```
from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import numpy as np

from scipy.integrate import odeint


"on va essayer de modéliser un champ magnétique géneré par une spire circulaire (donc densité de courant linéique circulaire),"

"situé dans le plan (xOy) et centré en O"


"On rajoute ici l'influence d'un champ électrique ponctuel attractif centrée en O"


q=-1.602*10**-19

"Charge électron"


w=10**-7

"mu0/4*pi"


I=5*10**-3

"Courant dans la spire"


m=10**-30

"Masse électron"


"R=10**-1"

r=5*10**(-3)

"le rayon de la spire"


T0=np.linspace(0,1000,1000000)
```

```
t=np.linspace(0,100000,1000000)

T=np.linspace(0,6000,1000000)

"Des intervalles de temps du problème"


x0=1.5

y0=0

z0=0.5

"position initial"


vx0=0

vy0=(1/1.4)*10**-3

vz0=(1/1.4)*10**-3

"vitesse initial"


U0=np.array([x0,y0,z0,vx0,vy0,vz0])

U1=np.array([0,0,0.25,10**-3,0,0])

U2=np.array([3*10**-1,0,0,0,10**-3,-5*10**-3])

U3=np.array([2*10**-1,0,10**-1,0,5*10**-3,5*10**-3])

U4=np.array([0.2,0.,0,0,-1*10**-9,10**-9])

U5=np.array([2,0,1,0,10**-3,1.5*10**-3])

U6=np.array([1.5,1.5,3,-5*10**-5,-5*10**-5,-5*10**-5])

U7=np.array([0.2,0.2,0.2,0,0.1*10**-3,0])

U8=np.array([0,0,-0.25,10**-3,0,0])

"Des vecteusr de coordonnée initiale de la particule. Dans l'ordre: x,y,z,vx,vy,vz"



"coordonné du champ magnétique dans le cas ou r<<R (R étant la distance depuis l'origine du point de calcul du champ mag)"
```

```
def Bx(x,y,z):
    return (I*w)*(3*np.pi*z*x*r**2)/((x**2+y**2+z**2)**(5/2))


def By(x,y,z):
    return (I*w)*(3*np.pi*z*y*r**2)/((x**2+y**2+z**2)**(5/2))


def Bz(x,y,z):
    return (I*w)*(((2*np.pi*r**2)/((x**2+y**2+z**2)**(3/2)))-
(((3*np.pi*r**2)*(x**2+y**2))/((x**2+y**2+z**2)**(5/2))))


"écriture du système d'équation différentielle, donné par la force de lorentz F=qv^B"
def f(u,t):
    du=np.zeros(u.shape)
    du[0]=u[3]
    du[1]=u[4]
    du[2]=u[5]
    du[3]=(q/m)*(u[4]*Bz(u[0],u[1],u[2])-u[5]*By(u[0],u[1],u[2]))
    du[4]=(q/m)*(u[5]*Bx(u[0],u[1],u[2])-u[3]*Bz(u[0],u[1],u[2]))
    du[5]=(q/m)*(u[3]*By(u[0],u[1],u[2])-u[4]*Bx(u[0],u[1],u[2]))
    return du


"à l'aide de odeint on résoud le système d'equa diff"
Uelec=odeint(f,U8,T0)


plt.figure(1,figsize=(5,5))
print(np.shape(U0))
plt.plot(Uelec[:,0],Uelec[:,1],'o',markersize='1.5')
plt.xlabel('Axe x')
```

```
plt.ylabel('Axe y')

plt.show()


plt.figure(1,figsize=(5,5))

np.shape(Uelec)

plt.plot(Uelec[:,0],Uelec[:,2],'o',markersize='1.5')

plt.xlabel('Axe x')

plt.ylabel('Axe z')

plt.show()

#%%



"Ici on va essayer de tracer la trajectoire de la particule directement en 3D"

from mpl_toolkits.mplot3d import Axes3D

from mpl_toolkits import mplot3d

%matplotlib inline


fig = plt.figure()

ax = plt.axes(projection='3d')


ax.plot3D(Uelec[:,0], Uelec[:,1], Uelec[:,2])

ax.plot3D(np.array([0.5,0,0,-0.5,0,0]),np.array([0,0.5,0,0,-0.5,0]),np.array([0,0,0.35,0,0,-0.35]),',')


"On modélise la sphère de la planeterrella"

u = np.linspace(0, 2 * np.pi, 100)

v = np.linspace(0, np.pi, 100)

x = 0.05 * np.outer(np.cos(u), np.sin(v))
```

```
y = 0.05 * np.outer(np.sin(u), np.sin(v))

z = 0.05 * np.outer(np.ones(np.size(u)), np.cos(v))


# Plot the surface

ax.plot_surface(x, y, z, color='b')

#%%


"Ici on modifie un peut le vecteur de coordonnée initial"

from mpl_toolkits.mplot3d import Axes3D

from mpl_toolkits import mplot3d

%matplotlib inline


fig = plt.figure()

ax = plt.axes(projection='3d')


U=odeint(f,U5,T)


ax.plot3D(U[:,0], U[:,1], U[:,2])

ax.plot3D(np.zeros((100,1)),np.zeros((100,1)),np.linspace(-5,5,100))



#%%

"!!! Cette partie semble ne pas offrir de résultat très probant !!!"


"On rajoute ici l'influence d'un champ électrique ponctuel attractif centrée en O"

K=1/(4*np.pi*8.85418782*10**-12)

"constante de 'coulomb'"
```

```
Q=10**4

"charge du point O"


"I=3.5*10**10"

I=5*10**-3

"Courant dans la spire"


T1=np.linspace(0,1000,1000000)


m=10**-30

"Masse électron"


"On va écrire les coordonées du champ électrique"

def Ex(x,y,z):

    return x/((x**2+y**2+z**2)**(3/2))


def Ey(x,y,z):

    return y/((x**2+y**2+z**2)**(3/2))


def Ez(x,y,z):

    return z/((x**2+y**2+z**2)**(3/2))


"On a le système d'équation différentiels donné par la force de lorentz F=q(v^B+E)"

def g(u,t):

    du=np.zeros(u.shape)

    du[0]=u[3]
```

```
    du[1]=u[4]
    du[2]=u[5]
    du[3]=((q)/m)*(u[4]*Bz(u[0],u[1],u[2])-u[5]*By(u[0],u[1],u[2]))+((K*Q*q)/m)*Ex(u[0],u[1],u[2])
    du[4]=((q)/m)*(u[5]*Bx(u[0],u[1],u[2])-u[3]*Bz(u[0],u[1],u[2]))+((K*Q*q)/m)*Ey(u[0],u[1],u[2])
    du[5]=((q)/m)*(u[3]*By(u[0],u[1],u[2])-u[4]*Bx(u[0],u[1],u[2]))+((K*Q*q)/m)*Ez(u[0],u[1],u[2])
    return du

"puis on trace le résultat"
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d
%matplotlib inline

fig = plt.figure()
ax = plt.axes(projection='3d')

"résolution du système:"
U=odeint(g,U7,T1)
print(I,Q)
ax.plot3D(U[:,0], U[:,1], U[:,2])
#%%
"Dans cette partie on va tenter de coder le solénoide qui compose l'électroaimant"
"pas du solénoide, en m"
d=2*10**-3

"on fait N+1 spires"
N=10
```

"On va sommer les champs de plusieurs spire superposé les unes sur les autres (car le champ mag est additif)"

```
def bx(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=Bx(x,y,z+i*d)
    return s


def by(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=By(x,y,z+i*d)
    return s


def bz(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=Bz(x,y,z+i*d)
    return s


def f(u,t):
    du=np.zeros(u.shape)
    d=5*10**-3
    du[0]=u[3]
    du[1]=u[4]
    du[2]=u[5]
    du[3]=(q/m)*(u[4]*bz(u[0],u[1],u[2])-u[5]*by(u[0],u[1],u[2]))
    du[4]=(q/m)*(u[5]*bx(u[0],u[1],u[2])-u[3]*bz(u[0],u[1],u[2]))
```

```
    du[5]=(q/m)*(u[3]*by(u[0],u[1],u[2])-u[4]*bx(u[0],u[1],u[2]))
    return du

"à l'aide de odeint on résoud le système d'equa diff"
Ui=odeint(f,U5,T)

plt.figure(1,figsize=(5,5))
print(np.shape(U0))
plt.plot(Ui[:,0],Ui[:,1],'o',markersize='1.5')
plt.xlabel('Axe x')
plt.ylabel('Axe y')
plt.show()

plt.figure(1,figsize=(5,5))
np.shape(Ui)
plt.plot(Ui[:,0],Ui[:,2],'o',markersize='1.5')
plt.xlabel('Axe x')
plt.ylabel('Axe z')
plt.show()

#%%

"Ici on va essayer de tracer la trajectoire de la particule directement en 3D, pour le solénoide"
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d
%matplotlib inline
```

```
X=np.linspace(-0.1,0.1,1000)

Y=np.linspace(-0.1,0.1,1000)



fig = plt.figure()

ax = plt.axes(projection='3d')


ax.plot3D(Ui[:,0], Ui[:,1], Ui[:,2])


u = np.linspace(0, 2 * np.pi, 100)

v = np.linspace(0, np.pi, 100)

x = 0.1 * np.outer(np.cos(u), np.sin(v))

y = 0.1 * np.outer(np.sin(u), np.sin(v))

z = 0.1 * np.outer(np.ones(np.size(u)), np.cos(v))


# Plot the surface

ax.plot_surface(x, y, z, color='b')

ax.plot3D(np.array([0.1,0,0,-0.1,0,0]),np.array([0,0.1,0,0,-0.1,0]),np.array([0,0,1,0,0,-1]),',')

#%%


"On va tracer des lignes de champ généré par la spire"

"rayon de la spire"

r=5*10**-3

"Courant parcourant la spire"

I=5*10**-3
```

```
def Bx(x,y,z):
    return I*w*(3*np.pi*z*x*r**2)/((x**2+y**2+z**2)**(5/2))

def By(x,y,z):
    return I*w*(3*np.pi*z*y*r**2)/((x**2+y**2+z**2)**(5/2))

def Bz(x,y,z):
    return I*w*(((2*np.pi*r**2)/((x**2+y**2+z**2)**(3/2)))-
(((3*np.pi*r**2)*(x**2+y**2))/((x**2+y**2+z**2)**(5/2))))

"On va pour cela regarder la tractoire d'une particule fictive dans le chanp suivant"
def f(u,t):
    du=np.zeros(u.shape)
    du[0]=Bx(u[0],0,u[1])/((Bx(u[0],0,u[1])**2+Bz(u[0],0,u[1])**2)**0.5)
    du[1]=Bz(u[0],0,u[1])/((Bx(u[0],0,u[1])**2+Bz(u[0],0,u[1])**2)**0.5)
    return du

def h(u,t):
    du=np.zeros(u.shape)
    du[0]=-Bx(u[0],0,u[1])/((Bx(u[0],0,u[1])**2+Bz(u[0],0,u[1])**2)**0.5)
    du[1]=-Bz(u[0],0,u[1])/((Bx(u[0],0,u[1])**2+Bz(u[0],0,u[1])**2)**0.5)
    return du

"Vecteur initial, et temps"
Ut0=np.array([0.2,0])
UT0=np.array([-0.2,0])
Jt0=np.array([0.1,0])
```

```
JT0=np.array([-0.1,0])

Ht0=np.array([0.05,0])

HT0=np.array([-0.05,0])

Ti=np.linspace(0,3,10000)


Ut=odeint(f,Ut0,Ti)

UT=odeint(f,UT0,Ti)

U_t=odeint(h,Ut0,Ti)

U_T=odeint(h,UT0,Ti)

Jt=odeint(f,Jt0,Ti)

JT=odeint(f,JT0,Ti)

J_t=odeint(h,Jt0,Ti)

J_T=odeint(h,JT0,Ti)

Ht=odeint(f,Ht0,Ti)

HT=odeint(f,HT0,Ti)

H_t=odeint(h,Ht0,Ti)

H_T=odeint(h,HT0,Ti)


plt.figure(1,figsize=(10,10))

plt.plot(Ut[:,0],Ut[:,1],'bo',markersize='0.5')

plt.plot(UT[:,0],UT[:,1],'bo',markersize='0.5')

plt.plot(U_t[:,0],U_t[:,1],'bo',markersize='0.5')

plt.plot(U_T[:,0],U_T[:,1],'bo',markersize='0.5')

plt.plot(Jt[:,0],Jt[:,1],'bo',markersize='0.5')

plt.plot(JT[:,0],JT[:,1],'bo',markersize='0.5')

plt.plot(J_t[:,0],J_t[:,1],'bo',markersize='0.5')

plt.plot(J_T[:,0],J_T[:,1],'bo',markersize='0.5')
```

```
plt.plot(Ht[:,0],Ht[:,1],'bo',markersize='0.5')

plt.plot(HT[:,0],HT[:,1],'bo',markersize='0.5')

plt.plot(H_t[:,0],H_t[:,1],'bo',markersize='0.5')

plt.plot(H_T[:,0],H_T[:,1],'bo',markersize='0.5')

plt.plot(0.05*np.cos(np.linspace(0,2*np.pi,1000)),0.05*np.sin(np.linspace(0,2*np.pi,1000)))

plt.plot(np.array([-r,r]),np.array([0,0]))

plt.axis([-0.3,0.3,-0.3,0.3])

plt.xlabel('Axe x')

plt.ylabel('Axe z')

plt.show()


#%%



"Dans cete partie on va s'intéresser à l'intensité de la coordonnée Bz le long de l'axe z"

I=5*10**2

"Courant dans la spire"


def Bx(x,y,z):

    return ((I*w)*(3*np.pi*z*x*r**2))/((x**2+y**2+z**2)**(5/2))


def By(x,y,z):

    return ((I*w)*(3*np.pi*z*y*r**2))/((x**2+y**2+z**2)**(5/2))


def Bz(x,y,z):

    return (I*w)*(((2*np.pi*r**2)/((x**2+y**2+z**2)**(3/2)))-
(((3*np.pi*r**2)*(x**2+y**2))/((x**2+y**2+z**2)**(5/2))))
```

```
def bx(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=Bx(x,y,z+i*d)
    return s

def by(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=By(x,y,z+i*d)
    return s

def bz(x,y,z):
    s=0
    for i in range(-N,N+1):
        s+=Bz(x,y,z+i*d)
    return s



Z=np.linspace(-0.1,0.1,1000000)
plt.plot(Z,Bz(0,0,Z))
plt.axis([-0.05,0.05,0,0.005])
plt.xlabel('z en m')
plt.ylabel('Bz en T')
plt.title('Intensité du champ mag le long de z pour la spire')
plt.show()
plt.axis([-0.05,0.05,0,20])
```

```
plt.plot(Z,bz(0,0,Z))
plt.xlabel('z en m')
plt.ylabel('Bz en T')
plt.title('Intensité du champ mag le long de z pour le solénoide')
plt.show()

#%%

"Cette fois ci on va tenter de générer les lignes de champs du solénoide"

def f(u,t):
    du=np.zeros(u.shape)
    du[0]=bx(u[0],0,u[1])/((bx(u[0],0,u[1])**2+bz(u[0],0,u[1])**2)**0.5)
    du[1]=bz(u[0],0,u[1])/((bx(u[0],0,u[1])**2+bz(u[0],0,u[1])**2)**0.5)
    return du

def h(u,t):
    du=np.zeros(u.shape)
    du[0]=-bx(u[0],0,u[1])/((bx(u[0],0,u[1])**2+bz(u[0],0,u[1])**2)**0.5)
    du[1]=-bz(u[0],0,u[1])/((bx(u[0],0,u[1])**2+bz(u[0],0,u[1])**2)**0.5)
    return du


"Vecteur initial, et temps"
Ut0=np.array([0.5,0])
UT0=np.array([-0.5,0])
Jt0=np.array([0.1,0])
```

```
JT0=np.array([-0.1,0])

Ht0=np.array([0.05,0])

HT0=np.array([-0.05,0])

Ti=np.linspace(0,1,100000)


Ut=odeint(f,Ut0,Ti)

UT=odeint(f,UT0,Ti)

U_t=odeint(h,Ut0,Ti)

U_T=odeint(h,UT0,Ti)




plt.figure(1,figsize=(10,10))

plt.plot(Ut[:,0],Ut[:,1],'bo',markersize='0.5')

plt.plot(UT[:,0],UT[:,1],'bo',markersize='0.5')

plt.plot(U_t[:,0],U_t[:,1],'bo',markersize='0.5')

plt.plot(U_T[:,0],U_T[:,1],'bo',markersize='0.5')



plt.plot(0.05*np.cos(np.linspace(0,2*np.pi,1000)),0.05*np.sin(np.linspace(0,2*np.pi,1000)))

for i in range(-N,N+1):

    plt.plot(np.array([-r,r]),np.array([i*d,i*d]),'r',markersize='0.7')

plt.axis([-0.3,0.3,-0.3,0.3])

plt.xlabel('Axe x')

plt.ylabel('Axe z')

plt.show()
```

```
#%%


"Zoom sur le champ généré dans le solénoide"

"!!! Cela n'est pas pertinent au vue de l'approximation sur laquelle repose l'ensemble de ce code !!!"

plt.figure(1,figsize=(10,10))

plt.plot(Ut[:,0],Ut[:,1],'bo',markersize='0.5')

plt.plot(UT[:,0],UT[:,1],'bo',markersize='0.5')

plt.plot(U_t[:,0],U_t[:,1],'bo',markersize='0.5')

plt.plot(U_T[:,0],U_T[:,1],'bo',markersize='0.5')


plt.plot(0.05*np.cos(np.linspace(0,2*np.pi,1000)),0.05*np.sin(np.linspace(0,2*np.pi,1000)))

for i in range(-N,N+1):

    plt.plot(np.array([-r,r]),np.array([i*d,i*d]),'r',markersize='0.7')

plt.axis([-0.021,0.021,-0.021,0.021])

plt.xlabel('Axe x')

plt.ylabel('Axe z')

plt.show()
```