

Supplementary Materials

Program code in Python written and debugged in PyCharm IDE for solving the integro-differential equation (7) by the iteration method.

```
1. #
2. # The program calculates the current distribution and the dipole moment in a wire
3. # inclusion (l - length, a - radius)
4. # with the impedance boundary conditions on its surface and surrounded with a
5. # medium with epsilon and mu.
6. #
7. # Yujie Zhao @ March 2021
8. #
9. # University of St. Andrews, Fife, Scotland
10. #
11.
12. import pandas as pd
13. import numpy as np
14. from scipy import integrate
15. from scipy.interpolate import CubicSpline
16.
17. # Wire and material parameters
18. epsilon = complex(4.0, 0.0) # effective dielectric constant of the matrix, (e1,e2)
19. mu = complex(1.0, 0.0) # effective magnetic constant of the matrix, (mu1,mu2)
20. l = 2.5 # wire length in cm
21. a = 9.3e-4 # wire radius in cm
22.
23. # Some constants
24. c = 2.9979250e+10 # speed of light in cm/s (cgs system)
25. i = complex(0.0, 1.0) # imaginary unit
26. unit = complex(1.0, 0.0) # real unit in the complex form
27. e0 = 1.0 / c # electric field amplitude
28.
29. # Calculation parameters
30. NCUR = 50 # number of points in the current distribution along the wire
31. accuracy1 = 1.0e-3 # relative error when integrating in the first approximation
32. # for the current
33. accuracy2 = 1.0e-3 # relative error when integrating in the dipole moments
34. accuracy3 = 1.0e-3 # relative error when integrating in the coefficients in the
35. # main loop by frequency
36. filename = 'Field5.50e_ZeroStrain.csv' # csv file for the experimental impedance
37. # dispersion (freq, real, imag)
38.
39. df = pd.read_csv(filename, sep = ',', header = None) # reading the impedance file
40. freq = df[0] # frequency array
41. NFREQ = len(df[0]) # number of frequency points
42. # Recalculating the experimental impedance (Ohms) into the surface impedance (cgs)
43. Impedance = (df[1] + df[2] * 1j) * 1.0e+9 * (a / l) / (2.0 * c)
44.
45. x1 = -l / 2.0 # wire left end
46. x2 = l / 2.0 # wire right end
47. XDATA = [] # points on the wire used for calculating the current distribution
48. for n in range(NCUR):
49.     XDATA.append(x1 + l * n / (NCUR - 1))
50.
51. # General Green's functions G(r) and Gf(r)
52. def G(r):
53.     return np.exp(-i * k * r) / (4.0 * np.pi * r)
54.
55. def Gf(r):
56.     return (a ** 2) * (unit + i * k * r) * np.exp(-i * k * r) / (2.0 * (r ** 3))
57.
58. # Function for calculating Q and Qf factors
59. def FQ(x):
60.     r = (x ** 2 + a ** 2)**0.5
61.     return (G(r)).real
62.
63. def FQf(x):
64.     r = (x ** 2 + a ** 2)**0.5
```

```

65.     return (Gf(r)).real
66.
67. # Functions for coef1 and coef2
68. def S1(x, p):
69.     r = ((x - p) ** 2 + a ** 2)**0.5
70.     return -i * (G(r)).imag / Q
71.
72. def S2(x, s, p):
73.     r = ((s - p) ** 2 + a ** 2)**0.5
74.     return (i * (kn ** 2 - k ** 2) / (Q * kn)) * np.sin(kn * (x - s)) * (G(r)).imag
75.
76. def S3(x, s, p):
77.     r = ((s - p) ** 2 + a ** 2)**0.5
78.     return (-f * epsilon * Z / (a * c * Q * kn)) * np.sin(kn * (x - s)) * (Gf(r)).imag
79.
80. def S23(x, s, p):
81.     return S2(x, s, p) + S3(x, s, p)
82.
83. def Re_F1a11(p, s):
84.     return (S23(1 / 2.0, s, p) * np.sin(kn * p)).real
85.
86. def Im_F1a11(p, s):
87.     return (S23(1 / 2.0, s, p) * np.sin(kn * p)).imag
88.
89. def Re_F2a11(p):
90.     return (S1(1 / 2.0, p) * np.sin(kn * p)).real
91.
92. def Im_F2a11(p):
93.     return (S1(1 / 2.0, p) * np.sin(kn * p)).imag
94.
95. def Re_F1a12(p, s):
96.     return (S23(1 / 2.0, s, p) * np.cos(kn * p)).real
97.
98. def Im_F1a12(p, s):
99.     return (S23(1 / 2.0, s, p) * np.cos(kn * p)).imag
100.
101. def Re_F2a12(p):
102.     return (S1(1 / 2.0, p) * np.cos(kn * p)).real
103.
104. def Im_F2a12(p):
105.     return (S1(1 / 2.0, p) * np.cos(kn * p)).imag
106.
107. def Re_Fa21(p):
108.     return (S1(-1 / 2.0, p) * np.sin(kn * p)).real
109.
110. def Im_Fa21(p):
111.     return (S1(-1 / 2.0, p) * np.sin(kn * p)).imag
112.
113. def Re_Fa22(p):
114.     return (S1(-1 / 2.0, p) * np.cos(kn * p)).real
115.
116. def Im_Fa22(p):
117.     return (S1(-1 / 2.0, p) * np.cos(kn * p)).imag
118.
119. def Re_F1B1(p, s):
120.     return (S23(1 / 2.0, s, p)).real
121.
122. def Im_F1B1(p, s):
123.     return (S23(1 / 2.0, s, p)).imag
124.
125. def Re_F2B1(p):
126.     return (S1(1 / 2.0, p)).real
127.
128. def Im_F2B1(p):
129.     return (S1(1 / 2.0, p)).imag
130.
131. def Re_FB2(p):
132.     return (S1(-1 / 2.0, p)).real
133.
134. def Im_FB2(p):
135.     return (S1(-1 / 2.0, p)).imag
136.
137. # Low, G(x), and upper, H(x), integration y-curves in a 2D integral (horizontal

```

```

138. # lines in our case)
139. def G1(x):
140.     return -1 / 2.0 + x - x
141.
142. def H1(x):
143.     return 1 / 2.0 + x - x
144.
145. # Current distribution - zero approximation
146. def j0(x):
147.     current = (i * f * epsilon * e0 / (2.0 * Q * (kn**2))) * (np.cos(kn * x) - np.cos(kn * l / 2.0)) /
np.cos(kn * l / 2.0)
148.     return current
149.
150. def Re_j0(x):
151.     return (j0(x)).real
152.
153. def Im_j0(x):
154.     return (j0(x)).imag
155.
156. # Current distribution - first iteration
157. def j1(x):
158.     x0 = x
159.     def Re_FA1j(p, s):
160.         return (S23(x0, s, p) * np.sin(kn * p)).real
161.
162.     def Im_FA1j(p, s):
163.         return (S23(x0, s, p) * np.sin(kn * p)).imag
164.
165.     def Re_FA2j(p):
166.         return (S1(x0, p) * np.sin(kn * p)).real
167.
168.     def Im_FA2j(p):
169.         return (S1(x0, p) * np.sin(kn * p)).imag
170.
171.     def Re_FB1j(p, s):
172.         return (S23(x0, s, p) * np.cos(kn * p)).real
173.
174.     def Im_FB1j(p, s):
175.         return (S23(x0, s, p) * np.cos(kn * p)).imag
176.
177.     def Re_FB2j(p):
178.         return (S1(x0, p) * np.cos(kn * p)).real
179.
180.     def Im_FB2j(p):
181.         return (S1(x0, p) * np.cos(kn * p)).imag
182.
183.     def Re_FC1j(p, s):
184.         return (S23(x0, s, p)).real
185.
186.     def Im_FC1j(p, s):
187.         return (S23(x0, s, p)).imag
188.
189.     def Re_FC2j(p):
190.         return (S1(x0, p)).real
191.
192.     def Im_FC2j(p):
193.         return (S1(x0, p)).imag
194.
195.     def G2(x):
196.         return -1 / 2.0 + x - x
197.
198.     def H2(x):
199.         return x0 + x - x
200.
201.     coef3 = -i * f * epsilon * e0 / (2.0 * Q * (kn ** 2))
202.
203.     Integral1 = list(integrate.dblquad(Re_FA1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
204.     Result1 = Integral1[0]
205.
206.     Integral2 = list(integrate.dblquad(Im_FA1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
207.     Result2 = Integral2[0]

```

```

208.
209.     Integral3 = list(integrate.quad(Re_FA2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
210.     Result3 = Integral[0]
211.
212.     Integral4 = list(integrate.quad(Im_FA2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
213.     Result4 = Integral[0]
214.
215.     current = coef1 * (np.sin(kn * x) + (Result1 + Result3) * unit + (Result2 + Result4) * i)
216.
217.     Integral1 = list(integrate.dblquad(Re_FB1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
218.     Result1 = Integral1[0]
219.
220.     Integral2 = list(integrate.dblquad(Im_FB1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
221.     Result2 = Integral2[0]
222.
223.     Integral3 = list(integrate.quad(Re_FB2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
224.     Result3 = Integral3[0]
225.
226.     Integral4 = list(integrate.quad(Im_FB2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
227.     Result4 = Integral4[0]
228.
229.     current = current + coef2 * (np.cos(kn * x) + (Result1 + Result3) * unit + (Result2 + Result4) *
i)
230.
231.     Integral1 = list(integrate.dblquad(Re_FC1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
232.     Result1 = Integral1[0]
233.
234.     Integral2 = list(integrate.dblquad(Im_FC1j, x1, x2, G2, H2, epsabs = accuracy1, epsrel =
accuracy1))
235.     Result2 = Integral2[0]
236.
237.     Integral3 = list(integrate.quad(Re_FC2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
238.     Result3 = Integral3[0]
239.
240.     Integral4 = list(integrate.quad(Im_FC2j, x1, x2, epsabs = accuracy1, epsrel = accuracy1))
241.     Result4 = Integral4[0]
242.
243.
244.
245.     current = current + coef3 * (unit + (Result1 + Result3) * unit + (Result2 + Result4) * i)
246.     return current
247.
248. # Sampling j1(x), its real and imaginary parts
249. def Re_j1():
250.     FDATA = []
251.     for n in range(NCUR):
252.         FDATA.append((j1(XDATA[n])).real)
253.     return FDATA
254.
255. def Im_j1():
256.     FDATA = []
257.     for n in range(NCUR):
258.         FDATA.append((j1(XDATA[n])).imag)
259.     return FDATA
260.
261. def moment():
262.     Integral = list(integrate.quad(Re_j0, x1, x2, epsabs = accuracy2, epsrel = accuracy2))
263.     Result = Integral[0]
264.     dip0 = unit * Result
265.     Integral = list(integrate.quad(Im_j0, x1, x2, epsabs = accuracy2, epsrel = accuracy2))
266.     Result = Integral[0]
267.     dip0 = - (dip0 + i * Result) * i / (2.0 * np.pi * f)
268.
269.     FDATA = Re_j1()
270.     cs = CubicSpline(XDATA, FDATA) # cubic spline interpolation of Re_j1(x)
271. # calculated in discrete points
272.     cubfun = lambda x: cs(x)
273.     Integral = list(integrate.quad(cubfun, x1, x2, epsabs = accuracy2, epsrel = accuracy2))
274.     Result = Integral[0]
275.     dip1 = unit * Result

```

```

276.
277.     FDATA = Im_j1()
278.     cs = CubicSpline(XDATA, FDATA)
279.     cubfun = lambda x: cs(x)
280.     Integral = list(integrate.quad(cubfun, x1, x2, epsabs = accuracy2, epsrel = accuracy2))
281.     Result = Integral[0]
282.     dip1 = - (dip1 + i * Result) * i / (2.0 * np.pi * f)
283.
284.     return [dip0, dip1]
285.
286.     dip0_real = [] # real part of the dipole moment - zero approximation
287.     dip0_imag = [] # imaginary part of the dipole moment - zero approximation
288.     dip1_real = [] # real part of the dipole moment - first approximation
289.     dip1_imag = [] # imaginary part of the dipole moment - first approximation
290.
291.     for n in range(NFREQ):
292.         f = freq[n] # frequency array
293.         Z = Impedance[n] # surface impedance array calculated from the experimental
294.         # impedance (Ohms)
295.         k = 2.0 * np.pi * f * (epsilon * mu)**0.5 / c # wavenumber in a medium with
296.         # epsilon and mu
297.
298.         Integral = list(integrate.quad(FQ, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
299.         Q = Integral[0]
300.         Integral = list(integrate.quad(FQf, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
301.         Qf = Integral[0]
302.
303.         kn = k * (unit - i * c * Z * Qf / (4.0 * (np.pi**2) * a * f * mu * Q))**0.5
304.         # normalised wavenumber
305.
306.         Integral = list(integrate.dblquad(Re_F1a11, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
307.         Result = Integral[0]
308.         a11 = unit * Result
309.         Integral = list(integrate.dblquad(Im_F1a11, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
310.         Result = Integral[0]
311.         a11 = a11 + i * Result
312.         Integral = list(integrate.quad(Re_F2a11, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
313.         Result = Integral[0]
314.         a11 = a11 + unit * Result
315.         Integral = list(integrate.quad(Im_F2a11, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
316.         Result = Integral[0]
317.         a11 = a11 + i * Result
318.
319.         Integral = list(integrate.dblquad(Re_F1a12, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
320.         Result = Integral[0]
321.         a12 = unit * Result
322.         Integral = list(integrate.dblquad(Im_F1a12, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
323.         Result = Integral[0]
324.         a12 = a12 + i * Result
325.         Integral = list(integrate.quad(Re_F2a12, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
326.         Result = Integral[0]
327.         a12 = a12 + unit * Result
328.         Integral = list(integrate.quad(Im_F2a12, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
329.         Result = Integral[0]
330.         a12 = a12 + i * Result
331.
332.         Integral = list(integrate.quad(Re_Fa21, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
333.         Result = Integral[0]
334.         a21 = unit * Result
335.         Integral = list(integrate.quad(Im_Fa21, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
336.         Result = Integral[0]
337.         a21 = a21 + i * Result
338.
339.         Integral = list(integrate.quad(Re_Fa22, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
340.         Result = Integral[0]
341.         a22 = unit * Result
342.         Integral = list(integrate.quad(Im_Fa22, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
343.         Result = Integral[0]
344.         a22 = a22+i * Result

```

```

345.
346.     Integral = list(integrate.dblquad(Re_F1B1, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
347.     Result = Integral[0]
348.     B1 = unit * Result
349.     Integral = list(integrate.dblquad(Im_F1B1, x1, x2, G1, H1, epsabs = accuracy3, epsrel =
accuracy3))
350.     Result = Integral[0]
351.     B1 = B1 + i * Result
352.     Integral = list(integrate.quad(Re_F2B1, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
353.     Result = Integral[0]
354.     B1 = B1 + unit * Result
355.     Integral = list(integrate.quad(Im_F2B1, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
356.     Result = Integral[0]
357.     B1 = B1 + i * Result
358.     B1 = (unit + B1) * i * f * epsilon * e0 / (2.0 * Q * (kn**2))
359.
360.     Integral = list(integrate.quad(Re_FB2, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
361.     Result = Integral[0]
362.     B2 = unit * Result
363.     Integral = list(integrate.quad(Im_FB2, x1, x2, epsabs = accuracy3, epsrel = accuracy3))
364.     Result = Integral[0]
365.     B2 = B2 + i * Result
366.     B2 = (unit + B2) * i * f * epsilon * e0 / (2.0 * Q * (kn**2))
367.
368.     coef2 = (B1 + B2) / 2.0
369.     coef2 = coef2 + (B2 - B1) * (a11 + a21) / (4.0 * np.sin(kn * l / 2.0) + 2.0 * (a11 - a21))
370.     coef2 = coef2 / (np.cos(kn * l / 2.0) + (a22 - a12) * (a11 + a21)
371.                      / (4.0 * np.sin(kn * l / 2.0) + 2.0 * (a11 - a21)) + (a12 + a22) / 2.0)
372.     coef1 = ((B1 - B2) + coef2 * (a22 - a12)) / (2.0 * np.sin(kn * l / 2.0) + (a11 - a21))
373.
374.     dip01 = moment() # [dip0, dip1] - array of the zero and first approximations
375.     # for the dipole moment
376.     dip0_real.append(c * (dip01[0]).real)
377.     dip0_imag.append(-c * (dip01[0]).imag)
378.     dip1_real.append(c * (dip01[1]).real)
379.     dip1_imag.append(-c * (dip01[1]).imag)
380.
381.     print('frequency point = ', n + 1)
382.
383.     disper0 = np.column_stack((freq, dip0_real, dip0_imag)) # dispersion of the dipole
384.     # moment - zero approximation
385.     disper1 = np.column_stack((freq, dip1_real, dip1_imag)) # dispersion of the dipole
386.     # moment - first approximation
387.
388.     np.savetxt('disper0.csv', disper0, delimiter=',')
389.     np.savetxt('disper1.csv', disper1, delimiter=',')
390.

```