EE2703 End Semester Report

Sagnik Ghosh, EE19B132

May 2021

1 Introduction

In the problem to be solved and analysed, we are given a current loop on the $X-Y$ plane centered at the origin such that the magnitude of the current is dependent on the azimuthal angle ϕ as follows:

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

As the computer cannot handle continuous-valued functions, we need to dicretize the problem at hand. This can be achieved by dividing the current loop into a finite number of current elements. The final problem is to compute and plot the magnetic field along the Z axis due to the current loop.

In order to do the above, we need to first compute the vector potential. The expression for vector potential is as follows:

$$\vec{A}(r,\phi,z) = \frac{\mu_0}{4\pi}\int \frac{I(\phi)\hat{\phi}e^{-jkR}ad\phi}{R}$$

The above expression, when simplified and discretized for the problem at hand, yields the following expression:

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

The individual terms in the above expression will be explained in the subsequent sections.

Finally, the magnetic field B can be calculated as follows:

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

Important Assumption: In the given expression for current I, we have a $\cos(\phi)$ term but if this is the case then diametrically opposite current elements will cancel each other. Hence, we will consider two cases:

- Case - A: We will assume in this case that the expression for I contains $|\cos(\phi)|$ instead of $\cos(\phi)$.
- Case - B: We will proceed with the given expression for I.

2 Problems

2.1 Pseudocode

The pseudocode used to compute the final magnetic field can be listed down as the following steps:

- Define a $(3, 3, 1000, 3)$ matrix of the vector potentials which stores the vector potentials of each point in the 3 by 3 by 1000 mesh surrounding the Z axis as 3-dimensional vectors.
- Initialise the parameters of the simulation such as k, radius of the loop, etc.
- Divide the loop into 100 sections. Create 100-dimensional vectors which store the magnitude of currents, X coordinates, Y coordinates, angular position ϕ.
- Plot the current elements using the vectors defined above and the quiver() function of the MatPlotLib Library.
- Define a function calc(l) which returns a $(3, 3, 1000)$ matrix containing the distances of each point in the 3 by 3 by 1000 grid from a particular current element indexed by l. This will be used later to compute the vector potentials.
- Define a function calc_A(l) which computes A by calling the function calc(l) defined earlier. This function returns a $(3, 3, 1000, 3)$ matrix containing 3-dimensional vectors corresponding to the vector potentials at each of the points in the 3 by 3 by 1000 grid for a particular current element indexed by l.
- Iterate over all the current elements and accumulate the vector potentials corresponding to each current element by calling the function calc_A(l) repeatedly.
- Compute the magnetic field B from the matrix containing the vector potentials computed earlier using the expression given in the introductory section.
- Plot the magnetic field along the Z axis computed above.
- Now, we are supposed to fit the magnetic field to a function of the form $B_z \approx cz^b$. We can solve an equivalent problem of fitting $\ln(B_z)$ to $b\ln(z) + \ln(c)$. Hence we can use np.linalg.lstsq() to determine b and $\ln(c)$.

2.2 Defining the Mesh

We can define the mesh by a 3 by 3 by 1000 grid around the Z axis using **np.meshgrid()**. In the code below, we define $(3, 3, 1000, 3)$ matrix which contains the coordinates of each point in the 3 by 3 by 1000 grid. We also initialize the vector potential matrix.

```
# Breaking the volume into 3*3*1000 grid
# creating arrays to store the x, y, z coordinates
x, y, z = np.meshgrid(np.arange(-1, 2), np.arange(-1, 2), np.arange(1, 1001))
# r contains the coordinates of the 3x3x1000 volume
r = np.stack((x, y, z), axis = -1)
# Matrix A can be indexed as (x_location, y_location, z_location)
A = np.zeros((3, 3, 1000, 3), dtype = 'complex128')
```

2.3 Plotting the Current Elements

In this section, we will plot the current elements of the current loop. This can be done using the quiver() function.

```
# Plotting current elements

element_angles = np.linspace(0, 2*np.pi, 101)[:-1] # angles of each current element
radius = 10
k=0.1
length = 2*np.pi*radius/100 # length of each element
# current vector in x direction
current_x = -length*np.sin(element_angles)*np.abs(np.cos(element_angles))*(10**7)
# current vector in y direction
current_y = length*np.cos(element_angles)*np.abs(np.cos(element_angles))*(10**7)
x_pos = radius*np.cos(element_angles) # x coordinate of elements
y_pos = radius*np.sin(element_angles) # y coordinate of elements

fig, ax = plt.subplots(figsize = (10,10))
fig = ax.quiver(x_pos, y_pos, current_x, current_y, width = 0.004)
plt.grid(True)
plt.xlabel(r'$X\rightarrow$')
plt.ylabel(r'$Y\rightarrow$')
plt.title(r'Plot of Current Elements in the Loop')
plt.show()
```

The above code is for plotting Case - A. It can be done similarly for Case - B by removing np.abs().

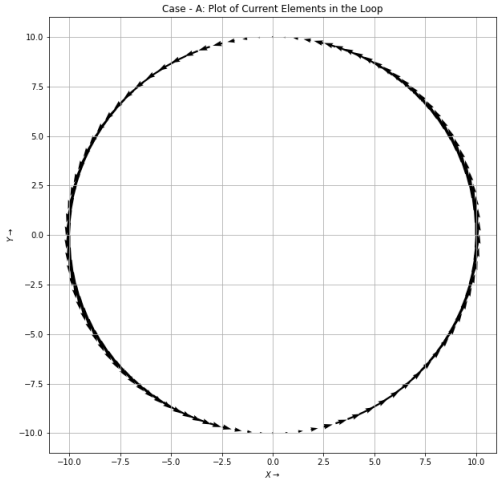

Figure 1: Case - A: Plot of Current Elements in the Loop

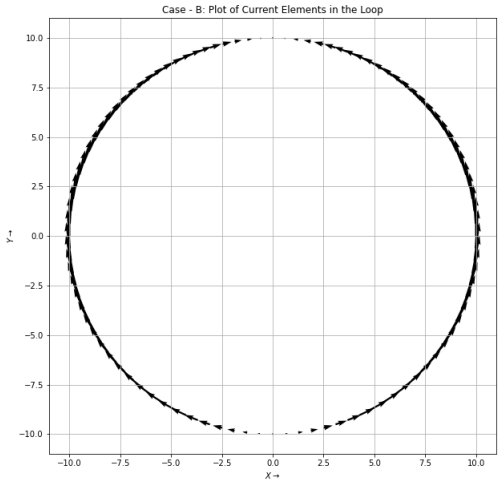

Figure 2: Case - B: Plot of Current Elements in the Loop

2.4 Obtaining the Vectors $\vec{r_l'}, \vec{dl}_l$

Vectors $\vec{r_l'}, \vec{dl}_l$ contain the coordinates and direction of each of the current elements in the loop respectively. These vectors can be obtained as follows:

```
# Finding rl' and dl'

rl = np.c_[x_pos, y_pos, np.zeros_like(x_pos)]
dl = np.c_[-length*np.sin(element_angles),
length*np.cos(element_angles), np.zeros_like(x_pos)]
```

2.5 Defining a Function to Compute $\vec{R}_{ijkl} =\mid \vec{r}_{ijk} - \vec{r_l'}\mid$

The defined function calc(l) returns R($\vec{R}_{ijkl}$) which is a $(3, 3, 1000, 3)$ matrix containing 3-dimensional vectors corresponding to the vector potentials at each of the points in the 3 by 3 by 1000 grid for a particular current element indexed by l.

```
def calc(l, rl):
    # creating arrays to store the x, y, z coordinates
    x, y, z = np.meshgrid(np.arange(-1, 2), np.arange(-1, 2), np.arange(1, 1001))
    # r contains the coordinates of the 3x3x1000 volume
    r = np.stack((x, y, z), axis = -1)
    # R contains |rijk-rl| for all i, j, k and is of shape 3x3x1000
    R = np.linalg.norm(r-rl[l], axis = -1)

    return R
```

2.6 Finding the Vector Potential A for a Particular Current Element

The expression for the Vector Potential A is given by:

$$\vec{A}_{ijk} = \sum_{l=0}^{N-1} \frac{\cos\left(\phi_l'\right) \exp(-jkR_{ijkl}) d\vec{l'}}{R_{ijkl}}$$

Note that each term in the summation is a function of $\vec{R}_{ijkl}$ which is returned by the earlier defined function calc(l). We will now define another function calc_A(l) which will call calc(l) and return the Vector Potential for a Particular Current Element.

```
def calc_A(l, rl, dl, element_angles):
    R = calc(l, rl)
    A = np.abs(np.cos(element_angles[l]))*np.exp(-1j*k*R)/R
    # Stacking A thrice so that multiplication with dl is vectorised
    A = np.stack((A, A, A), axis = -1)
    A = A*dl[l]

    return A
```

The above code is for Case - A. The code for Case - B can be derived by removing np.abs().

2.7 Iterating over Current Elements to find final Vector Potential

We have already defined a function calc_A(l) which returns the Vector Potential for a Particular Current Element. We need to sum the vector potentials contributed by every current element to obtain the final vector potential.

```
A1 = np.zeros((3, 3, 1000, 3), dtype = 'complex128')
# Accumulating A for every current element
for l in range(len(element_angles)):
    Al = calc_A(l, rl, dl, element_angles)
    A1 += Al
```

2.8 Computing Magnetic Field $\vec{B}$

The expression for Magnetic Field in terms of the Vector Potential is as follows:

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

This can be implemented as follows:

```
# Implementing the formula of B in terms of A
B = (A1[2, 1, :, 0] - A1[1, 2, :, 1] - A1[0, 1, :, 0] + A1[1, 0, :, 1])/4
```

2.9 Plotting Magnetic Field $\vec{B}$ Along Z Axis

We have already computed the Magnetic Field in the previous section. This can be plotted on a loglog plot as follows:

```
# Plotting B
plt.loglog(np.abs(B))
plt.xlabel(r'$Z\rightarrow$')
plt.ylabel(r'Magnetic Flux Density (B)$\rightarrow$')
plt.title(r'Case - A: Plot of Magnetic Flux Density as a Function of $z$')
plt.grid(True)
plt.show()
```

The following are the plots for both Case - A and Case - B. Note that in the below plots, for Case - A we get a smooth curve whereas we get a noisy curve which is nearly zero for Case-B which is expected as diametrically opposite current elements nullify each other.

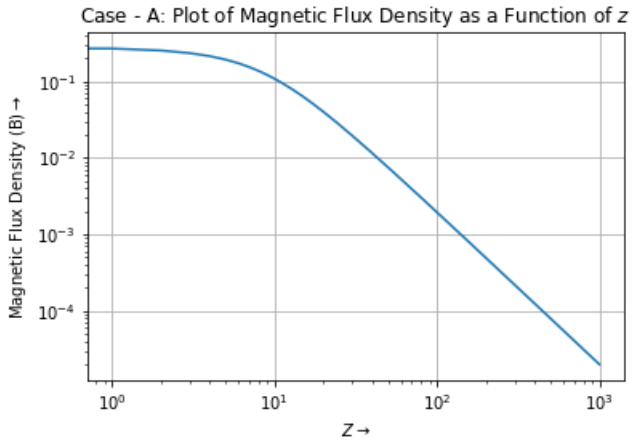

Figure 3: Case - A: Plot of Current Elements in the Loop

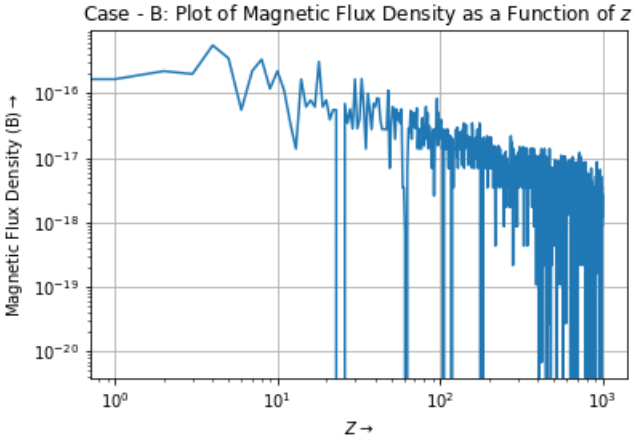

Figure 4: Case - B: Plot of Current Elements in the Loop

2.10 Fitting the Magnetic Field to $B_z \approx cz^b$

Fitting the Magnetic Field to $B_z \approx cz^b$ is equivalent to fitting $\ln(B_z)$ to $b\ln(z) + \ln(c)$. Hence we can use np.linalg.lstsq() to determine b and $\ln(c)$.

In order to do this, we need to generate a matrix whose columns are $ln(z)$ and all-ones. Using np.linalg.lstsq() will give us b and $\ln(c)$.

```
# Fitting Least Squares

def generate_M(z):
    a = np.log(z)
    b = np.ones_like(z)
    M = np.c_[a, b]
    return M
# Applying Least Squares
b, c = np.linalg.lstsq(generate_M(np.arange(1,1001)), np.log(np.abs(B)), rcond = None)[0]

c = np.exp(c) # Value of c returned by lstsq is actually ln(c)

plt.loglog(np.abs(B))
plt.loglog(c*(np.arange(1,1001))**b)
plt.legend(['Actual Magnetic Field', 'Least Squares Estimate'])
plt.xlabel(r'$Z\rightarrow$')
plt.ylabel(r'Magnetic Flux Density (B)$\rightarrow$')
plt.title(r'Plot of Magnetic Flux Density as a Function of $z$')
plt.grid(True)
plt.show()
```

The values of b and c thus obtained for Case - A are:

```
Value of b = -1.91
Value of c = 11.21
```

If we set k = 0, i.e., for the static case, we get:

```
Value of b = -2.83
Value of c = 68.68
```

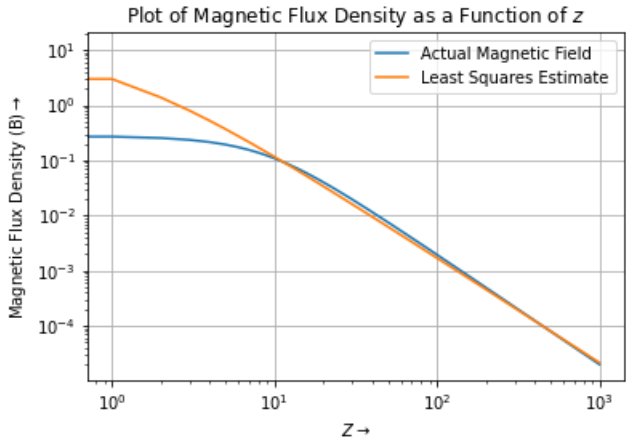

Figure 5: Plot of Magnetic Flux Density as a Function of z

2.11 Discussion

- For the case of the non-static case, we found that the **Magnetic Field decays as** $z^{-1.91}$ and for the static case the Magnetic Field decays as $z^{-2.83}$.

- For the case of the static case with uniform current, the magnetic field is as follows:

$$B_z = \frac{\mu_0 I}{2} \frac{R^2}{(R^2+z^2)^{3/2}}$$

  Hence, we expect the field to decay approximately as z^{-3} and we get the variation as $z^{-2.83}$ and hence the magnetic field falls off as expected.

- The difference mainly comes from the fact that the resolution we chose for the grid was not adequate. This can be rectified by choosing a grid of smaller physical dimensions and a larger number of points. Doing this would come at an additional computational cost.