EE - 2703 Applied Programming Lab

Assignment -4

Fitting Data to Models

Mohammed Khandwawala EE16B117

Contents

1 Introduction

The **Bessel functions** of the first kind $J_n(x)$ are defined as the solutions to the Bessel differential equation.

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - n^2)y = 0$$

which are nonsingular at the origin. They are sometimes also called cylinder functions or cylindrical harmonics. This Assignment is about fitting data to models. So we will be taking two models of Bessel function and fitting them using least squares. The Bessel Function is given by

$$\sqrt[2]{\frac{2}{x\pi}}cos(x - \frac{v\pi}{2} - \frac{\pi}{4}) \approx J_v(x) - (1)$$

First Model , lets call it model(b) . Approximate value of $J_1(x)$ can be given by,

$$Acos(x_i) + Bsin(x_i) \approx J_1(x_i)$$

Second Model , lets call it model(c) . Its approximation is as follows.

$$\frac{Acos(x_i)}{\sqrt[2]{x_i}} + \frac{Bsin(x_i)}{\sqrt[2]{x_i}} \approx J_1(x_i)$$

After fitting data to the model obtained from inbuilt Bessel function to obtain the constants . To estimate the measure of the fit we will try to get v from the constants. The closer and faster it reaches 1 the better fitting our model would be.

From both the above fits to estimate v. From eqation (1)

$$\phi = \frac{v\pi}{2} + \frac{\pi}{4}$$

and

$$cos(\phi) = \frac{A}{\sqrt[2]{A^2 + B^2}}$$

So,

$$v = \frac{2}{\pi}(cos^{-1}(\frac{A}{\sqrt[2]{A^2 + B^2}})) - \frac{\pi}{4})$$

2 About Least Squares Fitting

Solves the equation A X = B by computing a vector x that minimizes the Euclidean 2-norm $|||\ b - ax\ |||^{\ 2}$. The equation may be under, well, or over- determined (i.e., the number of linearly independent rows of a can be less than, equal to, or greater than its number of linearly independent columns). If a is square and of full rank, then X (but for round-off error) is the "exact" solution of the equation. From linear algebra package of numpy lstsq function does the work for us which is used in the assignment to solve matrix equations by least square.

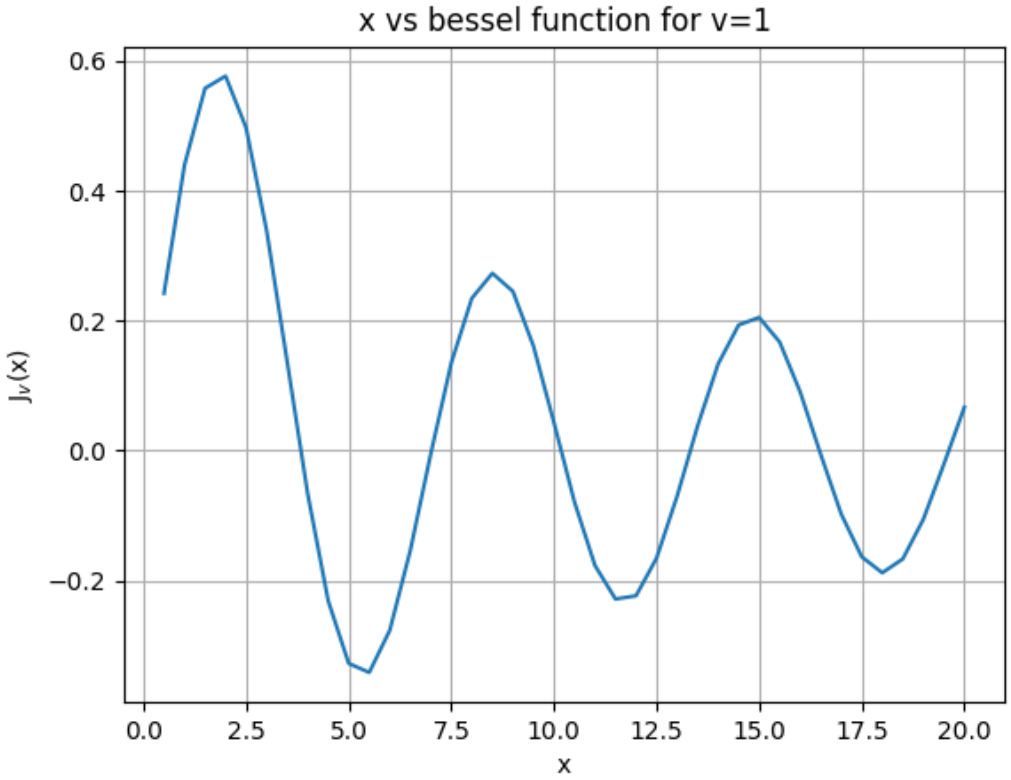

Figure 1: Plot of bessel function with v=1

2.1 Function Definition

Defining function to return the value of Bessel function of v=1.For this we are using inbuilt function from python. It is included under special functions in scipy package.

Listing 1: Defining Functions

```
from scipy import linalg,special
#defining approximate bessel function
def J(x,v):
        return special.jv(1,x)
```

As mentioned in the problem statement , a vector of 40 length from 0-20 with 40 step size is created and it is passed to bessel function. The functional values obtained are plotted against the vector. NOTE : Bessel funtion was not defined at 0 and thus only plotted from 0.5.See figure 1.

Listing 2: Plotting the functions

```
x = np.linspace(0.5,20,40)

#calculating bessel function for v = 1
jx1 =  J(x,1)

#plotting J1x
plt.plot(x,jx1)
plt.ylabel("J$_{v}$(x)")
plt.xlabel("x")
plt.grid(True)
plt.title("x vs bessel function for v=1")
plt.show()
```

2.2 Estimation of v

In earlier section model(b) and model(c) and the way to estimate v are already defined and the way to estimate v. Model(b) can be converted to the matrix equation given.

$$\begin{bmatrix} cos(x) & sin(x) \\ cos(x+h) & sin(x+h) \\ ... & ... \\ cos(x_o) & sin(x_o) \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} J_1(x) \\ J_1(x+h) \\ ... \\ J_1(x_o) \end{bmatrix} \tag{1}$$

Following naming convention for the corresponding matrix is used in the code.

$$M1M2 = M3$$

Here h is step size. Solving this equation multiple times for $0 < \mathrm{x} < \mathrm{x}_o$ using least square. For each solution v is calculated and plotted.

2.2.1 Estimation of v for model(b)

For x_o different A,B values are obtained and the v is calculated by comparing $\cos_{-1}\phi$ with phase. And then then the obtained values of v are plotted against x.See Figure 2.

```
v=[]
#To store v from second method
v_2=[]
for i in range(0,37):
        x1 = x[i:40]
        #to get matrix with sinusoids
        M1_1 = np.transpose(sincos(x1))
        #to get A,B using least square by matrix approach
        M3_1 = J(x1,1)
        coeff = linalg.lstsq(M1_1,M3_1)[0]
        phi = coeff[0]/np.sqrt(coeff[0]**2+coeff[1]**2)
        v.append((2/np.pi)*(np.arccos(phi)-np.pi/4))
```

2.2.2 Estimation of v for model(c)

Similar to model (b) just the M1 matrix will change Now,

$$M1 = \begin{bmatrix} \frac{cos(x)}{\sqrt[2]{x}} & \frac{sin(x_o)}{\sqrt[2]{x}} \\ \frac{cos(x+h)}{\sqrt[2]{x+h}} & \frac{sin(x+h)}{\sqrt[2]{x_o+h}} \\ ... & ... \\ \frac{cos(x_o)}{\sqrt[2]{x_o}} & \frac{sin(x_o)}{\sqrt[2]{x_o}} \end{bmatrix} \tag{2}$$

For x_o different A,B values are obtained and the v is calculated by comparing $\cos_{-1}\phi$ with phase. And then then the obtained values of v are plotted against x. See Figure 2.

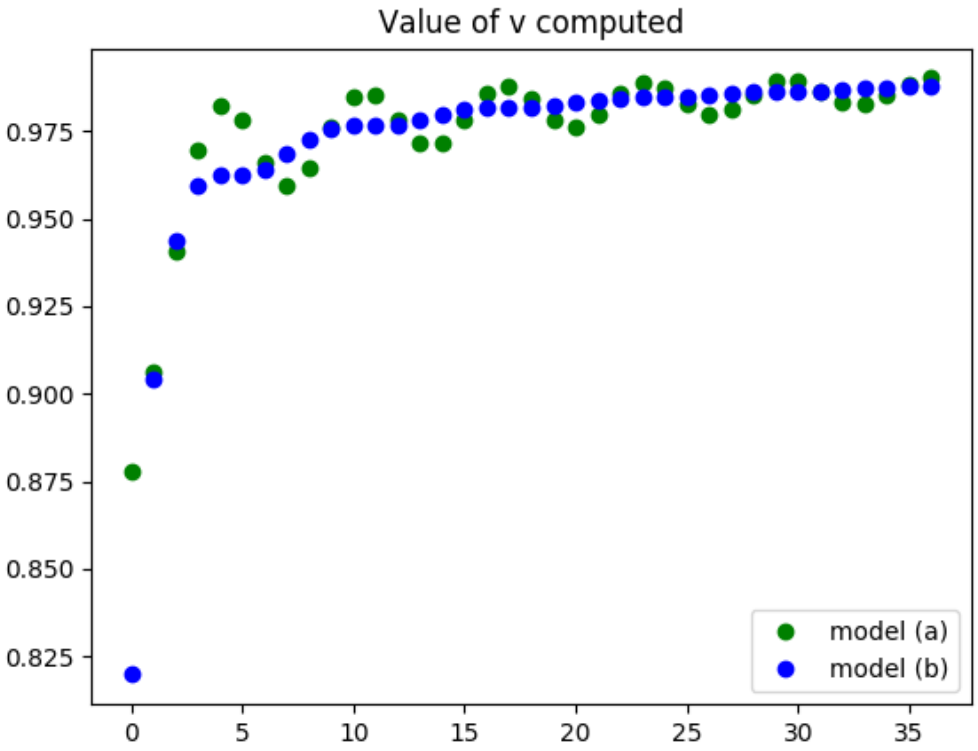

Figure 2: plotting value of v obtained by both the models

```
for i in range(0,37):
        x1 = x[i:40]
        #creating matrix with sinusoids by root x
        M1_2 = np.transpose(sincos2(x1))
        print M1_2
        #to get A,B using least square by matrix approach
        M3_2 = J(x1,1)
        print M3_2
        coeff2 = linalg.lstsq(M1_2,M3_2)[0]
        phi_2 = coeff2[0]/np.sqrt(coeff2[0]**2+coeff2[1]**2)
        print phi_2
        v_2.append((2/np.pi)*(np.arccos(phi_2)-np.pi/4))

plt.plot(v,"go")
plt.plot(v_2,"bo")
plt.title("Value of v computed")
plt.legend(["model (a)","model (b)"])
plt.show()
```

2.2.3 Figure 2 observation

As it can easily be seen from figure 2 that model (c) converges faster . And hence model(b) is more accurate than model (b). Convergence in both the models is less than but close to 1.

2.3 Defining function to calculate v

The Defined function is of the for as required in the problem statement

$$calcnu(x, x0, color, eps, model)$$

- x : input vector from 0 to 20 (any step size)
- xo : limit of x
- color : Color for plotting
- eps : noise value
- model : model(b) or (c)

Noise : We will add some noise to M3 matrix that is the matrix of functional value. Noise is added using python random.randn which returns sample form standard normal distribution.

```
def calcnu(x,xo,color,eps,model):
        if model == "b" :
                v=[]
                for i in range(0,(len(x)/20)*xo+1):
                        x1 = x[i:len(x)]
                        print i
                        M1_1 = np.transpose(sincos(x1))
                        M3_1 = J(x1,1)
                        coeff = linalg.lstsq(M1_1,M3_1+eps*np.random.randn(len(x1)))[0
                        #adding noise to the functional values
                        phi = coeff[0]/np.sqrt(coeff[0]**2+coeff[1]**2)
                        #computing phase and than obtaining v
                        v.append(((2/np.pi)*(np.arccos(phi)-np.pi/4)))
        if model == "c" :
                v=[]
                for i in range(0,(len(x)/20)*xo+1):
                        x1 = x[i:len(x)]
                        M1_2 = np.transpose(sincos2(x1))
                        M3_2 = J(x1,1)
                        coeff2 = linalg.lstsq(M1_2,M3_2+eps*np.random.randn(len(x1)))
                        #adding noise to the functional values
                        phi_2 = coeff2[0]/np.sqrt(coeff2[0]**2+coeff2[1]**2)
                        #computing phase and than obtaining v
                        v.append((2/np.pi)*(np.arccos(phi_2)-np.pi/4))
        plt.plot(x[0:(len(x)/20)*xo+1],v,color)
```

2.4 Plotting using calcnu function

Plotting three different plots. First plot is of model (b) with no noise , second plot of model (c) with no noise and third model with noise of O.O1 on model(c). See Figure 3.

```
calcnu(x,18,"go",0,"b")
calcnu(x,18,"bo",0,"c")
calcnu(x,18,"ro",0.01,"c")
plt.grid(True)
plt.legend(["e = 0, model (b)","e = 0, model (c)","e = 1.0e-02, model (c)"])
plt.show()
```

2.4.1 Observation from Figure 3

It can be seen that model(c) with noise has more variance . But mean still seems to be centered around the old convergence value. Adding noise increases error in out estimation of v for large value of x_o which earlier was more accurate.

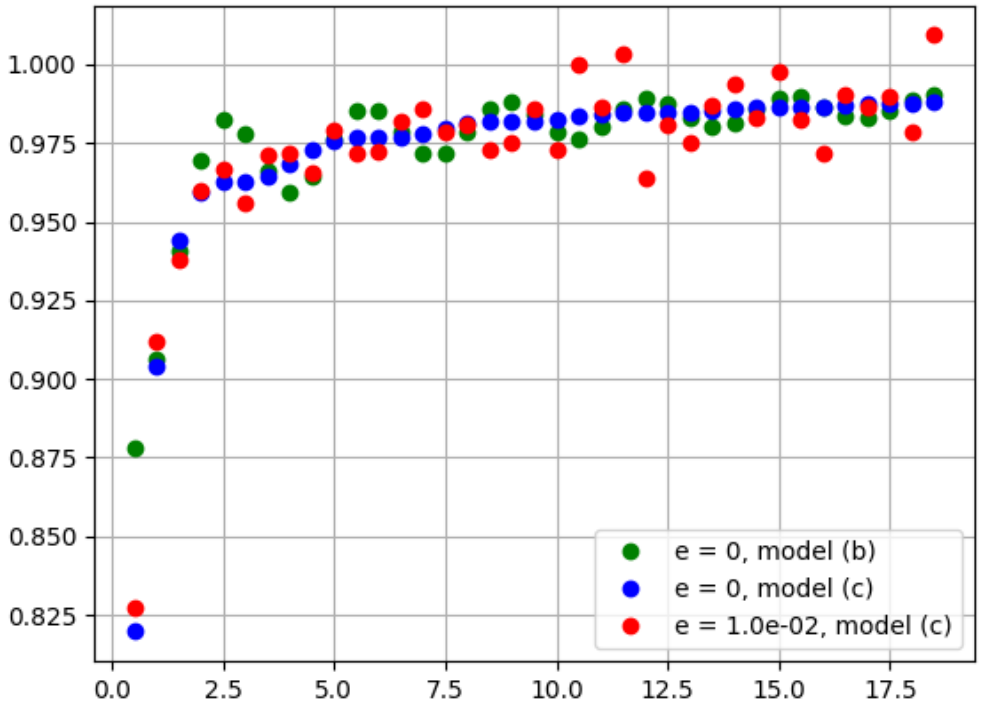

Figure 3: Plotting for v using model (b) with and without noise ans (c)

2.5 Effect on Varying measurements

The effect of number of measurements is not observed for no noise.In case of noise (See Figure 4 for model(b) and Figure 5 for model(c)) When the the number of measurements are increased and the same amount if noise is added , since our calculation has value of bessel function at more number of point effect of noise is reduced. Clearly from the plot the red triangles are more distorted with are calculated taking 40 points then blue crosses which were evaluated taking 1000 points.This result is consistent in both the models because it does not depend on the model but is a general property.

```
x_2 = np.linspace(0.5,20,40)
x_3 = np.linspace(0.5,20,1000)


calcnu(x_2,18,"r^",0.005,"b")
calcnu(x_3,18,"b+",0.005,"b")

plt.title("Variation in (b) model with number of measurements with noise")
plt.grid(True)
plt.legend(["with 40 measurements","with 1000 measurements"])
plt.show()

calcnu(x_2,18,"r^",0.005,"c")
calcnu(x_3,18,"b+",0.005,"c")

plt.title("Variation in (c) model with number of measurements with noise")
plt.grid(True)
plt.legend(["with 40 measurements","with 1000 measurements"])
```

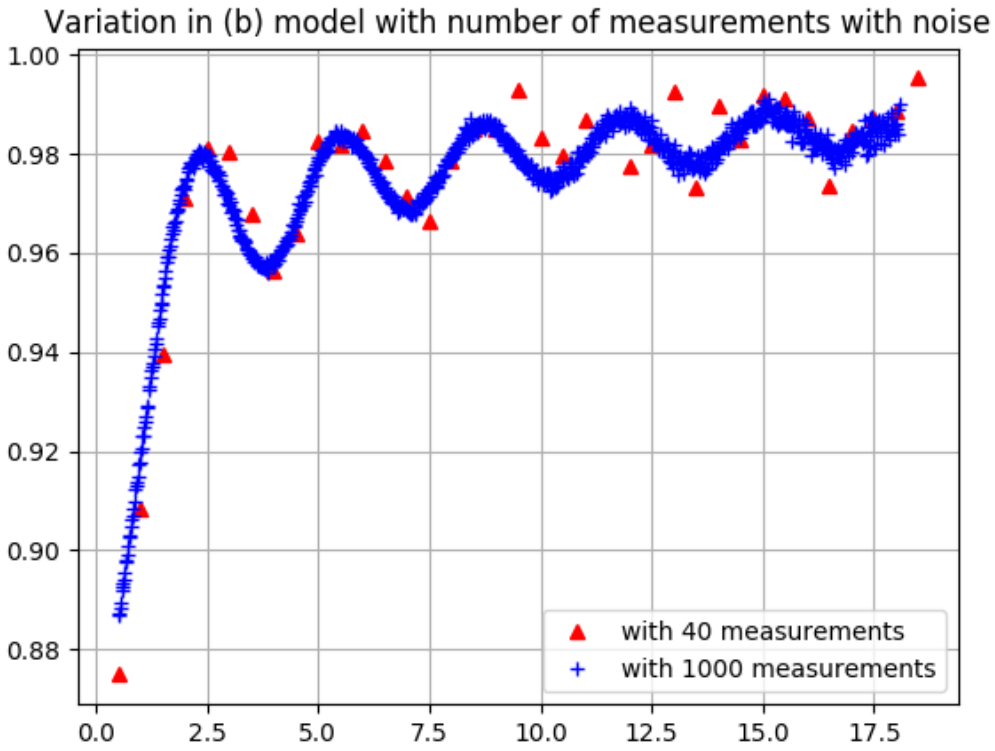

Figure 4: Variation in (b) model with number of measurements with noise

```
plt.show()
```

2.6 Effect of varying noise

Plotting both the models with different amount of noise 0 , 0.001, 0.003 and 0.005. Similar results are obtained in both the models. For Higher x_o values estimated v starts blowing up even for low noise content. For 0.01 nature of the curve is distinguishable and for 0.05 noise dominates. Quality of the fit reduces as the level of noise increases. See Figure 6 and Figure 7.

```
calcnu(x,18,"go",0,"b")
calcnu(x,18,"b^",0.05,"b")
calcnu(x,18,"c+",0.03,"b")
calcnu(x,18,"rx",0.01,"b")
plt.grid(True)
plt.title("Variation in (b) model with noice")
plt.legend(["e = 0, model (b)","e = 0.05, model (b)","e = 0.03, model (b)","e = 0.01,
plt.show()

calcnu(x,18,"go",0,"c")
calcnu(x,18,"b^",0.05,"c")
calcnu(x,18,"c+",0.03,"c")
calcnu(x,18,"rx",0.01,"c")
plt.grid(True)
plt.title("Variation in (c) model with noise")
plt.legend(["e = 0, model (c)","e = 0.05, model (c)","e = 0.03, model (c)","e = 0.01,
plt.show()
```

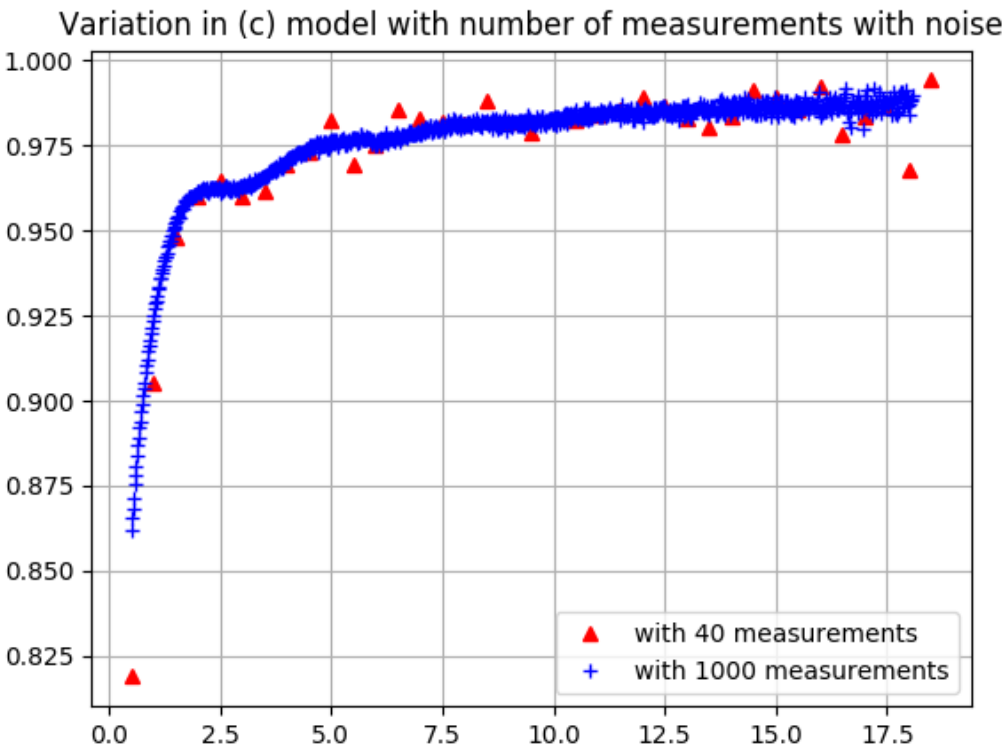

Figure 5: Variation in (c) model with number of measurements with noise

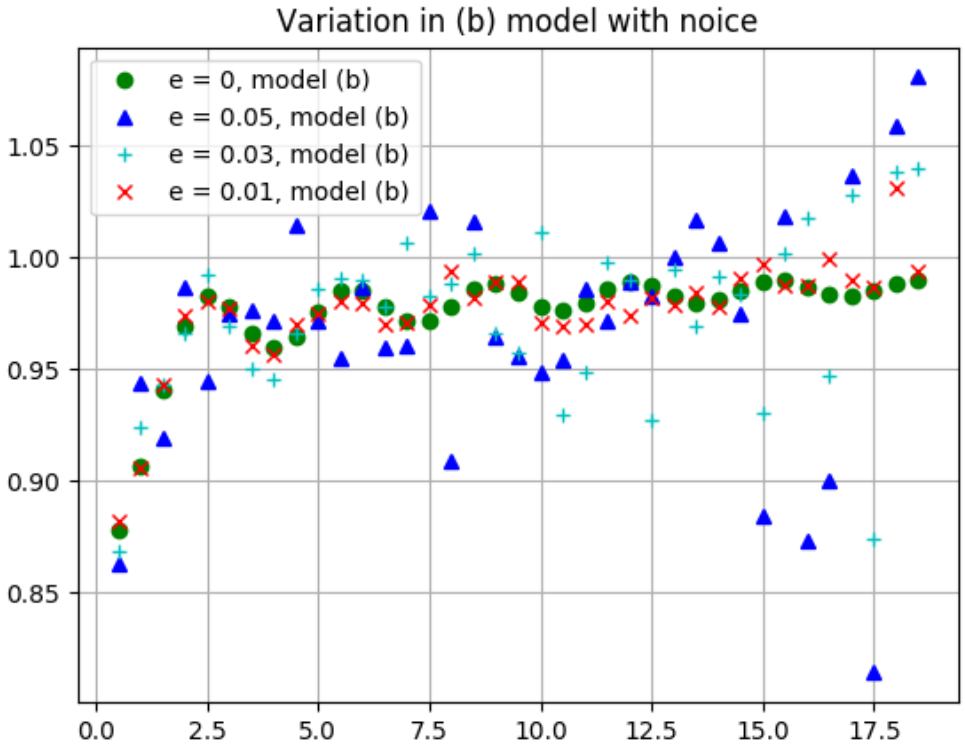

Figure 6: Variation in (B) model with noise

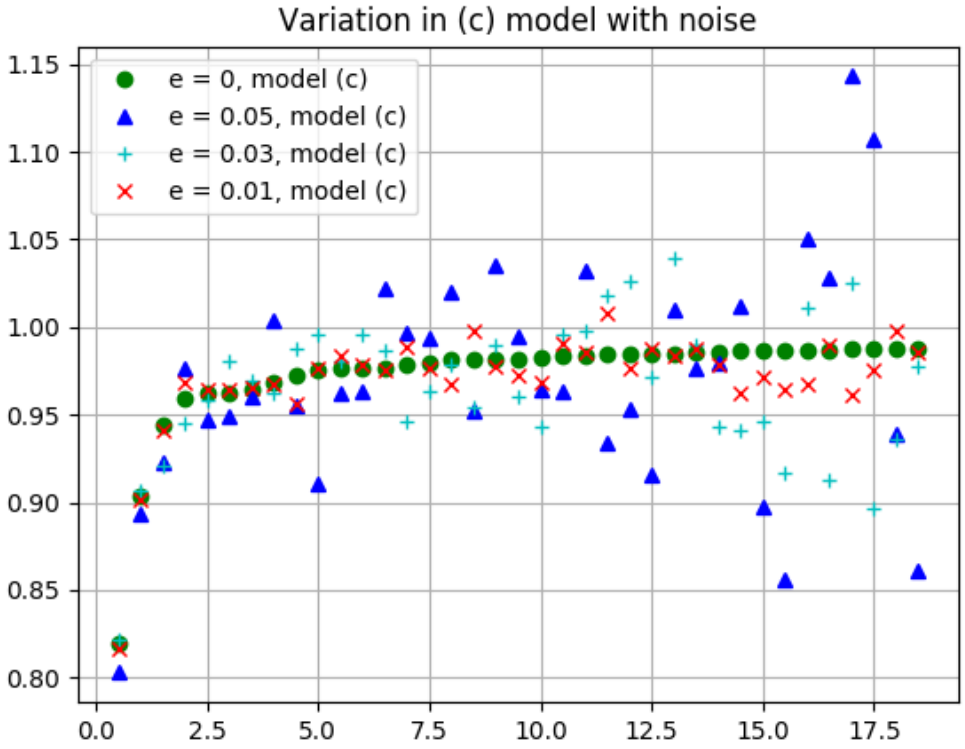

Figure 7: Variation in (C) model with noise

2.7 Quality of the Fit

To have good quality of fit from a model Increase in step-size can compensate for noise. Figure - 8 and Figure - 9 show model (b) and model (c) with varying noise just like Figure -6 and Figure -7. Significantly better result is obtained even for higher noise values .

```
calcnu(x_3,18,"go",0,"b")
calcnu(x_3,18,"b^",0.05,"b")
calcnu(x_3,18,"c+",0.03,"b")
calcnu(x_3,18,"rx",0.01,"b")
plt.grid(True)
plt.title("Variation in (b) model with noise 1000 points")
plt.legend(["e = 0, model (b)","e = 0.05, model (b)","e = 0.03, model (b)","e = 0.01,
plt.show()

calcnu(x_3,18,"go",0,"c")
calcnu(x_3,18,"b^",0.05,"c")
calcnu(x_3,18,"c+",0.03,"c")
calcnu(x_3,18,"rx",0.01,"c")
plt.grid(True)
plt.title("Variation in (c) model with noise 1000 points")
plt.legend(["e = 0, model (c)","e = 0.05, model (c)","e = 0.03, model (c)","e = 0.01,
plt.show()
```

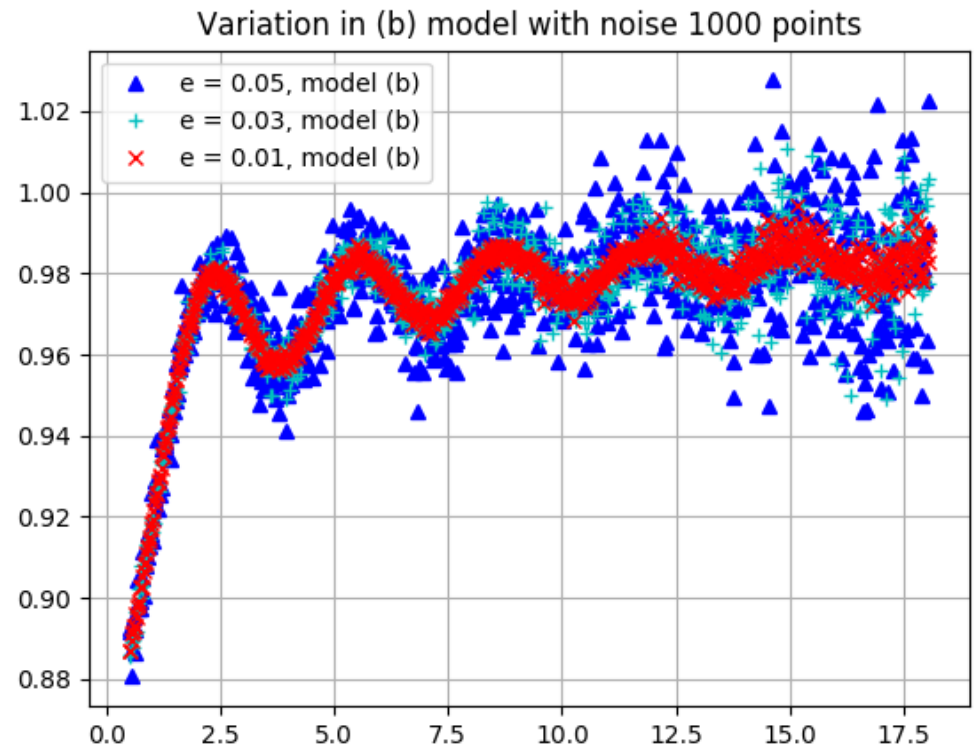

Figure 8: Variation in (b) model with noise and step-size

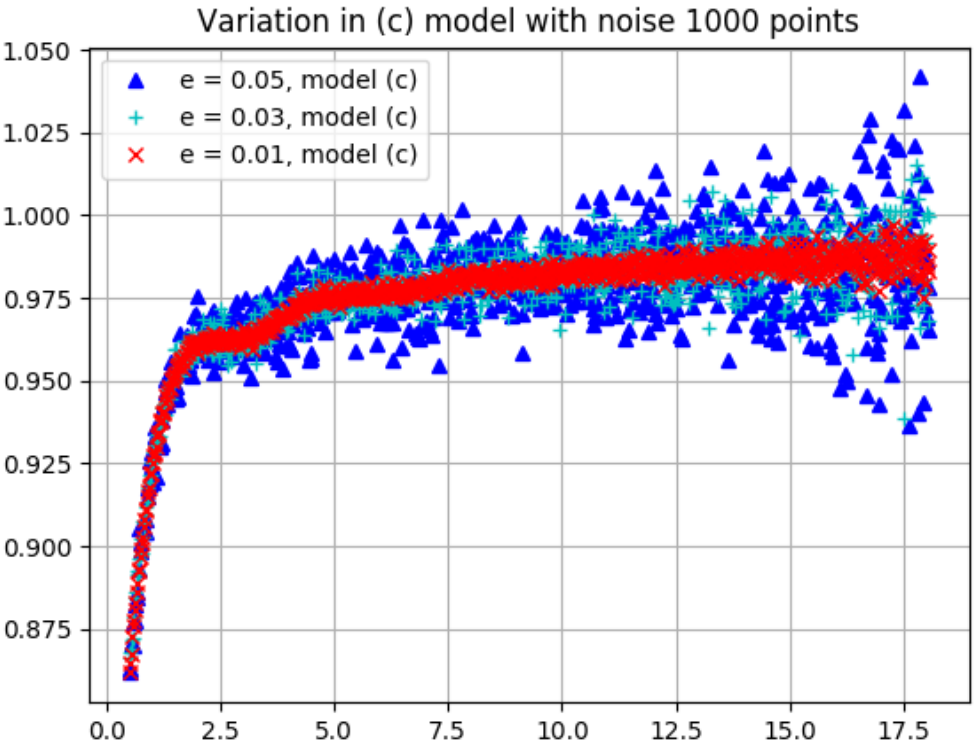

Figure 9: Variation in (C) model with noise and step-size