Déviation électrostatique

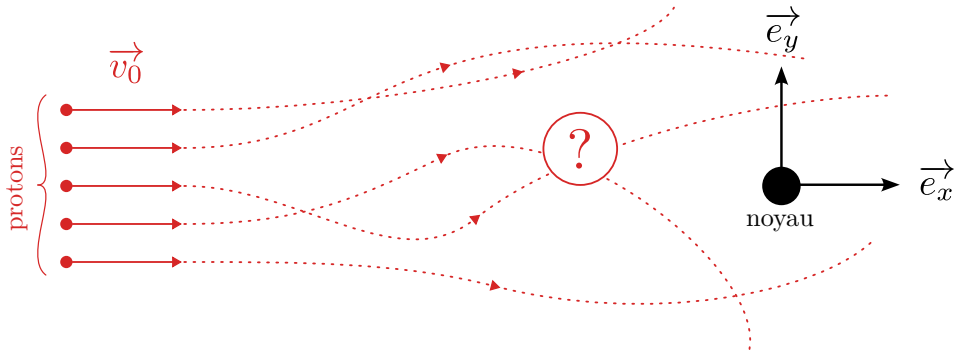
Capacités exigibles

- À l'aide d'un langage de programmation, obtenir des trajectoires d'un point matériel soumis à un champ de force centrale conservatif.

I Documents

Document 1 : Déviation électrostatique

On envoie sur le noyau d'un atome, un faisceau de protons de vitesse initiale $\overrightarrow{v_0}$. On cherche à étudier comment ces derniers vont être déviés par la présence du noyau.



La force d'interaction électrostatique entre deux charges q_1 et q_2, positionnées respectivement en M_1 et M_2 est donnée par la loi de Coulomb :

$$\overrightarrow{F} = \frac{q_1 q_2}{4\pi\epsilon_0 \, ||\mathrm{M}_1\mathrm{M}_2||^3} \overrightarrow{\mathrm{M}_1\mathrm{M}_2}$$

Avec $\epsilon_0 = 8.85 \cdot 10^{-11}\,\mathrm{F}\cdot\mathrm{m}^{-1}$ la permittivité du vide. Rappelons également la valeur de la charge élémentaire, ainsi que la masse d'un proton :

$$e = 1.60 \cdot 10^{-19}\,\mathrm{C}$$
$$m = 1.67 \cdot 10^{-27}\,\mathrm{kg}$$

II Énoncé

Dans tout ce problème on se place en coordonnées cartésiennes à deux dimensions (x, y), centrées sur le noyau (fixe). On notera Z le numéro atomique du noyau.

(1) Exprimer $\ddot{x}$ et $\ddot{y}$ en fonction de Z, e, m, ϵ_0, x et y.

(2) Montrer que le problème peut être ramené à l'étude d'une équation différentielle d'ordre 1 de la forme

$$\dot{U} = H(U, t)$$

Avec U un vecteur construit à partir des coordonnées cartésiennes, dont on déterminera le nombre de composantes, en donnant leurs définitions. Expliciter l'action de H sur ce vecteur.

(3) Écrire une fonction `coulomb(U, t)` représentant l'application $H(U, t)$.

(4) Créer une liste `temps` contenant toutes les valeurs des instants sur lesquels on va résoudre l'équation différentielle. On ira de $t_0 = 0$ à $t_{\max} = 2 \cdot 10^{-14}\,\mathrm{s}$ pour $N = 1000$ valeurs.

(5) Créer une liste `y0` de $n = 300$ valeurs d'ordonnées initiales (on va simuler n trajectoires de protons d'un coup). Ces positions seront choisies aléatoirement, selon une loi uniforme dans l'intervalle $[-x_{\min}, x_{\min}]$ avec $x_{\min} = 1\,\mathrm{nm}$.

(6) Écrire quelques lignes permettant de calculer et stocker dans une liste `trajectoires` les N valeurs prises par $x(t)$ et $y(t)$ pour chacun des protons envoyés. Cette variable `trajectoires` contiendra donc n tuples composés de deux listes de N valeurs : l'une pour x, l'autre pour y.

NB : On prendra $-x_{\min}$ comme abscisse initiale et $\overrightarrow{v_0} = v_0\,\overrightarrow{e_x}$ avec $v_0 = 1 \cdot 10^5\,\mathrm{m \cdot s^{-1}}$ comme vitesse initiale.

(7) Écrire quelques lignes afin d'afficher ces trajectoires. Pour faciliter la lecture, on pourra faire apparaître le noyau central, et rendre les repère orthonormé grâce aux lignes ci-dessous :

```
plt.scatter([0], [0], color='black')
plt.gca().set_aspect('equal')
plt.show()
```

(8) Faire varier les paramètres de la simulation, comme par exemple le numéro atomique Z du noyau, la vitesse initiale...

III Annexe

Pour ce TP vous aurez besoin des modules `numpy` et `pyplot` :

```
import numpy as np
import matplotlib.pyplot as plt
```

Vous aurez ensuite à utiliser les fonctions suivantes :

`slicing`

Il est très facile d'extraire et de modifier certaines valeurs d'un tableau `numpy`.
Exemple : Si on a une matrice 2D notée `M` :

```
M = array([[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]])
```

On peut extraire des valeurs :

```
>>> M[0]    # Renvoie la première ligne
array([1, 2, 3])
>>> M[0, 2]     # Renvoie la valeur en première ligne, troisième colonne
3
>>> M[:, 1]     # Renvoie toutes les valeurs de la deuxième colonne
array([2, 5, 8])
```

On remarque donc que les deux points ":" signifient "toutes les valeurs".

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

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

Exemple :

```
>>> np.linspace(0, 1, 11)
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

`np.random.random(n)`

Créer un tableau 1D de `n` valeurs aléatoires choisies uniformément entre 0 et 1 :
Exemple :

```
>>> np.random.random(4)
array([0.85568752, 0.94219879, 0.01725846, 0.2365631 ])
```

odeint

Dans la bibliothèque `scipy.integrate`, se trouve une fonction `odeint` permettant de résoudre numériquement des équations différentielles, et de manière bien plus précise que la méthode d'Euler :

```
from scipy.integrate import odeint

...

solutions = odeint(F, Y0, X)
```

Les arguments doivent être les suivants :

- `F` la fonction qui encode l'équation différentielle (ici par exemple une équation d'ordre n sur une fonction $y : x \to y(x)$) :
$$Y'(x) = F(Y(x), x)$$
Elle doit prendre elle-même en entrée :
  - `Y` un vecteur de taille n (liste python) contenant pour un x donné, les valeurs de $y(x)$ et ses $n-1$ premières dérivées :
$$Y(x) = \begin{pmatrix} y(x) \\ y'(x) \\ \vdots \\ y^{(n-1)}(x) \end{pmatrix}$$
  - `x` l'abscisse à laquelle on mène le calcul.

  Elle doit renvoyer le vecteur dérivé (toujours sous forme de liste python) :
$$Y'(x) = \begin{pmatrix} y'(x) \\ y''(x) \\ \vdots \\ y^{(n)}(x) \end{pmatrix}$$
- `Y0` les conditions initiales
$$Y(x_1) = \begin{pmatrix} y(x_1) \\ y'(x_1) \\ \vdots \\ y^{(n-1)}(x_1) \end{pmatrix}$$
- `X` La liste des abscisses sur lesquelles on va mener l'intégration :
$$X = [\, x_1, x_2, \ldots, x_N \,]$$
Souvent on utilise la fonction `np.numpy` (cf. précédemment), pour créer cette liste.

Alors `odeint` renvoie un tableau `numpy` de taille `N` $\times$ `n` avec

- `N` le nombre de points choisis pour intégrer l'équation
- `n` l'ordre de l'équation différentielle

$$\texttt{odeint} \implies \begin{pmatrix} y(x_1) & y'(x_1) & \ldots & y^{(n-1)}(x_1) \\ y(x_2) & y'(x_2) & \ldots & y^{(n-1)}(x_2) \\ \vdots & \vdots & & \vdots \\ y(x_N) & y'(x_N) & \ldots & y^{(n-1)}(x_N) \end{pmatrix}$$