HW1_2021

October 7, 2021

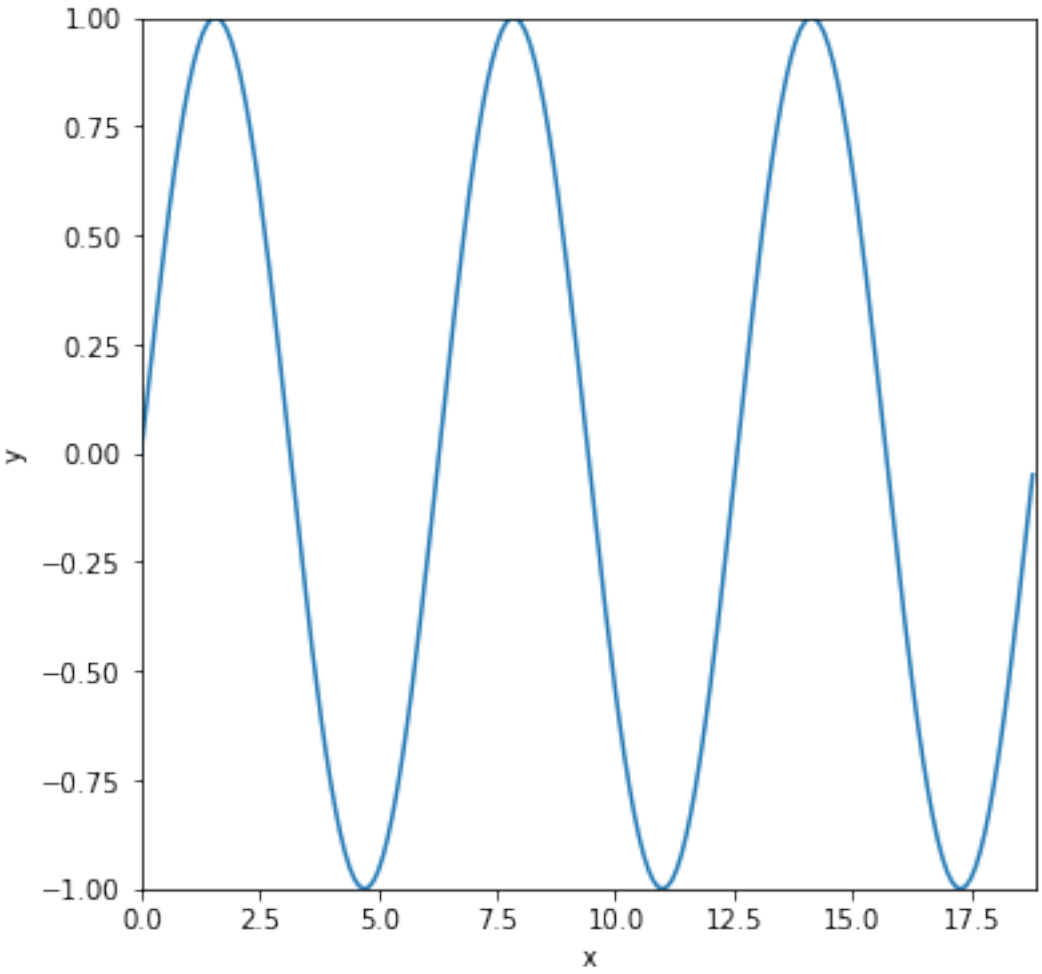
1 Basic plotting

Visualizing data and plotting functions are essential tasks. First we need to import the libraries.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In order to plot a function, you need a paired list of data. One way to do this is to create an array of values of the independent variable. You can then pass this array to the function you wish to plot

[2]:
```
x=np.arange(0,6*np.pi, 0.1)   #define the array of x-values
y=np.sin(x)                   #define the function, for instance a sine
plt.figure(figsize=(6,6))     #create a figure object
plt.plot(x,y)                 #plot the function
plt.xlabel('x')               #label the x-axis
plt.ylabel('y')               #label the y-axis
plt.axis([0,6*np.pi,-1,1]);    #set the x and y limits
```

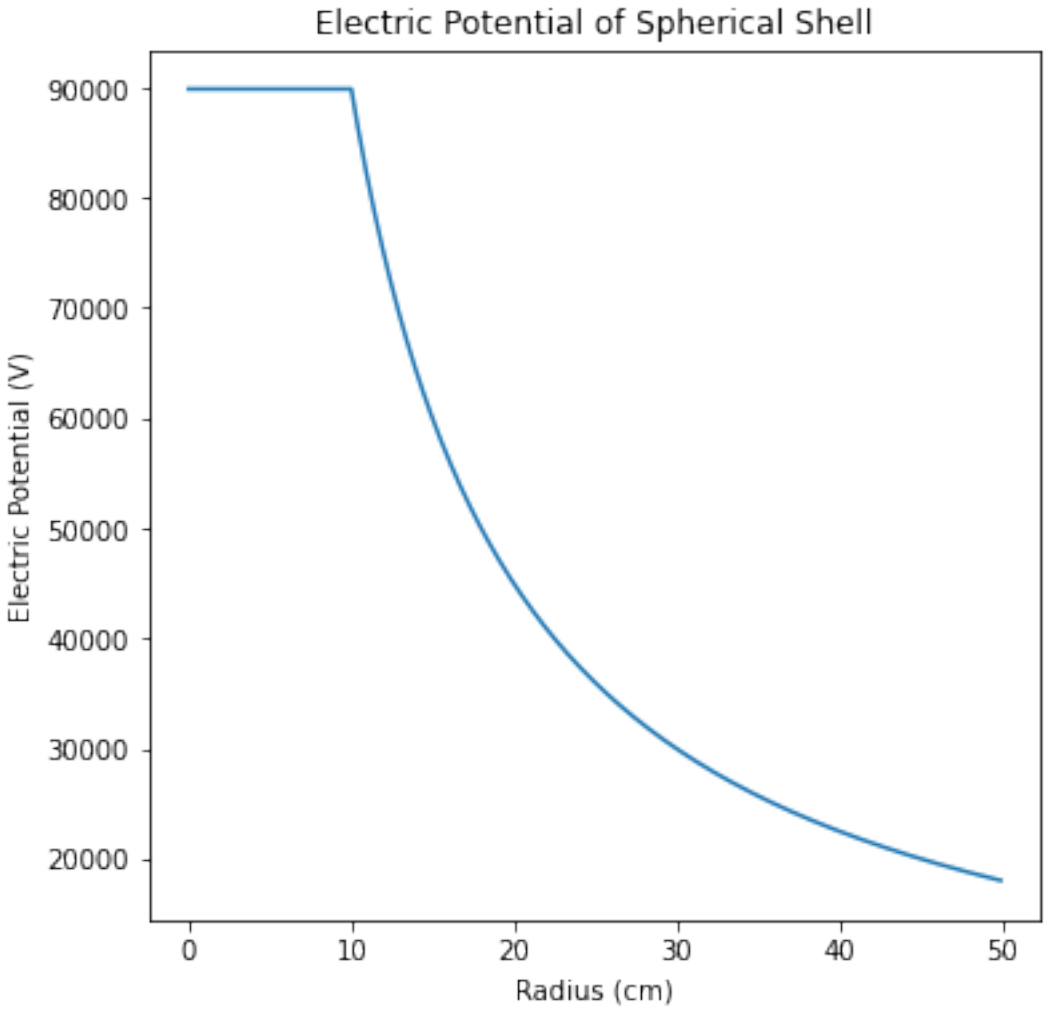


2 Assignment 1:

In first year E&M you derived the electric potential of a charged spherical shell. If you forgot the answer, you can find it in Griffiths example 2.7. Plot this potential as a function of distance r from the centre of the spherical shell of radius $R = 10cm$ that carries a total charge $Q = 1\,\mu C$ for $0 < r < 50\,cm$. Make sure to label the axes of the graph and indicate appropriate units.

[8]:
```
# Reimport in case previous cells aren't loaded when grading
import numpy as np
import matplotlib.pyplot as plt

EPSILON=8.854e-12
q = 1e-6
R = 10
r = np.arange(0, 50, 0.1)
V = [(100/R if s < R else 100/s) * q / (4*np.pi*EPSILON) for s in r]
```

```
plt.figure(figsize=(6,6))
plt.plot(r, V)
plt.xlabel('Radius (cm)')
plt.ylabel('Electric Potential (V)')
plt.title('Electric Potential of Spherical Shell');
```



[ ]:

3 Numerical integration

Another typical reason for using computers is to solve integrals that cannot be computed analytically. One of the most basic numerical approximations of a definite integral is the trapezoidal rule, which approximates the area under the function g(x) as a trapezoid. Let's calculate for instance $\int_0^{10} x^3 dx$:

[ ]:
```
def g(x):                               #define the function to integrate
    return x**3

N = 100                                 #number of grid points
a = 0
b = 10
h = (b-a)/N                             #grid spacing
s = 0                                   # value of integral

s = 0.5*g(a) + 0.5*g(b)                 #do the end points first
for k in range(1,N):                    # for loop sums up all points in between
    s += g(a+k*h)

I = s*h
print(I)
```

```
2500.2500000000005
```

The answer is a bit off the exact result 2500, and we can increase N to decrease the error. However, there are more sophisticated algorithms for which the error decreases faster than in the trapezoidal rule. One of them is provided by the quad function, which provides a much better answer.

[ ]:
```
from scipy.integrate import quad    #Import a specific integrator from scipy.
 ↪intergrate library
def g(x):                           #define the function to integrate
    return x**3
I = quad(g, 0,10)                   #perform the itegration
print(I[0],I[1])                    #quad returns a list, first value is the␣
 ↪answer, the second is an error estimate
```

```
2500.0000000000005 2.775557561562892e-11
```

4 Assignment 2:

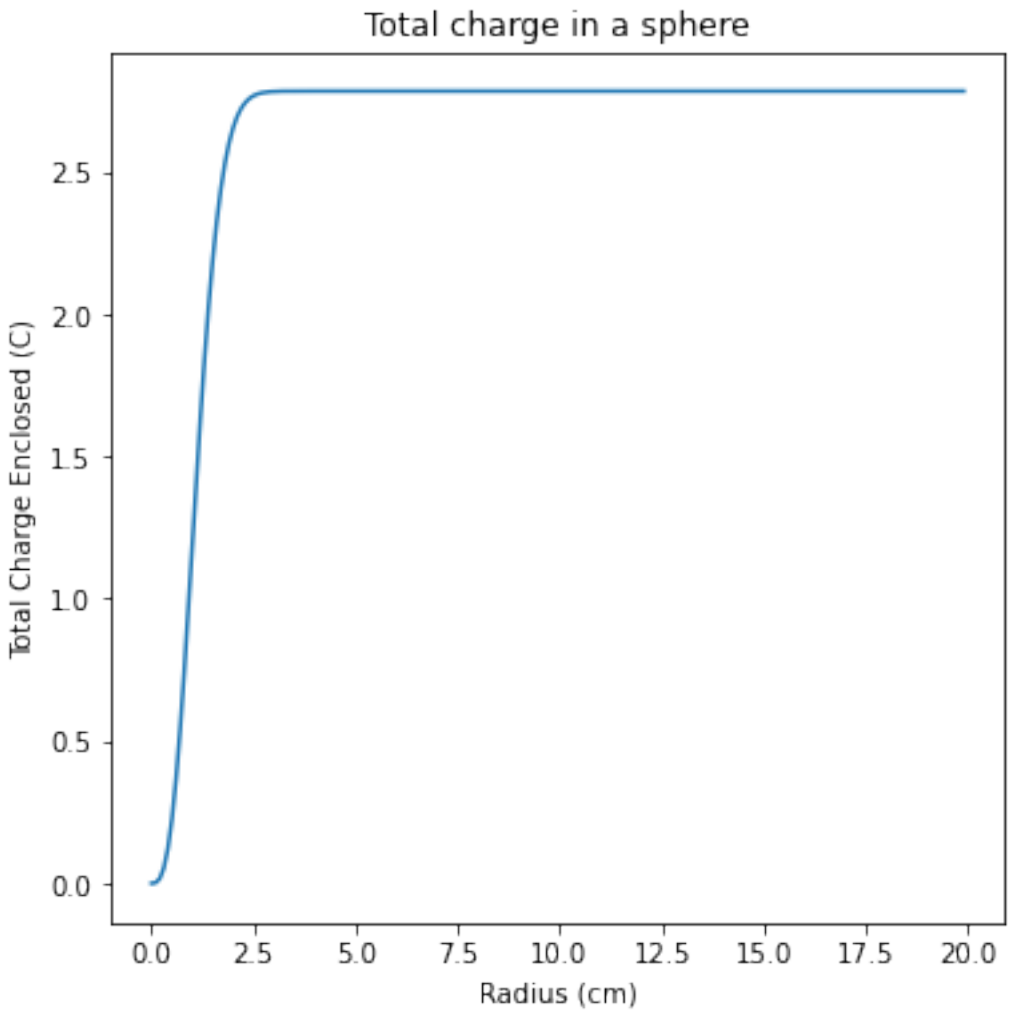
Consider a charged sphere of radius $R = 10\,cm$ with charge density

$$\rho(r) = 0.5e^{-r^2}\mathrm{C/m}^3$$

Plot the total charge enclosed by a Gaussian (i.e. spherical) surface of radius r as a function of r for $0 < r < 20\,cm$. Again don't forget to label the axes of the graph.

[16]:
```
# Reimport in case previous cells aren't loaded when grading
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def rho(r):
```

```
  return 0.5 * np.exp(-r**2)

def chargeInside(r):
  # Integrals are separable
  rInt = quad(lambda s: s**2 * rho(s), 0, r)[0]
  thetaInt = quad(lambda theta: np.sin(theta), 0, np.pi)[0]
  phiInt = 2*np.pi
  return rInt * thetaInt * phiInt


R = 10
r = np.arange(0, 20, 0.1)
chargeOutside = chargeInside(R)
C = [chargeInside(s) if s < 10 else chargeOutside for s in r]


plt.figure(figsize=(6,6))
plt.plot(r, C)
plt.xlabel('Radius (cm)')
plt.ylabel('Total Charge Enclosed (C)')
plt.title('Total charge in a sphere');
```


Total charge in a sphere
Total Charge Enclosed (C)
0.0
0.5
1.0
1.5
2.0
2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Radius (cm)