Radiation from a Loop Antenna

Sai Gautham Ravipati - EE19B053

July 19, 2021

Abstract

Antennas generate electromagnetic radiation and act as transducers by converting current and voltages to EM quantities and vice-versa. A loop antenna is a special kind of antenna formed by a loop of wire, with circulating current. Through this assignment, the radiation pattern of one such loop antenna is analysed using Python programming. The z-component of magnetic field is computed on a line parallel to z-axis passing through a point in the x-y plane, when the loop is aligned in the x-y plane. This is done using vectorised operations on Python arrays to obtain the field and later least squares approach is used to obtained a particular fit. The rest of the analysis follows.

Contents

1 Introduction

Consider a circular loop antenna of radius a = 10 cm. The current through the antenna is given as follows:

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

where μ_o is the permeability of free space, ϕ is the angle in the polar coordinate and ω is the frequency. Another relation of use is $\frac{1}{k} = \frac{c}{w}$, where c is the speed of light. The vector potential $\vec{A}$ is computed as follows :

$$\vec{A}(r,\phi,z) = \frac{\mu_o}{4\pi}\int \frac{I(\phi)\hat{\phi}e^{-jkR}ad\phi}{R} \tag{2}$$

where $\vec{R} = \vec{r} - \vec{r'}$, k is the wave-number and is taken to be 0.1. $\vec{r}$ is a point in space where the field is to be computed and $\vec{r'}$ is the point on the loop, which is given as $a\hat{r'}$. The magnetic field in space is computed as the curl of $\vec{A}$ and the same is given below :

$$\vec{B} = \boldsymbol{\nabla} \times \vec{A} \tag{3}$$

These equations when translated to their equivalent vector forms for Python computation shall give the magnetic field at required locations. The task is to compute the magnetic field along a line parallel to z-axis passing through (1,1,0) from z = 1 cm to 1000 cm. Then the same is fit to a function of the form Cz^b.

2 Pseudo-Code

The pseudo-code for the implementation is provided below. Please note that some terms shall be more clearer as we move further. The lines in actual code shall be related to pseudo code as we move on.

```
Start
Set radius, a to 10
Set number of sections, n to 100
Select the range of points in space separated by 1 cm
    x - [0,2];
    y - [0,2];
    z - [1,1000]
Form a mesh of points, r_ijk in space in the selected range

Break the loop to n segments
    polar coordinate, p - [0,2*pi) with n points
Set segment length, s to 2*pi*a/n
Compute scaled current, I as {-cos(p)*sin(p), cos(p)*cos(p)}
Compute location of segments, r' as {a*cos(p), a*sin(p)}
Compute segment vectors, dl' as {-s*sin(p), s*cos(p)}
Plot the vector currents, I and mid-points of segments, r'

Function calc
    Pass In: Integer corresponding to index of the segment
    Compute the distance of the all points of the mesh, R from the segment as
        |r_ijk - r'(segment)|
    Compute the individual terms in vector A corresponding to a given segment
        as {cos(p)*exp(-0.1j*R)/R}*dl'
```

```
22      Pass Out: A_l, Sub-vector to be added to vector A, corresponding to a given
            segment
23  Endfunction
24
25  Init vector A, call calc function return A_l, for the first segment
26
27  For each segment (l : 1 to n - 1)
28     call calc function, return A_l, for segment l
29     Accumulate vector A with A_l
30  Endfor
31
32  Compute z-component of field at (1,1) using the curl(A), where  A is given as
        {Ax,Ay}
33  Plot the magnetic field, Bz and the z values
34
35  Function fit
36      Pass In: Vector B, holding the magnetic field and vector z, holding the z
            points
37      Compute C,b for the fit B = Cz^b, translates to {b,log(C)}{log(z),1}.T =
            log(B), using least squares minimisation
38      Pass Out: Coefficients of the fit C,b
39  Endfunction
40
41  Fit Bz to Cz^b points using call fit function, return C,b
42  Plot the orginal field points Bz, the fit Cz^b and the z points on the same plot
43  End
```

3 Vector Currents and Segments

3.1 Setting up the Space

The Eq. 2, stated previously is an integral and is to be converted to a Riemann sum to solve numerically. The equation when converted to a sum reads as follows:

$$\vec{A}_{ijk} = \sum_{l=0}^{N-1} \frac{cos(\phi_l')e^{-jkR_{ijkl}}\vec{dl'}}{R_{ijkl}} \tag{4}$$

where $\vec{r}$ is at r_i, ϕ_l, z_k, and $\vec{r'}$ is at $a\cos(\phi_l')\hat{x}+a\sin(\phi_l')\hat{y}$. This equation is valid for any (x_i, y_j, z_k) and is summed over the current elements in the loop. This is as a vector operation over both l and over a vector of (x_i, y_j, z_k) values. To implement the same, we need to divide the volume into a 3 by 3 by 1000 mesh, with mesh points separated by 1cm. This is done by lines 4-7 in the below code and corresponds to lines 4-8 in the pseudo code. Once this is done we have a set of vector points (x_i, y_j, z_k). However we still need to divide the loop into segments, so that we have segments represented by index l, which gives us a vector operation over l. This is done by generating the variable phi $\in [0, 2\pi)$ such that there are n(100) points indicating the division of the loop into n(100) segments which are identified by the polar coordinate ϕ_l'. This is done by line 9 in the below code and corresponds to lines 10,11 in the pseudo-code. So till now we have divided the 3D volume and the ring into segments which gives us the indices going to be used in the computation of Eq. 4. Line 10 gives the length of each segment. The selection of location of mesh-grid points apart from the dimension is analysed again, once we have the plot of current in the later section.

```
a  = 10
n  = 100

d1 = np.linspace(0,2,3)
d2 = np.linspace(0,2,3)
d3 = np.linspace(1,1000,1000)
X,Y,Z = np.meshgrid(d1,d2,d3)

phi = np.linspace(0, 2*np.pi, n, endpoint = False)
s   = 2*np.pi*a/n
```

3.2 Plot of Vector currents and segments

Now we have the loop divided into segments, each segment can be represented by its mid-points whose position vector is given by $a\cos(\phi_l')\hat{x} + a\sin(\phi_l')\hat{y}$, this is generated in the python code below using line 2, and '*np.c_*' has been used to store the x-points $acos(phi)$ and y-points $asin(phi)$ where phi has been generated previously, into a single vector. We don't need the actual current as $4\pi/\mu_o$ gets cancelled out in Eq 2. So we look at the scaled current. The direction of current is along $\vec{dl}'$, so the current is taken as $[-sin(phi)cos(phi), cos(phi)sin(phi)]$ in line 1. $\vec{dl}'$ is the derivative of $\vec{r}'$, so it is given by $-s\sin(\phi_l')\hat{x} + s\cos(\phi_l')\hat{y}$, where s is the length of each segment and this translates to line 3 in the below code. The python computation done in this section corresponds to lines 12-15 in pseudo-code.

```
I  = np.c_[-np.sin(phi)*np.cos(phi),np.cos(phi)*np.cos(phi)]
r1 = a*np.c_[np.cos(phi),np.sin(phi)]
dl = s*np.c_[-np.sin(phi),np.cos(phi)]

plt.figure(0,figsize = (8,8))
plt.quiver(r1[:,0], r1[:,1], I[:,0], I[:,1], scale = 20, label = 'Vector
    Currents')
plt.plot(r1[:,0],r1[:,1],'r.',label = 'Centre Points of Elements',markersize =
    5)
plt.title('Current Elements in x-y plane, split into {} sections'.format(n),
    fontsize = 14)
plt.legend(loc = 'upper left')
plt.xlabel(r'$\leftarrow$ x (in cm) $\rightarrow$',fontsize = 12)
plt.ylabel(r'$\leftarrow$ y (in cm) $\rightarrow$',fontsize = 12)
plt.grid(True)
plt.axis('square')
plt.annotate('Max. Amplitude',(a,0),xytext = (-135,0),textcoords = 'offset
    points',bbox = bbox,arrowprops = arrowprops)
plt.annotate('Min. Amplitude',(0,-a),xytext = (0,45), textcoords = 'offset
    points',bbox = bbox,arrowprops = arrowprops)
plt.annotate('Max. Amplitude',(-a,0),xytext = (57,0), textcoords = 'offset
    points',bbox = bbox,arrowprops = arrowprops)
plt.annotate('Min. Amplitude',(0,a),xytext = (0,-50), textcoords = 'offset
    points',bbox = bbox,arrowprops = arrowprops)
plt.savefig('EE2703_plots/vector_current.png')
```

The segments and the current are plotted using 'plt.plot' and 'plt.quiver' respectively. The segments are represented by a their mid-point using red points, and the currents have been plotted as black vectors. The plot has been annotated at locations of maximum and minimum amplitude of current for better readability. The plot shall be saved as 'vector_current.png' in the directory 'EE2703_plots' upon execution of the code. The plot is presented in Fig 1.

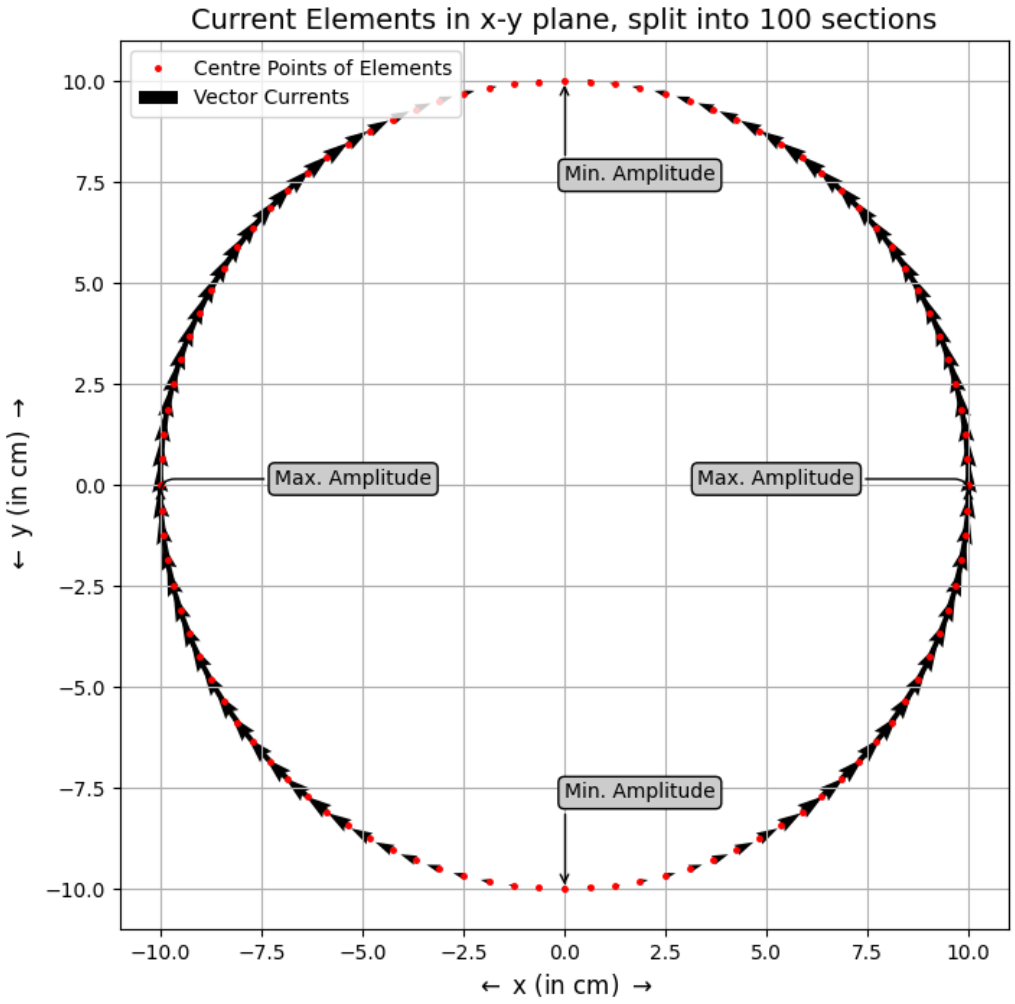

Figure 1: Segments and vector plot of currents

3.3 Location of mesh-grid

Clearly from the plots of current we can infer that if we consider the field along z-axis at origin, the field created due to left-half of the loop gets cancelled by the right-half section. So if we consider a mesh-grid symmetrical about origin, for say x ∈ [-1,1] and y ∈ [-1,1], at a later stage if we perform the curl operation, we shall get z-component of the magnetic field at locations on a line passing through origin along z-axis, which shall be very low due to finite precision of CPU and it is of no use for theoretical interest as decay rate obtained when we fit it is meaningless. So to add some theoretical value to the problem the mesh is taken off the axes, here in this case x ∈ [0,2] and y ∈ [0,2], and later when we compute the curl, the z-component of magnetic field is obtained at the center of the mesh which is a line parallel to z-axis passing through [1,1,0]. This field need not be zero and the decay rate obtained after fitting shall be of theoretical value. Please note that z-points vary from 1 cm to 1000 cm.

4 Computation of $\vec{A}$

4.1 Function calc(i)

We have obtained all the points required for computation and visualised the current pattern. Now the task is build the blocks for the computation of $\vec{A}$. The primary task is find the distance of all the points in the mesh from a particular segment indexed by l. One straight forward way to do this is by using for-loops. But numpy vectorisation, helps to find these distances from these points in the mesh grid parallely, unlike the for-loop which does the same in a sequential manner. Thus vectorisation saves time. This is implemented by function call given below. This function takes in a scalar index l, of a given segment. Then point-wise subtraction is done to remove x and y components of $\vec{r}'_l$ from every point in the mesh. Line 7 in the below code implements the same. Once we have this array in line 7, taking the norm shall give us the distance of each point in the mesh to a particular segment on the loop and the same is assigned to R, which is a (3,3,1000) array holding these distances. Now we have R_{ijk} in Eq. 4 for a given l, which is the same as $|\vec{r}_{ijk} - \vec{r}'_l|$. This array R is reshaped to (3,3,1000,1) to enable broadcasting with the vector dl, which is of the shape (1,2). Using dl as well, we have all the term required in basic term of Eq. 4 which when added up for different l indices shall give the vector potential . Thus for a given l, the same is computed by line 10, where point wise multiplication is done for operation on R-elements and the same is broadcasted with dl[l] vector computed previously. This returned value when summed up for various values of l shall give us the vector potential in space. Note that the vector returned is of the dimension (3,3,1000,2), where the last axes corresponds to x and y components of the same. Lines 18-23 in the pseudo code correspond to the same.

```
1  def calc(l):
2      """
3      Inputs - l : The index of the segment
4               (X,Y,Z,r1,dl taken from the global namespace)
5      Ouputs - A_l of the dimension (3,3,1000,2)
6      """
7      R1  = np.array((X-r1[l,0], Y-r1[l,1], Z))
8      R   = np.linalg.norm(R1, axis = 0)
9      R   = R.reshape((3,3,1000,1))
10     A_l = (np.cos(phi[l])*np.exp(-0.1j*R)/R)*dl[l]
11
12     return A_l
```

4.2 Accumulating over segments

Using the function calc(l) defined previously, We can use a for-loop to calculate the vector potential in space by accumulating the same for different indices of l, which mean different segments. Here, we can use for-loop for two reasons. The index l is assumed to be a scalar in the function calc(l) and and point-wise subtraction is done. To solve this calc(l) has to be modified to accept vector inputs for l. Once the modification is done, although we can get the vectorised norm for all values of l and the dimension of the returned array is of the form - (100,3,3,1000,2). We should sum over the first axis which has 100 elements, to get the A at all points in space. Here again np.sum or other equivalent implementations may involve matrix multiplication using np.dot to eventually get $\vec{A}$ in space of the dimension (3,3,1000). Each of the previous operations uses a for-loop internally by looking a C program, making it a bit

faster then the ordinary for-loop, but the time shall be of same order. Further the extra matrix operation in calc(l) adds a finite computation time which shall compensate the time saved by np.sum and at the end both the approaches end up almost at equal speeds, or at least at the same order, for say ms. So to avoid complexities in using calc(l), a single for-loop is used to compute $\vec{A}$, the input l to calc(l) is kept as a scalar. At the end we have $\vec{A}$ at locations in space. Note that both A_x and A_y are taken as single vector $\vec{A}$, and incremented at the same time in the loop which reduces the computation time when compared to the case of doing separately. The code which implements the same using for loops is given below and the same corresponds to lines 25-30 in the pseudo-code.

```
A = calc(0)
for i in range(1,n):
    A += calc(i)
```

5 Magnetic Field B_z

The magnetic field is given by the curl of vector potential by Eq. 3. The z component of magnetic field is given by the following equation.

$$B_z = \frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y} \tag{5}$$

The same when translated to a vectorial equation reads as follows :

$$B_z(z) = \frac{A_y(\Delta x + a, b, z) - A_y(-\Delta x + a, b, z)}{2\Delta x} - \frac{A_x(a, \Delta y + b, z) - A_x(a, -\Delta y + b, z)}{2\Delta y} \tag{6}$$

This is the field computed at the location (a,b,z). Vectorising equation 6 for all z, and computing the field at all locations from z : 1 cm to 1000 cm on a line parallel through z-axis passing through (1,1,0), translates to the line 1 of the python code given below. Note that the field center on the x-y plane is center of the mesh-gird which is (1,1,0). Further, usage of mesh-gird swaps the ij sub-indices of the vector A, and the same is reflected in the code. The field with z has been plotted both in log-log as well as linear plots. This corresponds to lines 32,33 in the pseudo-code. The plots have been presented in Fig 2,3.

```
Bz = 0.5*(A[1,2,:,1]-A[2,1,:,0]-A[1,0,:,1]+A[0,1,:,0])

print('\nData of magnetic field parallel to z-axis at (1,1,0)')
print('1.The mean value of field for z : 1 cm to 1000 cm is {:.3e} T.'.format(np
    .mean(abs(Bz))))
print('2.The max. value of field for z : 1 cm to 1000 cm is {:.3e} T.'.format(np
    .max(abs(Bz))))
print('3.The min. value of field for z : 1 cm to 1000 cm is {:.3e} T.'.format(np
    .min(abs(Bz))))

plt.figure(1, figsize = (9,6))
plt.plot(d3,np.abs(Bz),'-r.',label = 'Field Magnitude',markersize = 7.5)
plt.title('Magnetic field along z-axis at (1,1) - Linear',fontsize = 14)
plt.grid(True)
plt.legend(loc = 'upper right')
plt.xlabel(r'z (in cm) $\rightarrow$',fontsize = 12)
plt.ylabel(r'$B_z$ (in T) $\rightarrow$',fontsize = 12)
plt.savefig('EE2703_plots/field_linear.png')
```

```
16
17 plt.figure(2, figsize = (9,6))
18 plt.loglog(d3,np.abs(Bz),'-r.',label = 'Field Magnitude',markersize = 7.5)
19 plt.title('Magnetic field along z-axis at (1,1) - Loglog',fontsize = 14)
20 plt.grid(True)
21 plt.legend(loc = 'upper right')
22 plt.xlabel(r'z (in cm) (log) $\rightarrow$',fontsize = 12)
23 plt.ylabel(r'$B_z$ (in T) (log) $\rightarrow$',fontsize = 12)
24 plt.savefig('EE2703_plots/field_loglog.png')
```

The following lines are printed out in the terminal.

Data of magnetic field parallel to z-axis at (1,1,0)

1. The mean value of field for z : 1 cm to 1000 cm is 5.403e-04 T.
2. The max. value of field for z : 1 cm to 1000 cm is 6.939e-02 T.
3. The min. value of field for z : 1 cm to 1000 cm is 3.141e-06 T.

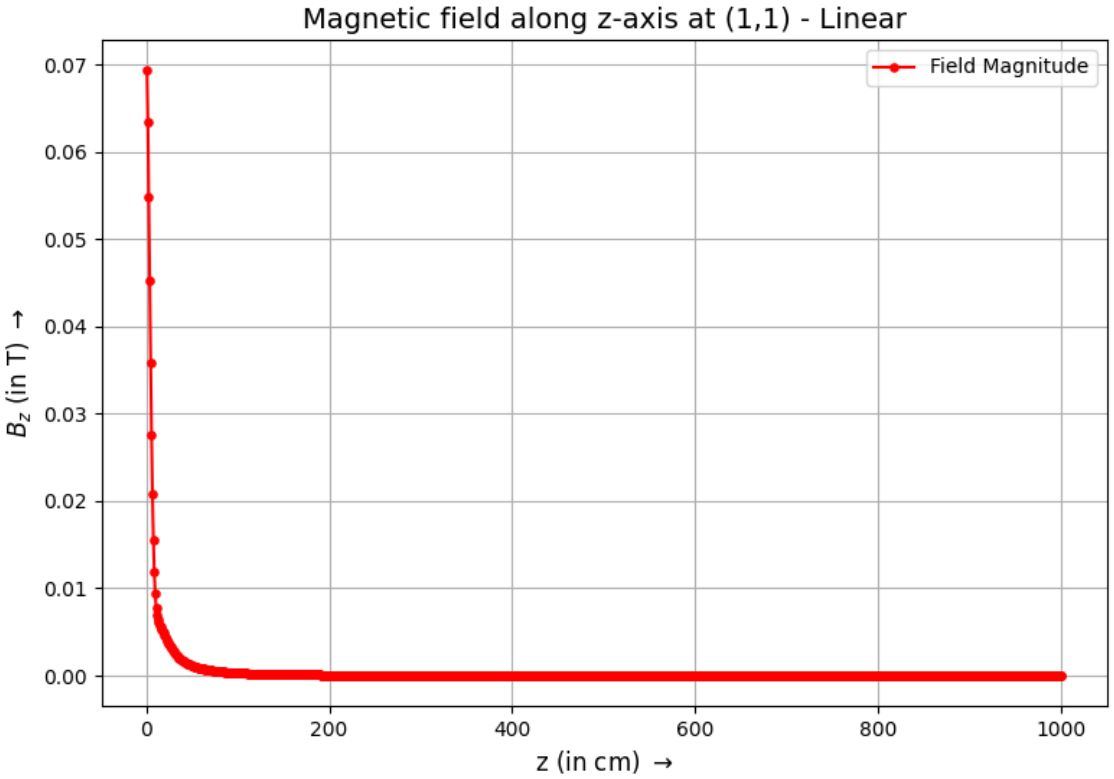

Figure 2: Plot of z-component of magnetic field (linear scale)

The log-log plot in Fig 3. is linear approximately from z : 30 cm, while its slightly non-linear in the initial parts. This indicates the dependence on several powers of z in the initial parts and single power of z in the later parts. This may be because the second can be considered as a far-zone field which is dominant by radiation while the first being near-zone field may have contribution of induction term as well, which leads to the non-linearity in the loglog curve.

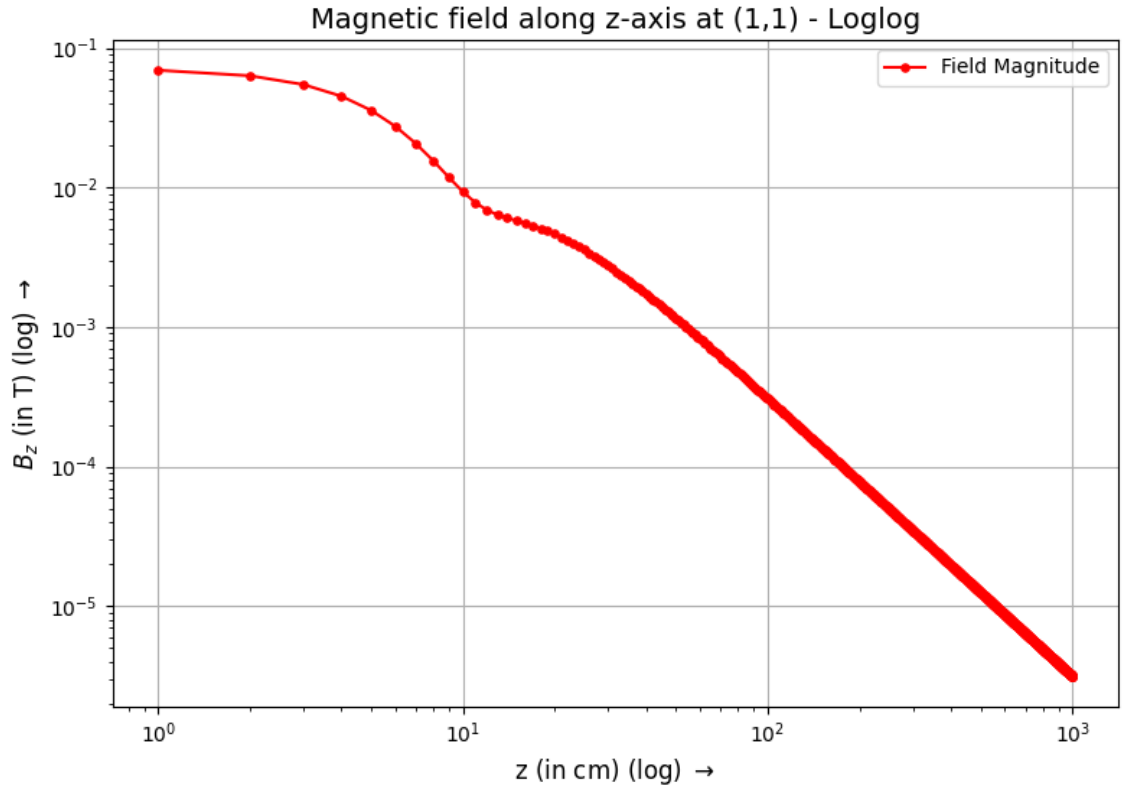

Figure 3: Plot of z-component of magnetic field (log-log scale)

6 Fit of magnetic field

Given the linear nature of the loglog plot from around 30 cm, a fit of the form Cz^b can be assumed. This equation can be expressed as linear matrix equation by taking log on both sides. Consider the following set of equations.

$$B_z = Cz^b \tag{7}$$

$$log(B_z) = log(C) + b * log(z) \tag{8}$$

$$\begin{pmatrix} 1 & log(z) \end{pmatrix} \quad \begin{pmatrix} log(C) \\ b \end{pmatrix} = \quad \begin{pmatrix} log(B_z) \end{pmatrix} \tag{9}$$

Eq 9. is a linear matrix equation and the same is solved using least squares approach, to get C and b. The following function implements the same and this corresponds to lines 35 to 39 in the pseudo-code.

```
1  def fit(B,z1):
2      """
3      Inputs - B : Field Vector
4               z1: z coordinates to obatin Cz^b
5      Ouputs - C,b : Coefficients of the fit obtained using least squares
6      """
7      x_in  = np.c_[np.log(z1), np.ones(len(z1))]
8      y_in  = np.log(B).T
9      b,C_l = np.linalg.lstsq(x_in,y_in,rcond=None)[0]
10     return np.exp(C_l),b
```

This function is used to fit the B data-points obtained. Two sets of points are considered to fit - all the points and points after 30 cm. The following lines of code are used to get and plot the fit. The plot has been presented in Fig 4. It can be seen that the fits deviate initially and almost coincide in the linear part of the curve.

```
C0,b0 = fit(np.abs(Bz),d3)
C1,b1 = fit(np.abs(Bz[30:]),d3[30:])

print('\nThe magnetic along line parallel to z-axis at (1,1,0) is fit to Cz^b')
print('The Coefficients with all points fit :')
print('1. C = {:.4f}'.format(C0))
print('2. b = {:.4f}'.format(b0))
print('The Coefficients with points after 30 cm fit :')
print('1. C = {:.4f}'.format(C1))
print('2. b = {:.4f}'.format(b1))

plt.figure(3, figsize = (9,6))
plt.loglog(d3,np.abs(Bz),'k.',label = 'Original Field Magnitude',markersize =
    7.5)
plt.loglog(d3,C1*d3**b1,label = 'Fit - Points from 30 cm', color = 'g',linestyle
    = '--')
plt.loglog(d3,C0*d3**b0,label = 'Fit - All points',color = 'r')
plt.title('Fit of Magnetic field along z-axis at (1,1) - Loglog',fontsize = 14)
plt.grid(True)
plt.legend(loc = 'upper right')
plt.xlabel(r'z (in cm) (log) $\rightarrow$',fontsize = 12)
plt.ylabel(r'$B_z$ (in T) (log) $\rightarrow$',fontsize = 12)
plt.savefig('EE2703_plots/field_fit.png')
```

The value of coefficients are given as follows :
The Coefficients with all points fit -

1. C = 1.4048
2. b = -1.8698

The Coefficients with points after 30 cm fit -

1. C = 2.7800
2. b = -1.9808

7 Theoretical findings

1. From the Fig 3. we can infer that magnetic field depends on multiple powers of z in the region from z : 1 cm to 30 cm, thus making it non-linear in log-log scale, while it varies linearly in the region after 30 cm, indicating the dependence of single power of z. This may be due to contribution of near-zone fields like the induction field for low values of z and only radiation field for higher values of z.

2. It can be seen from both the fits that B_z decays at a rate around (1.9-2). Considering only the case of far field ideally a radiation field falls as $1/r$ where r is the distance from center of the loop. For far-field r $\approx$ z at the location (1,1) on the mesh-grid. In addition to this as we are moving farther from origin on a straight line parallel to z-axis passing

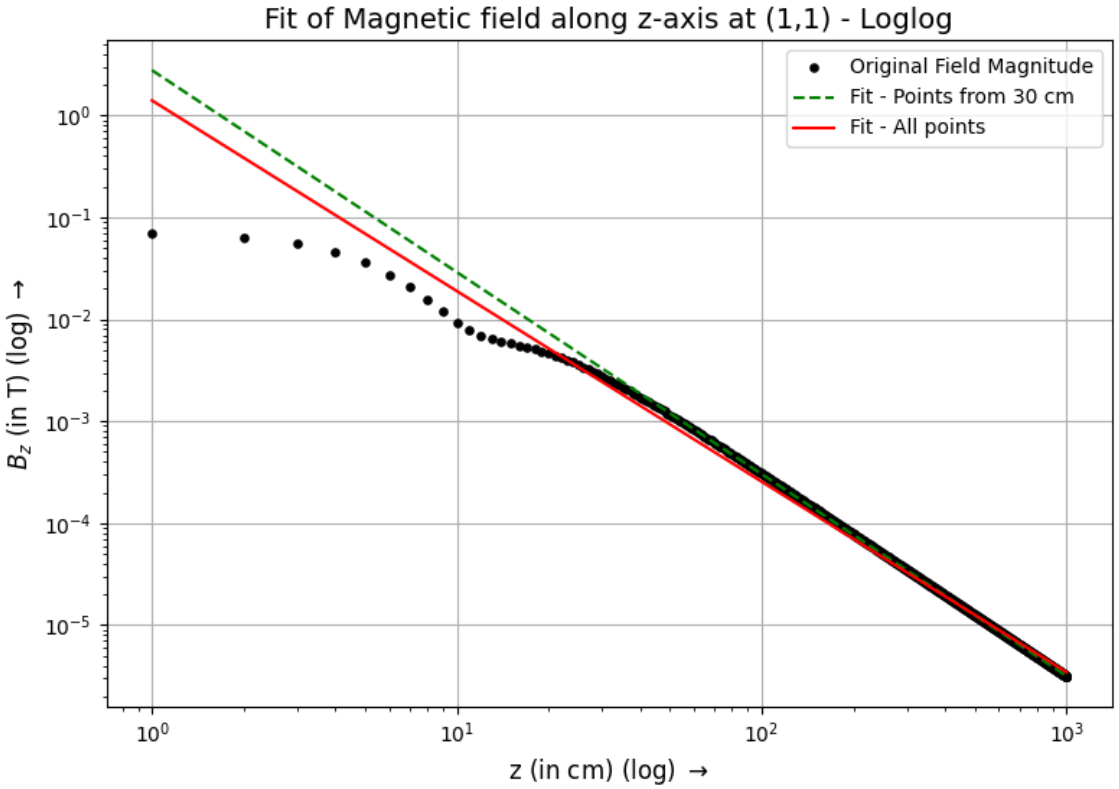

Figure 4: Fit of B_z to Cz^b

through (1,1,0) the position vector of the point is becoming closer and closer to the z-axis, indicating that the field tending to be closer and closer to zero (as both the left and right half components cancel out). So we can assume a dependence of $sin(\theta)$, where θ is the angle between position vector of point $\vec{r}$ and z-axis. After a certain set of points assuming far-field, $sin(\theta)$ can be approximated by θ, and z was already approximated by r. By taking in small-segment approximation for a circle arc, r*$\theta = \sqrt{2}$ (which is the distance between the line passing through (1,1,0) and z-axis). Taking the dependencies of both sin(θ) and $1/r$, it can be seen that the z-component of field falls as, $1/r^2$ and since r $\approx$ z, for far-field, we can see that the filed approximately decays at a rate around 2. Thus the B_z falls of as expected in the far-field region after z $>$ 30 cm.

3. For a static magnetic field, produced by constant current, the field can be found out using Biot-Savart law. For a constant current I, circulating through a loop of radius a, the magnetic field along the axis is given by :

$$\vec{B_z} = \frac{\mu_o I a^3}{2(z^2 + a^2)^{1.5}} \hat{z} \tag{10}$$

For far-field, $z >> a$, the following approximation can be made :

$$\vec{B_z} = \frac{\mu_o I a^3}{2z^3} \hat{z}$$

Assuming that the field to not to vary much at points off the axis, since these points are close to the axis, the field shall decay as $1/z^3$. Thus the decay rate would be 3 in case of a static-magnetic field.

4. The difference is due the fact that this field is caused by time-vary current, which is non-uniform and sinusoidal distributed. As a result of which the magnetic field along the z-axis is zero. Secondly this field is caused by radiation unlike the case of magneto-statics which was due to induction. Here in this case $1/r$ decay of radiation field and θ dependence for far-field are producing a decay rate of around 2. Whereas magneto-static field is a completely different from the former and has $1/z^3$ dependency. Put other way the former is being produced due to accelerated charges and the later due to moving charges.

8 Conclusion

The radiation pattern of a loop antenna has been analysed using Python programming. A non-uniform sinusoidal current through the loop antenna leads to a zero magnetic field on the z-axis. The magnetic field is being computed on a line parallel to z-axis passing through (1,1,0). Vectorised numpy operations have been used to compute the field vector $\vec{A}$ and they speed up the computation process. Further a for loop has been used for accumulating the vector field components due to each segment of the circular loop. The magnetic field is calculated as the curl of $\vec{A}$ and the same has been computed using a vectorised operation. The z-component of magnetic field is fit to Cz^b, and log-log plot was found to be linear at farther points and non-linear at nearer points indicating the contribution of near-field components. At far-field the magnetic field was found to decay as $1/z^2$ unlike the magneto-static case, and the same was reasoned through the phenomena of radiation. Thus although a fairly complicated problem was analysed through a few lines python code, indicating the usage of Python programming.