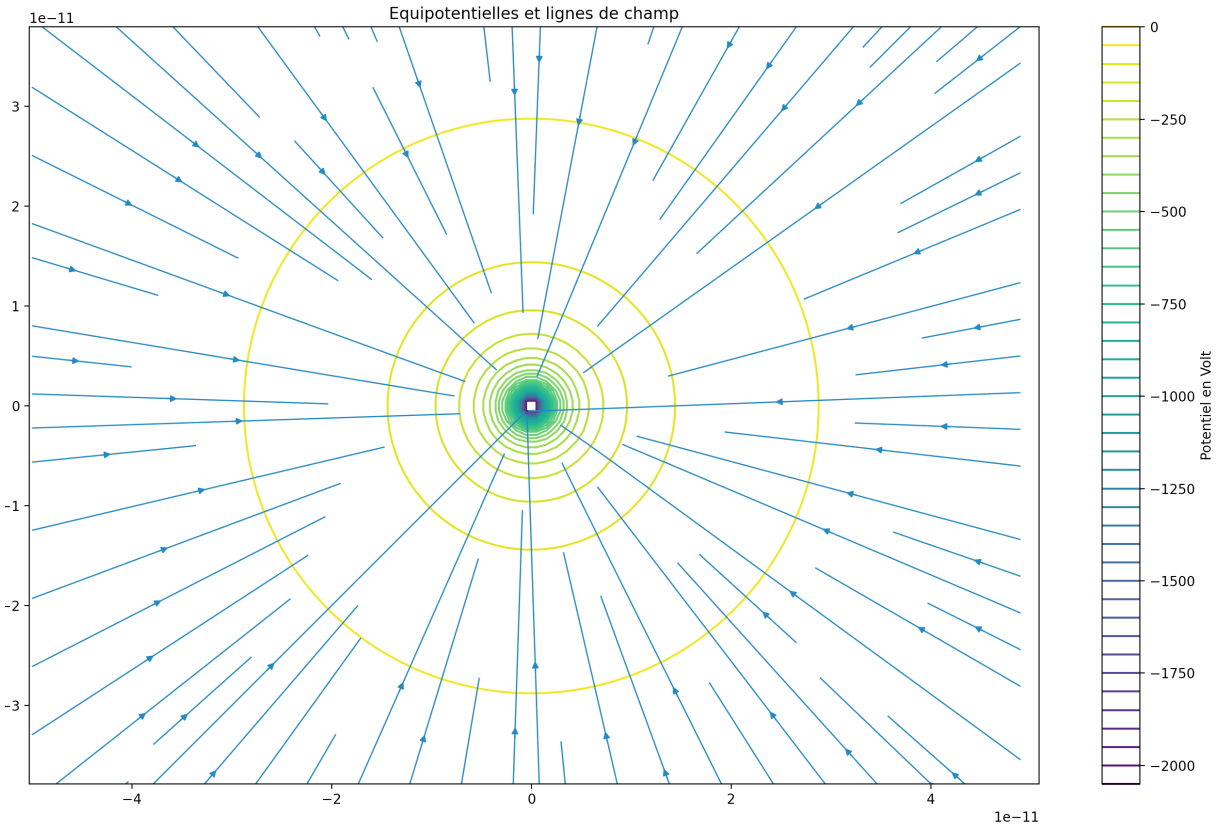


FIGURE 1 – Lignes de champ et équipotentielles autour d'un électron.

Le problème traité est invariant par rotation autour de tout axe contenant le point O.

On peut vérifier que les lignes de champ sont bien perpendiculaires aux équipotentielles et que les symétries et l'orientation des lignes de champ sont bien celles attendues.

B. Lignes de champ et équipotentielles autour d'un dipôle.

On veut étudier ici les lignes de champ électrique autour d'une molécule de chlorure d'hydrogène HCl.
Le chlore étant plus électronégatif que l'hydrogène, la molécule présente une charge partielle $+q$ sur l'atome d'hydrogène et une charge partielle $-q$ sur l'atome de chlore.
La molécule présente alors un moment dipolaire $p = qd$ où d est la distance entre les deux atomes.
Le moment dipolaire de la molécule HCl vaut : $p = 3{,}69 \cdot 10^{-30}$ C · m et la distance entre les deux atomes vaut $d = 127{,}4$ pm. La charge partielle q vaut donc : $q = 2{,}90 \cdot 10^{-20}$ C.

On reprend le même programme que précédemment en plaçant cette fois dans la liste objets les deux triplets (-d/2, 0, q) et (d/2, 0, - q).

On choisit de tracer les lignes de champ à l'échelle de la molécule de HCl, on choisit $L = 10^{-9}$ m et $a = L \,/\, 100$.

On obtient la figure suivante :

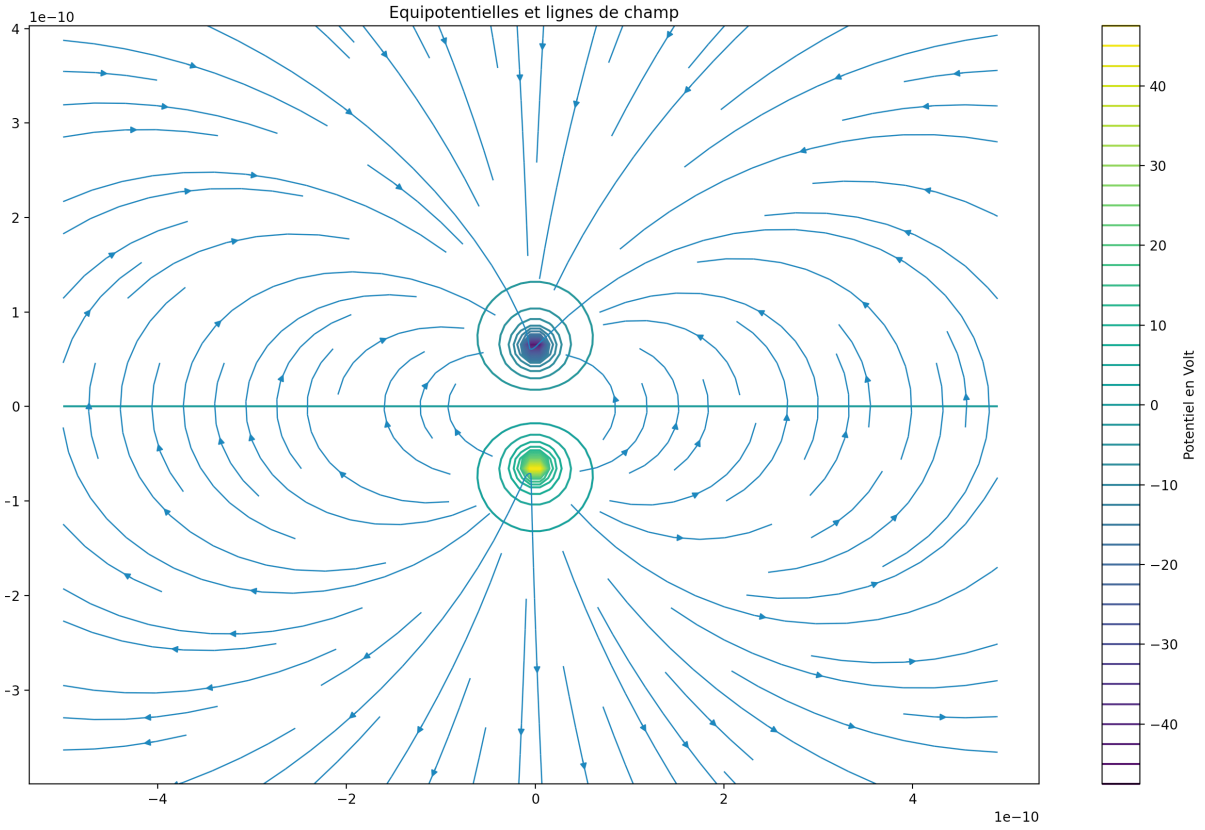
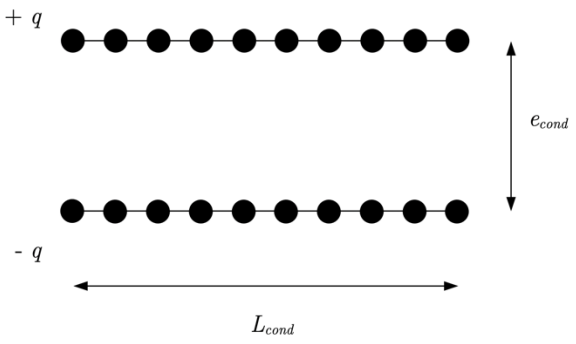


FIGURE 2 – Lignes de champ et équipotentielles autour d'une molécule HCl

Là encore, les lignes de champ sont bien perpendiculaires aux équipotentielles et les symétries ainsi que l'orientation des lignes de champ sont bien celles attendues.

C. Lignes de champ et équipotentielles dans un condensateur plan.

On souhaite visualiser le champ à l'intérieur d'un condensateur plan. On modélise chaque plaque de longueur L_{cond} comme constituée de 100 charges ponctuelles respectivement $+q$ et $-q$. Soit e_{cond} la distance entre les deux plaques.



On remplit la liste objets contenant les 100 charges + q sur la plaque supérieure et les 100 charges - q sur la plaque inférieure :

```
Xi = np.linspace(0, L_cond, 100)
for i in xi :
    ajouter_objet(-L_cond/2 + i, e_cond/2, q)

for i in xi :
    ajouter_objet(- L_cond/2 + i, - e_cond/2, -q)
```

La suite du programme est inchangée. On obtient alors la figure suivante :

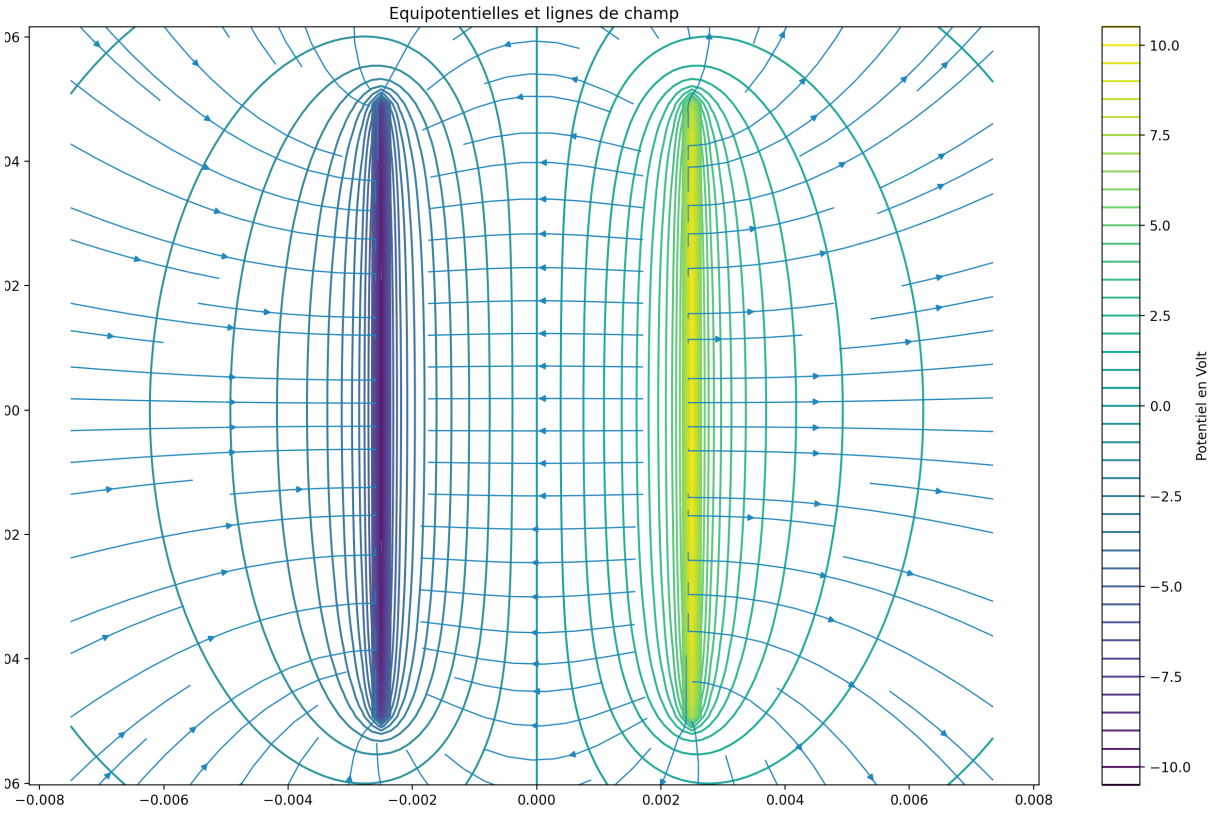


FIGURE 3 – Lignes de champ et équipotentielles autour dans un condensateur plan

On observe que le champ est uniforme à l'intérieur du condensateur et loin des extrémités, ce qui justifie la modélisation usuelle. La simulation permet d'illustrer les effets de bords.

Annexe A : Script Python

```python
## Importation des bibliothèques

import matplotlib.pyplot as plt
import numpy as np


## Paramétrage

e_0 = 8.85*10**(-12)
objets = []


## Construction de la distribution de charge et de la zone d'observation

def ajouter_objet(x,y,q):
    objets.append([x,y,q])

# Ajout d'une charge ponctuelle (enlever les commentaires pour ajouter la distribution)
"""
q_charge = - 1.6*10**(-19)
L = 10**(-10)
a = L / 100

ajouter_objet(0, 0, q_charge)
"""
# Ajout d'un dipôle (enlever les commentaires pour ajouter la distribution)
"""
q_dipole = 2.9*10**(-20)
d_dipole = 1.274*10**(-10)
L = 10**(-9)
a = L / 100

ajouter_objet(- d_dipole/2, 0, q_dipole)
ajouter_objet(+ d_dipole/2, 0, - q_dipole)
"""
# Ajout d'un condensateur (enlever les commentaires pour ajouter la distribution)
"""
q_cond = 10 ** (-14)
L_cond = 0.01 # Longueur des plaques du condensateur
e_cond = 0.005 # Distance entre les deux plaques
L = 0.015 # Longueur du domaine
a = L/100 # pas spatial

xi = np.linspace(0, L_cond, 100)
for i in xi:
    ajouter_objet(-L_cond/2 + i, e_cond/2, q_cond)

for i in xi:
    ajouter_objet(- L_cond/2 + i, - e_cond/2, - q_cond)
"""
```

```
## Calcul du potentiel

def calculV(xM,yM,q,xO,yO):
    """
    calcul du potentiel en xM, yM créé par une charge ponctuelle q placée en xO,yO
    """

    # problème pour la position de coordonnées xO,yO
    if xM == xO and yM == yO :
        return 0

    else :
        return q/(4*np.pi*e_0)*1/((xM-xO)**2+(yM-yO)**2)**(1/2)

x = np.linspace(- L/2, L/2, int(L/a))
y = np.linspace(- L/2, L/2, int(L/a))
X, Y = np.meshgrid(x,y)

nM = len(X)
no = len(objets)
V = np.zeros((nM,nM))

for i in range(0,nM) :
    for j in range (0,nM):
        for k in range(0,no):
            V[i,j] = V[i,j] + calculV(x[i], y[j], objets[k][2], objets[k][0], objets[k][1])


## Calcul du champ électrique

Ex = np.zeros((nM - 1, nM - 1))
Ey = np.zeros((nM - 1, nM - 1))

for i in range (0, nM-2) :
    for j in range (0, nM-2) :
        Ex[i,j] = - (V[i,j+1] - V[i,j]) / a
        Ey[i,j] = - (V[i+1,j] - V[i,j]) / a


## Ajustage des matrices V, X et Y

Vajust = np.zeros((nM - 1, nM - 1))

for i in range (0, nM - 1) :
    for j in range (0, nM - 1) :
        Vajust[i,j] = V[i,j]

Xajust = np.zeros((nM - 1, nM - 1))
for i in range (0, nM - 1):
    for j in range (0, nM - 1):
        Xajust[i,j] = X[i,j]

Yajust = np.zeros((nM - 1, nM - 1))
for i in range (0, nM - 1):
    for j in range (0, nM - 1):
        Yajust[i,j] = Y[i,j]


## Affichage des courbes

schema = plt.contour(Xajust, Yajust, Vajust, 40)

cbar = plt.colorbar()
cbar.set_label("Potentiel en Volt")
plt.axis("equal")
plt.title("Equipotentielles et lignes de champ")

schema = plt.streamplot(Xajust, Yajust, Ex, Ey, linewidth=1, density=1)

plt.show()
```