Assignment-4

Q1

```
#-------------------------------------------------------------
# Importing all libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import optimize, stats

global data, x, y, sigma_y

#-------------------------------------------------------------
# Define the data array
data = np.array([[ 0.417022004703, 0.720324493442, 0.000114374817345,
0.302332572632,
                   0.146755890817, 0.0923385947688, 0.186260211378,
0.345560727043,
                   0.396767474231, 0.538816734003, 0.419194514403,
0.685219500397,
                   0.204452249732, 0.878117436391, 0.0273875931979,
0.670467510178,
                   0.417304802367, 0.558689828446, 0.140386938595, 0.198101489085
],
                 [ 0.121328306045, 0.849527236006, -1.01701405804, -
0.391715712054,
                  -0.680729552205, -0.748514873007, -0.702848628623, -
0.0749939588554,
                   0.041118449128, 0.418206374739, 0.104198664639, 0.7715919786,
                  -0.561583800669, 1.43374816145, -0.971263541306,
0.843497249235,
                  -0.0604131723596, 0.389838628615, -0.768234900293, -
0.649073386002 ],
                 [ 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
```

```python
#-------------------------------------------------------------
# Defining the required functions for polynomial fitting
def polynomial_fit(theta, x):
    # Polynomial model of degree (len(theta) - 1)
    return sum(t * x ** n for (n, t) in enumerate(theta))

#-------------------------------------------------------------
# Defining the required logL function
def logL(theta):
    # Gaussian log-likelihood of the model at theta
    y_fit = polynomial_fit(theta, x)
    return sum(stats.norm.logpdf(*args)
               for args in zip(y, y_fit, sigma_y))

#-------------------------------------------------------------
# Defining the function that returns the best theta for the fitting
def best_theta(degree):
    theta_0 = (degree + 1) * [0]
    neg_logL = lambda theta: -logL(theta)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)

#-------------------------------------------------------------
# Defining the function that computes chi2
def compute_chi2(degree, data = data):
    x, y, sigma_y = data
    theta = best_theta(degree)
    resid = (y - polynomial_fit(theta, x)) / sigma_y
    return np.sum(resid ** 2)

#-------------------------------------------------------------
# Defining the function that computes DOF
def compute_dof(degree, data = data):
    return data.shape[1] - (degree + 1)

#-------------------------------------------------------------
# Defining the function that computes chi2 likelihood
def chi2_likelihood(degree, data = data):
    chi2 = compute_chi2(degree, data)
    dof = compute_dof(degree, data)
    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))
```

```
theta1 = best_theta(1)
theta2 = best_theta(2)
theta3 = best_theta(3)

# Print the Log L values
print("Log L values")
print("  Linear model:     logL = ", logL(best_theta(1)))
print("  Quadratic model:  logL = ", logL(best_theta(2)))
print("  Cubic model:      logL = ", logL(best_theta(3)))
print("  Chi2 likelihood")

# Print the chi2 likelihood
print("Chi2 likelihood")
print("  Linear model:    ", chi2_likelihood(1))
print("  Quadratic model: ", chi2_likelihood(2))
print("  Cubic model:     ", chi2_likelihood(3))

# Print the p values
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))
print('From the p values of frequentist analysis, as both the p values are
greater than 0.05, we cannot reject our Null Hypothesis.')

#-------------------------------------------------------------
# Bayesian Analysis

# Compute the AIC values
AIC1 = -2*logL(theta1) + (2.0*2*20)/(17.0)
AIC2 = -2*logL(theta2) + (2.0*3*20)/(16.0)
AIC3 = -2*logL(theta3) + (2.0*4*20)/(15.0)

# Compute the BIC values
BIC1 = -2*logL(theta1) + 2*np.log(x.shape[0])
BIC2 = -2*logL(theta2) + 3*np.log(x.shape[0])
BIC3 = -2*logL(theta3) + 4*np.log(x.shape[0])

# Print the AIC values
print("AIC values")
print("  Linear model:    ", AIC1)
print("  Quadratic model: ", AIC2)
print("  Cubic model:     ", AIC3)

# Computing 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('-> For quadratic model, 0 < delta AIC < 2, which implies that quadratic
model also has a substantial support.')
print('-> For cubic model, 2 < delta AIC < 4, which implies that cubic model has
considerably less support.')

# Print the BIC values
print("BIC values")
print("  Linear model:    ", BIC1)
print("  Quadratic model: ", BIC2)
print("  Cubic model:     ", BIC3)

# Computing 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)

print('-> For quadratic model, 0 < delta BIC < 2, which implies that there is no
evidence against quadratic model.')
print('-> For cubic model, 2 < delta AIC < 4, which implies that there is
positive evidence against cubic model.')

# Plotting
t =  np.linspace(0, 1, 1000)
fig, ax = plt.subplots(figsize=(10, 10))
plt.plot(t, polynomial_fit(theta1, t), label = r'$Linear\ fitting$')
plt.plot(t, polynomial_fit(theta2, t), label = r'$Quadratic\ fitting$')
plt.plot(t, polynomial_fit(theta3, t), label = r'$Cubic\ fitting$')
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 = 22.92491031200274

Cubic model: logL = 23.130409258797624

Chi2 likelihood

Chi2 likelihood

Linear model: 0.04538379558592013

Quadratic model: 0.03660844755014173

Cubic model: 0.042152806010143505

p_values

Quadratic model: 0.178132756953164

Cubic model: 0.32887884419640656

From the p values of frequentist analysis, as both the p values are greater than 0.05, we cannot reject our Null Hypothesis.

AIC values

Linear model: -39.330804463131344

Quadratic model: -38.34982062400548

Cubic model: -35.59415185092858

Delta AIC values

Linear model: 0.0

Quadratic model: 0.9809838391258623

Cubic model: 3.736652612202761

-> For quadratic model, 0 < delta AIC < 2, which implies that quadratic model also has a substantial support.

-> For cubic model, 2 < delta AIC < 4, which implies that cubic model has considerably less support.

BIC values

Linear model: -38.04522226896454

Quadratic model: -36.86262380334351

Cubic model: -34.277889423379285

Delta BIC values

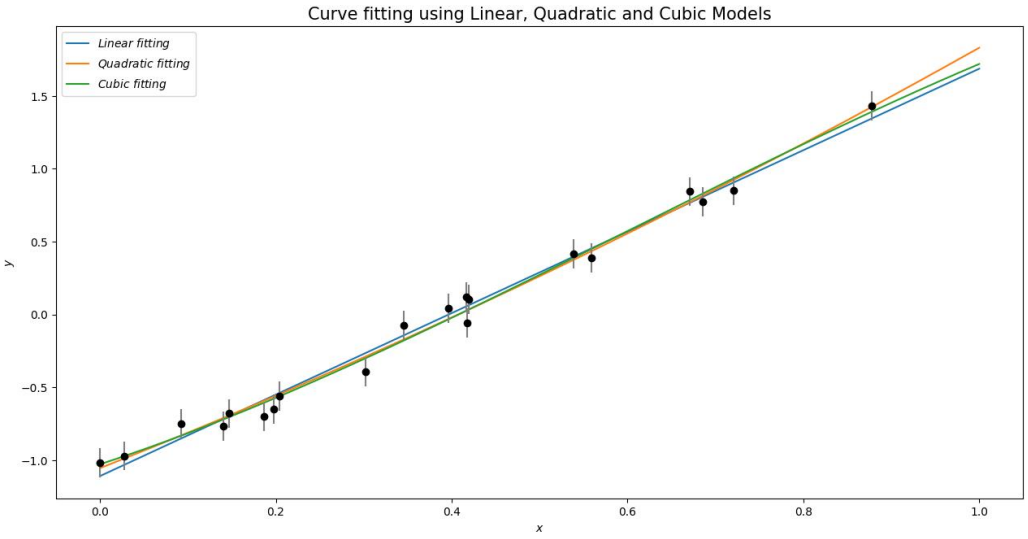
Linear model: 0.0

Quadratic model: 1.1825984656210267

Cubic model: 3.7673328455852513

-> For quadratic model, 0 < delta BIC < 2, which implies that there is no evidence against quadratic model.

-> For cubic model, 2 < delta AIC < 4, which implies that there is positive evidence against cubic model.



Q2

```
#-------------------------------------------------------------
# Importing all libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize, stats

global data, x, y, sigma_y

#-------------------------------------------------------------
# Define the data array
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

#-------------------------------------------------------------
# Defining the required functions for polynomial fitting
def polynomial_fit(theta, x):
    # Polynomial model of degree (len(theta) - 1)
    return sum(t * x ** n for (n, t) in enumerate(theta))

#-------------------------------------------------------------
# Defining the required logL function
def logL(theta):
    # Gaussian log-likelihood of the model at theta
    y_fit = polynomial_fit(theta, x)
    return sum(stats.norm.logpdf(*args)
               for args in zip(y, y_fit, sigma_y))

#-------------------------------------------------------------
# Defining the function that returns the best theta for the fitting
```

```
def best_theta(degree):
    theta_0 = (degree + 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)

#-------------------------------------------------------------
# Bayesian Analysis

# Compute the AIC values
AIC1 = -2*logL(theta1) + (2.0*2*20)/(17.0)
AIC2 = -2*logL(theta2) + (2.0*3*20)/(16.0)

# Compute the BIC values
BIC1 = -2*logL(theta1) + 2*np.log(x.shape[0])
BIC2 = -2*logL(theta2) + 3*np.log(x.shape[0])

# Print the AIC values
print("AIC values")
print("  Linear model:    ", AIC1)
print("  Quadratic model: ", AIC2)

# Print the BIC values
print("BIC values")
print("  Linear model:    ", BIC1)
print("  Quadratic model: ", BIC2)

#-------------------------------------------------------------
# Defining the function that computes chi2
def compute_chi2(degree, data = data):
    x, y, sigma_y = data
    theta = best_theta(degree)
    resid = (y - polynomial_fit(theta, x)) / sigma_y
    return np.sum(resid ** 2)

#-------------------------------------------------------------
# Defining the function that computes DOF
def compute_dof(degree, data = data):
    return data.shape[1] - (degree + 1)

#-------------------------------------------------------------
# Defining the function that computes chi2 likelihood
def chi2_likelihood(degree, data = data):
```

```
    chi2 = compute_chi2(degree, data)
    dof = compute_dof(degree, data)
    return stats.chi2(dof).pdf(chi2)

# Print the chi2 likelihood
print("chi2 likelihood")
print("  Linear model:    ", chi2_likelihood(1))
print("  Quadratic model: ", chi2_likelihood(2))

# Computing delta AIC
AIC_min = min(AIC1, AIC2)
print("Delta AIC values")
print("  Linear model:    ", AIC1-AIC_min)
print("  Quadratic model: ", AIC2-AIC_min)
print('-> For linear model, 0 < delta AIC < 2, which implies that quadratic model
also has a substantial support.')
print('-> For quadratic model, 0 < delta AIC < 2, which implies that quadratic
model also has a substantial support.')

# Computing delta BIC
BIC_min = min(BIC1, BIC2)
print("Delta BIC values")
print("  Linear model:    ", BIC1-BIC_min)
print("  Quadratic model: ", BIC2-BIC_min)
print('-> For linear model, 0 < delta BIC < 2, which implies that there is no
evidence against quadratic model.')
print('-> For quadratic model, 0 < delta BIC < 2, which implies that there is no
evidence against quadratic model.')

# Plotting
fig, ax = plt.subplots()
for degree, color in zip([1, 2], ['green', 'red']):
    v = np.linspace(0, 40, 1000)
    chi2_dist = stats.chi2(compute_dof(degree)).pdf(v)
    chi2_val = compute_chi2(degree)
    chi2_like = chi2_likelihood(degree)
    ax.fill(v, chi2_dist, alpha=0.3, color = color, label = 'Model {0} (degree =
{0})'.format(degree))
    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()
```

```
print('-> We can see visually here how this procedure corrects for model complexity: even though the chi2 value for the quadratic model is lower (shown by the vertical lines), the characteristics of the chi2 distribution mean the likelihood of seeing this value is lower (shown by the horizontal lines), meaning that the degree=1 linear model is favored.')
```

Output

AIC values

Linear model: -39.31585166028391

Quadratic model: -38.383027173007946

BIC values

Linear model: -38.030269466117105

Quadratic model: -36.895830352345975

chi2 likelihood

Linear model: 0.04552443406372872

Quadratic model: 0.03625617489379681

Delta AIC values

Linear model: 0.0

Quadratic model: 0.9328244872759655

-> For linear model, 0 < delta AIC < 2, which implies that quadratic model also has a substantial support.

-> For quadratic model, 0 < delta AIC < 2, which implies that quadratic model also has a substantial support.

Delta BIC values

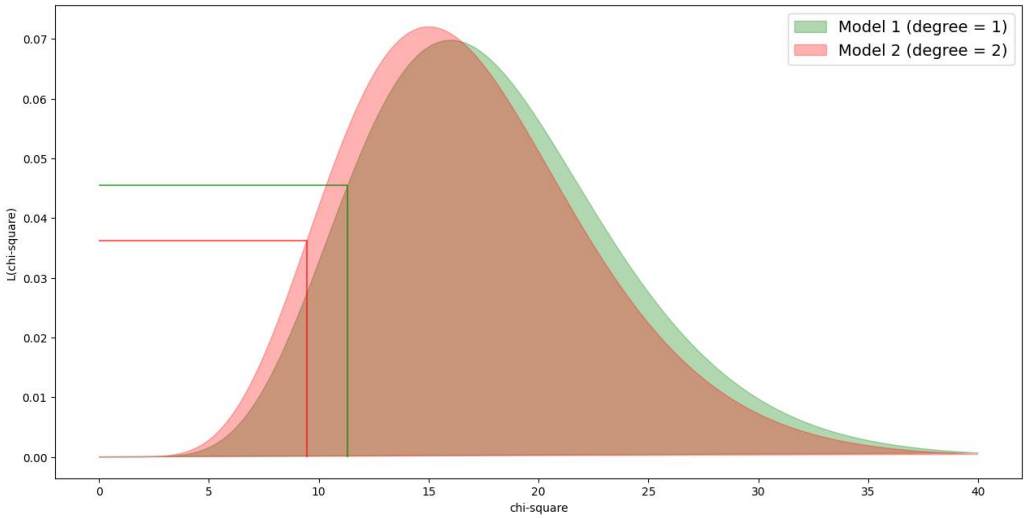
Linear model: 0.0

Quadratic model: 1.1344391137711298

-> For linear model, 0 < delta BIC < 2, which implies that there is no evidence against quadratic model.

-> For quadratic model, 0 < delta BIC < 2, which implies that there is no evidence against quadratic model.

-> We can see visually here how this procedure corrects for model complexity: even though the chi2 value for the quadratic model is lower (shown by the vertical lines), the characteristics of the chi2 distribution mean the likelihood of seeing this value is lower (shown by the horizontal lines), meaning that the degree=1 linear model is favored.


Model 1 (degree = 1)
Model 2 (degree = 2)
L(chi-square)
chi-square
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0
5
10
15
20
25
30
35
40

Q3

Paper on Fast and Robust Spectrum Sensing via Kolmogorov-Smirnov Test can be found using link

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

The method of periodically monitoring a specific frequency band in order to detect the presence or absence of primary users is known as spectrum sensing.

The Guowei Zhang; Xiaodong Wang; Ying-Chang Liang; Ju Liu (all authors) of this publication proposed an idea to use of a sequence of noise-only samples to create the null hypothesis distribution F0. The null hypothesis test is then accepted or rejected using the two-sample K-S test based on the received signal samples.

However, since this received signal is complex, a 2D KS test must be considered.

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

The noise statistics are gathered by the cognitive user by collecting M0 noise-only sample vectors and the related amplitude or quadrature statistics. It estimates the empirical 1D or 2D noise cdf $F0$. Spectrum sensing: The cognitive user collects M received signal vectors and form the corresponding amplitude or quadrature statistics. It then calculates the maximum difference D and declares the presence of the primary users based on that; otherwise, no primary users are present.

However, according to https://asaip.psu.edu/Articles/beware-the-kolmogorov-smirnov-test/, the kolmogorov-smirnov test should not be applied to the data with more than 1 Dimension, which is done in this paper.

Q4

```
#-------------------------------------------------------------
# Importing all libraries
import scipy
from scipy import stats

# Computing significance in terms of no of sigmas HIGGS
significance_Higgs = scipy.stats.norm.isf(1.7e-9)
print('Significance in terms of no of sigmas HIGGS: ',significance_Higgs)
# Which is matching with the value given in tha paper

# Computing significance in terms of no of sigmas LIGO
significance_Ligo = scipy.stats.norm.isf(2e-7)
print('Significance in terms of no of sigmas LIGO: ',significance_Ligo)

# Goodness of fit according to the best parameters taken from mentioned paper
p_value = 1-scipy.stats.chi2(67).cdf(65.2)
print('Goodness of fit using the best-fit: ',p_value)
```

Output

Significance in terms of no of sigmas HIGGS: 5.911017938341624

Significance in terms of no of sigmas LIGO: 5.068957749717791

Goodness of fit using the best-fit: 0.5394901931099038