

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import random
4 import mcint
5
6
7 def diff_r_rprim(r, r_p, theta, theta_p, phi, phi_p):
8     '''
9     Returns the difference between two arbitrary vectors r and r',
10    given their position.
11    '''
12    term1 = r**2 + r_p**2
13    term2 = 2*r*r_p*(np.sin(theta)*np.sin(theta_p)*np.cos(theta-
14    phi)+np.cos(theta)*np.cos(theta_p))
15    diff_sqrt = np.sqrt(term1-term2)
16    return diff_sqrt
17
18 def k_dot_r(k,r,theta):
19     '''
20     Returns the value of the dot product between k and r,
21     with the coordinates aligned with the z || z' axes.
22     '''
23     return k*r*np.cos(theta)
24
25
26 """ def kp_dot_rp(alpha, beta, kp, rp, thetap, phip):
27     '''
28     alpha and beta are the azimuthal and the polar angles for
29     k' respectively. The integral should be independent of
30     beta, due to spherical symmetry.
31     '''
32     return kp*rp*(np.sin(alpha)*np.cos(beta-hip)+np.cos(alpha)*np.cos(thetap))
33
34
35 def integrand(x, alpha, beta, k, kp):
36     # Dimensions to integrate over
37     r = x[0]
38     phi = x[1]
39     theta = x[2]
40     rp = x[3]
41     phip = x[4]
42     thetap = x[5]
43
44     factor1 = np.exp(-1j*kp_dot_rp(alpha, beta, kp, rp, thetap, phip))
45     factor2 = np.exp(1j*k_dot_r(k, r, theta))
46     diffrrp = diff_r_rprim(r, rp, theta, thetap, phi, phip)
47     factor3 = np.exp(1j*k*diffrrp)/diffrrp
48     return factor1*factor2*factor3 """
49
50 # Start from scratch
51 def volume(R):
52     return 2*np.pi**2 * R
53
54
55 def differens(r, r_p, theta, theta_p, phi, phi_p):
56     '''
57     Returns the difference between two arbitrary vectors r and r',
58     given their position.
59     '''
60     return np.sqrt(r**2 + r_p**2 - 2*r*r_p*
61     (np.sin(theta)*np.sin(theta_p)*np.cos(phi-phi_p) + np.cos(theta)*np.cos(theta_p)))
62

```

```

63 def dotprod1(k, r_p, alpha, theta_p, beta, phi_p):
64     '''
65     alpha and beta are the azimuthal and the polar angles for
66     k' respectively. The integral should be independent of
67     beta, due to spherical symmetry.
68     '''
69     return k*r_p*(np.sin(alpha)*np.sin(theta_p)*np.cos(beta-phi_p) +
70 np.cos(alpha)*np.cos(theta_p))
71
72 def dotprod2(k,r,theta):
73     '''
74     Returns the value of the dot product between k and r,
75     with the coordinates aligned with the z || z' axes.
76     '''
77     return k*r*np.cos(theta)
78
79 def integrand(k, r, r_p, theta, theta_p, alpha, beta, phi, phi_p):
80     ''' The integrand that we wish to approximate. '''
81     exp1 = np.exp(-1j*dotprod1(k,r_p,alpha,theta_p,beta,phi_p))
82     exp2 = np.exp(1j*dotprod2(k,r,theta))
83     exp3 = np.exp(1j*differens(r, r_p, theta, theta_p, phi, phi_p))
84     fac = r**2*r_p**2*np.sin(theta)*np.sin(theta_p)/differens(r, r_p, theta,
85 theta_p, phi, phi_p)
86     return exp1*exp2*exp3*fac*volume(R)**2
87
88
89 def calc_q(alpha,k):
90     return 2*k*np.sin(alpha/2)
91
92
93 def calc_f1(R, k, alpha):
94     q = calc_q(alpha, k)
95     return (np.sin(q*R)-R*q*np.cos(q*R))
96
97
98 # constants
99 N = int(1e7)
100 eV = 1.6e-19 # J
101 c = 3e8 # m/s
102 hbar = 1.054e-34 # ev s
103 R = 1e-10 # 1 ångström
104 #v0 = (1e-38)*eV # simon test
105 v0 = 1*eV # 1 eV
106 E = 10*eV # 1 ev
107 m = 938e6*eV/c**2 # 938 MeV/c^2
108 k = np.sqrt(2*m*E)/hbar
109 kp = k # Elastic
110
111 # R = 1
112 # hbar = 1
113 # E = 100
114 # v0 = 1
115 # m = 1
116 # k = 70
117 # kp = 1
118 # v0 = 1
119
120
121
122 r = R*np.random.rand(N)
123 r_p = R*np.random.rand(N)
124 theta = np.pi*np.random.rand(N)

```

```

125 theta_p = np.pi*np.random.rand(N)
126 phi     = 2*np.pi*np.random.rand(N)
127 phi_p   = 2*np.pi*np.random.rand(N)
128
129 alpha = np.linspace(0,0.6,100)
130 beta = 0
131
132 f_1 = np.zeros(len(alpha))
133 f_2 = np.zeros(len(alpha))
134
135 for j,alph in enumerate(alpha):
136     print(f'{j+1} of {len(alpha)}')
137     I = integrand(k, r, r_p, theta, theta_p, alph, beta, phi, phi_p)
138     s = (I/N).sum()
139     f_1[j] = -4*m*v0/(hbar**2 * calc_q(alph, k)**3) * calc_f1(R, k, alph)
140     f_2[j] = s
141
142
143 # Insert prefactor for f_2 and plot
144 f_2 = (m*v0/hbar**2)**2 * f_2
145 sigma_contrib2 = np.abs(f_2)**2
146 sigma_contrib1 = np.abs(f_1)**2
147 sigma_contrib_tot = np.abs(f_1 + f_2)**2 # Total differential cross section
148 fig, ax = plt.subplots(3,1)
149 fig.suptitle("Problem 8: Comparison of first and second order")
150 ax[0].plot(alpha, sigma_contrib1, 'b--')
151 ax[0].set_xlabel("alpha, rad")
152 ax[0].set_ylabel("|f1|^2, m^2")
153 ax[0].grid()
154
155 ax[1].plot(alpha, sigma_contrib2, 'r--')
156 ax[1].set_xlabel("alpha, rad")
157 ax[1].set_ylabel("|f2|^2, m^2")
158 ax[1].grid()
159
160 ax[2].plot(alpha, sigma_contrib1, 'b--')
161 ax[2].plot(alpha, sigma_contrib2, 'r--')
162 ax[2].plot(alpha, sigma_contrib_tot, 'g-')
163 ax[2].set_xlabel("alpha, rad")
164 ax[2].set_ylabel("|f1 + f2|^2, m^2")
165 ax[2].grid()
166 plt.tight_layout()
167 plt.savefig("problem8.png")
168 plt.show()
169

```