# End Semester Exam - EE2703

Nihal Gajjala

May 2021

## Introduction

This is a problem based on radiation from a loop antenna. The problem is to compute and plot the magnetic field $\vec{B}$ along the z-axis at a distance of 1cm to 1000 cm from the origin (wire). We plot the graphs and fit it the obtained data to $\mid \vec{B} \mid = cz^b$.

The main challenge in this exam is to handle Python arrays efficiently, and also to understand the accuracy obtained by particular choices of grid.

## Problem Statement

A long wire carries a current $(I)$ through a loop of wire. The wire is placed in the x-y plane and centered at the origin. The radius of the loop is taken to be 10 cm and is also equal to $\frac{1}{k} = \frac{c}{\omega}$ i.e. the circumference of the loop is $\lambda$ (length of antenna).

$$I = \frac{4\pi}{\mu_0}\cos(\phi)\exp(j\omega t) \tag{1}$$

Here, $\phi$ is the angle in polar coordinates i.e. in (r,$\phi$,z) coordinates. Using the right hand thumb rule, we observe that the magnetic field $(\vec{B})$ is aligned along the z-axis. The computation of $\vec{B}$ involves the calculation of the vector potential $(\vec{A})$.

$$\vec{A}(r,\phi,z) = \frac{4\pi}{\mu_0}\int \frac{I(\phi)\hat{\phi}\exp(-jkR)a\,d\phi}{R} \tag{2}$$

where $\vec{R} = \vec{r} - \vec{r'}$ and $k = \frac{\omega}{c} = 0.1$. $\vec{r}$ is the point where we want the field, and $\vec{r'} = a\hat{r}'$ is the point on the loop. We can approximate the above equation from an integral to a summation.

$$\vec{A}_{ijk} = \sum_{l=0}^{N-1} \frac{\cos(\hat{\phi_l}') \exp(-jkR_{ijkl})\,dl'}{R_{ijkl}} \tag{3}$$

where $\vec{r}$ is at $r_i$, $\phi_j$, $z_k$ and $\vec{r'}$ is at $a\cos\phi_l'\hat{x} + a\sin\phi_l'\hat{y}$. From $\vec{A}$, you can obtain $\vec{B}$ as $\vec{B} = \nabla * \vec{A}$. Along the z-axis this becomes ($\vec{A}$ is along $\hat{\phi}$ and the curl gives only a $B_z$ component along the z-axis).

$$B_z(z) = \frac{A_y(\triangle x,0,z) - A_x(0,\triangle y,z) - A_y(-\triangle x,0,z) + A_x(0,-\triangle y,z)}{4\triangle x\triangle y} \tag{4}$$

## Pseudo Code

- Construct a meshgrid (x,y,z) where
  - size along x-direction is 3
  - size along y-direction is 3
  - size along z-direction is 1000

  using np.linspace and np.meshgrid.
- Break the loop into 100 sections using np.linspace and create radius vector, dl vector, phi array and position vector using np.linspace, np.vstack and np.array.
- Plot the Current Element in the x-y plane using pylab library
- Write a function to calculate the direction and magnitude of current and plot Current in the x-y plane. Generate arrows using built-in quiver function.
- Write a function calc(l) that calculates and returns $\vec{R_{ijkl}} = \mid \vec{r_{ijk}} - \vec{r_l}' \mid$ for all $\vec{r_{ijk}}$.
- Calculate $\vec{A_{ijk}}$ and $\vec{B_z}$. Plot the variation of $\vec{B_z}$ along the z-axis.
- Using the least square method calculate the best fit for b. The built-in function to calculate least squares is np.linalg.lstsq.

## Current Elements

We divided the loop into 100 equal sections using np.linspace. The position of each current element is a function of radius (r) and angle it makes with x-axis in the x-y plane ($\phi$). The x-coordinate is calculated using radius*np.cos(phi) whereas the y coordinate is calculated using radius*np.sin(phi). Here phi is an array or equally spaced number in the range 0 to $2\pi$ given by np.linspace(0,2*np.pi,sections).

Similarly, the x-coordinate of dl is given by (2*np.pi*10/sections) (radius*np.cos(phi)) and y-coordinate of dl is given by (2*np.pi*10/sections) (radius*np.sin(phi)).

We next plotted a graph showing the position of all the current elements (100 sections). We use plot function in pylab to display the current elements as red dots. Other functions used include xlabel, ylabel, title, grid and show to make the graph easy to understand.

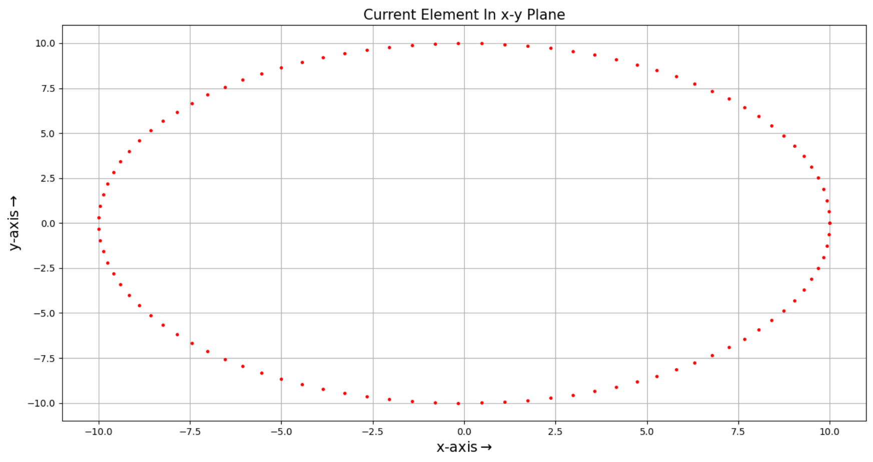


Figure 1: Current Elements In x-y Plane

In the above figure we can observe that the loop is circular with a radius on 10 units. We can also see that the current elements are placed at equal distances.

## Current Vectors

We now calculate the current at each and every current element. The x-component of current is given by y*np.cos(np.arctan(y/x)) and the y-component of current is given by -x*np.cos(np.arctan(y/x)). A separate function named current computes the above components and returns the values to the main function.

We next plotted a graph showing the direction and magnitude of current at all the current elements. We use quiver function in pylab to display the currents as arrows. Higher magnitudes of current have longer arrows. The arrow size can be customised using scale, headwidth, headlength and width. Other functions used include xlabel, ylabel, title, grid and show to make the graph easy to understand.

pylab.quiver(*args, data=None, **kw)

quiver([X, Y], U, V, [C], **kw)

X, Y define the arrow locations, U, V define the arrow directions, and C optionally sets the color

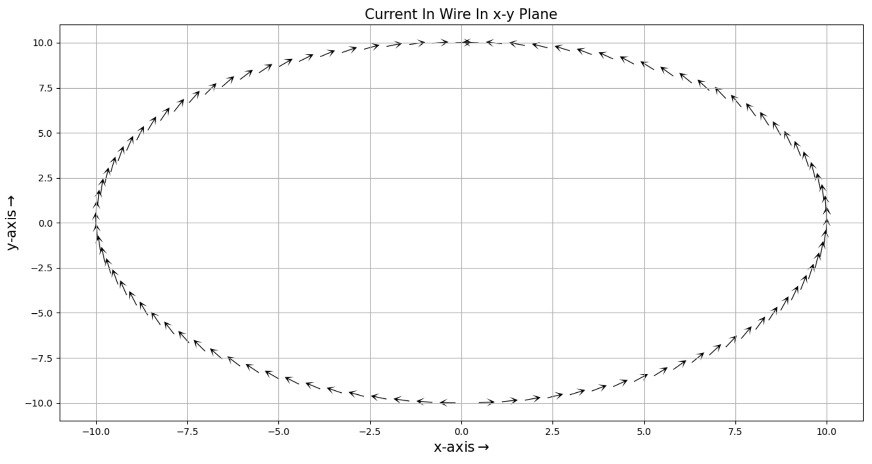


Figure 2: Currents In x-y Plane

We observe that the magnitude of current is maximum near the x-axis and minimum near the y-axis. We even see that the y-component of current changes direction when $\phi$ is $90^0$ and $270^0$. Similarly the x-component of current changes direction when $\phi$ is $0^0$ and $180^0$.

## Magnetic field

We write a function named calc which takes l as an argument and returns $R_{ijkl}$. We calculate the norm using the built-in function in numpy (np.linalg.norm). We used .reshape function to gives a new shape to an array without changing its data. Using the $R_{ijkl}$ obtained from the calc(l) function we calculate the potential vector using eqn3.

The x-component of potential vector is given by:
np.sum(cosine*dl*np.exp(1j*Rijkl/10)*dl/Rijkl,axis=0)
where dl is dlvector[:,0].reshape((100,1,1,1))

Similarly, y-component of potential vector is given by
np.sum(cosine*dl*np.exp(1j*Rijkl/10)*dl/Rijkl,axis=0)
where dl is dlvector[:,1].reshape((100,1,1,1)).

Using eqn 4 we can calculate $\vec{B}$
Bz=(Ay[1,0,:]-Ax[0,1,:]-Ay[-1,0,:]+Ax[0,-1,:])/(4)

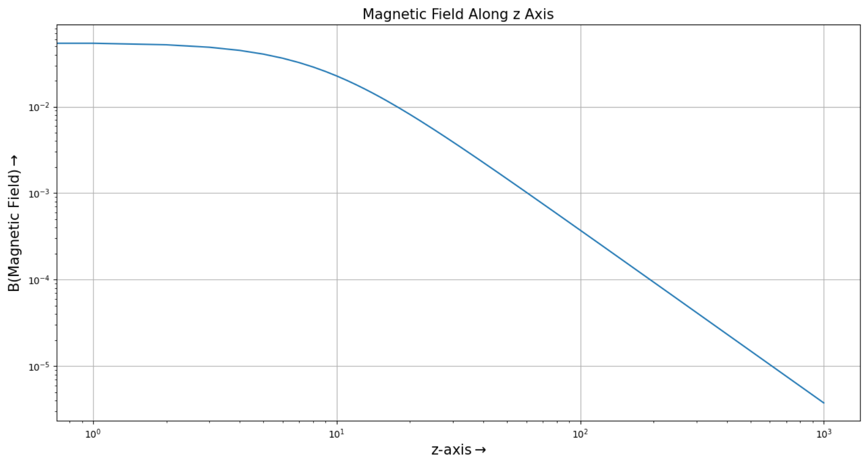
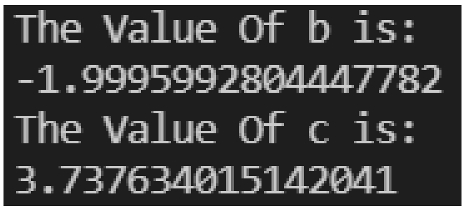


Figure 3: Magnetic Field Along z Axis

From the above figure we can say that the magnetic field decreases non-linearly initially when we are close to the wire. This decrease slowly becomes linear for larger distances from the wire. This is due to the fact that at smaller distances the effect of z is dominant and this gives rise to the non-linear behaviour but at larger distances the effect due to z becomes negligible compared to the exponential term giving rise to a linear graph in the log scale.

## Least Squares

The dependence of magnetic field on z can be calculated using the least square method. We can approximate the magnetic field to be of the form $cz^b$ where b and c are constants and z is the height or depth of the point at which the magnetic field is being calculated from the wire. b is given by the slope in figure 3. On solving using least squares we get b as -1.9995992804447782 and c as 3.737634015142041.



```
The Value Of b is:
-1.9995992804447782
The Value Of c is:
3.737634015142041
```

Figure 4: Least Square Solution

## Conclusion

We can conclude that in the log scale the magnetic field has a non-linear dependence on z for small values of z. The dependence of magnetic field slowly becomes linear as we increase z. At large values of z, the magnetic field changes linearly with z in the log scale. Here z is the height or depth of the point at which the magnetic field is being calculated from the wire.

This is similar to what we calculated theoretically. Initially at smaller distances the effect of z is dominant and this gives rise to the non-linear behaviour but at larger distances the effect due to z becomes negligible compared to the exponential term giving rise to a linear graph in the log scale.

The expected decay rate in case of a static magnetic field is -2. The decay we got in the above setup is -1.9995992804447782 which is approximately equal to 2. This error is due to the fact that the magnetic field is non-linear for small distances. In the least square solution we consider these non-linear points as well which results in the decay rate deviating from the expected decay rate.

## Code

End_Semester_Exam.py

```
"""
Course: EE2703-Applied Programming Lab
Name: Nihal Gajjala
Roll Number: EE19B044
End Semester Exam
"""

# Importing Libraries
import numpy as np
import pylab

# Function To Calculate Rijkl
def calc(l):
    return np.linalg.norm(np.tile(rijk,(100,1,1,1)).reshape((100,3,3,3,1000))-np.hstack
        ((r_vector,np.zeros((100,1)))).reshape((100,3,1,1,1)),axis=1)
'''
Rijkl = | rijk - rl |
.reshape - Gives A New Shape To An Array Without Changing Its Data
np.hstack - Stacks Arrays In Sequence Horizontally (Column Wise)
np.linalg.norm - Returns The Norm Of rijk - rl
np.zeros - Returns An Array Of Given Shape And Type, Filled With Zeros
'''

# Function To Calculate The Current Along x-direction And y-direction
def current(x,y):
    return np.array([-np.sin(np.arctan(y/x)),np.cos(np.arctan(y/x))])
'''
np.array - Constructs An Array
np.arctan - Tangent Inverse Function
np.cos - Cosine Function
'''

# Creating Meshgrid
# Size Along x Direction = 3
# Size Along y Direction = 3
# Size Along z Direction = 1000
x=np.linspace(0,2,num=3)           # Constructing An Array Of Evenly Spaced Numbers
    Over A Specified Interval
```

```
y=np.linspace(0,2,num=3)            # Constructing An Array Of Evenly Spaced Numbers
     Over A Specified Interval
z=np.linspace(0,999,num=1000)       # Constructing An Array Of Evenly Spaced Numbers
     Over A Specified Interval
X,Y,Z=np.meshgrid(x,y,z)            # Creating Coordinate Matrices From Coordinate
     Vectors

# Declaring The Radius Of Wire And Number Of Sections It Is Divided Into
radius=10                     # Radius Of Wire
sections=100                  # Number Of Sections

# Radius Vector -> (x - cos(phi), y - sin(phi))
r_vector=np.vstack((radius*np.cos(np.linspace(0,2*np.pi,sections)).T,radius*np.sin(np.
     linspace(0,2*np.pi,sections)).T)).T
# dl Vector -> (x - Acos(phi), y - Asin(phi))
# A = np.pi/5
dl_vector=2*np.pi*radius/sections*np.vstack((np.cos(np.linspace(0,2*np.pi,sections)).T,
     np.sin(np.linspace(0,2*np.pi,sections)).T)).T
# Phi
phi=np.linspace(0,2*np.pi,sections)
# Position Vector
rijk=np.array((X,Y,Z))
'''
.T - Transposes The Array
np.array - Constructs An Array
np.cos - Cosine Function
np.sin - Sine Function
np.linspace - Constructs An Array Of Evenly Spaced Numbers Over A Specified Interval
np.vstack - Stacks Arrays In Sequence Vertically (Row Wise)
'''

# Plotting Current Elements In x-y Plane
pylab.figure(1)
pylab.plot(r_vector[:,0],r_vector[:,1],'ro',markersize=2.5)              # Plotting y vs
     x As Lines And Markers
pylab.xlabel(r'x-axis$\rightarrow$',fontsize=15)                         # Setting The
     Label For The x-axis
pylab.ylabel(r'y-axis$\rightarrow$',fontsize=15)                         # Setting The
     Label For The y-axis
pylab.title('Current Elements In x-y Plane',fontsize=15)                  # Setting Title
      Of The Graph
pylab.grid()                                                             # Displaying The
      Grid
pylab.show()                                                             # Displaying The
      Figure

# Currents Along x-direction And y-direction
ix,iy=current(r_vector[:,0],r_vector[:,1])          # Calculating x And y Component Of
     Current Using 'current' Function

# Plotting Current Direction In x-y Plane
pylab.figure(2)
pylab.quiver(r_vector[:,0],r_vector[:,1],ix,iy,scale=50,headwidth=10,headlength=10,width
     =0.001)
'''
Plotting A 2-D Field Of Arrows
r_vector[:,0] - The x Coordinates Of The Arrow Locations
r_vector[:,1] - The y Coordinates Of The Arrow Locations
ix - The x-direction Components Of The Arrow Vectors
iy - The y-direction Components Of The Arrow Vectors
scale - The Arrow Length Unit
headwidth - Head Width As Multiple Of Shaft Width
headlength - Head Length As Multiple Of Shaft Width
width - Shaft Width In Arrow Units
'''
pylab.xlabel(r'x-axis$\rightarrow$',fontsize=15)                         # Setting The
     Label For The x-axis
pylab.ylabel(r'y-axis$\rightarrow$',fontsize=15)                         # Setting The
     Label For The y-axis
pylab.title('Current In Wire In x-y Plane',fontsize=15)                  # Setting Title
     Of The Graph
pylab.grid()                                                             # Displaying The
      Grid
pylab.show()                                                             # Displaying The
      Figure

Rijkl=calc(0)                                                    # Calculating Rijkl
     Using 'calc' Function
cosine=np.cos(phi).reshape((100,1,1,1))                          # Construction Cosine
     Vector
dl=dl_vector[:,0].reshape((100,1,1,1))                           # dl Vector Component
     Along x-direction
Ax=np.sum(cosine*dl*np.exp(1j*Rijkl/10)*dl/Rijkl,axis=0)         # Potential Along x-
```

```
     direction
dl=dl_vector[:,1].reshape((100,1,1,1))                          # dl Vector Component
     Along y-direction
Ay=np.sum(cosine*dl*np.exp(1j*Rijkl/10)*dl/Rijkl,axis=0)        # Potential Along y-
     direction
'''
j - Imaginary Unit
np.cos - Cosine Function
np.exp - Exponential Function
np.sum - Summation Function
'''

Bz=(Ay[1,0,:]-Ax[0,1,:]-Ay[-1,0,:]+Ax[0,-1,:])/(4)              # Calculating Magnetic
     Field
magneticField=(np.zeros(1000).T,np.zeros(1000).T,Bz.T)          # Vectorizing Magnetic
     Field
'''
Magnetic Field Is Zero Along x-direction
Magnetic Field Is Zero Along y-direction
Magnetic Field Is Bz Along z-direction
'''

# Plotting Magnetic Field Along z-axis
pylab.figure(3)
pylab.loglog(z,np.abs(Bz))                                               # Making A Plot
     With Log Scaling On Both The Axis
pylab.xlabel(r'z-axis$\rightarrow$',fontsize=15)                         # Setting The
     Label For The x-axis
pylab.ylabel(r'B(Magnetic_Field)$\rightarrow$',fontsize=15)              # Setting The
     Label For The y-axis
pylab.title('Magnetic_Field_Along_z_Axis',fontsize=15)                   # Setting Title
     Of The Graph
pylab.grid()                                                             # Displaying The
      Grid
pylab.show()                                                             # Displaying The
      Figure

# Solving For B Using Least Squares
A=np.hstack([np.ones(len(Bz[300:]))[:,np.newaxis],np.log(z[300:])[:,np.newaxis]])
log_c,b=np.linalg.lstsq(A,np.log(np.abs(Bz[300:])),rcond=None)[0]        # Returns log(c)
      and b
c=np.exp(log_c)                  # Exponential Function
print("The_Value_Of_b_is:")
print(b)                         # Printing The Value Of 'b' In The Terminal
print("The_Value_Of_c_is:")
print(c)                         # Printing The Value Of 'c' In The Terminal
```