#Question-1

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import optimize, stats
import csv

#read the excel file
global data, x, y, sigma_y, val1, val2, val3
data = pd.read_excel("C:\\Users\\Heera Baiju\\Downloads\\Data
Science\\Assignment 4\\D4data.xlsx")
x = data['x']
y = data['y']
sigma_y = data['sigma_y']

#initialise theta for all the fits to zero
val1 = np.array([0, 0])
val2 = np.array([0, 0, 0])
val3 = np.array([0, 0, 0, 0])
def linearFunc(x, val1):
    return val1[1]*x+val1[0]

def quadraticFunc(x, val2):
    return val2[2]*x**2 + val2[1]*x + val2[0]

def cubicFunc(x, val3):
    return val3[3]*x**3 + val3[2]*x**2 + val3[1]*x + val3[0]

#The log-likelihood function is maximum for best fitting value of theta
def logL(theta, n):
    if n==1:
        y_fit = linearFunc(x, theta)
    elif n==2:
        y_fit = quadraticFunc(x, theta)
    elif n==3:
        y_fit = cubicFunc(x, theta)
    return sum(stats.norm.logpdf(*args)
               for args in zip(y, y_fit, sigma_y))
#This function returns the best theta for the fitting
def best_theta(n, theta_val):
    if n==1:
        theta_0 = (n+1)*[0]
        neg_logL = lambda theta: -logL(theta, 1)
        return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)
    if n==2:
        theta_0 = (n+1)*[0]
```

```
        neg_logL = lambda theta: -logL(theta, 2)
        return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)
    if n==3:
        theta_0 = (n+1)*[0]
        neg_logL = lambda theta: -logL(theta, 3)
        return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)

#compute chi2 likelihood for frequentist
def compute_chi2(n):
    if n==1:
        theta = best_theta(n, val1)
        resid = ((y - linearFunc(x, theta)) / sigma_y)
    elif n==2:
        theta = best_theta(n, val2)
        resid = ((y - quadraticFunc(x, theta)) / sigma_y)
    elif n==3:
        theta = best_theta(n, val3)
        resid = ((y - cubicFunc(x, theta)) / sigma_y)

    return np.sum(resid ** 2)

def compute_dof(degree, data=data):
    return data.shape[0] - (degree + 1)

def chi2_likelihood(n):
    chi2 = compute_chi2(n)
    dof = compute_dof(n)
    return stats.chi2(dof).pdf(chi2)
#Compute the p value for the fit using linear model as the null hypothesis
def p_val(n):
    return 1-stats.chi2(n-1).cdf(compute_chi2(1) - compute_chi2(n))
#Compute the optimized values of the parameters
r1 = best_theta(1, val1)
r2 = best_theta(2, val2)
r3 = best_theta(3, val3)
print("Log L values")
print("linear model:    logL =", logL(best_theta(1, val1), 1))
print("quadratic model: logL =", logL(best_theta(2, val2), 2))
print("cubic model:    logL =", logL(best_theta(3, val3), 3))

print("chi2 likelihood")
print("- linear model:    ", chi2_likelihood(1))
print("- quadratic model: ", chi2_likelihood(2))
print("- cubic model: ", chi2_likelihood(3))

print("p_values") #The p value for null hypothesis will not be defined as the
delta chi square value is zero
print("- quadratic model: ", p_val(2))
```

```
print("- cubic model: ", p_val(3))


#Compute the AICc values as number of data points is considerably small
AIC1 = -2*logL(r1, 1) + (2.0*2*20)/(17.0)
AIC2 = -2*logL(r2, 2) + (2.0*3*20)/(16.0)
AIC3 = -2*logL(r3, 3) + (2.0*4*20)/(15.0)

#Compute the BIC values
BIC1 = -2*logL(r1, 1) + 2*np.log(x.shape[0])
BIC2 = -2*logL(r2, 2) + 3*np.log(x.shape[0])
BIC3 = -2*logL(r3, 3) + 4*np.log(x.shape[0])
print("AIC values")
print("- linear model:    ", AIC1)
print("- quadratic model: ", AIC2)
print("- cubic model: ", AIC3)

#Delta AIC
AIC_min = min(AIC1, AIC2, AIC3)
print("Delta AIC values")
print("- linear model:    ", AIC1-AIC_min)
print("- quadratic model: ", AIC2-AIC_min)
print("- cubic model: ", AIC3-AIC_min)


print("BIC values")
print("- linear model:    ", BIC1)
print("- quadratic model: ", BIC2)
print("- cubic model: ", BIC3)

#Delta BIC
BIC_min = min(BIC1, BIC2, BIC3)
print("Delta BIC values")
print("- linear model:    ", BIC1-BIC_min)
print("- quadratic model: ", BIC2-BIC_min)
print("- cubic model: ", BIC3-BIC_min)


t =  np.linspace(0, 1, 1000)
fig, ax = plt.subplots(figsize=(10, 10))
plt.plot(t, linearFunc(t, r1), label='linear_fitting')
plt.plot(t, quadraticFunc(t, r2), label='quadratic_fitting')
plt.plot(t, cubicFunc(t, r3), label='cubic_fitting')
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title("Curve fitting using Linear, Quadratic and Cubic Models", size=15)
ax.errorbar(x, y, sigma_y, fmt='ok', ecolor = 'gray')
plt.show()
```

OUTPUT

Log L values

linear model: logL = 22.01834340803626

quadratic model: logL = 22.924910312002666

cubic model: logL = 23.130409258798014

chi2 likelihood

- linear model: 0.04538379558592013
- quadratic model: 0.036608447550143294
- cubic model: 0.04215280601013452

p_values

- quadratic model: 0.17813275695318243
- cubic model: 0.3288788441962769

AIC values

- linear model: -39.330804463131344
- quadratic model: -38.34982062400533
- cubic model: -35.594151850929364

Delta AIC values

- linear model: 0.0
- quadratic model: 0.9809838391260115
- cubic model: 3.736652612201979

BIC values

- linear model: -38.04522226896454
- quadratic model: -36.86262380334336
- cubic model: -34.27788942338007

Delta BIC values

- linear model: 0.0
- quadratic model: 1.1825984656211759

- cubic model: 3.7673328455844697

Comments and inferences

The linear model is considered as the null hypothesis
From the p values of frequentist analysis, as both the p values are greater than 0.05, we cannot reject our null hypothesis.
For quadratic model, 0 < delta AIC < 2, implies that quadratic model has a substantial support.
For cubic model, 2 < delta AIC < 4, implies that cubic model has considerably less support.
For quadratic model, 0 < delta BIC < 2, implies that there is no evidence against quadratic model.
For cubic model, 2 < delta AIC < 4, implies that there is positive evidence against cubic model.

#Question-2

```
import numpy as np
from scipy import optimize, stats
import matplotlib.pyplot as plt
global data, x, y, sigma_y


data = np.array([[ 0.42,  0.72,  0.  ,  0.3 ,  0.15,
                   0.09,  0.19,  0.35,  0.4 ,  0.54,
                   0.42,  0.69,  0.2 ,  0.88,  0.03,
                   0.67,  0.42,  0.56,  0.14,  0.2  ],
                 [ 0.33,  0.41, -0.22,  0.01, -0.05,
                  -0.05, -0.12,  0.26,  0.29,  0.39,
                   0.31,  0.42, -0.01,  0.58, -0.2 ,
                   0.52,  0.15,  0.32, -0.13, -0.09 ],
                 [ 0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1  ]])
x,y,sigma_y = data

def p_fit(theta, x):


    return sum(a * x ** n for (n, a) in enumerate(theta))


def logL(theta):

    y_fit = p_fit(theta, x)
    return sum(stats.norm.logpdf(*args)
for args in zip(y, y_fit, sigma_y))

def best_theta(deg):
    theta_0 = (deg + 1) * [0]
    neg_logL = lambda theta: -logL(theta)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)
theta1 = best_theta(1)
theta2 = best_theta(2)

AIC1 = -2*logL(theta1) + (2.0*2*20)/(17.0)
AIC2 = -2*logL(theta2) + (2.0*3*20)/(16.0)

BIC1 = -2*logL(theta1) + 2*np.log(x.shape[0])
BIC2 = -2*logL(theta2) + 3*np.log(x.shape[0])
```

```
print("AIC values")
print(" Linear : ", AIC1)
print(" Quadratic : ", AIC2)

print("BIC values")
print(" Linear : ", BIC1)
print(" Quadratic : ", BIC2)

def compute_chi2(deg, data = data):
    x, y, sigma_y = data
    theta = best_theta(deg)
    resid = (y - p_fit(theta, x)) / sigma_y
    return np.sum(resid ** 2)

def compute_dof(deg, data = data):
    return data.shape[1] - (deg + 1)

def chi2_likelihood(deg, data = data):
    chi2 = compute_chi2(deg, data)
    dof = compute_dof(deg, data)
    return stats.chi2(dof).pdf(chi2)
# Print the chi2 likelihood
print("chi2 likelihood")
print(" Linear : ", chi2_likelihood(1))
print(" Quadratic : ", chi2_likelihood(2))
# Computing delta AIC
AIC_min = min(AIC1, AIC2)
print("Delta AIC values")
print(" Linear : ", AIC1-AIC_min)
print(" Quadratic : ", AIC2-AIC_min)
# Computing delta BIC
BIC_min = min(BIC1, BIC2)
print("Delta BIC values")
print(" Linear : ", BIC1-BIC_min)
print(" Quadratic : ", BIC2-BIC_min)
# Plotting
fig, ax = plt.subplots()
for deg, color in zip([1, 2], ['blue', 'red']):
    v = np.linspace(0, 40, 1000)
    chi2_dist = stats.chi2(compute_dof(deg)).pdf(v)
    chi2_val = compute_chi2(deg)
    chi2_like = chi2_likelihood(deg)
    ax.fill(v, chi2_dist, alpha=0.3, color = color, label = 'Model {0} (deg =
{0})'.format(deg))
    ax.vlines(chi2_val, 0, chi2_like, color = color, alpha = 0.6)
    ax.hlines(chi2_like, 0, chi2_val, color = color, alpha = 0.6)
    ax.set(ylabel='L(chi-square)')
    ax.set_xlabel('chi-square')
```

```
ax.legend(fontsize=14)
plt.show()
```

OUTPUT

AIC values

Linear : -39.315851660283926

Quadratic : -38.383027173007996

BIC values

Linear : -38.03026946611712

Quadratic : -36.895830352346024

chi2 likelihood

Linear : 0.04552443406372872

Quadratic : 0.03625617489379636

Delta AIC values

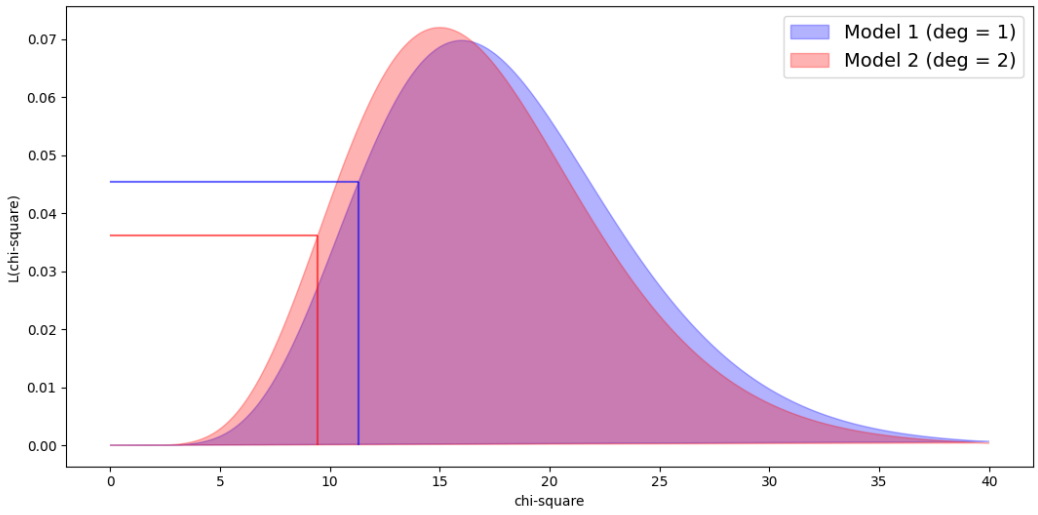
Linear : 0.0

Quadratic : 0.9328244872759299

Delta BIC values

Linear : 0.0

Quadratic : 1.1344391137710943



#Question-3

Link to the paper in question

http://ieeexplore.ieee.org/document/5666474/

(Fast and Robust Spectrum Sensing via Kolmogorov-Smirnov Test)

The K-S test is a non-parametric method to measure the goodness of fit. The basic procedure involves computing the empirical cumulative distribution function (ECDF) of some decision statistic obtained from the received signal and comparing it with the ECDF of the channel noise samples. A sequential version of the K-S-based spectrum sensing technique is also proposed.

In the paper, a sequence of noise-only samples is used to create the null hypothesis distribution F0. A two-sample K-S test from the received signal samples is performed and the null hypothesis test is either accepted or rejected.

But in this paper, a 2D KS test is used due to complex signals.

The following steps are then involved in the K-S-based spectrum sensing:

The M0 noise-only sample vectors and the related amplitude or quadrature statistics are considered. And the empirical 1D or 2D noise cdf $F0$ is calculated. The M received signal vectors is collected to form amplitude or quadrature statistics and calculates the maximum difference D. Presence or absence of primary users is derived.

But another paper disapproves the use of the Kolmogorov-Smirnov-test in more than 1 Dimensional data.

https://asaip.psu.edu/Articles/beware-of-the-kolmogorov-smirnov-test

#Question-4

```
import scipy
from scipy import stats
significance_Higgs =scipy.stats.norm.isf(1.7e-9)

#Higgs boson
print('The significance in terms of number of sigmas of the Higgs boson
discovery claim from the p value given in the abstract of the ATLAS discovery
paper, ',significance_Higgs)

#Ligo
significance_Ligo =scipy.stats.norm.isf(2e-7)
print('Significance in terms of number of sigmas LIGO',significance_Ligo)

#Goodness of fit
p_value=1-scipy.stats.chi2(67).cdf(65.2)
print('GOF using the best-fit',p_value)
```

OUTPUT

The significance in terms of number of sigmas of the Higgs boson discovery claim from the p value given in the abstract of the ATLAS discovery paper, 5.911017938341624

Significance in terms of number of sigmas LIGO 5.068957749717791

GOF using the best-fit 0.5394901931099038