Rapport de stage

Abdessamad Mbardi

Master 1 Physique fondamentale et applications

Simulation d'une image d'un nuage d'ions Be^+/H_2^+ piégés

sous la direction de

Laurent HILICO

Laboratoire Kastler Brossel

Sorbonne Université

Table des matières

1 Introduction

A partir des images de fluorescence obtenues expérimentalement d'un nuage constitué des ions H_2^+ et Be^+ piégés par un piège à ion radiofréquence, le problème est de détecter puis compter le nombre d'ions H_2^+ présents dans le nuage d'ions que l'on ne peut pas voir car ils ne fluorescent pas. L'objectif principal de ce stage est d'écrire un code de simulation d'une image d'un nuage constitué d'un nombre connu d'ions H_2^+ et Be^+ piégés par un piège de Paul linéaire, pour écrire ensuite un autre code qui, à partir de cette image, détecte puis compte le nombre d'ions H_2^+ présents dans le nuage. Au paragraphe 2, je présente le principe de fonctionnement des pièges à ions radiofréquences, la notion de diagramme de stabilité des trajectoires, pour finir au paragraphe 3, à expliquer comment j'ai pu simuler une image du nuage d'ions et détecter les ions H_2^+ invisible.

2 Piège à ion radiofréquence

Pour confiner une particule chargé en trois dimensions, il faut établir un potentiel de piégeage présentant un minimum au point de l'espace où on veut piéger la particule. Un tel potentiel peut s'écrire, au voisinage du minimum situé en $x = y = z = 0$ sous la forme :

$$U = Ax^2 + By^2 + Cz^2 + U_{min} \tag{1}$$

avec A, B, C des constantes. Si le champ est de nature électrostatique, il vérifie $\Delta U = 0$ et donc les constantes A, B, C vérifient $A+B+C = 0$. Elle ne peuvent pas être toutes les trois strictement supérieures à 0, donc le potentiel est nécessairement déconfinant dans une des trois directions de l'espace. En 1958, l'idée de Wolfgang Paul [5] a été d'utiliser un potentiel dépendant sinusoïdalement du temps, de la forme $U(t) = \gamma(t)(Ax^2 + By^2 + Cz^2)$ et de montrer qu'il existe des conditions pour lesquelles les ions sont piégés. Deux configurations sont couramment utilisées :

$$A = B = 1 \text{ et } C = -2 \tag{2}$$

$$A = -B = 1 \text{ et } C = 0 \tag{3}$$

Les solutions (2) et (3) correspondent respectivement aux pièges de Paul hyperbolique et piège de Paul linéaire. Dans la suite, nous allons nous intéresser au piège linéaire uniquement.

2.1 Piège de Paul linéaire

Ce piège est constitué de quatre conducteurs segmentés placés parallèlement, équidistants et qui sont à la même distance R de l'axe des z comme illustré dans la figure (1). Les électrodes RF sont alimentées avec une tension $V_{RF} = U_0 + V_0 \cos(\Omega t)$ tandis que les électrodes endcap sont alimentées avec un voltage constant U_1.

2.2 Le potentiel de piégeage

Pour des électrodes très longues ($L >> R$), le potentiel créé dans le plan transverse xy est donné par :

$$\Phi(x, y, t) = \frac{x^2 - y^2 + R^2}{2R^2}(U_0 + V_0 cos\Omega t) \tag{4}$$

Le champ électrique $\vec{E} = -g\vec{r}ad\Phi$ est :

$$\vec{E} = \frac{U_0 + V_0 \cos(\Omega t)}{R^2}(-x\vec{x} + y\vec{y}) \tag{5}$$

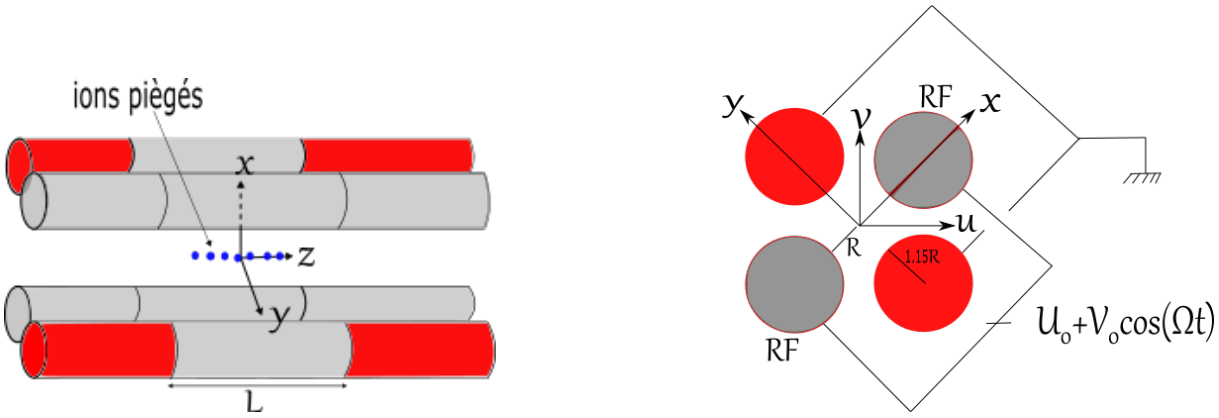


FIGURE 1 – *Gauche : Piège de Paul linéaire, constitué des électrodes RF et centrales (gris) et quatre électrodes endcap (rouge) pour le confinement le long de la direction z. Droite : Piège vu selon l'axe oz. Les traits fins indiquent les connections électriques : les électrodes rouges sont à la masse , la tension* $U_0 + V_0 cos(\Omega t)$ *est appliquée aux deux autres électrodes,R=3.5mm.*

La force exercée sur les ions de charge Q par le piége est :

$$\vec{F} = Q\vec{E} \tag{6}$$

2.3 Equations du mouvement

Les équations du mouvement des ions soumis à la force $\vec{F}$ dans le plan xy sont données par :

$$\ddot{x} = -\frac{Q}{mR^2}(U_0 + V_0 \cos \Omega t)x \tag{7}$$

$$\ddot{y} = \frac{Q}{mR^2}(U_0 + V_0 \cos \Omega t)y. \tag{8}$$

On les écrit sous la forme d'équations de Mathieu [1] :

$$\frac{d^2x}{d\tau^2} + (a_x + 2q_x \cos \Omega t)x = 0 \tag{9}$$

$$\frac{d^2y}{d\tau^2} + (a_y + 2q_y \cos \Omega t)y = 0. \tag{10}$$

où $a_x = -a_y = \frac{4QU_0}{m\Omega^2R^2}$, $q_x = -q_y = \frac{2QV_0}{m\Omega^2R^2}$ sont les paramètres de stabilité et $\tau = \frac{\Omega t}{2}$ le temps en unité réduite.

Dans l'approximation adiabatique ($|a_x|, |a_y| << 1$ et $|q_x|, |q_y| << 1$), on montre que les solutions des équations (9) et (10) sont stables et ont pour forme :

$$x(t) = x_0 \cos(\omega_x t + \varphi_x)(1 + \frac{q_x}{2} \cos \Omega t) \tag{11}$$

$$y(t) = y_0 \cos(\omega_y t + \varphi_y)(1 + \frac{q_y}{2} \cos \Omega t). \tag{12}$$

où $\omega_x = \frac{\Omega}{2}\sqrt{\frac{q_x^2}{2} + a_x}$ et $\omega_y = \frac{\Omega}{2}\sqrt{\frac{q_y^2}{2} + a_y}$.

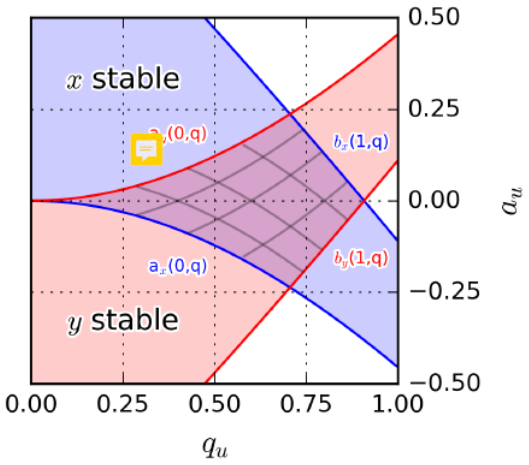


FIGURE 2 – *Diagramme de stabilité suivant x et y pour un piège linéaire, la partie hachurée correspond aux solutions stables.*[4].

	$U_0(V)$	$V_0(V)$	a_x	q_x
H_2^+	0.1	200	$2{,}36.10^{-4}$	0.236
Be^+	0.1	200	$5{,}02.10^{-5}$	0.050
H_2^+	9	450	0.0212	0.5312
Be^+	9	450	$4.722.10^{-3}$	0.11

TABLE 1 – *Quelques valeurs des paramètres de stabilité pour les ions H_2^+ et Be^+ avec $\Omega = 2\pi \times 13.5 MHz$.*

D'après les solutions (11) et (12), le mouvement d'un ion dans le plan (xy) est la superposition d'un mouvement séculaire (lent) de fréquences ω_x et ω_y et d'un micro mouvement (rapide) de fréquence Ω.
Notons que le choix de U_0, V_0 et U_1 permet de varier le potentiel effectif de piégeage dans chacune des directions, et donc d'obtenir des nuages cylindriques (symétriques en x/y) ou bien plats. Nous donnons dans le tableau (1(quelques valeurs du couple (V_0,U_0) et Ω qui représentent un point de fonctionnement typique du piège pour les ions H_2^+ et Be^+ car elles vérifient l'approximation adiabatique et donnent bien des solutions stables (figure 2) suivant x et y comme illustré dans les figures (3) et (4), où les solutions de l'équation de Mathieu sont calculées avec le logiciel Mathematica (Voir l'annexe).

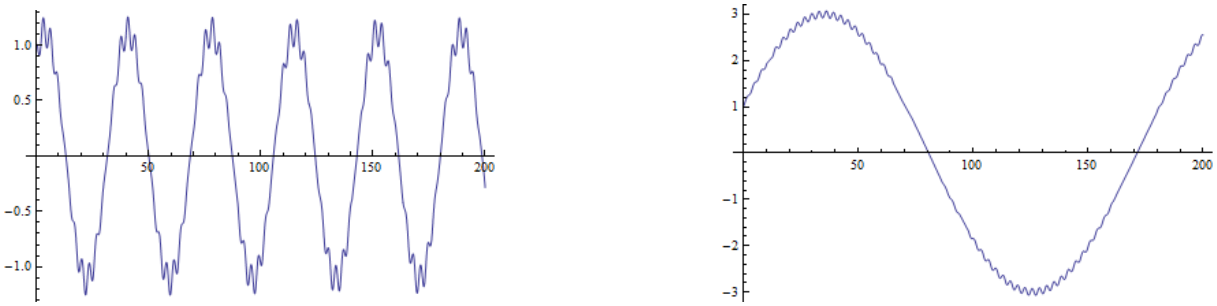


FIGURE 3 – *Gauche : trajectoire suivant x d'un ion H_2^+ avec $a_x = 2,36.10^{-4}$ et $q_x = 0.236$, Droite : trajectoire d'un ion Be^+ avec $a_x = 5,02.10^{-5}$ et $q_x = 0.050, \Omega = 2\pi \times 13.5$* MHz.

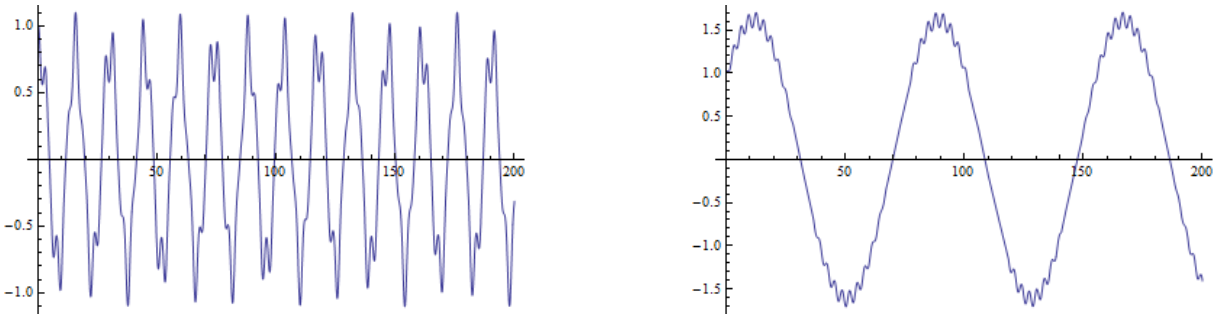


FIGURE 4 – *Gauche : trajectoire suivant x d'un ion* H_2^+ *avec* $a_x = 0.0212$ *et* $q_x = 0.5312$. *Droite : trajectoire d'un ion* Be^+ *avec* $a_x = 4.722.10^{-3}$ *et* $q_x = 0.11, \Omega = 2\pi \times 13.5$ MHz.

3 Simulation d'une image

3.1 Simulation numérique du nuage : Code Fortran

Pour construire une image du nuage, nous utilisons un code écrit en Fortran [2] qui simule le comportement d'un ensemble d'ions soumis à la force du piège de Paul linéaire, à la répulsion mutuelle coulombienne entre les ions et à l'interaction avec le laser. Nous donnons ici les principales fonctions de ce code :

1. **Stockage** dans la mémoire pour chaque ion, la position, la vitesse, l'accélération et l'état interne de chaque atome (état fondamental ou état excité).
2. **Calcul de la force de coulomb** : ce morceau de code calcule la force de coulomb exercé sur la particule i par les autres ions $j \neq i$.

$$\vec{F_i} = \sum_{j \neq i} \vec{F_{ij}} = \sum_{j \neq i} \frac{q_i q_j \vec{r_{ij}}}{4\pi\epsilon_0 r_{ij}^3}. \tag{13}$$

3. **Force du piège de Paul linéaire** : donnée par l'équation (6).
4. **Interaction avec le laser de refroidissement** : Cette partie utilise une approche en termes de probabilité par unité de temps pour décider si l'ion Be^+ va absorber un photon pour passer à l'état excité, émettre un photon dans le mode du laser par émission stimulée ou spontanée. En cas d'absorption ou d'émission d'un photon par un ion, la vitesse est modifiée de $\pm\frac{\hbar\vec{k}}{m}$ où $\vec{k}$ est l'impulsion du photon (+ si absorption, - si émission).
5. **Initialisation des ions** : Ce morceau sert à initialiser les ions tout en passant par quatre étapes :
   - L'initialisation des ions dans une distribution cylindrique uniforme des vitesses initiales ayant une distribution thermique de température T.
   - Relaxation non physique : Cette phase non physique permet de relaxer les ions vers leur positions d'équilibre dans un nuage en appliquant une force de frottement non physique de type $-k\vec{v}$, la force du piège et la force coulombienne, seule la force du laser est omise.
   - Relaxation physique avec le laser : le code supprime la force de frottement non physique et prend en compte l'interaction avec le laser de refroidissement.
6. **Refroidissement sympathique** : Maintenant, on peut dire que le nuage a été bien préparé. Le code passe donc à la simulation de refroidissement sympathique

des ions qui n'ont subit aucune force durant la préparation du nuage (les ions H_2^+ pour notre cas).

7. **La résolution** des équations différentielles du mouvement des ions est faite en utilisant un algorithme Verlet-vitesse [3] modifié permettant d'utiliser un pas de temps variable.

FIGURE 5 – *Visualisation d'un nuage plat constitué de* $100Be^+/10H_2^+$ *via le logiciel VMD (Visualizing Molecular Dynamics),* $U_0 = 9V, V_0 = 450$ *et* $\Omega = 2\pi \times 13.5$MHz.

3.2 Le fichier *.photon

On a besoin de modéliser l'image du nuage d'ions sur la caméra d'observation. L'idée est d'utiliser l'optique géométrique pour construire une image du nuage à travers une lentille convergente. Dans le code fortran, j'ai rajouté un sous programme (voir l'annexe) qui, lors de l'exécution, sélectionne les photons d'émission spontanée émis qui passent à travers la lentille de collection de la fluorescence (voir figure 6) et écrit dans un fichier le numéro de l'ion, sa position(x_B, y_B, z_B) et les angles (θ, ϕ) sous lesquels le photon est émis. En effet, l'angle sphérique doit vérifier $\theta_B < \theta_{max}$ où θ_{max} dépend des coordonnées du point B et des dimensions de la lentille.

3.3 Simulation d'une image des ions piégés

La figure (6) montre le chemin optique parcouru par un photon émis dans la direction donnée par le vecteur $\overrightarrow{BM}$ par un ion situé en B afin qu'il arrive à la caméra en passant par une lentille convergente de focale f centrée en C. De manière ensuite à pouvoir reconstruire l'image du nuage, j'ai écrit un code python qui, pour chaque photon enregistré dans le fichier *.photon, calcule les coordonnées (u_p, v_p, z_p) du point P, puis en déduit les indices (i, j) du pixel de la caméra dans lequel il est situé le point P et cela construit finalement dans une figure une image du nuage d'ions.

3.3.1 Détails du code

(Voir l'annexe)

- Notre code contient une fonction pixel qui prend comme arguments x_B, y_B, z_B, θ_B et ϕ_B du photon émis par un ion placé dans le point B, $xscreen$ position du plan de la caméra, pour calculer et retourner les indices (i, j) du pixel dans lequel il est situé $P(u_p, v_p, z_p)$ le point d'arrivé du photon à la caméra.
- Il applique cette fonction à chaque photon enregistré dans le fichier *.photon.
- Il crée un tableau carré représentant les pixels de la caméra et indiquant le nombre de photons reçus par chaque pixel.
- Finalement, il trace dans une figure les différents éléments du tableau pour donner une image de notre nuage.

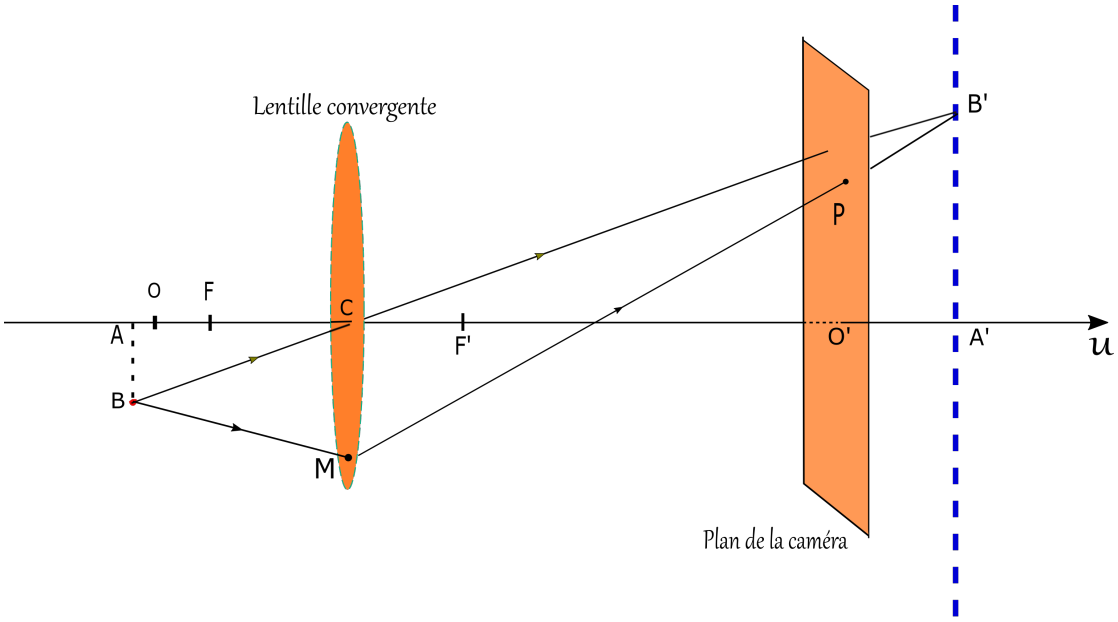


FIGURE 6 – *Banc optique : O est le centre du piége.*

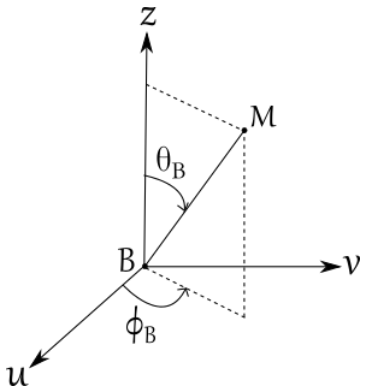


FIGURE 7 – *Définition des angles sphériques θ_B et ϕ_B du vecteur $\overrightarrow{BM}$ dans le repère $(B, \vec{u}, \vec{v}, \vec{z})$.*

3.3.2 Résultats

la figure (8) montre les images des nuages plats d'ions obtenues avec les tensions $U_0 = 450V, V_0 = 9V$ et $\Omega = 2\pi \times 13.5$MHz. Les images à droite et à gauche correspondent respectivement aux nombres d'ions $100Be^+/10H_2^+$ et $110Be^+$ (Be^+ pur). Remarquons dans l'image à gauche l'existence des trous qui correspondent aux ions H_2^+ qu'on ne peut pas les voir car ces ions refroidis sympathiquement sont insensibles au laser de refroidissement des ions Be^+, cela est dû au fait que l'ion H_2^+ n'a aucune transition dipolaire électrique permise.

3.4 Détermination du nombre d'ions H_2^+ dans le nuage

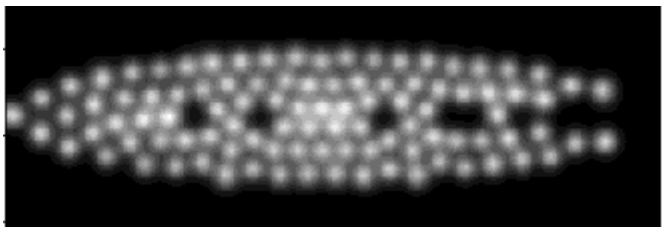
3.4.1 Code python

Afin de pouvoir déterminer le nombre d'ions H_2^+ qui existent dans le nuage, j'ai écrit un code python qui détecte dans un premier temps les trous correspondants, puis donne leur nombre et qu'on va l'appliquer à l'image du nuage qu'on a dans la figure (8) à gauche pour laquelle on connait le nombre d'ions H_2^+. Ce code contient la fonction **number-ions-**

FIGURE 8 – *Gauche : Image produite avec 190254 photons, du nuage constitué de* $100Be^+/10H_2^+$ *, Droite : Image produite avec 216784 photons, du nuage constitué de* $110Be^+$.

flat qui prend comme argument le dossier qui contient le fichier image qu'on va utiliser et qui se déroule en trois étapes (voir l'annexe) :

- **Floutage gaussien** en utilisant la fonction cv2.GaussianBlur



- **"Seuillage" ou "Thresholding"** : cette étape est très importante pour la détection. La fonction utilisée est cv2.threshold contenue dans la bibliothèque cv2, elle permet d'avoir des surfaces enlacées par des contours fermés qui correspondent aux trous.

- **Inversion** : Noir $<--->$ Blanc

- **Détection** : Le code peut maintenant détecter les trous qui existent dans l'image en utilisant la fonction cv2.SimpleBlobDetector contenue dans cv2. Dans le code, j'ai choisi le filtrage par surface et par couleur.

3.4.2 Résultats et Difficultés

la figure (9) montre les trous correspondants aux ions H_2^+ qu'on peut détecter avec le code. Dans ce cas le code affiche un nombre de 5 qui ne correspond pas au nombre réel d'ions présents dans le nuage. Mais remarquons que parmi les cercles en rouge, il y en a un qui représente 2 ions. Les 4 ions qui restent sont visibles sur la figure 5, mais je n'ai pas pu les détecter car ils ne sont pas entourés du noir (contours ouverts). Ce que j'ai fait montre qu'il est possible de détecter les ions qui sont à l'intérieur du nuage. Pour les autres, il faut trouver une autre idée car en effet, le contour non fermé ne permet pas à l'analyse d'image.
A ce moment là, la physique peut aider. On sait que les ions sont poussés à droite par le laser et qu'on a une petite suite d'ions H_2^+ à mettre. On peut dire que les ions B_e^+ qui sont à droite ne peuvent être en équilibre que si on place des ions entre eux. Bien sûr, c'est plus facile à dire qu'à faire !

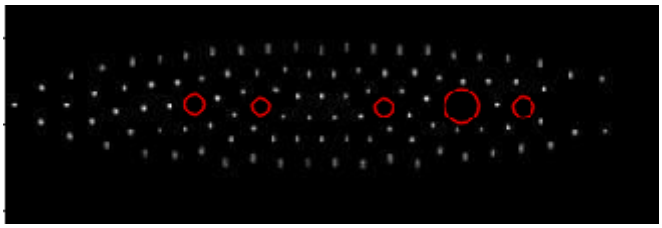


FIGURE 9 – *les différents trous détectés par le code python (entourés par des cerles rouge).*

4 Conclusion

Après avoir introduit la théorie de la physique étudiée, nous avons motivé les besoins de simulation d'une image du nuage d'ions. Au paragraphe 2 nous avons simulé une image du nuage en utilisant le code python pour finir ensuite dans le paragraphe 3, à présenter un autre code qui détecte les trous. Pour l'étude de la spectroscopie de l'ion H_2^+, il est très important de connaître le nombre d'ions H_2^+ présent dans le nuage. Le nombre de trous qu'on détecté ne correspond pas au nombre réel d'ions H_2^+ présents dans le nuage, donc il faut donc chercher une autre méthode numérique ou physique. On va se poser la question si on peut dans le futur déterminer le nombre d'ions qui ne sont pas inclus dans les Be^+, car cela nous permettra de déduire le nombre des ions H_2^+.

5 Annexe

5.1 Fichier *.photon

c'est la partie ajoutée au code fortran qui permet la création du fichier *.photon.

```
subroutine save_photon(indice,theta_loc,phi_loc)
use laser_cooling
use coordinates
use cte
implicit none
integer indice
double precision theta_loc,phi_loc
double precision x_M,y_M,z_M,eta

eta=(x_lens-u(indice-5))/(sin(theta_loc)*cos(pi/4-phi_loc))
! calcul de M
if ((eta).ge.0) then
x_M=x_lens
y_M=u(indice-4)+eta*sin(theta_loc)*sin(pi/4-phi_loc)
z_M=u(indice-3)+eta*cos(theta_loc)
if (((y_M**2+z_M**2).le.(lens_diameter/2)**2).and.(eta.gt.0)) then

  write(127,'(i4,2x,5(e26.19,2x))')indice/6,u(indice-5),u(indice-4),u(indice-3),
theta_loc,pi/4-phi_loc
endif
endif
return
end
```

5.2 Calcul des solutions de l'équation de mathieu

c'est le programme Mathematica qui résout une équation de Mathieu.

```
a = 0.04249
q = 1.0624
s = NDSolve[{u''[t] + (a + 2 q Cos[2 t]) u[t] == 0, u[0] == 1,
   u'[0] == 0.1}, u, {t, 0, 200}]
Plot[Evaluate[{u[t]} /. s], {t, 0, 200}, PlotStyle -> Automatic]
```

5.3 Programme Python de calcul de l'image

```
import os
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import scipy.misc

dirname=os.chdir("Z:/dyn-molec/v81/100Be10H2-450V-9V")
filename="run_100Be_10H2p_12341.photon"
data=np.loadtxt(filename)
data=data[:,1:]

def pixel(xB,yB,zB,thetaB,phiB,xscreen):
```

```
    a=1.3e-5
    f=0.025
    Gr=-7
    CO=f*(1-Gr)/Gr
    OC=-CO

    ''' calcul des coordonnées du point M '''
    lamb = (OC-xB)/(math.sin(thetaB)*math.cos(phiB))
    xM=OC
    yM=yB+lamb*math.sin(thetaB)*math.sin(phiB)
    zM=zB+lamb*math.cos(thetaB)
    #print("M:",xM,yM,zM)

    ''' calcul de xAprime '''
    xAprime=OC+((xB-OC)*f)/((xB-OC)+f)

    ''' calcul de Bprime : intersection de (BC) avec le plan passant par Aprime '''
    t=(xAprime-OC)/(OC-xB)
    xBprime=xAprime
    yBprime=-yB*t
    zBprime=-zB*t

    '''calcul des coordonnées de P '''
    v=(xscreen-xM)/(xBprime-xM)
    xP=xscreen
    yP=yM+(yBprime-yM)*v
    zP=zM+(zBprime-zM)*v
    #print("P:",xP,yP,zP)
    i=math.floor((zP/a)+0.5)
    j=math.floor((yP/a)+0.5)
    return i,j

dim = 380
camsensor = np.zeros((dim,dim))
for d in data[0:]:
    #print(d)
    x,y=pixel(d[0],d[1],d[2],d[3],d[4],0.22837280001120017)
    x = x + int(dim/2)
    y = y + int(dim/2)
    #print(x,y)
    if(x<dim and y<dim and x>=0 and y>=0):
        camsensor[x,y] =  camsensor[x,y] + 1

plt.figure(figsize=(10,10))
print(filename)
print(filename.replace('photon','png'))
scipy.misc.imsave(filename.replace('photon','png'), camsensor.T)
plt.imshow(camsensor.T,vmin=9,vmax=10,interpolation='nearest', cmap=cm.Greys)
plt.show()
```

5.4 Programme Python de calcul du nombre d'ions H_2^+

Le code python qui détecte puis compte le nombre de trous dans l'image.

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

import os
import glob
import re
import cv2
import scipy.misc

" on définit la fonction qui permet de détecter les ions H_2^+ "

def number_ions_flat(file,showimg=False):
    img = cv2.imread(file,cv2.IMREAD_GRAYSCALE)
    img_clean = img[:]

    '''on affiche dans une figure l'image initiale'''
    print('Initial image')
    plt.figure(figsize=(12,6))
    plt.imshow(img,cmap=plt.get_cmap('gray'))
    plt.show()

    '''étape 1 : floutage gaussien'''
    img = cv2.GaussianBlur(img, (27, 27), 0)
    print('Blur')
    plt.figure(figsize=(12,6))
    plt.imshow(img,cmap=plt.get_cmap('gray'))
    plt.show()

    '''étape 2 : seuillage ou thresholding'''
    img = cv2.threshold(img, 2, 255, cv2.THRESH_BINARY)[1]
    print('Threshold')
    plt.figure(figsize=(12,6))
    plt.imshow(img,cmap=plt.get_cmap('gray'))
    plt.show()

    '''étape 3 : Inversion'''
    img=255-img
    print('B<->W')
    plt.figure(figsize=(12,6))
    plt.imshow(img,cmap=plt.get_cmap('gray'))
    plt.show()

    '''la dérnière étape : la détection des trous  '''
    params = cv2.SimpleBlobDetector_Params()
    params.minThreshold = 200
    params.maxThreshold = 255
    params.filterByArea = True
```

```
    params.minArea = 60
    params.maxArea = 1000
    params.minDistBetweenBlobs = 1
    params.filterByCircularity = False
    params.filterByInertia = False
    params.filterByConvexity = False
    params.filterByColor = True
    params.blobColor = 255


    detector = cv2.SimpleBlobDetector_create(params)

    keypoints = detector.detect(img)

    im_with_keypoints = cv2.drawKeypoints(img, keypoints, np.array([]), (0,0,255),
     cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    for kk in range(len(keypoints)):
        print(kk,keypoints[kk].pt,keypoints[kk].size)

    if showimg:
        print('img with KP')
        plt.figure(figsize=(12,6))
        plt.imshow(cv2.drawKeypoints(img, keypoints, np.array([]), (255,0,0),
         cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS))
        plt.show()

        plt.figure(figsize=(12,6))
        plt.imshow(cv2.drawKeypoints(img_clean, keypoints, np.array([]), (255,0,0),
         cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS))
        plt.show()
        scipy.misc.imsave(f.replace('jpg','jpg'), img)
        scipy.misc.imsave(f.replace('jpg','jpg'), img_clean)

    cv2.waitKey(0)

    return(len(keypoints),keypoints)

''' notre fichier image est contenu dans le dossier "detection" '''
foldername = 'Z:\\detection\\'

files = glob.glob(foldername+'*.png')

ion_numbers = np.zeros(len(files))
''' j'ai utilisé la boucle for car on avait appliqué la fonction number_ions_flat
 à plusieurs image'''
for i,f in enumerate(files[:]):
    print(f)
    img = scipy.misc.imread(f)
    print(img)
```

```
temp = img[:,:]
scipy.misc.imsave(f.replace('png','jpg'), temp)
ion_numbers[i],keypoints = number_ions_flat(f.replace('png','jpg'),showimg=True)


print(ion_numbers[i])
```

Références

[1] F. G. Major, V. N. Gheorghe, G. Werth, Charged particle traps, Ed. Springer.

[2] N. Silitoe, Production of state-selected H_2^+ ions and numerical simulations of sympathetic cooling in RF traps, (2017).

[3] Loup Verlet, "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules", Physical Review 159 pp. 98-103, (1967).

[4] Johannes Heinrich. A Be+ Ion Trap for H2+ Spectroscopy. Atomic Physics [physics.atom-ph]. Sorbonne Université, Faculté des Sciences et Ingénierie, 2018. English.

[5] Wolfgang.Paul, Otto.Osberghaus, Erhardt.Fischer, An ion cage, physikalisches institut deer Universitat Bonn, 1958.