1 Introduction

This code generates the trajectory followed by a massive, charged particle in the influence of electromagnetic fields.
The setup of the code allows to either import a field as a numpy array (e.g: generated by github.com/ACauqui13/magnetic-field-calculator) or make it internally on the go.

2 non-imported fields

This section provides an example for a non-imported, pregenerated field.
Usage of trajectorysolver-nongenerated.py in the repository.

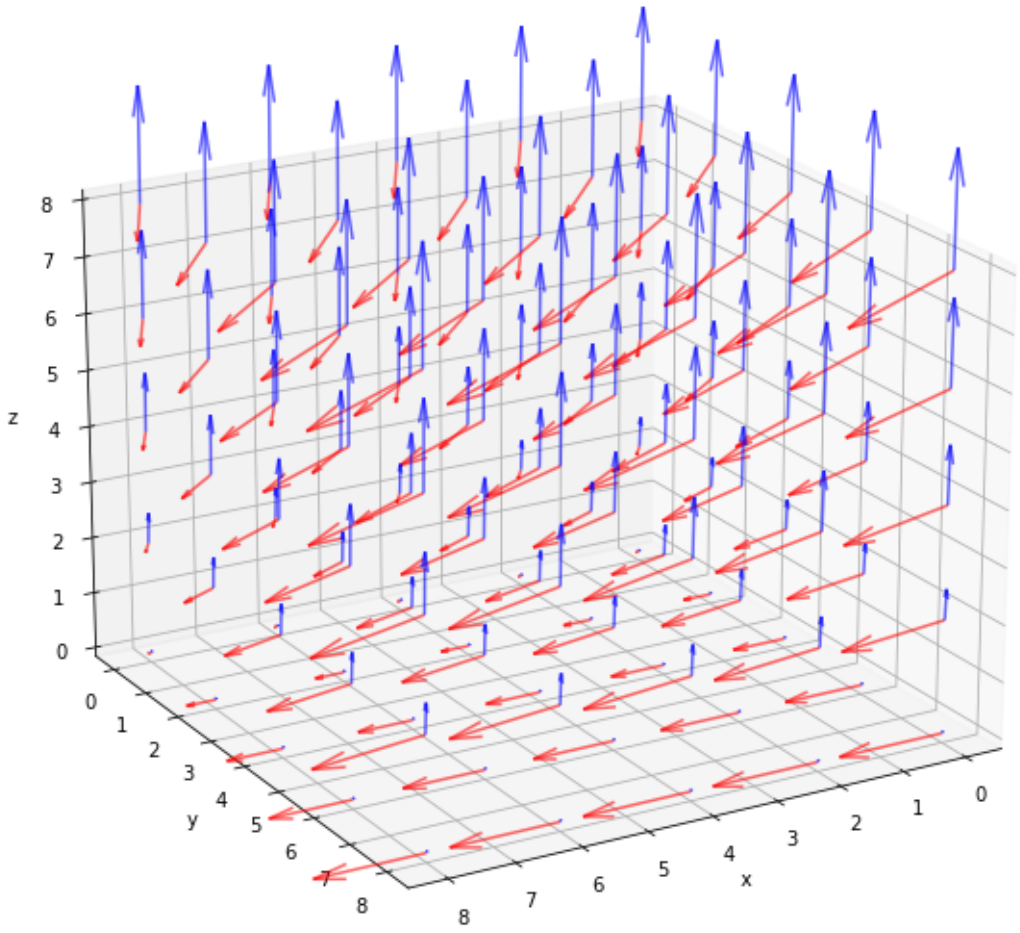
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import axes3d

h=0.05
x = np.arange(0,10,h)
y = np.arange(0,10,h)
z= np.arange(0,10,h)
q=1.6e-19
m=9.31e-20

def pos(current_state,t):
    rx,ry,rz,vx,vy,vz=current_state
    E0=[0.5+2*ry,0,-0.010-0.8*rz]
    B0=[0,5,10+50*rz]
    B=(q*B0[2])/m
    C=(q*B0[1])/m
    D=(q*E0[1])/m
    E=(q*B0[0])/m
    F=(q*B0[2])/m
    G=(q*E0[2])/m
    H=(q*B0[1])/m
    I=(q*B0[0])/m


    d2rxdt2=q*E0[0]/m+B*vy-C*vz
    d2rydt2=D+E*vz-F*vx
    d2rzdt2=G+H*vx-I*vy

    return [vx,vy,vz,d2rxdt2,d2rydt2,d2rzdt2]

y0=[5.700,6,2.5,0.1,-0.1,-0.1]                    #initial conditions [x0,y0,z0,vx0,vy0,
    vz0]
t=np.arange(0,100,0.001)                          #time interval of simulation
xyz=odeint(pos,y0,t)                              #Creates an array of values for the
    position of the particle
```

In the definition of the function pos(current_state,t) one can find the corresponding equations of motion to solve. E0=[Ex,Ey,Ez] and B0=[Ex,Ey,Ez] are the electric and magnetic fields to which the particle is subject. They can be set constant or can be made to vary with position. In the example above, rx,ry and rz are the current position components of the particle, so I made a electromagnetic field that varies with the latter as follows:

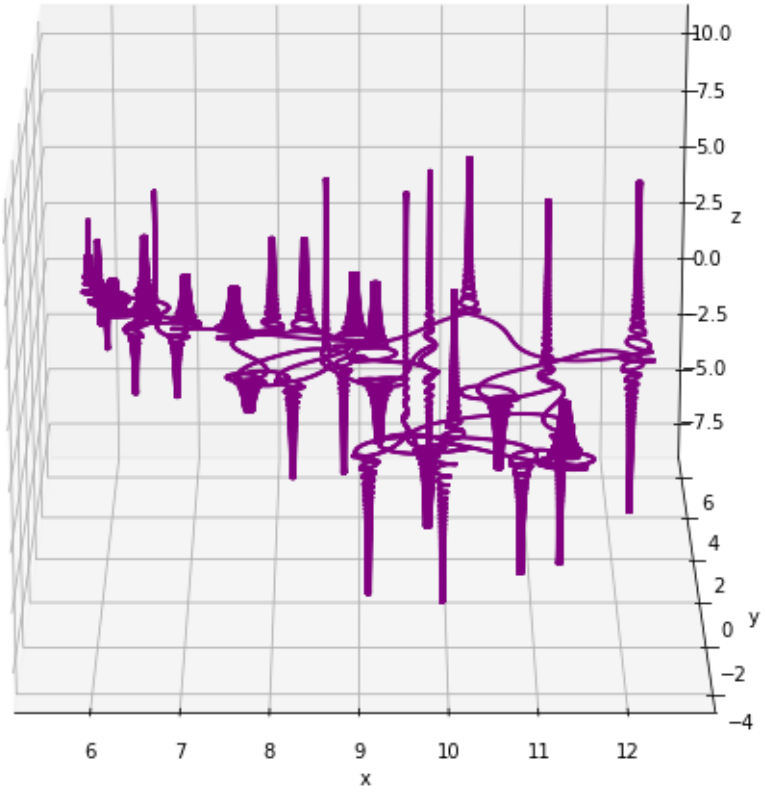
$$\vec{E} = (0.5 + 2y)\vec{i} + 0\vec{j} + (-0.010 - 0.8z)\vec{k}$$
$$\vec{B} = (0)\vec{i} + 5\vec{j} + (10 + 50z)\vec{k}$$

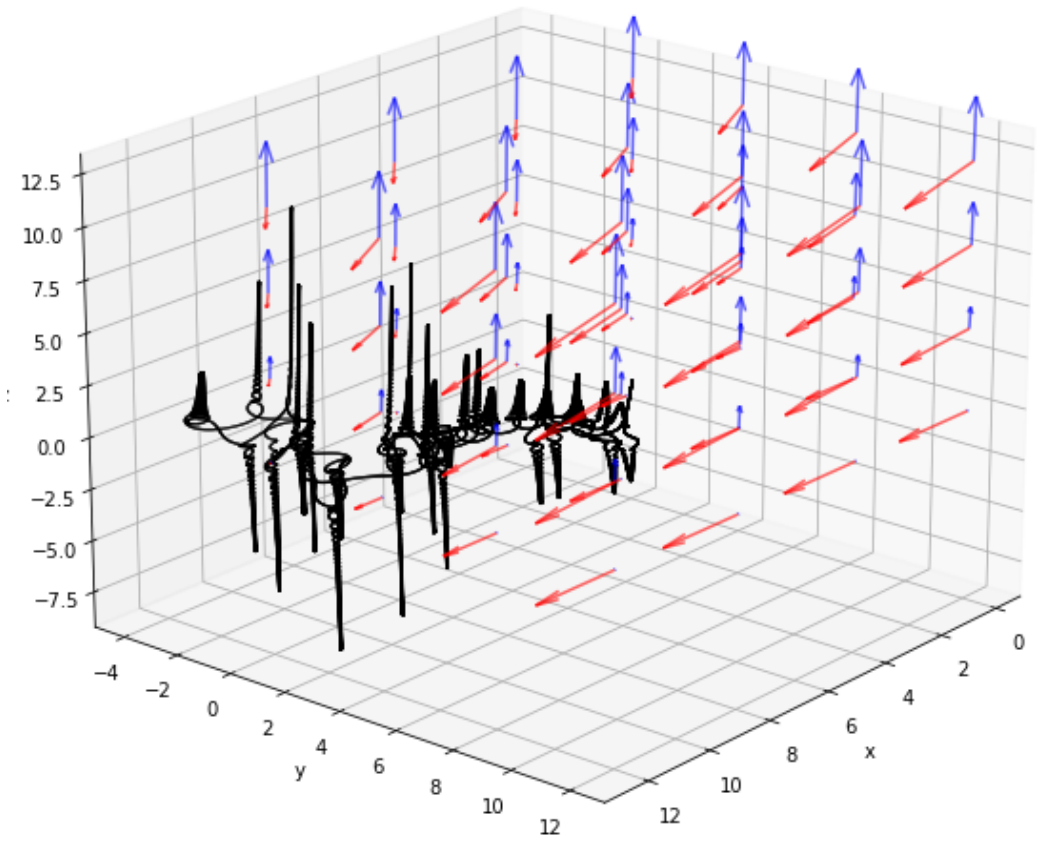
Which we can plot:

```
1
2  X, Y, Z =  np.meshgrid(x,
3                         y,
4                         z)
5  Ex = 0.5+2*Y
6  Ey = 0
7  Ez=-0.010-0.8*Z
8
9  Bx=0
10 By=5
11 Bz=10 + 50*Z
12
13 fig = plt.figure(figsize=(10,10))
14 ax = fig.add_subplot(projection='3d')
15 ax.quiver(X, Y, Z, Ex,Ey, Ez, length=0.1,alpha=0.6,color='red')
16 ax.quiver(X, Y, Z, Bx, By, Bz, length=0.005,alpha=0.6,color='Blue')
17 ax.set_xlabel('x')
18 ax.set_ylabel('y')
19 ax.set_zlabel('z')
20 ax.view_init(elev=20,azim=60)
```



The execution of odeint(pos,y0,t) yields the following (using the graph generator.py on the repository):


10.0
7.5
5.0
2.5
z
0.0
−2.5
−5.0
−7.5
6
4
2
0
y
−2
−4
6
7
8
9
10
11
12
x



3 Imported field

the code in the repository github.com/ACauqui13/magnetic-field-calculator generates a magnetic field using as input a current of any given shape, size and magnitude. The generated field is exported as a np array, which can be used as input in this algorithm:

```
Bx=np.load('your filepath bx.npy')
By=np.load('your filepath by.npy')
Bz=np.load('your filepath bz.npy')
```

The function which defines the current position of the particle now uses the values from these arrays to evaluate the magnetic field.I also implemented a electric field which would vary with time. In order to feed the algorithm the correct values, i defined a function to find the closest index in the magnetic field arrays to any given number. This is, find the 'position' of the particle in the magnetic field array, which is a discrete mesh, from the continuous-valued position:

```
def indexfindx(r):
    h=0.05
    x = np.arange(0,10,h)
```

```
    for i in x:
        if r>i:
            continue
        if r==i:
            break
        if r<i:
            break
    return np.where(x==i)[0][0]

def indexfindy(r):
    h=0.05
    y = np.arange(0,10,h)
    for i in y:
        if r>i:
            continue
        if r==i:
            break
        if r<i:
            break
    return np.where(y==i)[0][0]

def indexfindz(r):
    h=0.05
    z = np.arange(0,10,h)
    for i in z:
        if r>i:
            continue
        if r==i:
            break
        if r<i:
            break
    return np.where(z==i)[0][0]
```

So if the x-position of the particle is 4.567cm it is approx 4.6cm which corresponds to index 92 in the mesh.

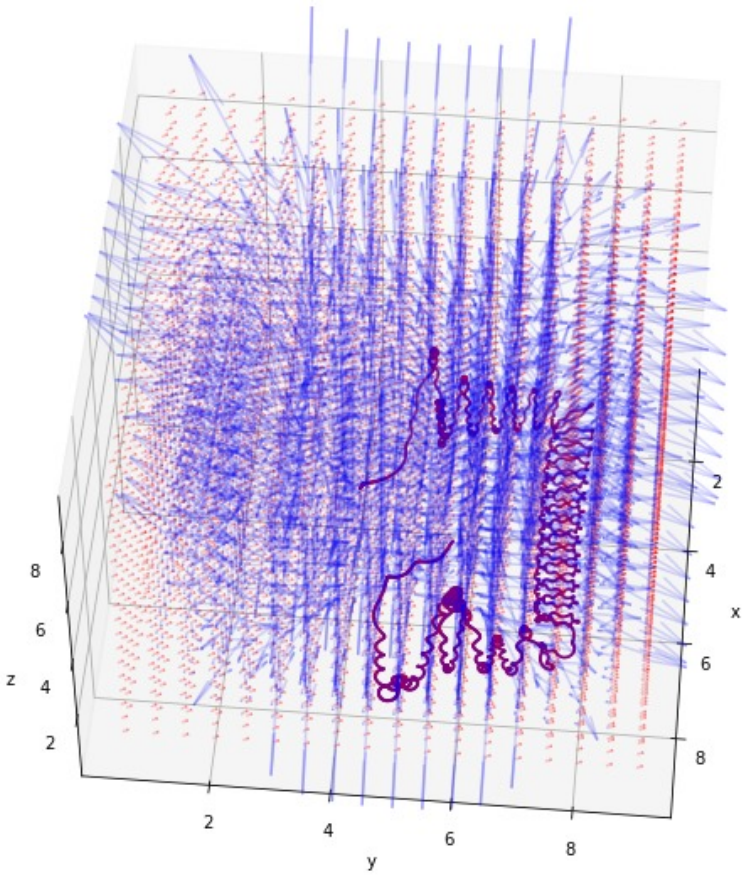
```
def Etime(t,t0):
    return square(2*np.pi*(1/(2*t0))*t)

def pos11(current_state,t):
    rx,ry,rz,vx,vy,vz=current_state
    E0=[0,Etime(t,25)*0.01,-Etime(t,25)*0.01]
    B0=[Bx[indexfindx(rx),indexfindy(ry),indexfindz(rz)],By[indexfindx(rx),indexfindy(ry
    ),indexfindz(rz)],Bz[indexfindx(rx),indexfindy(ry),indexfindz(rz)]]
    B=(q*B0[2])/m
    C=(q*B0[1])/m
    D=(q*E0[1])/m
    E=(q*B0[0])/m
    F=(q*B0[2])/m
    G=(q*E0[2])/m
    H=(q*B0[1])/m
    I=(q*B0[0])/m

```

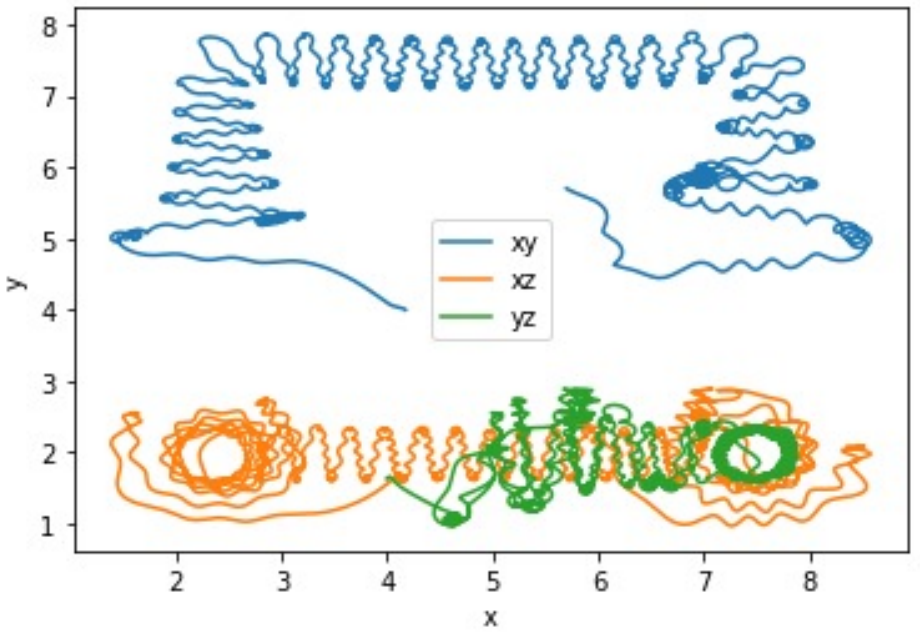
```
17
18      d2rxdt2=q*E0[0]/m+B*vy-C*vz
19      d2rydt2=D+E*vz-F*vx
20      d2rzdt2=G+H*vx-I*vy
21
22      return [vx,vy,vz,d2rxdt2,d2rydt2,d2rzdt2]
```

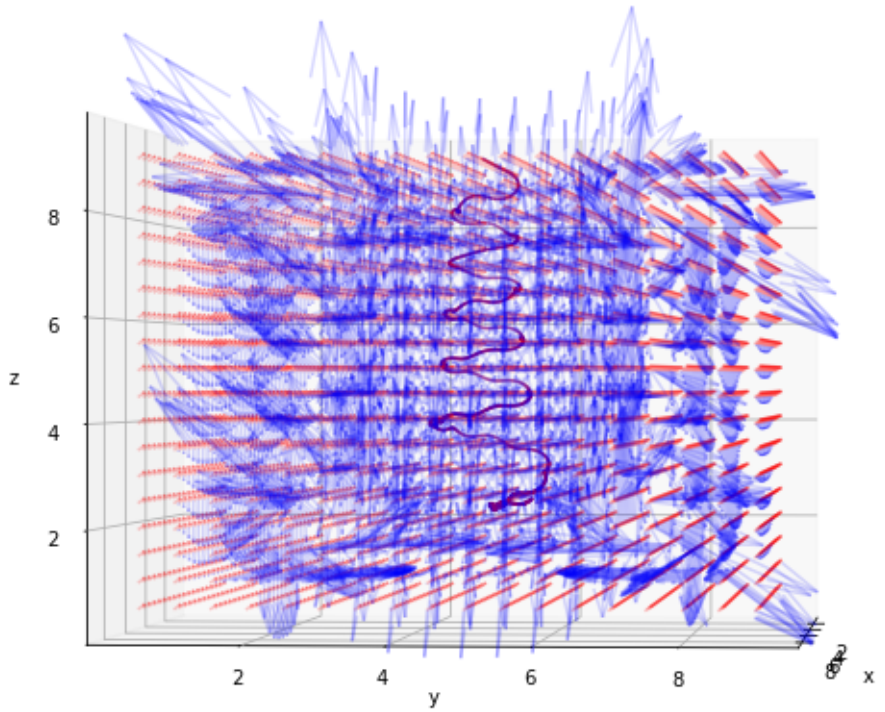
the function Etime(t,t0) generates a square wave with half period (change of sign) of t0.This is fed to the current state function to generate a field which oscillates in time.The function uses the indexfind functions to extract the values of the components of the magnetic field imported in any position of the simulation. The results are then generated as before:

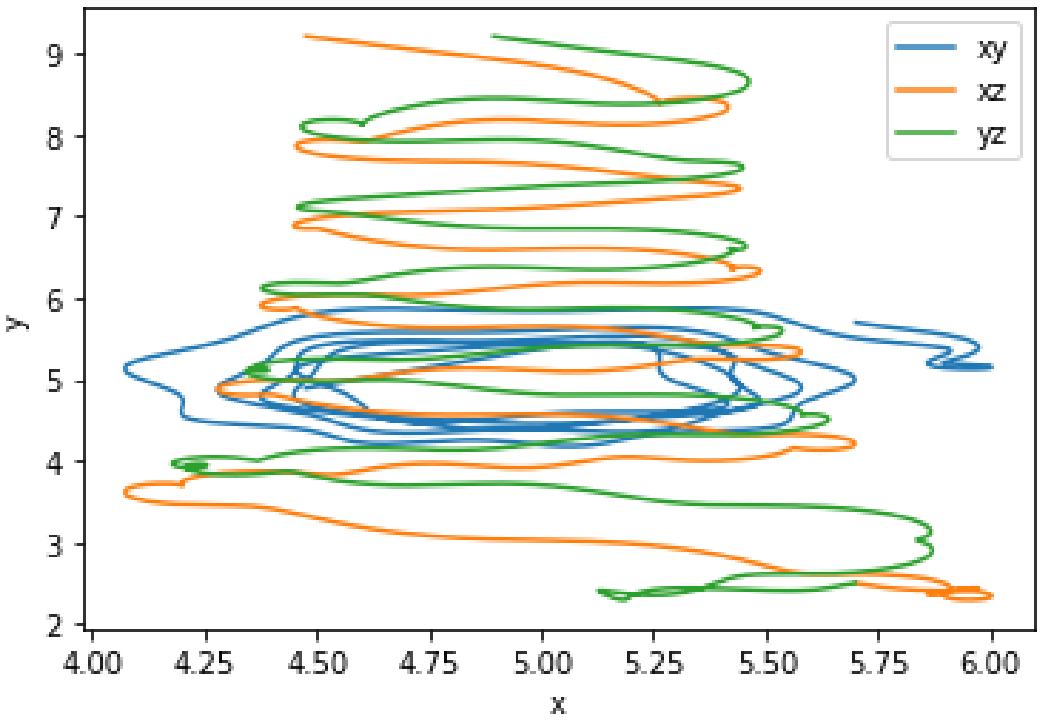
```
1 y0=[xo,yo,zo,vxo,vyo,vzo]
2 t=np.arange(0,100,0.001)
3 xyz=odeint(pos,y0,t)
```

Here are some examples i did:




xy
xz
yz
x
y


x
y
z


xy
xz
yz
9
8
7
6
5
4
3
2
y
4.00
4.25
4.50
4.75
5.00
5.25
5.50
5.75
6.00
x