

PHYS-512, PS2

Muath Hamidi

September, 2022

Problem 1

Well, we can recall the electric field

$$E(r) = \frac{1}{4\pi\epsilon_0} \int \frac{\hat{r}}{r^2} dq \quad (1)$$

Substitute the model,

$$E_z = \frac{1}{4\pi\epsilon_0} \int \frac{z - R \cos \theta}{(R^2 + z^2 - 2Rz \cos \theta)^{3/2}} \sigma R^2 \sin(\theta) d\theta d\phi \quad (2)$$

Integrate over ϕ , and let $u = \cos \theta$

$$E_z = \frac{\sigma R^2}{2\epsilon_0} \int_{-1}^1 \frac{z - Ru}{(R^2 + z^2 - 2Rzu)^{3/2}} du \quad (3)$$

Now, use the code:

```
1 #=====
2 # Course: PHYS 512
3 # Problem: PS2 P1
4 #=====
5 # By: Muath Hamidi
6 # Email: muath.hamidi@mail.mcgill.ca
7 # Department of Physics, McGill University
8 # September 2022
9
10 #=====
11 # Libraries
12 #=====
13 import numpy as np # For math
14 import matplotlib.pyplot as plt # For graphs
15 from scipy.integrate import quad # For math
16
17 #=====
18 # Parameters
19 #=====
20 # Take the whole constant beside the integral --> 1
21 R = 1 # R
22 Z = np.linspace(0, 3*R, 401) # z values, from 0 to 3R
23 u1 = -1 # Integral's lower limit
24 u2 = 1 # Integral's upper limit
25
26 #=====
27 # Integrand
28 #=====
```

```

29 def Integrand(u):
30     Integrand = (z - R*u) / (R**2 + z**2 - 2*R*z*u)**(1.5)
31     return Integrand
32
33 #=====
34 # Integration
35 #=====
36 def Integration(Integrand,u1,u2,tol): # This is Jon's function with modification
37     us = np.linspace(u1,u2,5) # u partition
38     du = us[1] - us[0]
39     y = Integrand(us)
40     area1 = (y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])*du/3 # 3-point with du
41     area2 = (y[0]+4*y[2]+y[4])*(2*du)/3 # 3-point with 2du
42     Error = np.abs(area1-area2)
43     if Error < tol:
44         return area1
45     else:
46         mid = (u1+u2)/2
47         int1 = Integration(Integrand, u1, mid, tol/2)
48         int2 = Integration(Integrand, mid, u2, tol/2)
49         return int1 + int2
50
51 # My Integration
52 MyE = []
53 for i in range(0, len(Z)):
54     z = Z[i]
55     if z == 1: # Here where we have singularity
56         MyE.append(np.nan)
57     else:
58         MyE.append(Integration(Integrand,u1,u2,tol=1e-6))
59
60 # Quad Integration
61 Quad = []
62 for i in range(0, len(Z)):
63     z = Z[i]
64     ans = quad(Integrand, -1, 1)
65     Quad.append(ans[0])
66
67 #=====
68 # Plot
69 #=====
70 plt.plot(Z, Quad, c='green', label="Quad")
71 plt.plot(Z, MyE, ls=':', lw=5, label="My E")
72 plt.title("$E$ vs $z/R$")
73 plt.xlabel("$z/R$")
74 plt.ylabel("$E$")
75 plt.legend()
76 plt.show
77 plt.savefig('2.1.pdf', format='pdf', dpi=1200)

```

Both numerical solutions agrees with the analytic solution, where we have zero field inside the spherical shell, at $z = R$ it jumps to a value equivalent to the electric field value from a dot charge in the center of the sphere, and then decay in rate of $1/z^2$.

There is a singularity in the integral at $z = R$. It seems that quad doesn't care about it, while my integrator does. That's why we needed to do something about it.

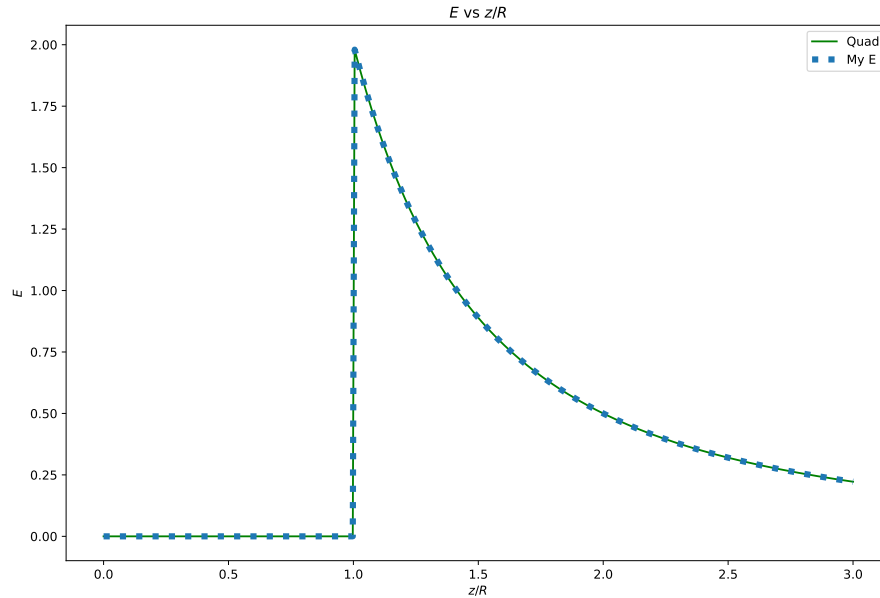


Figure 1: E vs z/R.

Problem 2

Following the Code:

```

1 #=====
2 # Course: PHYS 512
3 # Problem: PS2 P2
4 #=====
5 # By: Muath Hamidi
6 # Email: muath.hamidi@mail.mcgill.ca
7 # Department of Physics, McGill University
8 # September 2022
9
10 #=====
11 # Libraries
12 #=====
13 import numpy as np # For math
14 import matplotlib.pyplot as plt # For graphs
15
16 #=====
17 # Functions
18 #=====
19 def heaviside(x):
20     return 1.0*(x>0)
21
22 def offset_gauss(x):
23     return 1+10*np.exp(-0.5*x**2/(0.1)**2)
24
25 def cos(x):
26     return np.cos(x)
27

```

```

28 #=====
29 # Integrate Adaptive
30 #=====
31 def integrate_adaptive(fun,a,b,tol,extra=None):
32     global counter
33     if extra==None:
34         xs = np.linspace(a,b,5) # x partition
35         dx = xs[1] - xs[0]
36         y = fun(xs)
37         counter += len(y)
38         area1 = (y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])*dx/3 # 3-point with dx
39         area2 = (y[0]+4*y[2]+y[4])*(2*dx)/3 # 3-point with 2dx
40         Error = np.abs(area1-area2)
41         if Error < tol:
42             return area1
43         else:
44             mid = (a+b)/2
45             int1 = integrate_adaptive(fun, a, mid, tol/2, extra=[y[0], y[1], y[2],
dx]]
46             int2 = integrate_adaptive(fun, mid, b, tol/2, extra=[y[2], y[3], y[4],
dx])
47             return int1 + int2
48
49     else:
50         x = np.array([a+0.5*extra[3],b-0.5*extra[3]])
51         y = fun(x)
52         counter += len(y)
53         dx = extra[3]/2
54         area1 = dx*(extra[0]+4*y[0]+2*extra[1]+4*y[1]+extra[2])/3
55         area2 = 2*dx*(extra[0]+4*extra[1]+extra[2])/3
56         Error = np.abs(area1-area2)
57         if Error < tol:
58             return area1
59         else:
60             mid=(a+b)/2
61             int1 = integrate_adaptive(fun, a, mid, tol/2, extra = [extra[0] ,y[0],
extra[1], dx])
62             int2 = integrate_adaptive(fun, mid, b, tol/2, extra = [extra[1] ,y[1],
extra[2], dx])
63             return int1 + int2
64
65 #=====
66 # Integration (Jon's Function)
67 #=====
68 def Integration(fun,a,b,tol): # This is Jon's function with modification
69     global counter
70     xs = np.linspace(a,b,5) # x partition
71     dx = xs[1] - xs[0]
72     y = fun(xs)
73     counter += len(y)
74     area1 = (y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])*dx/3 # 3-point with dx
75     area2 = (y[0]+4*y[2]+y[4])*(2*dx)/3 # 3-point with 2dx
76     Error = np.abs(area1-area2)
77     if Error < tol:
78         return area1
79     else:

```

```

80     mid = (a+b)/2
81     int1 = Integration(fun, a, mid, tol/2)
82     int2 = Integration(fun, mid, b, tol/2)
83     return int1 + int2
84
85 #=====
86 # Calculations
87 #=====
88 # Heaviside
89 # Integration Adaptive
90 counter = 0
91 Integ = integrate_adaptive(heaviside, a=0.1, b=1, tol=1e-6, extra=None)
92 print("Number of function calls, Heaviside, Mine:", counter)
93 # Jon's Function
94 counter = 0
95 Integ = Integration(heaviside, a=0.1, b=1, tol=1e-6)
96 print("Number of function calls, Heaviside, Jon's:", counter)
97
98 # Cos
99 # Integration Adaptive
100 counter = 0
101 Integ = integrate_adaptive(cos, a=-1, b=1, tol=1e-6, extra=None)
102 print("Number of function calls, Cos, Mine:", counter)
103 # Jon's Function
104 counter = 0
105 Integ = Integration(cos, a=-1, b=1, tol=1e-6)
106 print("Number of function calls, Cos, Jon's:", counter)
107
108 # Gauss
109 # Integration Adaptive
110 counter = 0
111 Integ = integrate_adaptive(offset_gauss, a=-1, b=1, tol=1e-6, extra=None)
112 print("Number of function calls, Gauss, Mine:", counter)
113 # Jon's Function
114 counter = 0
115 Integ = Integration(offset_gauss, a=-1, b=1, tol=1e-6)
116 print("Number of function calls, Gauss, Jon's:", counter)

```

The results we have:

Number of function calls, Heaviside, Mine: 5
 Number of function calls, Heaviside, Jon's: 5
 Number of function calls, Cos, Mine: 65
 Number of function calls, Cos, Jon's: 155
 Number of function calls, Gauss, Mine: 409
 Number of function calls, Gauss, Jon's: 1015

Where in the Heaviside, $x \in [0.1, 1]$. In the Cos, $x \in [-1, 1]$. In the Gauss, $x \in [-1, 1]$.

Problem 3

Our function here is

$$f(x) = \log_2(x) \quad (4)$$

in the interval $x \in [0.5, 1]$. Using `np.polynomial.chebyshev.chebfit` in the code:

```
1 #=====
2 # Course: PHYS 512
3 # Problem: PS2 P3
4 #=====
5 # By: Muath Hamidi
6 # Email: muath.hamidi@mail.mcgill.ca
7 # Department of Physics, McGill University
8 # September 2022
9
10 #=====
11 # Libraries
12 #=====
13 import numpy as np # For math
14 import matplotlib.pyplot as plt # For graphs
15 import numpy.polynomial.chebyshev as chebyshev
16 from scipy.special import legendre
17
18 #=====
19 # Chebyshev
20 #=====
21 def Chebyshev(z):
22     x = np.linspace(0.5,1,101)
23     rescaled_x = 4 * x - 3
24     y = np.log2(x)
25     order = 7 # Using this degree gives 1e-07 to 1e-06 error magnitude. Which is
26               # what we want. Known by plotting later.
27     return chebyshev.chebval(4 * z - 3, chebyshev.chebfit(rescaled_x, y, order))
28
29 #=====
30 # Plot Chebyshev
31 #=====
32 z = np.linspace(0.5,1,101)
33 rms = np.sqrt(np.sum((np.log2(z) - Chebyshev(z))**2)/len(z)) # RMS in Chebyshev
34 print("RMS in Chebyshev:",rms)
35 plt.plot(z, np.log2(z), label="$\log_2$")
36 plt.plot(z, Chebyshev(z), ls=':', lw=5, label="Chebyshev")
37 plt.title("Chebyshev")
38 plt.xlabel("x")
39 plt.ylabel("y")
40 plt.legend()
41 plt.savefig('2.3.1.pdf', format='pdf', dpi=1200)
42 plt.show()
43 plt.close()
44
45 Error = Chebyshev(z) - np.log2(z)
46 plt.plot(z, Error) # Error plot
47 plt.title("Error in Chebyshev")
48 plt.xlabel("x")
49 plt.ylabel("y")
50 plt.savefig('2.3.2.pdf', format='pdf', dpi=1200)
51 plt.show()
52 plt.close()
53 #=====
```

```

54 # mylog2
55 #=====
56 def mylog2(z):
57     mantissa, exponent = np.frexp(z)
58     x = np.linspace(0.5,1,101)
59     rescaled_x = 4 * x - 3
60     y = np.log(x)
61     order = 7
62     cheb = chebyshev.chebval(4 * mantissa - 3, chebyshev.chebfit(rescaled_x, y,
63         order))
64     return exponent * np.log(2) + cheb
65 #=====
66 # Plot mylog2
67 #=====
68 z = np.linspace(0.01,100,10000)
69 rms = np.sqrt(np.sum((np.log(z) - mylog2(z))**2)/len(z)) # RMS in mylog2
70 print("RMS in mylog2:",rms)
71 plt.plot(z, np.log(z), label="$log$")
72 plt.plot(z, mylog2(z), ls=':', lw=5, label="mylog2")
73 plt.title("mylog2")
74 plt.xlabel("x")
75 plt.ylabel("y")
76 plt.legend()
77 plt.savefig('2.3.3.pdf', format='pdf', dpi=1200)
78 plt.show()
79 plt.close()
80
81 Error = mylog2(z) - np.log(z)
82 plt.plot(z, Error) # Error plot
83 plt.title("Error in mylog2")
84 plt.xlabel("x")
85 plt.ylabel("y")
86 plt.savefig('2.3.4.pdf', format='pdf', dpi=1200)
87 plt.show()
88 plt.close()
89
90 #=====
91 # Legendre Polynomial (Bonus)
92 #=====
93 def Legendre(z):
94     x = np.linspace(0.5,1,101)
95     rescaled_x = 4 * x - 3
96     order = 7
97     y = legendre(order)(z)
98     cheb = chebyshev.chebval(4 * z - 3, chebyshev.chebfit(rescaled_x, y, order))
99     return cheb
100
101 #=====
102 # Plot Legendre
103 #=====
104 order = 7
105 z = np.linspace(0.5,1,101)
106 rms = np.sqrt(np.sum((legendre(order)(z) - Legendre(z))**2)/len(z)) # RMS in
    Legendre
107 print("RMS in Legendre:",rms)

```

```

108 plt.plot(z, legendre(order)(z), label="$Legendre true$")
109 plt.plot(z, Legendre(z), ls=':', lw=5, label="Legendre")
110 plt.title("Legendre")
111 plt.xlabel("x")
112 plt.ylabel("y")
113 plt.legend()
114 plt.savefig('2.3.5.pdf', format='pdf', dpi=1200)
115 plt.show()
116 plt.close()
117
118 Error = Legendre(z) - legendre(order)(z)
119 plt.plot(z, Error) # Error plot
120 plt.title("Error in Legendre")
121 plt.xlabel("x")
122 plt.ylabel("y")
123 plt.savefig('2.3.6.pdf', format='pdf', dpi=1200)
124 plt.show()
125 plt.close()

```

The results are:

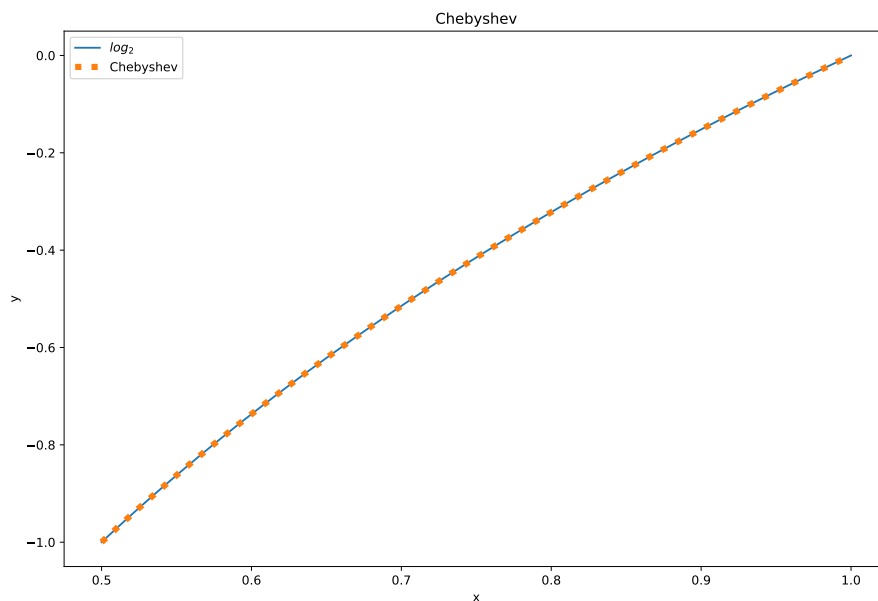


Figure 2: Chebyshev.

Where we have the RMSs:

RMS in Chebyshev: 1.7989069481332663e-07

RMS in mylog2: 1.2150018872283483e-07

RMS in Legendre: 2.2416770311835436e-16

We observe high accuracy in matching the three functions, specially the Legendre polynomial which has accuracy down to machine precision.

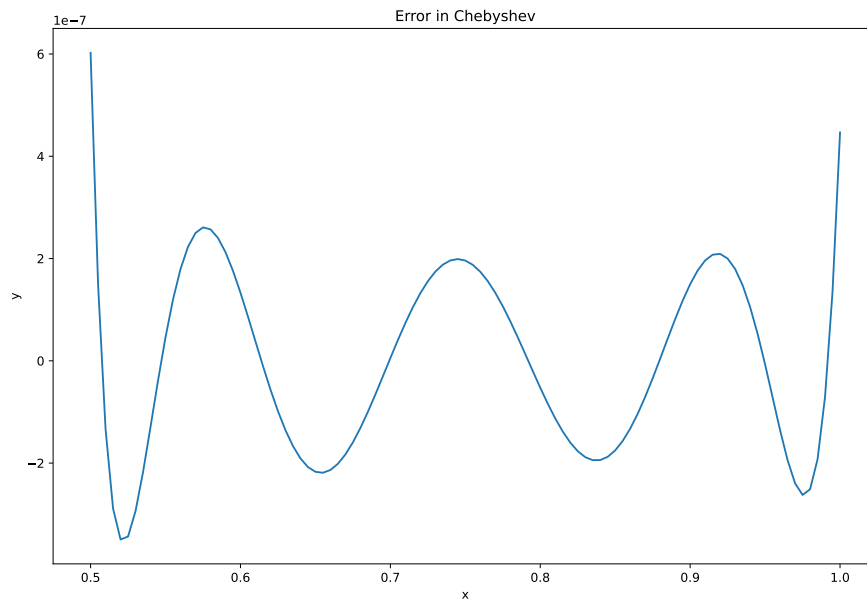


Figure 3: Error in Chebyshev.

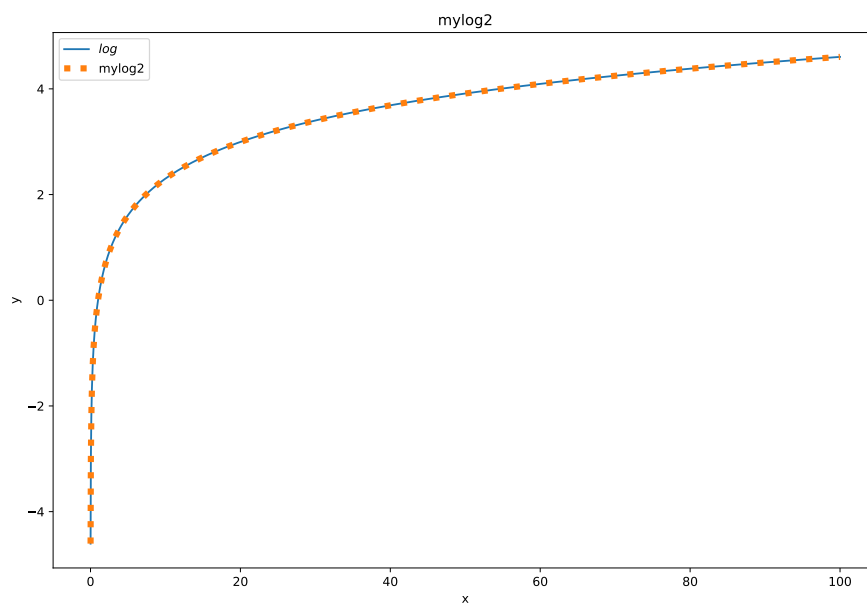


Figure 4: mylog2.

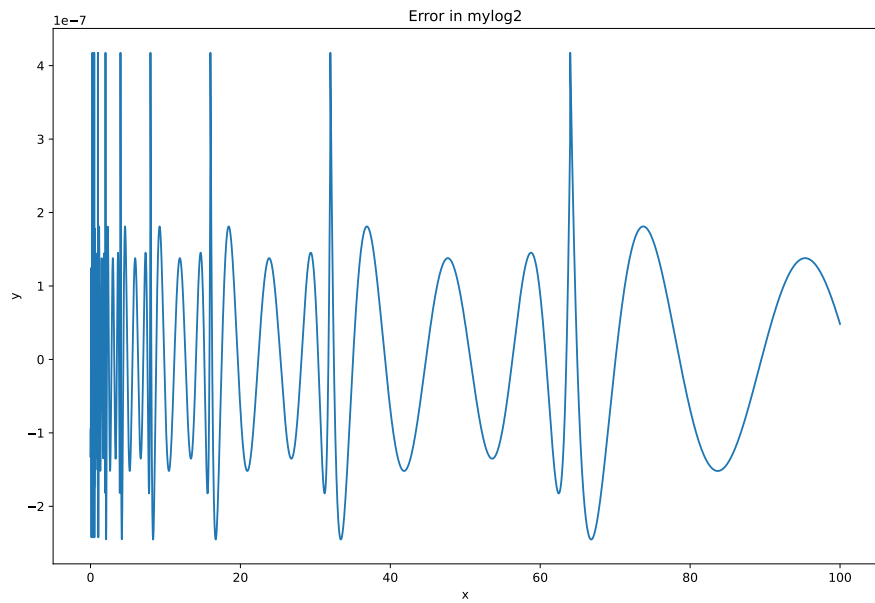


Figure 5: Error in mylog2.

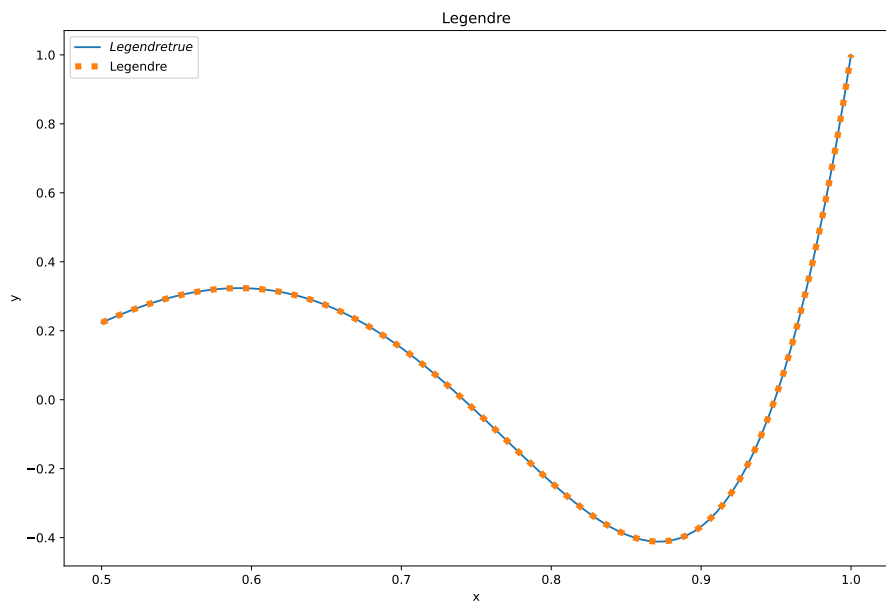


Figure 6: Legendre.

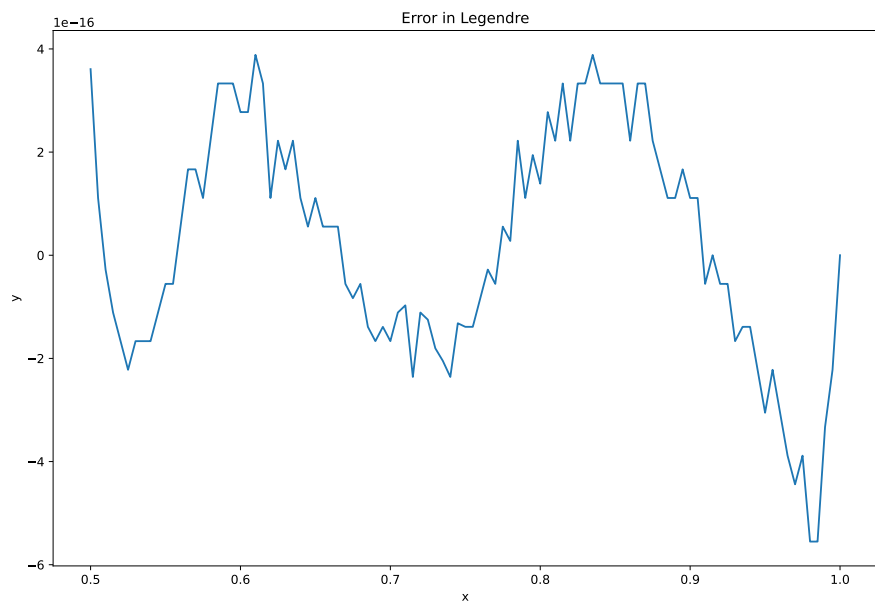


Figure 7: Error in Legendre.