End Semester: Half Wave Dipole Antennas

Hariharan P EE20B042

May 12, 2022

Brief Up:

The tasks of this assignment are the following:

- Compute current through each section
- Check the obtained current with current obtained using direct formula
- Plot both of them against z and analyse the difference

Introduction:

- A long wire carries a current I(z) in a dipole antenna with half length of 50cm - so the antenna is a metre long, and has a wavelength of 2 metres. We want to determine the currents in the two wires of the antenna. The standard analysis assumes that the antenna current is given by

$$I = \{Imsin(k(l - z)); 0 \leq z \leq l\}$$
$$I = \{Imsin(k(l + z)); -l \leq z < 0\}$$

- The problem is to determine if this is a good assumption.
- We will compute the antenna current through this assignment in procedure and compare it with the above current and explain any discrepancies

Initialising necessary library and variables:

We declare the following variables to efficiently run the program.

Algorithm 1 Pseudo code to initialise values

```
from pylab import everything without any specific calling name
n=4
dz=0.5/n
a=0.01
k=pi
mu=(4*pi*(10**-7))/(4*pi)
muo=4*pi*(10**-7)
```

The python code to execute the above tasks is as following:

```
from pylab import *
n=4  #no. of sections
dz=0.5/n # length of section
a=0.01 #radius
k=pi # wavenumber
mu=(4*pi*(10**-7))/(4*pi) #mu0/4pi
muo=4*pi*(10**-7) #mu0
```

Section1:Initialising array to store section points:

- We shall divide the antenna into n sections each of length dz=0.5/n. Construct a vector z such that

$$z = i \times dz, -N \leq i \leq +N$$

$$u = i \times dz, -(N-1) \leq i \leq -1 \cup 1 \leq i \leq N-1$$

- Now initialise the z and u vectors as points along the antenna but u vector doesn't have the terms corresponding to known currents, ie., at i=0,N,2N

- Initialise two vectors I and J , one to store currents calculated using the direct formula and another to store unknown current respectively.

- We use arange to initialise the vectorised z and u and I and z have sizes

$$2 \times N + 1$$

and J and u have sizes

$$2 \times N - 2$$

Algorithm 2 Initialising I,J,z,u

I=zeros vector of length (2*n+1)
J=zeros vector of length (2*n-2)
z=numpy vector from -0.5 to 0.5 with dz as step
u=numpy vector from -0.5+dz to -dz union dz to 0.5-dz with dz as step

The python code to execute the above tasks is as following:

```
I=zeros(2*n+1) #current vector using equation given
J=zeros(2*n-2) #current vector to be computed
z=array(arange(-0.5,0.5+dz,dz)) #vector z
u=array(append(arange(-0.5+dz,0,dz),arange(dz,0.5,dz)))
```

Lets see what the z and u matrices are initialised with (for N=4)
z:
[-0.5 -0.38 -0.25 -0.12 0. 0.12 0.25 0.38 0.5]
u:
[-0.38 -0.25 -0.12 0.12 0.25 0.38]

Section2: M matrix function:

We need to obtain M matrix described below. Thus we define a function to return M matrix when called

$$\begin{pmatrix} H_\phi[z_1] \\ \dots \\ H_\phi[z_{N-1}] \\ H_\phi[z_{N+1}] \\ \dots \\ H_\phi[z_{2N-1}] \end{pmatrix} = \frac{1}{2\pi a} \begin{pmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} J_1 \\ \dots \\ J_{N-1} \\ J_{N+1} \\ \dots \\ J_{2N-1} \end{pmatrix} = M * J$$

Algorithm 3 function to return m

defining a function called getM():
creating an identity matrix of size (2n-2,2n-2)
dividing this matrix by 2*pi*a
returning the matrix m

The python code to execute the above tasks is as following:

```
def getM():
  m=identity(2*n-2)
  m=m/(2*pi*a) #m matrix is 1/(2*pi*a) times indentity matrix
  return m
```

Lets see what the M matrix looks like for N=4:

m:
[[15.92 0. 0. 0. 0. 0.]
[0. 15.92 0. 0. 0. 0.]
[0. 0. 15.92 0. 0. 0.]
[0. 0. 0. 15.92 0. 0.]
[0. 0. 0. 0. 15.92 0.]
[0. 0. 0. 0. 0. 15.92]]

Section3: Computation of P and P_B matrices:

- We shall express vector potential as

$$A_{z,j} = \sum_j \frac{\mu 0}{4\pi} \frac{exp(-jkR_{ij}) \times dz_j}{R_{ij}}$$

- This sum is reduced to summation of P matrix with J vector . With this,we also include the portion from already known current, which would be Pb matrix times In.

$$A_{z,j} = \sum_j P_{ij} \times I_j + P_B \times I_N$$

- P is a matrix with 2N - 2 columns and 2N 2 rows. Note that the vector potential is driven by all the currents, which is why we use the I vector.

- PB is the contribution to the vector potential due to current IN, and is given by

$$P_B = \frac{\mu 0}{4\pi} \frac{exp(-jkR_{iN}) \times dz_j}{R_{iN}}$$

- We now require Rij and RiN matrices to compute P and Pb.

- R is the distance between observer and source.R is given by

$$R_{ij} = \sqrt{a^2 + (z_i - z_j)^2}$$

- By calculating this expression for i and j from 0 to 2N+1 gives Rz and i and j for indices corresponding to u gives Ru.

- RiN is obtained by slicing the middle column from Rz and deleting the indices 0,N,2N
- To compute the exp(-jkRu) terms, we define a function called epower and we vectorize it. Thus we can use the vectorization property to compute P and Pb which will be very quick than nested for loops.
- We use meshgrid to use vectorized Zi and Zj and Ui and Uj to compute Rz and Ru

Algorithm 4 computing P and Pb matrices

meshgrid for z,z
meshgrid for u,u
Rz as numpy array of (a**2+(zi-zj)**2)**0.5
Ru as numpy array of (a**2+(ui-uj)**2)**0.5
initialisng P as complex matrix of size 2n-2,2n-2
initialisng Pb as complex matrix of size 2n-2,2n-2
RiN=slicing the middle column from Rz and deleting the first,last,middle elements
defining a function called epower(x):
return (complex(cos(x),sin(x)))
vectorizing the function epower
epower1 =epower(-1*Ru*k)
epower2 =epower(-1*k*RiN)
epower3 =epower(pi/2)
P is assigned values as (mu*dz*epower1)/(Ru)
Pb is assigned values as (mu*dz*epower2)/(RiN)

The python code to execute the above tasks is as following:

```
(zj,zi)=meshgrid(z,z)
(uj,ui)=meshgrid(u,u)
Rz=array((a**2+(zi-zj)**2)**0.5)
Ru=array((a**2+(ui-uj)**2)**0.5)

P=zeros((2*n-2,2*n-2),dtype=complex_)
Pb=zeros(2*n-2,dtype=complex_)

RiN=delete(Rz[:,n],[0,n,2*n])
def epower(x):
  return (complex(cos(x),sin(x)))

epower=vectorize(epower)
epower1 =epower(-1*Ru*k)
epower2 =epower(-1*k*RiN)
epower3 =epower(pi/2)
P=(mu*dz*epower1)/(Ru)
```

Lets look at P and P_B matrices (multiplied by 10^8 for N=4:
$P \times 10^8$:
[[124.94-3.93j , 9.2 -3.83j , 3.53-3.53j , -0. -2.5j , -0.77-1.85j , -1.18-1.18j]
[9.2 -3.83j ,124.94-3.93j , 9.2 -3.83j , 1.27-3.08j , -0. -2.5j, -0.77-1.85j]
[3.53-3.53j , 9.2 -3.83j , 124.94-3.93j , 3.53-3.53j , 1.27-3.08j, -0. -2.5j]
[-0. -2.5j , 1.27-3.08j , 3.53-3.53j , 124.94-3.93j , 9.2 -3.83j , 3.53-3.53j]
[-0.77-1.85j , -0. -2.5j , 1.27-3.08j , 9.2 -3.83j , 124.94-3.93j, 9.2 -3.83j]
[-1.18-1.18j , -0.77-1.85j , -0. -2.5j , 3.53-3.53j , 9.2 -3.83j, 124.94-3.93j]]

$P_b \times 10^8$:
[1.27-3.08j 3.53-3.53j 9.2 -3.83j 9.2 -3.83j 3.53-3.53j 1.27-3.08j 1.27-3.08j]

Section4: Computing Q and Q_B matrices from magnetic field intensity :

- We shall compute Q and Q_B matrices from the expression of magnetic field intensity expression along ϕ direction component.

$$H_\phi(r, Z_i) = \sum P_{ij} \frac{a}{\mu 0}(\frac{jk}{R_{ij}} + \frac{1}{R_{ij}^2})I_j + P_B \frac{a}{\mu 0}(\frac{jk}{R_{iN}} + \frac{1}{R_{iN}^2})I_m$$

$$H_\phi(r, Z_i) = \sum Q_{ij} I_j + Q_{Bi} I_m$$

- Now comparing the both expressions, we shall get the expression for Q and Q_B

$$Q = P_{ij} \frac{a}{\mu 0}(\frac{jk}{R_{ij}} + \frac{1}{R_{ij}^2})$$

$$Q_B = P_B \frac{a}{\mu 0}(\frac{jk}{R_{iN}} + \frac{1}{R_{iN}^2})$$

- The Q_{ij} in the equation is over all the currents. However this needs to be split into the unknown currents, J_j and the boundary currents. Only one of the boundary currents is non-zero, namely the feed current at i = N. The matrix corresponding to $J_j 4$ we call Qi_j, and the boundary current we call $Q_B = Q_{iN}$

Algorithm 5 Computing Q and Qb matrices:

Q=commplex matrix with values zeros of size (2*n-2,2*n-2)
Qb=commplex vector with values zeros of size (2*n-2,2*n-2)
Q is initialised with values as P*(a/muo)*(((k*epower3)/Ru)+(1/(Ru**2)))
Qb is initialised with values as Pb*(a/muo)*(((k*epower3)/RiN)+(1/(RiN**2)))

The python code to execute the above tasks is as following:

```
Q=zeros((2*n-2,2*n-2),dtype=complex_) #initialising Q and Qb
Qb=zeros(2*n-2,dtype=complex_)
Q=P*(a/muo)*(((k*epower3)/Ru)+(1/(Ru**2))) #preparing Q
Qb=Pb*(a/muo)*(((k*epower3)/RiN)+(1/(RiN**2))) #preparing Qb
```

Lets see what is initialised inside Q and Q_B matrices:
$Q \times 10^4$:
[[9.95e+05-10.28j 5.42e+02-10.12j 8.02e+01 -9.66j 1.24e+01 -7.96j 5.83e+00 -6.82j 2.26e+00 -5.59j]

[5.420e+02-10.12j 9.95e+05-10.28j 5.42e+02-10.12j 2.77e+01 -8.92j 1.24e+01 -7.96j 5.83e+00 -6.82j]

[8.02e+01 -9.66j 5.42e+02-10.12j 9.95e+05-10.28j 8.02e+01 -9.66j 2.77e+01 -8.92j 1.24e+01 -7.96j]

[1.24e+01 -7.96j 2.77e+01 -8.92j 8.02e+01 -9.66j 9.95e+05-10.28j 5.42e+02-10.12j 8.02e+01 -9.66j]

[5.83e+00 -6.82j 1.24e+01 -7.96j 2.77e+01 -8.92j 5.42e+02-10.12j 9.95e+05-10.28j 5.42e+02-10.12j]

[2.26e+00 -5.59j 5.83e+00 -6.82j 1.24e+01 -7.96j 8.02e+01 -9.66j 5.42e+02-10.12j 9.95e+05-10.28j]]

$Q_B \times 10^4$:
[27.72 -8.92j 80.2 -9.66j 542.08-10.12j 542.08-10.12j 80.2 -9.66j 27.72 -8.92j]

Section5: Obtaining J vector from m and Q matices:

- The matrices M,Q,Q_B and J are connected by the equation

$$MJ = QJ + Q_B I_m$$

- Thus J vector can be obtained by matrix multiplication of inv(M-Q) and Q_B I_m

- This current vector can be compared to the standard expression given at the top of the assignment as I vector.
- We shall add the Boundary currents (zero at i=0, i=2N, and I_m at i=N).
- We shall also initialise I vector using the expression in the first page of the assignment.

Algorithm 6 Computing J from Q,Qb,M:

calling the function getM()
J is obtained by dot multiplication of inv(m-Q),(Qb)
inserting values 0 at first and the last places and 1 at the middle index
computing I matrix by first computing it's z vector

The python code to execute the above tasks is as following:

```
m=getM()
J=dot(linalg.inv(m-Q),(Qb*1.0))
J=insert(J,0,0)
J=append(J,0)
J=insert(J,n,1)
z1=z[:n]
z2=z[n:2*n+1]
I1=array(sin(k*(0.5+z1)))
I2=array(sin(k*(0.5-z2)))
I=concatenate((I1,I2))
```

Lets see what are the values of I and J vectors
I:
[0. 0.38 0.71 0.92 1. 0.92 0.71 0.38 0.]
abs($J \times 10^3$):
[0.0 0.03 0.1 0.65 1000 0.65 0.1 0.03 0.0]
We notice that the value of J vector is very small comparing to I vector other than at the middle index ie. value=1.
This large difference is reduced when we increase the value of N.

Analysing current vectors by plotting:

- We shall now plot these two current vectors vs z vector for N=4 and N=100
- We notice a large deviation in N=4 plot and quite similar curve for N=100

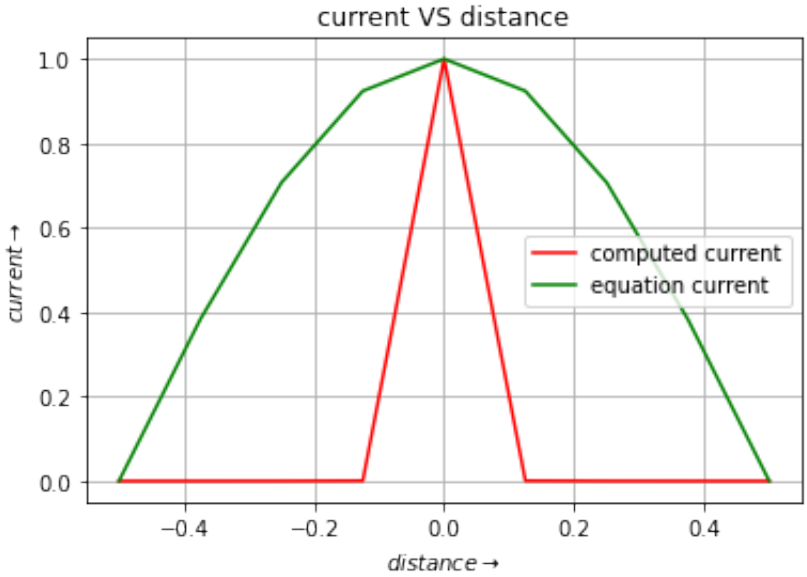

Figure 1: Current VS distance for N=4

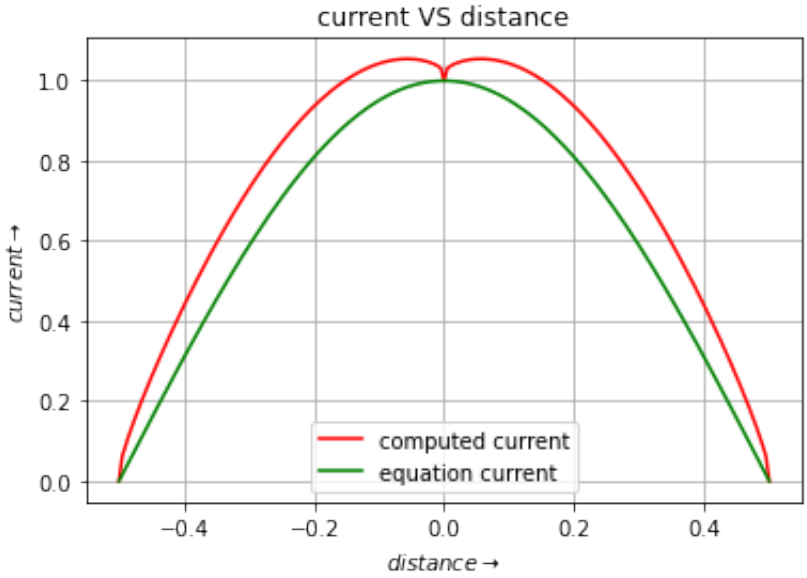

Figure 2: Current VS distance for N=100

- The reason for this discrepancy is because we aren't considering the section to be of infinitesimally small. For N=4, we get large errors as expected and for for large value like N=100, we see the difference since we do summation instead of integration which would otherwise give

more similar plot. The sudden matching value at distance 0 is because we explicitly give it's value to be 0. Otherwise we would have seen a smooth curve.

Conclusion:

Thus we have analysed and observed how the current of half wave dipole antennas exist and how do they vary from the standard expression current.We have also inferred about the matrices we used and how vectorization is quick and easy than for loops.