EE2703 : Applied Programming Lab
Final Examination
Radiation from Loop Antenna

Bachotti Sai Krishna Shanmukh
EE19B009

May 29, 2021

Introduction

We aim to understand the current distribution in a Loop Antenna and compute the magnetic field along z-axis. We also try to find the relationship between the magnetic field and z.

1 Breaking the volume into Mesh

1.1 Creating mesh points

We first create Numpy arrays in space which represent the coordinates of points in Cartesian subspace. These points are known as mesh points. Let's first define the space for which we need to create these mesh points.

$$x \in [0, 2]$$

$$y \in [0, 2]$$

$$z \in [1, 1000]$$

Each mesh point is separated by 1cm (1 unit). We use np.meshgrid() function to get the coordinates for each mesh point.

```
x=np.linspace(0,2,3)
y=np.arange(2,-1,-1)
z=np.linspace(1,1000,1000)

Y,Z,X=np.meshgrid(y,z,x)
```

The mesh grids are created in such a manner that each grid takes indices in the following order:

```
Array['Z Co-ordinate index','Y Co-ordinate index','X Co-ordinate index']
```

To get a better feel for this, let's use some surface plotting.

1.2 Z = 1 Plane

In the array z, the entry 1 is at 0-th index. Hence to get Z = 1 plane, we use the slice : X[0], Y[0], Z[0]

```
surf = ax.plot_surface(X[0],Y[0],Z[0])
```

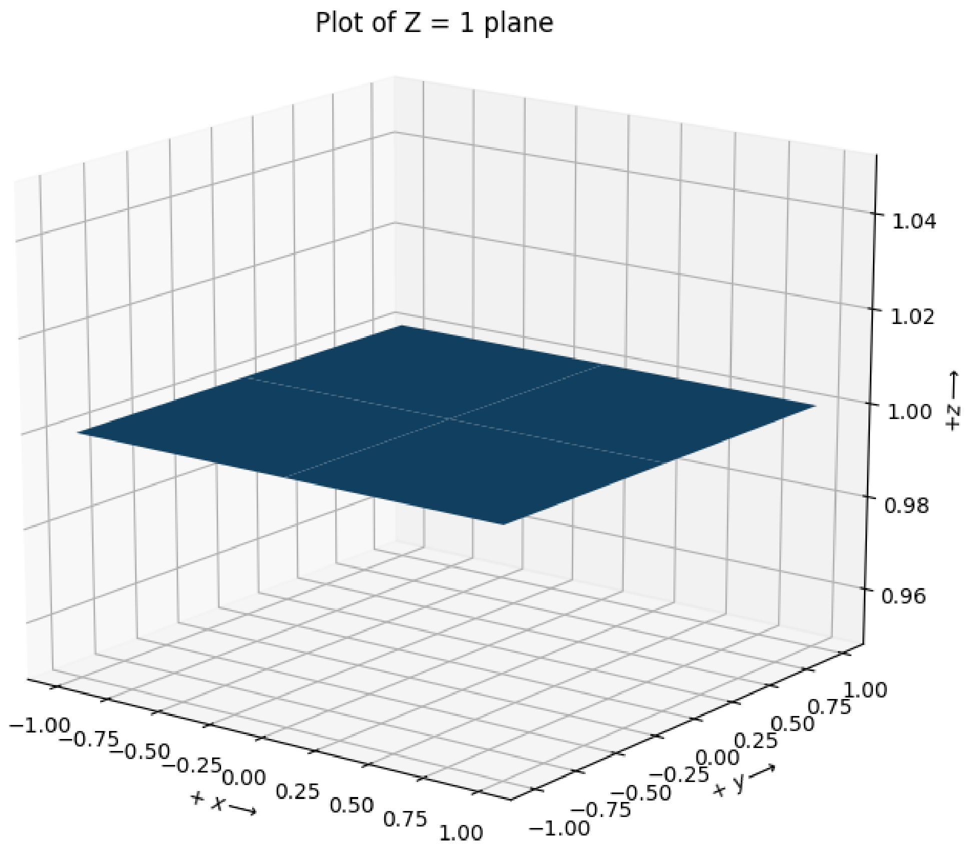

Figure 1: Z = 1 Plane

1.3 Y = 2 Plane

In the array y, the entry 2 is at 0-th index. Hence to get Y = 2 plane, we use the slice : X[: , 0 , :], Y[: , 0 , :], Z[: , 0 , :]

```
surf = ax.plot_surface(X[:,0,:],Y[:,0,:],Z[:,0,:])
```

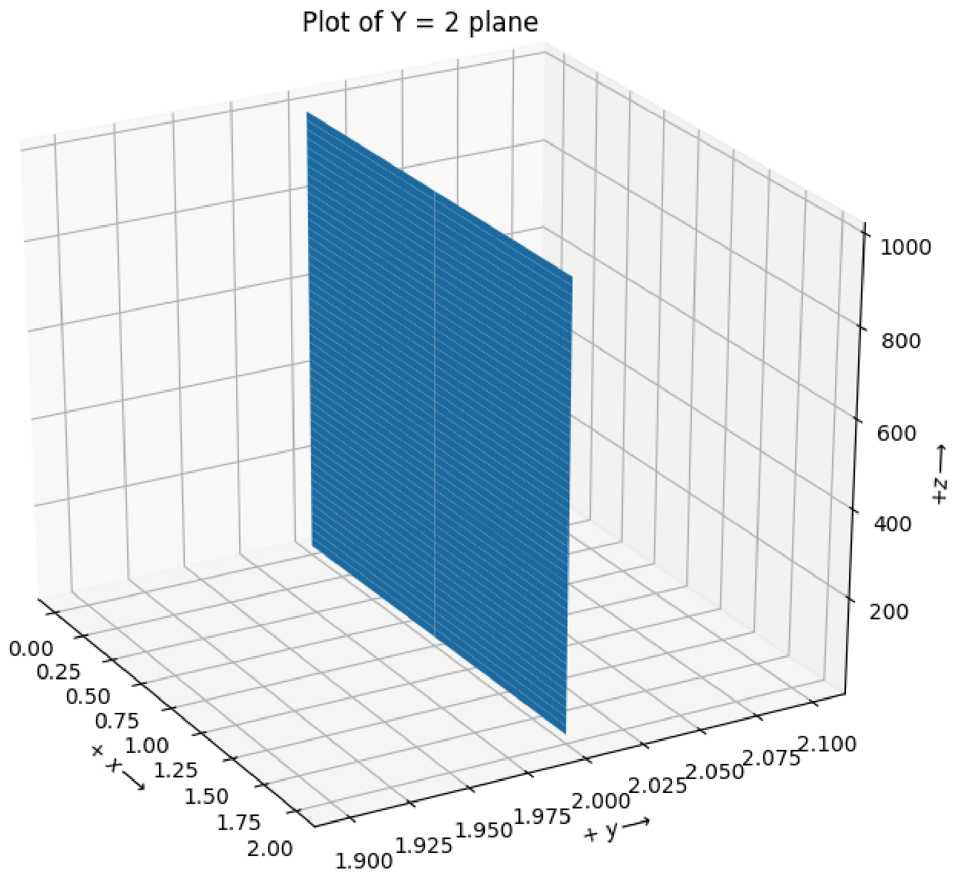

Figure 2: Y = 2 Plane

1.4 X = 0 Plane

In the array x, the entry 0 is at 0-th index. Hence to get X = 0 plane, we use the slice : X[: , : , 0], Y[: , : , 0], Z[: , : , 0]

```
surf = ax.plot_surface(X[:,:,0],Y[:,:,0],Z[:,:,0])
```

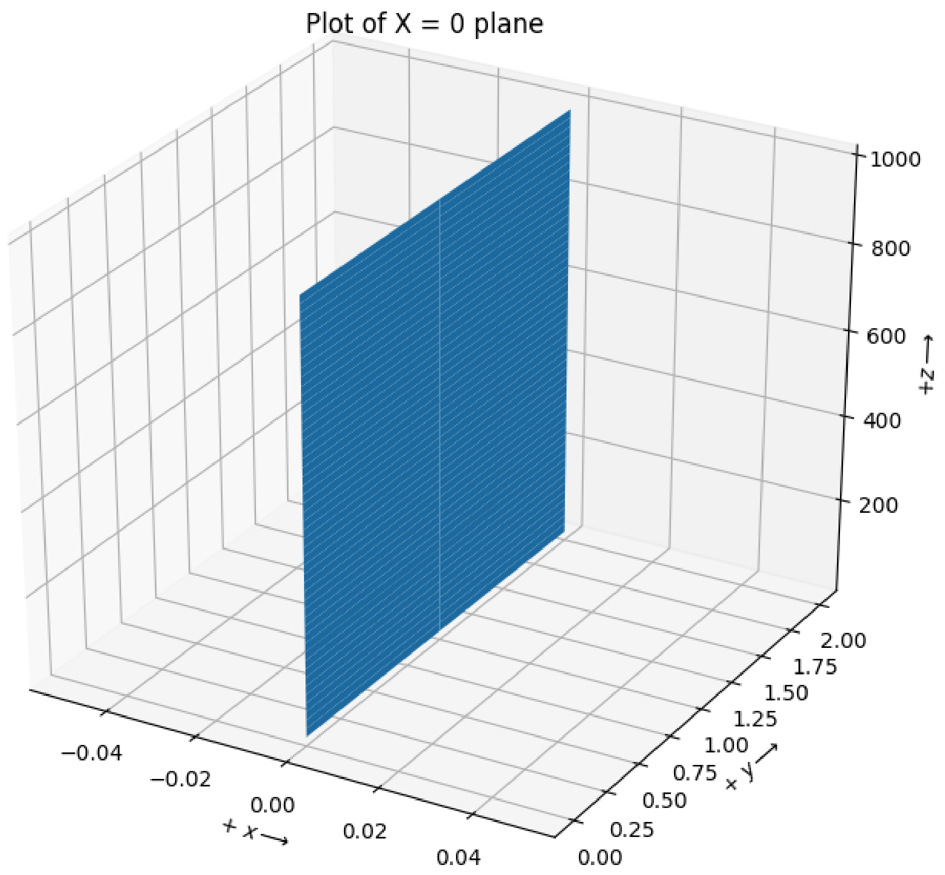

Figure 3: X = 0 Plane

2 Current distribution in Loop

Given, the radius(R) of loop is 10 cm. Let's break the loop into 'N' sections. The below figure depicts the midpoints of these sections. The angular positions of these midpoints can be found out by the following code.

```
midpt_angles = np.linspace(0,2*np.pi,N+1)[:-1]
coses = np.cos(midpt_angles)
sines = np.sin(midpt_angles)
x_c = rad*coses # Coordinates of ...
y_c = rad*sines # ... Midpoints
```

The position vectors of the midpoints are given by:

$$\vec{r} = R\cos(\phi)\hat{e_x} + R\sin(\phi)\hat{e_y}$$

If (x,y) are the coordinates of the point then $cos(\phi)$ and $sin(\phi)$ can be computed by

$$\cos\phi = \frac{x}{\sqrt{x^2+y^2}}$$

$$\sin\phi = \frac{y}{\sqrt{x^2+y^2}}$$

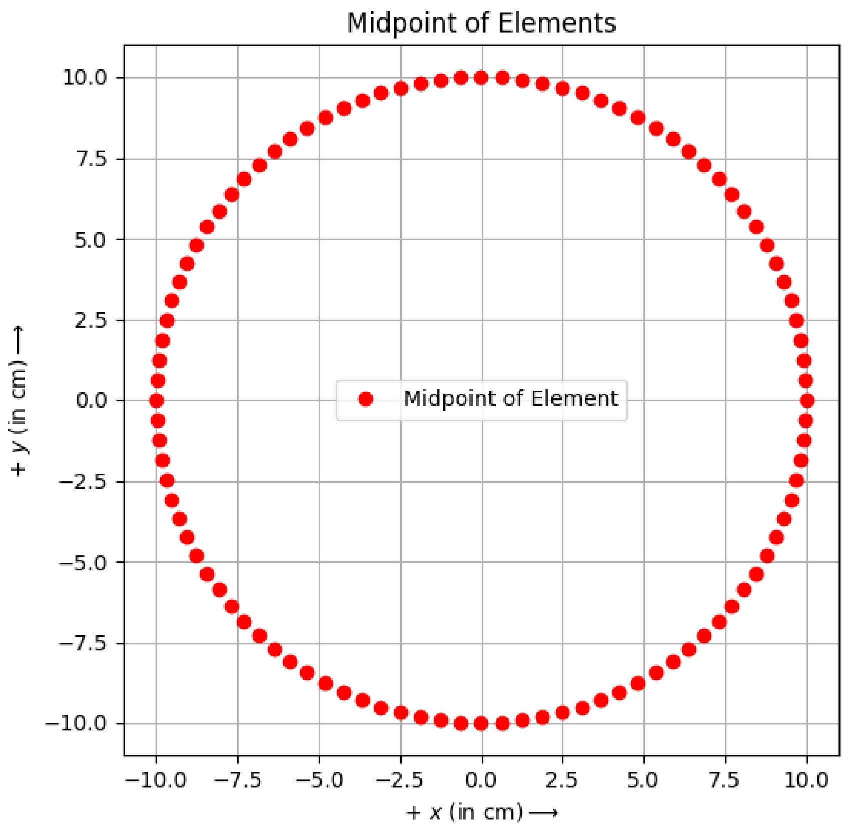

Figure 4: N = 100 sections

The current distribution in the loop is given by the following function, where ϕ is the angular position.

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

The tangential vector $\vec{dl}$ for given angular position ϕ is given by:

$$\hat{dl} = -\sin(\phi)\hat{e_x} + \cos(\phi)\hat{e_y}$$

$$|dl| = r \times d\theta = r \times \frac{2\pi}{N}$$

$$\vec{dl} = |dl|\hat{dl}$$

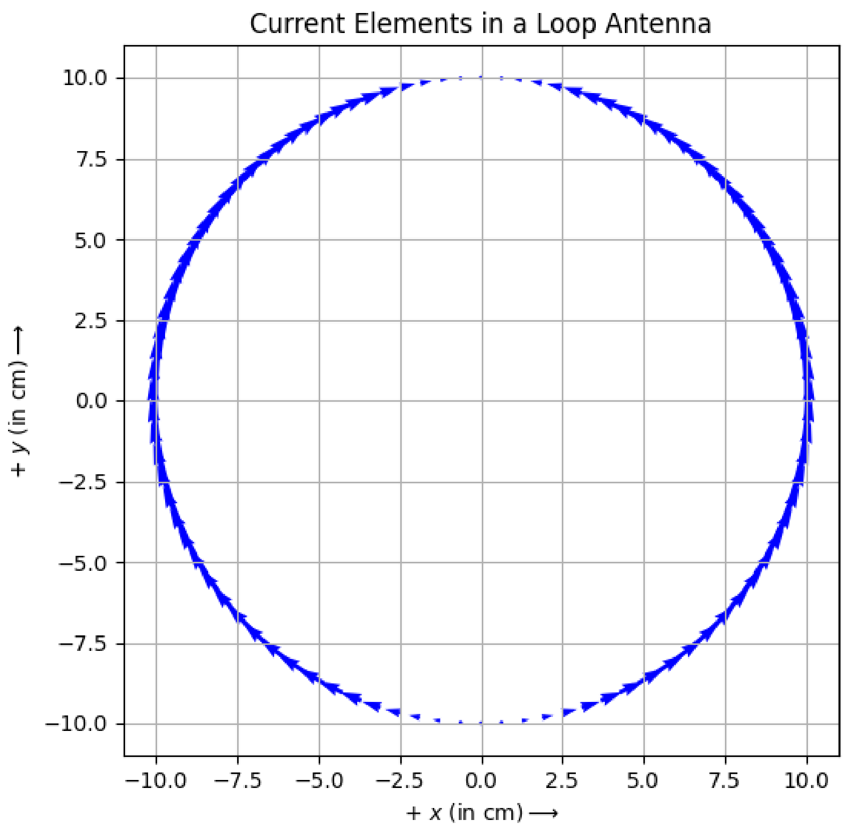

Figure 5: Quiver Plot of Current Elements

The current element vectors $I\vec{dl}$ in the loop can be found out by the following defined function code:

```python
def currentelement_spatial(phi,dl):
    """
    Returns current element vector for given index(l) of segment
    4pi/mu_0 = 1e7
    """
    return dl*np.array([-1e7*np.cos(phi)*np.sin(phi),1e7*np.cos(phi)**2]).T

dl = 2*np.pi*rad/N                             # Magnitude of dl = r*d_theta
Idl = currentelement_spatial(midpt_angles,dl)    # Function call
fig,ax = plt.subplots(figsize=(6,6))
ax.quiver(pts[:,0],pts[:,1],Idl[:,0],Idl[:,1],color='b')
```

The $\vec{dl}$ vector can also be found out separately by the following code:

```python
dl_vec = dl*np.vstack((-sines,coses)).T
```

3 Computing Vector Potential and Magnetic Fields

The vector potential $\vec{A}$ is given by

$$\vec{A} = \frac{\mu_0}{4\pi} \int \frac{I(\phi)e^{-jkR}\vec{dl'}}{R} \tag{2}$$

This integral expression can be approximated to a summation as shown below:

$$\vec{A}_{ijk} = \sum_{l=0}^{N-1} \frac{cos(\phi'_l)e^{-jkR_{ijkl}}\vec{dl'}}{R_{ijkl}} \tag{3}$$

In the above expression, R_{ijkl} is the matrix with distances between midpoint of l-th segment of the loop antenna to all mesh points in the grid. To compute this distance, calc(l), a vectorized function is defined as follows:

```python
def calc(l):
    """
    Input : l lies between 0 to N-1
    Return :
    This function returns the vector A field for all points in grid
    due to the l-th segment in the loop.
    """
    coords = pts[l]  # Midpoint of l-th segment
    xl = coords[0]   # x coordinate
    yl = coords[1]   # y coordinate
    x_diff = X - xl
    y_diff = Y - yl
    distance = np.sqrt(x_diff**2 + y_diff**2 + Z**2)  # L2 distance
    distance = distance.reshape((1000,3,3,1))
    # Reshaping the array to get the advantage of broadcasting
    return coses[l]*np.exp(-1j*0.1*distance)*dl_vec[l]/distance
A = 0   #Initialising variable
for l in range(N):
    A += calc(l)   #Increment
```

The above function is extended to return the term inside the summation, in equation (3). We can now use the calc(l) function to find $\vec{A}$ due to each segment in the loop and vector-add them by using a 'for loop'. The usage of 'for loop', here, can be justified by the fact that even if we vectorize the computation of $\vec{A}$ for all segments, we will have to use the np.sum() function along a specific axis to get the resultant field. This operation has almost same computational complexity as using of a 'for loop'. Hence, the latency in both methods do not show a significant difference.

```
Shape of distance before reshaping is (1000,3,3)
Shape of A is (1000,3,3,2)
```

In the array A, the first dimension is the z-coordinate index, the second dimension is y-coordinate index, the third dimension is the x-coordinate index and the last dimension is

the direction index of the field.
Hence the x- component of $\vec{A}$ is given by A[: , : , : , 0] and the y-component of $\vec{A}$ is given by A[: , : , : , 1].

To compute the magnetic field $\vec{B}$ we can use the below formula:

$$\vec{B} = \nabla \times \vec{A} \tag{4}$$

$$B_z = \frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}$$

We approximate the above operation by using the following equation.

$$B_z(1,1,z) = \frac{A_y(1+\Delta x,1,z) - A_y(1-\Delta x,1,z)}{2\Delta x} - \frac{A_x(1,1+\Delta y,z) - A_x(1,1-\Delta y,z)}{2\Delta y} \tag{5}$$

We know that, Δx and Δy are each 1 cm.

- $A_y(1+\Delta x,1,z)$ can be written as A[: , 1 , 2 , 1], since y = 1 is at index 1 and x = 2 is at index 2. The last dimension index is 1 which represents the field along y - direction.

- $A_y(1-\Delta x,1,z)$ can be written at A[: , 1 , 0 , 1], since y = 1 is at index 1 and x = 0 is at index 0. The last dimension index is 1 which represents the field along y - direction.

- $A_x(1,1+\Delta y,z)$ can be written at A[: , 0 , 1 , 0], since y = 2 is at index 0 and x = 1 is at index 1. The last dimension index is 0 which represents the field along x - direction.

- $A_x(1,1-\Delta y,z)$ can be written at A[: , 2 , 1 , 0], since y = 0 is at index 2 and x = 1 is at index 1. The last dimension index is 0 which represents the field along x - direction.

Below is the code for finding the B_z field.

```python
B = (A[:,1,2,1] - A[:,1,0,1] + A[:,2,1,0] - A[:,0,1,0])/2
```

To understand the how B_z varies with z, we plot the magnitude of B_z vs z in log-log scale. Below is the code and plot:

```python
plt.loglog(z,np.abs(B),'bo-',markersize=3)
plt.xlabel(r'+ $z$ (in cm)$\longrightarrow$')
plt.ylabel(r'+ $B_z$ (in Tesla) $\longrightarrow$')
plt.title('Log-Log plot of Magnetic Field along +z direction')
plt.grid()
```

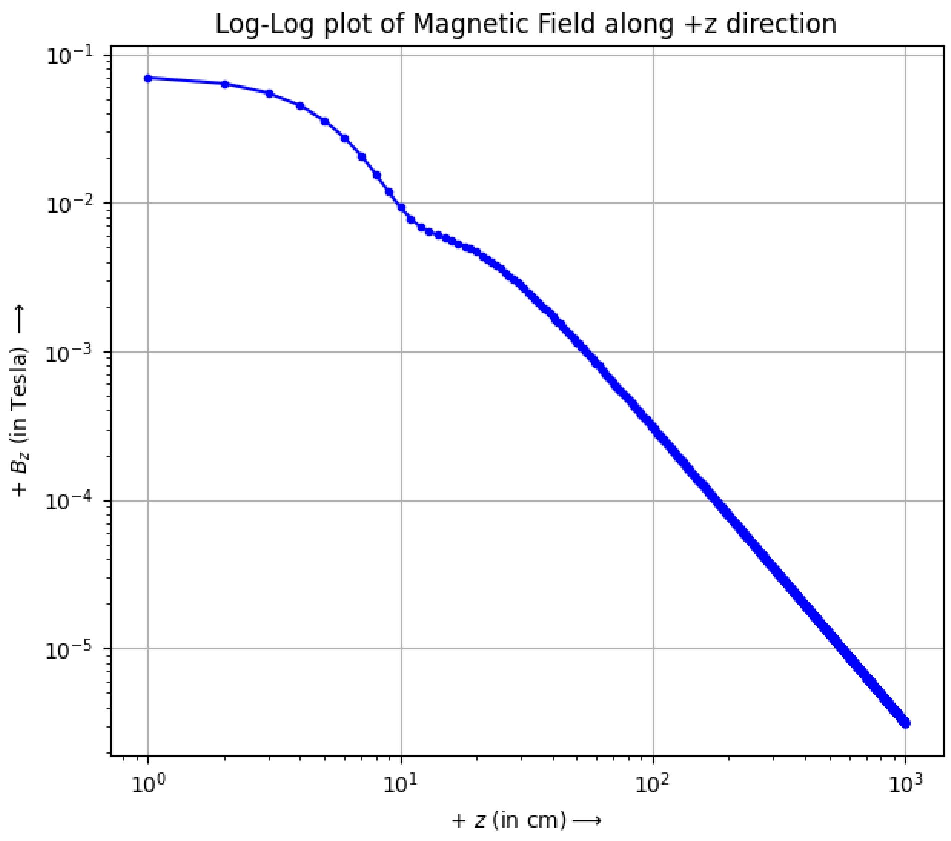

Figure 6: B_z vs z in log-log scale

4 Finding the Least Squares Fit

We can use the method of least squares to find a suitable approximate relationship between magnitude of B_z vs z. Let's try to model this relationship to the following equation

$$B_z = cz^b \tag{6}$$

The above equation can be modified as :

$$log(B_z) = log(c) + b \times log(z)$$

By least squares method, we can estimate the parameters log(c) and b by using the previously computed B_z array and z. Below is the code for the same:

```python
C = np.log(np.abs(B))
A = np.c_[np.ones(1000),np.log(z)]
params = slg.lstsq(A,C)[0]    # Lstsq fit parameters

c = np.exp(params[0])
b = params[1]
```

The parameters b and c computed using the Least Squares method are shown below:

```
b is  -1.8698222544166256
c is  1.4048479369070908
```

Below is the plot of least squares fit along with the original curve.

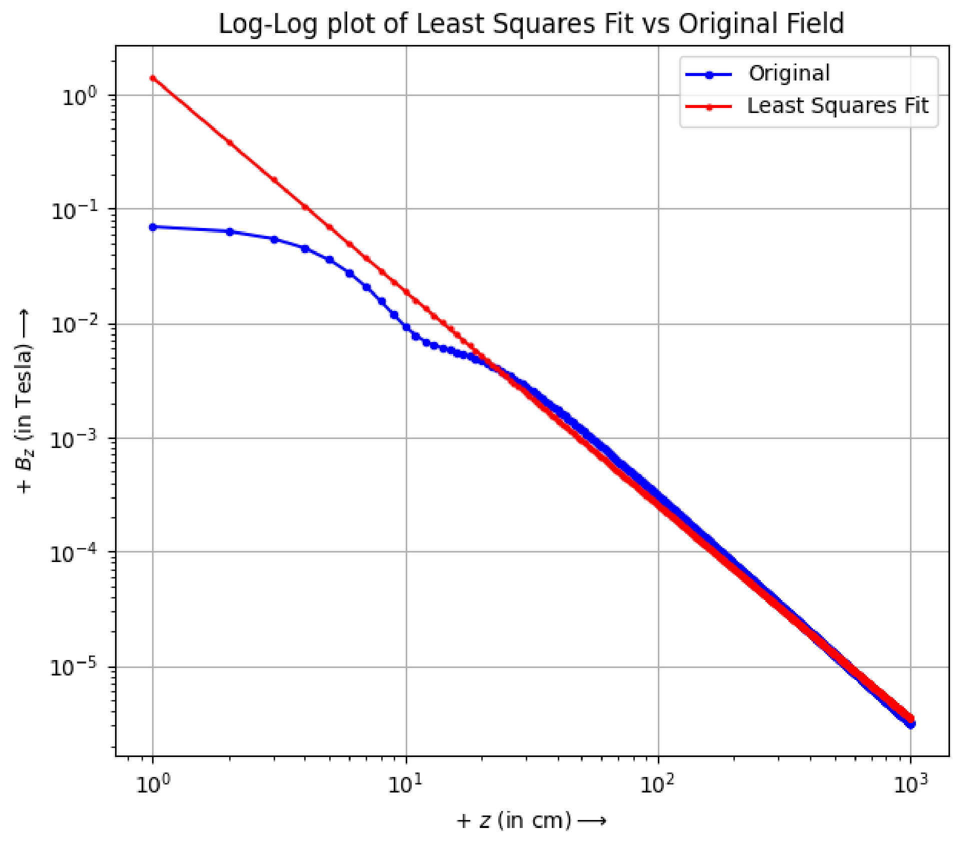

Figure 7: Least Squares fit and Original magnitude of B_z

From the least squares method, we can show that the magnetic field component B_z falls off as:

$$B_z \approx \frac{1.404}{z^{1.869}} \tag{7}$$

In case of magneto-static current distribution, the magnetic field due to a loop of radius R is given by

$$\vec{B} = \frac{\mu_0 I r^2}{2(z^2 + R^2)^{3/2}} \hat{e_z} \tag{8}$$

If $z >> R$

$$\vec{B} \approx \frac{\mu_0 I r^2}{2z^3} \hat{e_z}$$

This difference in B_z vs z relationship in loop antenna arises due to the spatial and time variation of current in the loop.

Conclusion

In this assignment, we first defined the space for which the fields are to be found out. The loop was divided into 'N' sections and the position vectors of midpoint of these sections were found. We then computed the current element vectors for given current distribution of the loop and understood the current element direction and distribution through quiver plot. We later found the vector potential due to each section using calc(l) function and then found the net vector potential due to all sections. The curl equation was approximated to a difference equation and B_z was computed at an axis passing through (1,1) in the 3×3 grid plane and along the z-axis using this approximate equation. The B_z component in a loop antenna falls off w.r.t z with a decay of 1.869 which is approximately estimated using the Least squares method.