Assignment 4

Pushkal Mishra
EE20BTECH11042

Importing Libraries

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import stats
from scipy.optimize import curve_fit
```

Question 1

[2]:
```
def calculate_linear(x, a0, a1):
    return a0 + a1 * x

def calculate_quadratic(x, a0, a1, a2):
    return a0 + (a1 * x) + a2 * (x ** 2)

def calculate_cubic(x, a0, a1, a2, a3):
    return a0 + (a1 * x) + a2 * (x ** 2) + a3 * (x ** 3)

def estimated_y(x, degree, param):
    estimate = 0
    for i in range(degree + 1):
        estimate += (param[i] * (x ** i))

    return estimate

def chi2_likelihood(data, degree, param):
    x, y, sigma_y = data[0], data[1], data[2]
    dof = data[0].shape[0] - degree - 1

    estimate = estimated_y(x, degree, param)
    errors = y - estimate
    chi2 = np.sum((errors / sigma_y) ** 2)

    return stats.chi2(dof).pdf(chi2)
```

```
def calculate_MLE(data, degree, param):
    x, y, sigma_y = data[0], data[1], data[2]

    estimate = estimated_y(x, degree, param)
    log_likelihood = np.sum(stats.norm.logpdf(y, estimate, sigma_y))

    return log_likelihood

def calculate_AIC(data, degree, param):
    log_likelihood = calculate_MLE(data, degree, param)
    k = degree + 1

    return 2 * (k - log_likelihood)

def calculate_AICc(data, degree, param):
    k = degree + 1
    N = len(data[0])

    AIC = calculate_AIC(data, degree, param)
    AICc = AIC + ((2 * k * (k + 1)) / (N - k - 1))

    return AICc

def calculate_BIC(data, degree, param):
    k = degree + 1
    N = len(data[0])

    log_likelihood = calculate_MLE(data, degree, param)
    BIC = k * np.log(N) - 2 * log_likelihood

    return BIC

def calculate_p_value_wrt_linear(data, degree, param, linear_param):
    x, y, sigma_y = data[0], data[1], data[2]
    dof = degree - 1

    linear_estimate = estimated_y(x, 1, linear_param)
    deg_estimate = estimated_y(x, degree, param)

    chi2_linear = np.sum(((y - linear_estimate) / sigma_y) ** 2)
    chi2_deg = np.sum(((y - deg_estimate) / sigma_y) ** 2)

    p_value = stats.chi2(dof).sf(chi2_linear - chi2_deg)

    return p_value
```

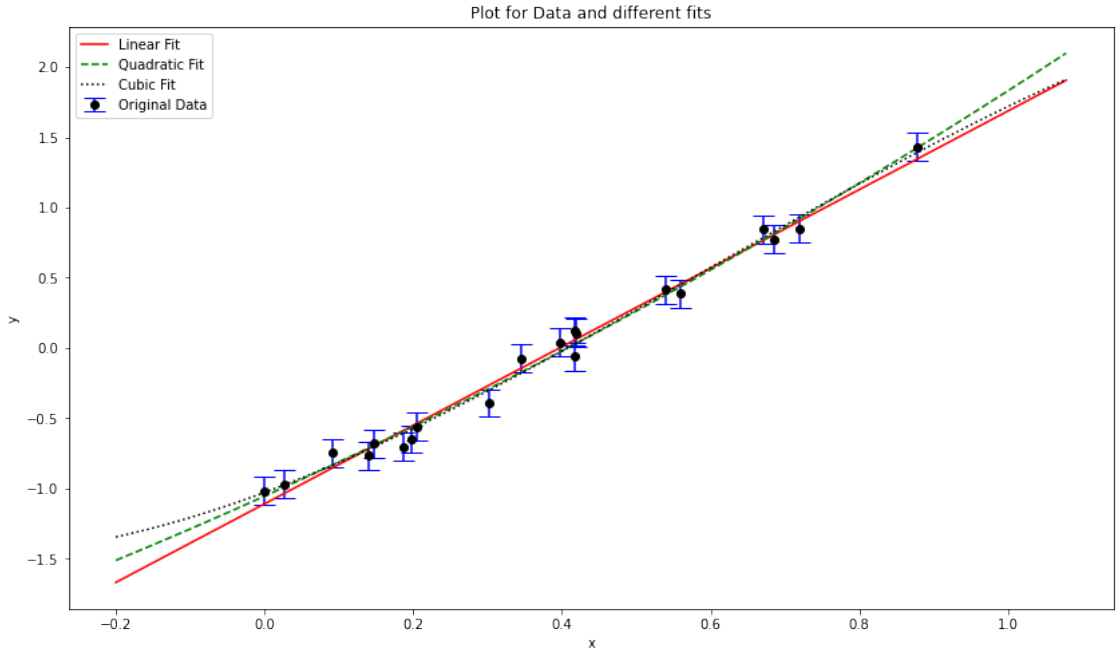
[3]:
```
data = np.loadtxt("q1_data.csv", delimiter = " ", dtype = str)

x, y, sigma_y = [], [], []
for elem in data[1 : ]:
    x.append(float(elem[0]))
    y.append(float(elem[1]))
    sigma_y.append(float(elem[2]))

x = np.array(x)
y = np.array(y)
sigma_y = np.array(sigma_y)
data = [x, y, sigma_y]

linear_param, _ = curve_fit(f = calculate_linear, xdata = x, ydata = y, sigma =␣
 ↪sigma_y)
quad_param, _ = curve_fit(f = calculate_quadratic, xdata = x, ydata = y, sigma =␣
 ↪sigma_y)
cubic_param, _ = curve_fit(f = calculate_cubic, xdata = x, ydata = y, sigma =␣
 ↪sigma_y)

print(f"Parameters for Linear Fit (y = a0 + a1 * x): \na0 =␣
 ↪{linear_param[0]}\na1 = {linear_param[1]}\n")
print(f"Parameters for Quadratic Fit (y = a0 + a1 * x + a2 * x^2):"
      f"\na0 = {quad_param[0]}\na1 = {quad_param[1]}\na2 = {quad_param[2]}\n")
print(f"Parameters for Cubic Fit (y = a0 + a1 * x + a2 * x^2 + a3 * x^3):"
      f"\na0 = {cubic_param[0]}\na1 = {cubic_param[1]}\na2 =␣
 ↪{cubic_param[2]}\na3 = {cubic_param[3]}\n")

chi2_linear = chi2_likelihood(data, 1, linear_param)
chi2_quad = chi2_likelihood(data, 2, quad_param)
chi2_cubic = chi2_likelihood(data, 3, cubic_param)

print("Chi-Square values for-")
print(f"Linear model: {chi2_linear}")
print(f"Quadratic model: {chi2_quad}")
print(f"Cubic model: {chi2_cubic}")

print(f"Since Linear model has the highest Chi-Squared value, it fits the data␣
 ↪best.\n")

AIC_linear = calculate_AIC(data, 1, linear_param)
AIC_quad = calculate_AIC(data, 2, quad_param)
AIC_cubic = calculate_AIC(data, 3, cubic_param)

print("AIC values for-")
print(f"Linear model: {AIC_linear}")
print(f"Quadratic model: {AIC_quad}")
```

```
print(f"Cubic model: {AIC_cubic}")

print(f"Since Linear model has the lowest AIC value, it fits the data best.\n")

AICc_linear = calculate_AICc(data, 1, linear_param)
AICc_quad = calculate_AICc(data, 2, quad_param)
AICc_cubic = calculate_AICc(data, 3, cubic_param)

print("AICc values for-")
print(f"Linear model: {AICc_linear}")
print(f"Quadratic model: {AICc_quad}")
print(f"Cubic model: {AICc_cubic}")

print(f"Since Linear model has the lowest AICc value, it fits the data best.\n")

BIC_linear = calculate_BIC(data, 1, linear_param)
BIC_quad = calculate_BIC(data, 2, quad_param)
BIC_cubic = calculate_BIC(data, 3, cubic_param)

print("BIC values for-")
print(f"Linear model: {BIC_linear}")
print(f"Quadratic model: {BIC_quad}")
print(f"Cubic model: {BIC_cubic}")

print(f"Since Linear model has the lowest BIC value, it fits the data best.\n")

p_value_quad = calculate_p_value_wrt_linear(data, 2, quad_param, linear_param)
p_value_cubic = calculate_p_value_wrt_linear(data, 3, cubic_param, linear_param)

print("p values for-")
print(f"Quadratic fit with Linear fit as null hypothesis: {p_value_quad}")
print(f"Cubic fit with Linear fit as null hypothesis: {p_value_cubic}")
```

```
Parameters for Linear Fit (y = a0 + a1 * x):
a0 = -1.1102808170221141
a1 = 2.7978986063937183

Parameters for Quadratic Fit (y = a0 + a1 * x + a2 * x^2):
a0 = -1.055789148253146
a1 = 2.384751844139253
a2 = 0.502612970397154

Parameters for Cubic Fit (y = a0 + a1 * x + a2 * x^2 + a3 * x^3):
a0 = -1.0291046143544065
a1 = 1.971840396239574
a2 = 1.744513808670288
a3 = -0.9672503050014959
```

```
Chi-Square values for-
Linear model: 0.045383795585918596
Quadratic model: 0.036608447550140304
Cubic model: 0.04215280601005979
Since Linear model has the highest Chi-Squared value, it fits the data best.

AIC values for-
Linear model: -40.03668681607269
Quadratic model: -39.849820624005616
Cubic model: -38.26081851760257
Since Linear model has the lowest AIC value, it fits the data best.

AICc values for-
Linear model: -39.330804463131514
Quadratic model: -38.349820624005616
Cubic model: -35.59415185093591
Since Linear model has the lowest AICc value, it fits the data best.

BIC values for-
Linear model: -38.04522226896471
Quadratic model: -36.862623803343645
Cubic model: -34.27788942338661
Since Linear model has the lowest BIC value, it fits the data best.

p values for-
Quadratic fit with Linear fit as null hypothesis: 0.1781327569531638
Cubic fit with Linear fit as null hypothesis: 0.3288788441952259
```

[4]:
```
x_plot = np.linspace(min(x) - 0.2, max(x) + 0.2, 10000)

y_linear = calculate_linear(x_plot, linear_param[0], linear_param[1])
y_quad = calculate_quadratic(x_plot, quad_param[0], quad_param[1], quad_param[2])
y_cubic = calculate_cubic(x_plot, cubic_param[0], cubic_param[1],␣
 ↪cubic_param[2], cubic_param[3])

fig = plt.figure(figsize = (14, 8))
plt.errorbar(x, y, sigma_y, fmt = "ok", lw = 1.5, capsize = 8, ecolor = "blue",␣
 ↪label = "Original Data")
plt.plot(x_plot, y_linear, color = "red", label = "Linear Fit")
plt.plot(x_plot, y_quad, color = "green", ls = "--", label = "Quadratic Fit")
plt.plot(x_plot, y_cubic, color = "black", ls = ":", label = "Cubic Fit")
plt.title("Plot for Data and different fits")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



Question 2

[5]:
```
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

linear_param, _ = curve_fit(f = calculate_linear, xdata = x, ydata = y, sigma =␣
 ↪sigma_y)
quad_param, _ = curve_fit(f = calculate_quadratic, xdata = x, ydata = y, sigma =␣
 ↪sigma_y)

print("AIC values for-")
print(f"Linear model: {calculate_AIC(data, 1, linear_param)}")
print(f"Quadratic model: {calculate_AIC(data, 2, quad_param)}")
```

```
print(f"Linear model has a lesser AIC value, hence fits the data better.\n")

print("AICc values for-")
print(f"Linear model: {calculate_AICc(data, 1, linear_param)}")
print(f"Quadratic model: {calculate_AICc(data, 2, quad_param)}")
print(f"Linear model has a lesser AICc value, hence fits the data better.\n")

print("BIC values for-")
print(f"Linear model: {calculate_BIC(data, 1, linear_param)}")
print(f"Quadratic model: {calculate_BIC(data, 2, quad_param)}")
print(f"Linear model has a lesser BIC value, hence fits the data better.\n")

print(f"Yes these results agree with Frequentist Model comparisons as shown on␣
 ↪JVDP's blog.")
```

```
AIC values for-
Linear model: -40.02173401322526
Quadratic model: -39.8830271730082
Linear model has a lesser AIC value, hence fits the data better.

AICc values for-
Linear model: -39.31585166028409
Quadratic model: -38.8830271730082
Linear model has a lesser AICc value, hence fits the data better.

BIC values for-
Linear model: -38.03026946611728
Quadratic model: -36.89583035234623
Linear model has a lesser BIC value, hence fits the data better.

Yes these results agree with Frequentist Model comparisons as shown on JVDP's
blog.

AIC and BIC values are used to compare data fitting models based on
strength of evidence rules. If a model has 2 units lower AIC value than
the other model, it is said to fit the data better. Similarly for BIC
values also.
```

Question 3

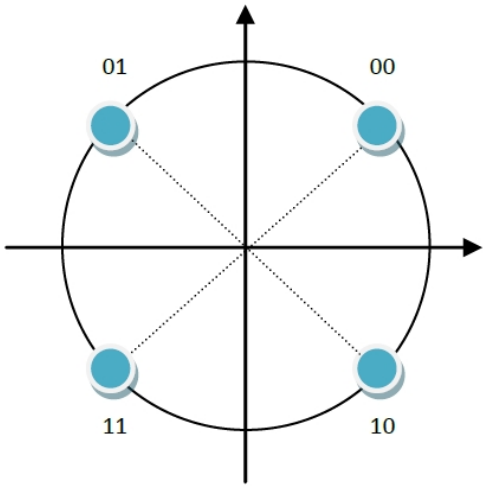
Title: Fold-based Kolmogorov–Smirnov Modulation Classifier

Modulation classification determines the modulation scheme that was used on a signal at the transmitter before sending it through a wireless channel, given that this scheme is unknown at the receiver.

In this paper, feature-based K-S test is used to improve the modulation classification accuracy on

a folded signal in the signal space. This method has various military and civilian applications, and can also be used in spectrum surveillance.

Consider an example, for QPSK modulation the signal can take 4 points in the signal space as illustrated by the below image-



When the number of samples for the received signal is less, it becomes hard to identify these points/hot spots. Therefore we perform signal folding, i.e. folding along both X and Y axes which makes the signal points concentrate around one point in the signal space whose location depends on the modulation scheme.

The K-S test is used as follows-
We consider K possible modulation schemes, i.e. the received signal can follow any modulation scheme from the set $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_K\}$. So the modulation classification problem corresponds to identifying a signal constellation $\mathcal{M}$ for which the received signals belong to. We are given noisy samples of the transmitted signal $\{y_1, y_2, ..., y_N\}$ with the assumption that the noise follows a circularly symmetric Complex Gaussian Distribution $\mathcal{CN}_c(0, \sigma^2)$. To carry out modulation classification, a sequence of decision statistics $\{z_n\}$ are formed from $\{y_n\}_{n=1}^N$. These decision statistics also include the folding operation as mentioned above. Then the Emperical CDF $\hat{F}_1$ of $\{z_n\}$ is computed along with the Theoretical CDF F_0^k corresponding to $\mathcal{M}_k$ for every k in the set (all possible modulation schemes). Finally, the K-S distance is defined as follows-

$$\hat{\mathcal{D}}_k = \max_{1 \leq n \leq N} |\hat{F}_1(z_n) - F_0^k(z_n)|$$

By the Law of Large numbers, if $\hat{F}_1$ converges to F_0^k then the received signal is said to follow $\mathcal{M}_k$. In other words from the Kolmogorov–Smirnov test, $\mathcal{M}_k$ is the optimal modulation scheme for k that minimizes $\hat{\mathcal{D}}_k$.

You can find this paper here.

Question 4

[6]:
```
p_value_higgs = 1.7 * 10 ** -9
print(f"Significance for Higgs Boson discovery for p = 1.7e-9: {stats.norm.
 ↪isf(p_value_higgs)} sigmas\n")

p_value_higgs_from_graph = [10 ** -1, 10 ** -2, 10 ** -3, 10 ** -5, 10 ** -7, 10␣
 ↪** -9]
print(f"Significance for Higgs Boson discovery from graph-")
for p_value in p_value_higgs_from_graph:
    print(f"For p value: {p_value} : {stats.norm.isf(p_value)} sigmas")

p_value_ligo = 2 * (10 ** -7)
significance_ligo = stats.norm.isf(p_value_ligo)
print(f"\nSignificance for LIGO discovery: {significance_ligo} sigmas\n")

chi2_super_K = 65.2
dof_super_K = 67
chi2_gof_super_K = 1 - stats.chi2(dof_super_K).cdf(chi2_super_K)
print(f"The Chi-Squared goodness of fit for Super-K discovery:␣
 ↪{chi2_gof_super_K}")
```

```
Significance for Higgs Boson discovery for p = 1.7e-9: 5.911017938341624 sigmas

Significance for Higgs Boson discovery from graph-
For p value: 0.1 : 1.2815515655446004 sigmas
For p value: 0.01 : 2.3263478740408408 sigmas
For p value: 0.001 : 3.090232306167813 sigmas
For p value: 1e-05 : 4.264890793922825 sigmas
For p value: 1e-07 : 5.1993375821928165 sigmas
For p value: 1e-09 : 5.9978070150076865 sigmas

Significance for LIGO discovery: 5.068957749717791 sigmas

The Chi-Squared goodness of fit for Super-K discovery: 0.5394901931099038 sigmas
```