Corrigé

Étude numérique de l'équation de Poisson

Question 1. On a $\dfrac{\partial^2 V}{\partial x^2} = \dfrac{1}{L^2}\dfrac{\partial^2 V}{\partial X^2}$ et $\dfrac{\partial^2 V}{\partial y^2} = \dfrac{1}{L^2}\dfrac{\partial^2 V}{\partial Y^2}$ donc $\Delta V + \dfrac{\rho}{\epsilon_0} = 0 \iff \dfrac{\partial^2 V}{\partial X^2} + \dfrac{\partial^2 V}{\partial Y^2} + \rho' = 0$ avec $\boxed{\rho' = \dfrac{L^2\rho}{\epsilon_0}.}$

Question 2. On a $V(X+h,Y) = V(X,Y) + h\dfrac{\partial V}{\partial X}(X,Y) + \dfrac{h^2}{2}\dfrac{\partial^2 V}{\partial X^2}(X,Y) + O(h^3)$ donc

$$V(X+h,Y) + V(X-h,Y) = 2V(X,Y) + h^2\frac{\partial^2 V}{\partial X^2}(X,Y) + O(h^3).$$

De même, $V(X,Y+h) + V(X,Y-h) = 2V(X,Y) + h^2\dfrac{\partial^2 V}{\partial Y^2}(X,Y) + O(h^2)$. On en déduit :

$$\frac{\partial^2 V}{\partial X^2} + \frac{\partial^2 V}{\partial Y^2} = \frac{V(X_i+h,Y_j) + V(X_i-h,Y_j) + V(X_i,Y_j+h) + V(X_i,Y_j-h) - 4V(X_i,Y_j)}{h^2} + O(h).$$

Question 3. En remplaçant dans l'équation obtenue à la question (1) et en négligeant le terme en $O(h)$ on obtient l'équation :

$$V(i+1,j) + V(i-1,j) + V(i,j+1) + V(i,j-1) - 4V(i,j) + \rho''(i,j) = 0$$

avec $\boxed{\rho'' = h^2\rho' = \dfrac{h^2L^2\rho}{\epsilon_0} = \dfrac{L^2\rho}{N^2\epsilon_0}.}$

Question 4.

```
def nouveau_potentiel(V, rhos, frontiere, i, j):
    if frontiere[i, j]:
        return V[i, j]
    else:
        return (V[i+1, j] + V[i-1, j] + V[i, j+1] + V[i, j-1] + rhos[i, j]) / 4
```

Question 5. Mis à part les cases du bord, le potentiel au rang $k+1$ d'une case dépend du potentiel au rang k de ses quatre voisine. Ainsi, si on modifie le potentiel d'une case, on ne pourra plus modifier le potentiel de ses quatre voisines.

Question 6.

```
def itere_J(V, rhos, frontiere):
    Vc = V.copy()
    N = V.shape[0] - 1
    e = 0
    for i in range(1, N):
        for j in range(1, N):
            V[i, j] = nouveau_potentiel(Vc, rhos, frontiere, i, j)
            e += (V[i, j] - Vc[i, j])**2
    return np.sqrt(e) / N
```

Question 7.

```
def poisson(f_iter, V, rhos, frontiere, eps):
    e = f_iter(V, rhos, frontiere)
    while e > eps:
        e = f_iter(V, rhos, frontiere)
```

Question 8. Si les cases de V sont traitées dans le sens des i et j croissants, les cases de rang $(i-1,j)$ et $(i,j-1)$ sont déjà modifiées lorsque vient le tour de la case de rang (i,j), et les cases $(i+1,j)$ et $(i,j+1)$ pas encore. Les valeurs contenues dans ces cases peuvent donc être utilisées pour appliquer la formule (5), et il n'est donc plus nécessaire d'utiliser les anciennes valeurs stockées dans une copie de V pour appliquée celle-ci. Pour les mêmes raisons il n'est pas nécessaire de modifier la fonction `nouveau_potentiel`.

Question 9.

```
def itere_GS(V, rhos, frontiere):
    N = V.shape[0] - 1
    e = 0
    for i in range(1, N):
        for j in range(1, N):
            vc = V[i, j]
            V[i, j] = nouveau_potentiel(V, rhos, frontiere, i, j)
            e += (V[i, j] - vc)**2
    return np.sqrt(e) / N
```

Question 10.

```
def nouveau_potentiel_SOR(V, rhos, frontiere, i, j, omega):
    if frontiere[i, j]:
        return V[i, j]
    else:
        return (1 - omega) * V[i, j] + omega * (V[i+1, j] + V[i-1, j] + V[i, j+1] +
                                                V[i, j-1] + rhos[i, j]) / 4
```

Question 11.

```
def itere_SOR(V, rhos, frontiere):
    N = V.shape[0] - 1
    omega_opt = 2 / (1 + np.pi / N)
    e = 0
    for i in range(1, N):
        for j in range(1, N):
            vc = V[i, j]
            V[i, j] = nouveau_potentiel_SOR(V, rhos, frontiere, i, j, omega_opt)
            e += (V[i, j] - vc)**2
    return np.sqrt(e) / N
```

Question 12. La fonction `nouveau_potentiel_SOR` est de complexité constante, donc la fonction `itere_SOR` est de complexité quadratique $O(N^2)$. D'après l'énoncé, pour la valeur ω_{opt} le nombre d'itérations nécessaire pour atteindre la précision ϵ est linéaire en $O(N)$, donc la complexité totale de cet appel est en $O(N^3)$.

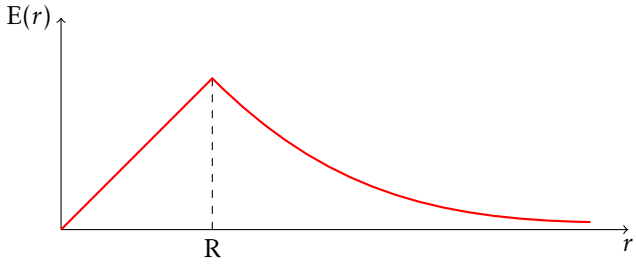
Question 13.

a) Les charges sont invariantes par rotation autour de l'axe (Oz) et par translation le long de cet axe, donc le champ $\vec{E}$ est indépendant de θ et de z. Par ailleurs en tout point M de l'espace les plans $(M,\vec{u_r},\vec{u_\theta})$ et $(M,\vec{u_r},\vec{u_z})$ sont des plans de symétrie des charges donc $\vec{E}(M)$ appartient à ces plans. Ainsi, $\vec{E} = E(r)\vec{u_r}$.
Les équipotentielles sont perpendiculaires au champ dont sont des cylindres d'axe Oz.

b) Si on considère une portion de longueur ℓ de ce fil le théorème de Gauss donne (en considérant le cylindre de rayon r et de hauteur ℓ) :

- si $r \geqslant R$, $2\pi r\ell E(r) = \dfrac{\pi R^2 \rho \ell}{\epsilon_0}$ soit $E(r) = \dfrac{\rho R^2}{2\epsilon_0 r}$;
- si $r \leqslant R$, $2\pi r\ell E(r) = \dfrac{\pi r^2 \rho \ell}{\epsilon_0}$ soit $E(r) = \dfrac{\rho r}{2\epsilon_0}$.

On obtient l'allure suivante :



c) La valeur maximale du champ est obtenue pour $r = \mathrm{R}$ et donne $\mathrm{E}(\mathrm{R}) \approx 28{,}2 \cdot 10^3\ \mathrm{V \cdot m^{-1}}$.

Question 14. Le point de coordonnées (x, y) est dans le cercle si et seulement si $(x - x_c)^2 + (y - y_c)^2 \leqslant \mathrm{R}^2$, d'où la fonction :

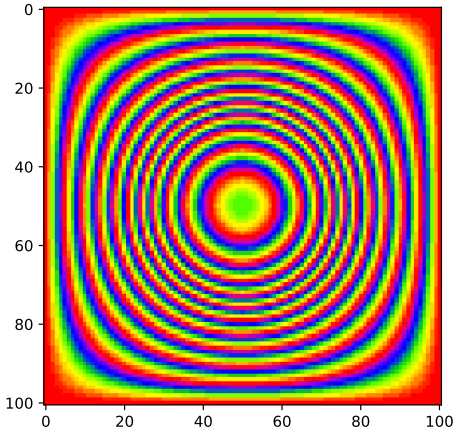
```
def dans_cylindre(x, y, xc, yc, R):
    return (x - xc)**2 + (y - yc)**2 <= R**2
```

Question 15. Le couple $(\mathrm{X}_i, \mathrm{Y}_j)$ correspond au point de coordonnées $x = \dfrac{\mathrm{L}i}{\mathrm{N}}$, $y = \dfrac{\mathrm{L}j}{\mathrm{N}}$, le centre du cylindre a pour coordonnées $x_c = \dfrac{\mathrm{L}}{2}$, $y_c = \dfrac{\mathrm{L}}{2}$ et le rayon a pour valeur $\mathrm{R} = \dfrac{\mathrm{L}}{4}$. Quant à la frontière, elle est constituée des couples (i, j) tels que $i \in \{0, \mathrm{N}\}$ ou $j \in \{0, \mathrm{N}\}$.

```
for i in range(N+1):
    for j in range(N+1):
        if dans_cylindre(i/N, j/N, 1/2, 1/2, 1/4):
            rhos[i, j] = L**2 * rho / eps0 / N**2
        if i == 0 or i == N or j == 0 or j == N:
            frontiere[i, j] = True
```

Question 16.

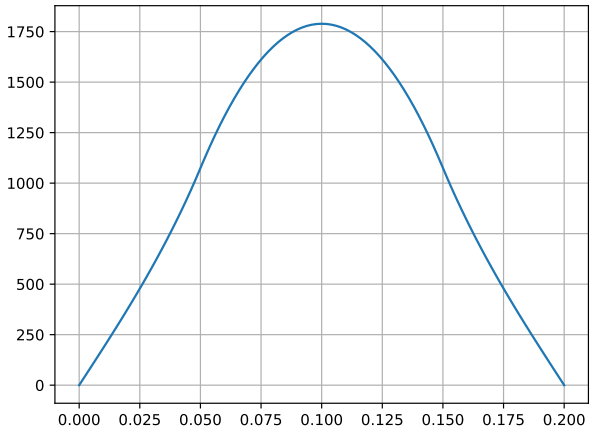
```
poisson(itere_SOR, V, rhos, frontiere, 1e-5)
plt.imshow(V, cmap='prism')
plt.show()
```



Question 17.

Il s'agit de tracer le graphe de la fonction $x \mapsto \mathrm{V}(x, \mathrm{L}/2)$ pour $x \in [0, \mathrm{L}]$.

```
X = [L * i / N for i in range(N+1)]
P = [V[i, N//2] for i in range(N+1)]
plt.plot(X, P)
plt.grid()
plt.show()
```



Question 18. On a $\vec{E} = -\text{grad}\, V$ donc $E_x = -\dfrac{\partial V}{\partial x} = -\dfrac{1}{L}\dfrac{\partial V}{\partial X}$.

Or $V(X+h,Y) = V(X,Y) + h\dfrac{\partial V}{\partial X}(X,Y) + O(h^2)$ donc $\dfrac{\partial V}{\partial X}(X,Y) = \dfrac{V(X+h,Y) - V(X,Y)}{h} + O(h)$.

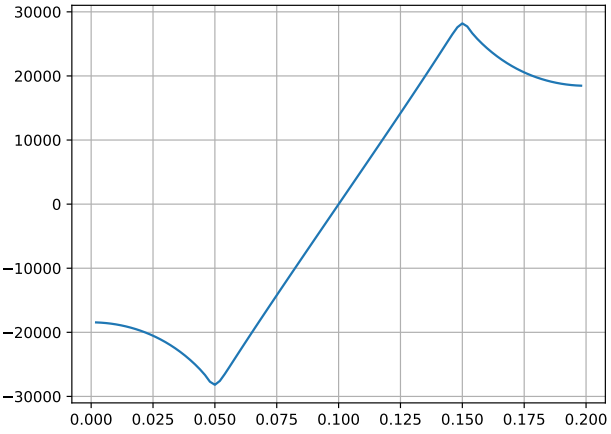
On peut donc approcher $E_x(i,j)$ par $\dfrac{V(i,j) - V(i+1,j)}{hL}$. Il existe néanmoins une meilleure approximation, en poussant le développement limité plus loin :

$V(X+h,Y) = V(X,Y) + h\dfrac{\partial V}{\partial X}(X,Y) + \dfrac{h^2}{2}\dfrac{\partial^2 V}{\partial X^2}(X,Y) + O(h^3)$ et $V(X-h,Y) = V(X,Y) - h\dfrac{\partial V}{\partial X}(X,Y) + \dfrac{h^2}{2}\dfrac{\partial^2 V}{\partial X^2}(X,Y) + O(h^3)$ donc

$$\frac{\partial V}{\partial X}(X,Y) = \frac{V(X+h,Y) - V(X-h,Y)}{2h} + O(h^2)$$

ce qui conduit à approcher $E_x(i,j)$ par $\dfrac{V(i-1,j) - V(i+1,j)}{2hL}$ (pour $1 \leqslant i \leqslant N-1$).

```
X = [L * i / N for i in range(1, N)]
E = [(V[i-1, N//2] - V[i+1, N//2]) * N / 2 / L for i in range(1, N)]
plt.plot(X, E)
plt.grid()
plt.show()
```



Question 19. Il s'agit cette fois d'un cylindre creux chargé. On remplace la fonction `dans_cylindre` par la fonction :

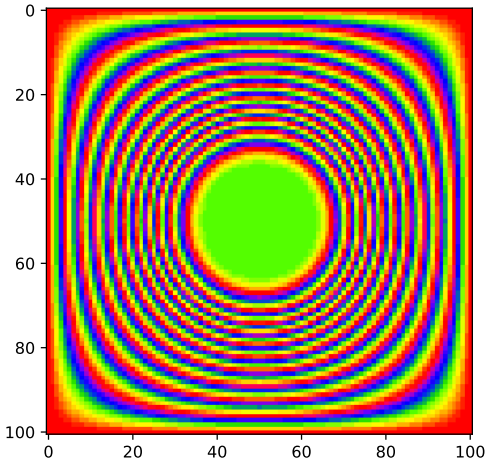
```
def dans_couronne(x, y, xc, yc, r, R):
    return (x - xc)**2 + (y - yc)**2 <= R**2 and (x - xc)**2 + (y - yc)**2 >= r**2
```

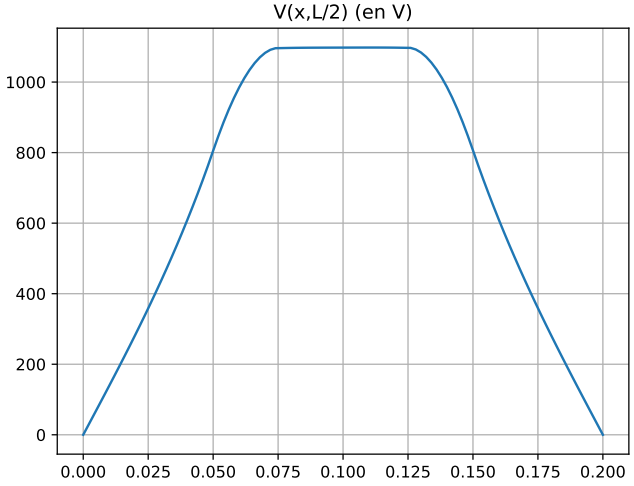
puis on redéfinit les valeurs du tableau `rhos` :

```
rhos = np.zeros((N+1, N+1))

for i in range(N+1):
    for j in range(N+1):
        if dans_couronne(i/N, j/N, 1/2, 1/2, 1/8, 1/4):
            rhos[i, j] = L**2 * rho / eps0 / N**2
```

Il reste à ré-exécuter les scripts précédents pour obtenir les graphes ci-dessous :


0
20
40
60
80
100
0
20
40
60
80
100


V(x,L/2) (en V)
1000
800
600
400
200
0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200