Report - Assignment 4

-Akshay Anand (EE16B046)

February 28, 2018

Abstract

In this week's assignment, the bessel function of the first type ($J_v(x)$) was plotted for 41 values from 0 to 20. Then, an approximate equation in the form of $A\cos(x_i) + B\sin(x_i) \approx J_1(x_i)$ was first solved by least squares approximation to get the values of A and B, from which the value of ν used in the approximate bessel function equation was calculated. Then, a more accurate equation of the form $A\frac{\cos(x_i)}{\sqrt{x}} + B\frac{\sin(x_i)}{\sqrt{x}} \approx J_1(x_i)$ was solved using the same method to approximate the value of ν and the results obtained were noted. Finally, some noise (with mean 0) was also added to the calculated value of the bessel function and ν was calculated again and the deviation obtained was noted.

Approximate Bessel function equation used for finding ν:

$$J_v(x) \approx \sqrt{\frac{2}{\pi x}}\cos\left(x - \frac{\nu\pi}{2} - \frac{\pi}{4}\right)$$

Libraries Used

```
import numpy as np
import math
from scipy.special import jv
from numpy.linalg import lstsq
import matplotlib.pyplot as plt
```

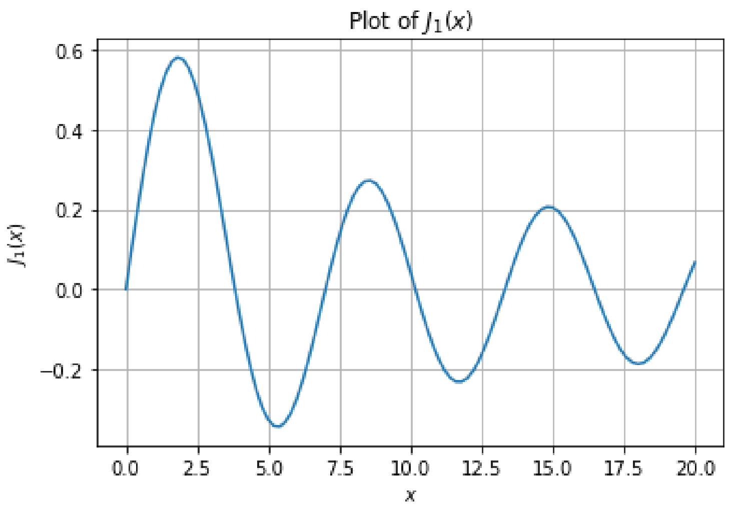
1 Initializing vectors

A vector of 41 values from 0 to 20 is initialized in a variable x and the value of the bessel function of the first type for ν = 1 was calculated at each of these points and plotted.

```
def J(x,v):
    return jv(v,x)    #in-built bessel function in scipy.special
h = 0.5  # step-size
np.random.seed(2)  # A seed given so that uniform noise is generated across step-size values
(and hence different runs of same program)
x = np.linspace(0.0, 20.0, (20.0)/h + 1) # vector initialised
y = J(x,1)
```

The bessel function is now plotted from 0 to 20 with 101 values so as to obtain a smooth graph.

```
x_plot = np.linspace(0.0,20.0,101)
plt.plot(x_plot,J(x_plot,1))
plt.grid()
plt.title('Plot of $J_1(x)$')
plt.ylabel('$J_1(x)$')
plt.xlabel('$x$')
plt.show()
```



2 Function for calculating ν

The function for calculating ν is defined as follows

```
def calcnu(x,x0,eps,model):
    i = np.where(x==x0)[0][0]   # index in x corresponding to x0 calculated
    x_new = x[i:len(x)] # sub vector of x extracted based on starting index
    y_new = y[i:len(x)] # sub vector of y extracted based on starting index
    y_new = y_new + eps*np.random.randn(len(y_new))
    A = np.zeros((len(x_new),2)) # The 2D matrix is initialised
    if (model == 'b'): # model 'b' corresponds to the equation in question (b)
        A[:,0]=np.cos(x_new)
        A[:,1]=np.sin(x_new)
    elif (model == 'c'): # model 'c' corresponds to the equation in question (c)
        A[:,0]=np.cos(x_new)/np.sqrt(x_new)
        A[:,1]=np.sin(x_new)/np.sqrt(x_new)
    c = lstsq(A,y_new)[0] # Values of A and B found as c[0] and c[1] respectively
    phi = math.acos(c[0]/(np.sqrt(c[0]*c[0] + c[1]*c[1]))) # phi calculated
    v = phi - (math.pi/4)
    v = v / (math.pi/2) # nu finally calculated and returned
    return v
```

In the above function the parameters are:

x: The vector ranging from 0 to 20 with h step-size.

x0: The element in x from where the sub-vector is to be taken.

eps: The amount of noise to be added. $eps = 0$ means no noise is added.

model: The equation to be used for finding ν. The one in question (b) is taken if model = 'b', and the one in 'c' is taken if model = 'c'

3 Calculation and plot of ν

The different ν values obtained using model 'b', model 'c' with no noise and model 'c' with noise (eps = 0.01) are calculated in the sub-vector of x starting from x_0 where x_0 ranges from 0.5 to 18.

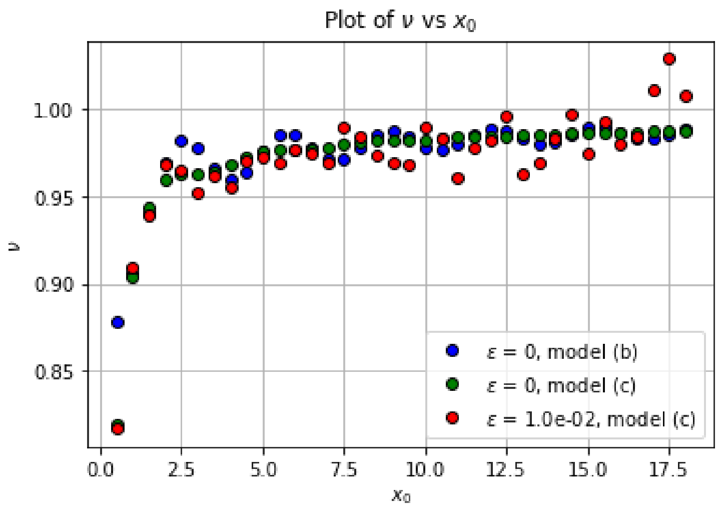
```
x0 = np.linspace(0.5,18.0,(18.0-0.5)/h + 1)
nu_b = []
nu_c_no_noise = []
nu_c_with_noise = []
for i in x0:
        nu_b.append(calcnu(x,i,0,'b'))
        nu_c_no_noise.append(calcnu(x,i,0,'c'))
        nu_c_with_noise.append(calcnu(x,i,0.01,'c'))
    print ("The maximum error between the calculated nu values with and without noise is:
    ",max(np.absolute(np.array(nu_c_no_noise)-np.array(nu_c_with_noise))))
```

In this case, the output obtained was:

```
The maximum error between the calculated nu values with and without noise is:  0.0413326561536
```

The above calculated values are now plotted against x_0.

```
plt.plot(x0, nu_b, 'bo', markeredgecolor='black')
plt.plot(x0, nu_c_no_noise, 'go', markeredgecolor='black')
plt.plot(x0, nu_c_with_noise, 'ro', markeredgecolor='black')
plt.legend(('$\epsilon$ = 0, model (b)','$\epsilon$ = 0, model (c)','$\epsilon$ = 1.0e-02, model (c)'))
plt.grid()
plt.title(r'Plot of $\nu$ vs $x_0$')
plt.xlabel('$x_0$')
plt.ylabel(r'$\nu$')
plt.show()
```

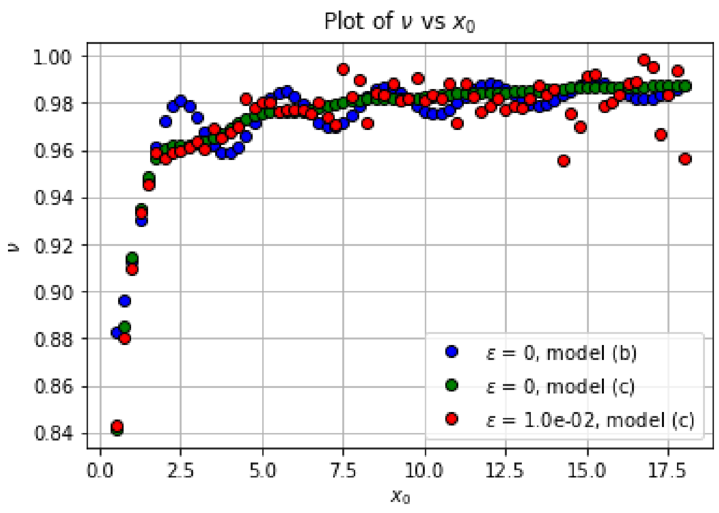


4 Further calculations

Now, the number of measurements are increased (i.e step-size in x is reduced) and ν is calculated again to observe the changes with respect to that calculated with higher step-size.

4.1 Step-size = 0.25

With this step-size, the final plot of ν vs x_0 obtained is:

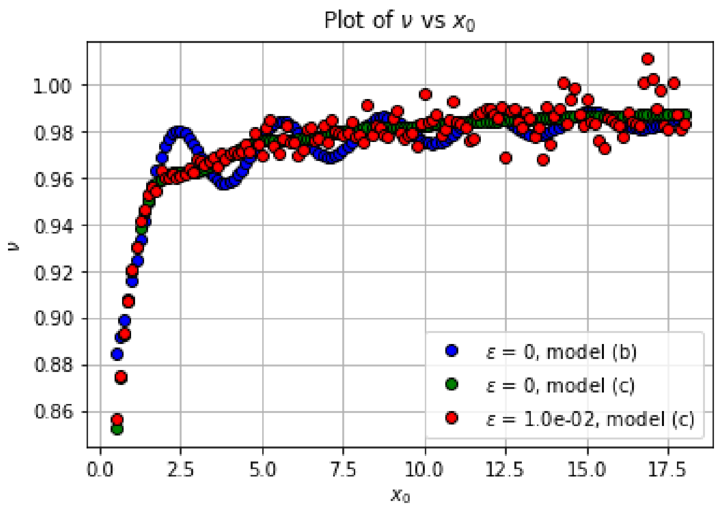


In this case, output obtained was:

```
The maximum error between the calculated nu values with and without noise is:  0.0316178012703
```

4.2 Step-size = 0.125

With this step-size, the final plot of ν vs x_0 obtained is:



In this case, output obtained was:

```
The maximum error between the calculated nu values with and without noise is:  0.0239385036311
```

5 Inferences

Thus, we find the final fit depends on:

1. Model Accuracy: We find that the plot corresponding to model 'b' (blue dots) which is a less accurate model for the bessel function, varies quite a bit initially and even near the end does not completely approach the correct value of 1 and the oscillatory variation is still present. However, for model 'c' (green dots), which is a more accurate estimate of the bessel function, such oscillatory variations are not there and the estimated ν values steadily increases with x_0 which means it could approach 1 provided range is increased and hence accuracy is more for model 'c' than model 'b'.

2. Effect of noise: We see from the above plots that with noise, the calculated values vary quite a bit from the actual values without noise, this becoming more prominent as x_0 increases. This is because for lower x_0 values, there are more elements in the sub-vector taken and hence, the net effect of noise added through numpy.random.randn (which follows a standard normal distribution) will average out to almost 0 whereas if number of elements is less, the effect of noise will be more pronounced. This is also seen in the graph where the deviation from the actual model 'c' plot is more at larger x_0 values than at smaller ones.

3. Number of measurements: We find that even if the number of measurements is increased, model 'b' and model 'c' look similar and hence are not very affected by the number of measurements, but it is observed that the maximum error between ν values of model 'c' with noise and without noise, reduces as number of measurements increases because of the same reason as to why the effect of noise is minimal in lower x_0 (above point). Hence, as number of measurements increases, it stabilizes the plot more.