Tanmay Garg

CS20BTECH11063

Data Science Analysis Assignment 4

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import astroML
from astroML.stats import sigmaG
import pandas as pd
```

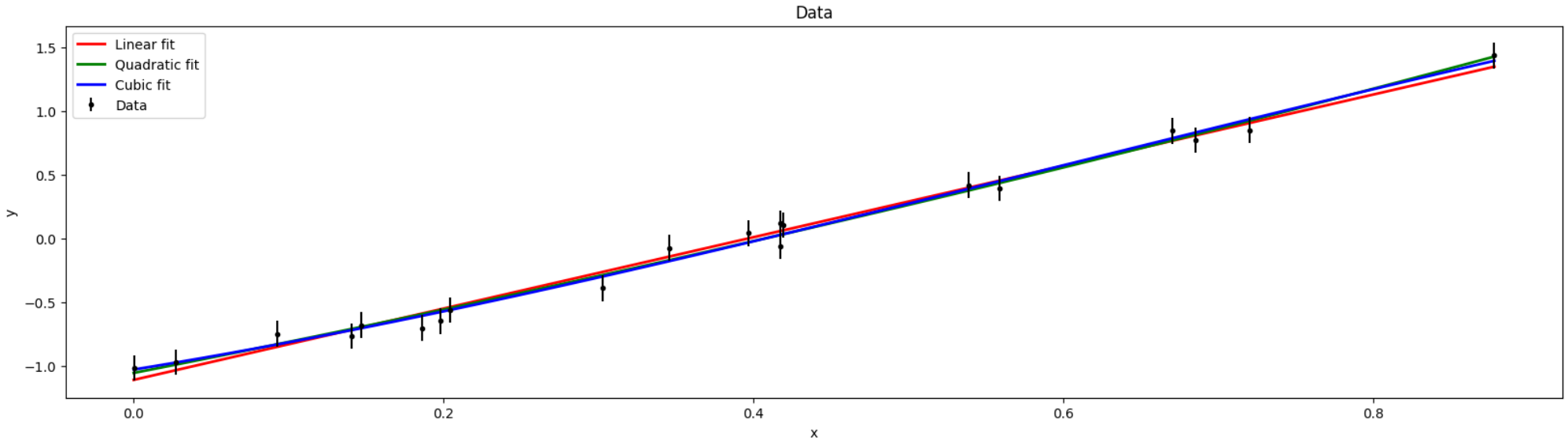
Q1

In [2]:
```
from scipy.optimize import curve_fit

df = pd.read_csv('testdata.csv', sep=' ')

def linear(x, a, b):
    return a + b*x
def quadratic(x, a, b, c):
    return a + b*x + c*x**2
def cubic(x, a, b, c, d):
    return a + b*x + c*x**2 + d*x**3

linear_fit, linear_cov = curve_fit(linear, df['x'], df['y'], sigma=df['sigma_y'])
quadratic_fit, quadratic_cov = curve_fit(quadratic, df['x'], df['y'], sigma=df['sigma_y'])
cubic_fit, cubic_cov = curve_fit(cubic, df['x'], df['y'], sigma=df['sigma_y'])

# Print expression of all fits
print('Linear fit: y = {:.2f} + {:.2f}x'.format(linear_fit[0], linear_fit[1]))
print('Quadratic fit: y = {:.2f} + {:.2f}x + {:.2f}x^2'.format(quadratic_fit[0], quadratic_fit[1], quadratic_fit[2]))
print('Cubic fit: y = {:.2f} + {:.2f}x + {:.2f}x^2 + {:.2f}x^3'.format(cubic_fit[0], cubic_fit[1], cubic_fit[2], cubic_fit[3]))

# plot the data
fig, ax = plt.subplots(figsize=(20, 5))
x = np.linspace(0, max(df['x']), 100)
plt.plot(x, linear(x, linear_fit[0], linear_fit[1]), color='r', linestyle='-', linewidth=2)
```

```
plt.plot(x, quadratic(x, quadratic_fit[0], quadratic_fit[1], quadratic_fit[2]), color='g', linestyle='-', linewidth=2)
plt.plot(x, cubic(x, cubic_fit[0], cubic_fit[1], cubic_fit[2], cubic_fit[3]), color='b', linestyle='-', linewidth=2)
ax.errorbar(df['x'], df['y'], yerr=df['sigma_y'], fmt='.k', ecolor='black')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Data')
plt.legend(['Linear fit', 'Quadratic fit', 'Cubic fit', 'Data'])
# plt.xlim(0, 0.3)
# plt.ylim(-1.2, -0.5)
plt.show()


def AIC_func(degree, parameters, x, y, sigma_y):
    l_max = 0
    if degree == 1:
        fit_func = linear(x, parameters[0], parameters[1])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    if degree == 2:
        fit_func = quadratic(x, parameters[0], parameters[1], parameters[2])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    if degree == 3:
        fit_func = cubic(x, parameters[0], parameters[1], parameters[2], parameters[3])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    return -2*l_max + 2*(degree+1)

def BIC_func(degree, parameters, x, y, sigma_y, N):
    l_max = 0
    if degree == 1:
        fit_func = linear(x, parameters[0], parameters[1])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    if degree == 2:
        fit_func = quadratic(x, parameters[0], parameters[1], parameters[2])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    if degree == 3:
        fit_func = cubic(x, parameters[0], parameters[1], parameters[2], parameters[3])
        l_max = np.sum(stats.norm.logpdf(y, loc=fit_func, scale=sigma_y))
    return -2*l_max + np.log(N)*(degree+1)

def AICc_func(degree, parameters, x, y, sigma_y, N):
    l_max = 0
    return AIC_func(degree, parameters, x, y, sigma_y) + 2*(degree+1)*(degree+2)/(N-degree-2)

def error_fit(degree, parameters, x, y, sigma_y):
    if degree == 1:
        fit_func = linear(x, parameters[0], parameters[1])
    if degree == 2:
```

```
        fit_func = quadratic(x, parameters[0], parameters[1], parameters[2])
    if degree == 3:
        fit_func = cubic(x, parameters[0], parameters[1], parameters[2], parameters[3])
    return (y-fit_func)/sigma_y

def chi2_func(degree, parameters, x, y, sigma_y):
    return np.sum(error_fit(degree, parameters, x, y, sigma_y)**2)

def dof_func(x, degree):
    return len(x) - degree - 1

def chi2_likelihood(degree, parameters, x, y, sigma_y):
    return stats.chi2.pdf(chi2_func(degree, parameters, x, y, sigma_y), dof_func(x, degree))

# Print chi2 likelihood for all fits
print('Chi2 Likelihood Linear fit: {}'.format(chi2_likelihood(1, linear_fit, df['x'], df['y'], df['sigma_y'])))
print('Chi2 Likelihood Quadratic fit: {}'.format(chi2_likelihood(2, quadratic_fit, df['x'], df['y'], df['sigma_y'])))
print('Chi2 Likelihood Cubic fit: {}'.format(chi2_likelihood(3, cubic_fit, df['x'], df['y'], df['sigma_y'])))

# Print AIC, BIC, AICc for all fits in table format
print('AIC\tBIC\tAICc\tFit')
print('{:.2f}\t{:.2f}\t{:.2f}\tLinear'.format(AIC_func(1, linear_fit, df['x'], df['y'], df['sigma_y']),
                                              BIC_func(1, linear_fit, df['x'], df['y'], df['sigma_y'], len(df['x'])),
                                              AICc_func(1, linear_fit, df['x'], df['y'], df['sigma_y'], len(df['x']))))
print('{:.2f}\t{:.2f}\t{:.2f}\tQuadratic'.format(AIC_func(2, quadratic_fit, df['x'], df['y'], df['sigma_y']),
                                                 BIC_func(2, quadratic_fit, df['x'], df['y'], df['sigma_y'], len(df['x'])),
                                                 AICc_func(2, quadratic_fit, df['x'], df['y'], df['sigma_y'], len(df['x']))))
print('{:.2f}\t{:.2f}\t{:.2f}\tCubic'.format(AIC_func(3, cubic_fit, df['x'], df['y'], df['sigma_y']),
                                             BIC_func(3, cubic_fit, df['x'], df['y'], df['sigma_y'], len(df['x'])),
                                             AICc_func(3, cubic_fit, df['x'], df['y'], df['sigma_y'], len(df['x']))))

# print p-value of quadratic fit and cubic fit wrt linear fit
print('p-value quadratic fit wrt linear fit: {}'.format(stats.chi2.sf(chi2_func(2, quadratic_fit, df['x'], df['y'], df['sigma_y']),
                                                                      dof_func(df['x'], 2))))
print('p-value cubic fit wrt linear fit: {}'.format(stats.chi2.sf(chi2_func(3, cubic_fit, df['x'], df['y'], df['sigma_y']),
                                                                  dof_func(df['x'], 3))))
```

```
Linear fit: y = -1.11 + 2.80x
Quadratic fit: y = -1.06 + 2.38x + 0.50x^2
Cubic fit: y = -1.03 + 1.97x + 1.74x^2 + -0.97x^3
```




```
Chi2 Likelihood Linear fit: 0.045383795585918596
Chi2 Likelihood Quadratic fit: 0.036608447550140304
Chi2 Likelihood Cubic fit: 0.04215280601005979
AIC      BIC      AICc     Fit
-40.04   -38.05   -39.33   Linear
-39.85   -36.86   -38.35   Quadratic
-38.26   -34.28   -35.59   Cubic
p-value quadratic fit wrt linear fit: 0.9234043003657486
p-value cubic fit wrt linear fit: 0.9098699709614965
```

It can be seen from the above results that Linear Model would be the best fit as it has the highest χ^2 Likelyhood value. Linear Model also has the least AIC, BIC, AICc values. Hence, Linear Model is the best fit for the given data.

Q2

In [3]:
```
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_fit, linear_cov = curve_fit(linear, x, y, sigma=sigma_y)
quadratic_fit, quadratic_cov = curve_fit(quadratic, x, y, sigma=sigma_y)

# print AIC and BIC for linear fit and quadratic fit in table format
print('AIC\tBIC\tFit')
print('{:.2f}\t{:.2f}\tLinear'.format(AIC_func(1, linear_fit, x, y, sigma_y),
                                      BIC_func(1, linear_fit, x, y, sigma_y, len(x))))
print('{:.2f}\t{:.2f}\tQuadratic'.format(AIC_func(2, quadratic_fit, x, y, sigma_y),
                                         BIC_func(2, quadratic_fit, x, y, sigma_y, len(x))))
```

```
AIC     BIC     Fit
-40.02  -38.03  Linear
-39.88  -36.90  Quadratic
```

As we can see that Linear Model has the smaller AIC and BIC value, hence it is the best fit model.

AIC and BIC is used to compare models based on strength of evidence rules. If a model has an AIC value that is 2 or more units smaller than the AIC value of another model, then the first model is considered to be significantly better than the second model. Similarly, lower BIC value indicates a better model.

Q3

We will be using the paper titled, *First-Passage Time Distribution of Brownian Motion as a Reliability Model* by *Y. S. Sherif* and *M. L. Smith*, to understand the usage of Kolmogoorov-Smirnov test.

This paper tries to fit the inversr Gaussian Distribution model to observed failure data of high speed steel tools.

The Inverse Gaussian Distribution is given by:

$$f(x;\mu,\lambda) = \sqrt{\frac{\lambda}{2\pi x^3}}\exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)$$

where μ and λ are the mean and shape parameter respectively.

They substitute λ with $\mu\gamma^2$, where γ is the dimensionless shape parameter.

They use the failure data reported by *Wager* and *Barash* in the paper titled *Study of the distribution of tool life in high speed steel cutting tools*.

They hypothesize that the failure data is a sample from the Inverse Gaussian Distribution. They use the Kolmogorov-Smirnov test to test the hypothesis.

The value of K-S test value $D_{max} = 0.1120$ and is smaller than the expected value at 5% significance level, $D_{105}^{0.05} = 0.1130$. Using lognormal distribution, the value of D_{max} is 0.1331 and they concluded that Inverse Gaussian Distribution is a better fit for the data.

However, the concern here is that KS test was used incorrectly here as KS test should not be used to compare distributions when the hypothesized distribution is calculated from the data itself as mentioned on the Penn State website.

References:

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

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5220910

https://medium.com/@pabaldonedo/kolmogorov-smirnov-test-may-not-be-doing-what-you-think-when-parameters-are-estimated-from-the-data-2d5c3303a020

https://www.jstor.org/stable/pdf/2280095.pdf?refreqid=excelsior%3Ae3b35e6171e506d41fd5ad8bdf0340fc&ab_segments=&origin=&initiator=&acceptTC=1

Q4

In [4]:
```
# P vals for Higgs Boson
pval_higgs = [10**-1, 10**-2, 10**-3, 10**-5, 10**-7, 10**-9]
significance_higgs = stats.norm.isf(pval_higgs)

print('Significance for Higgs Boson: {}'.format(significance_higgs))

# P vals for LIGO
pval_ligo = 2 * 10 ** (-7)
significance_ligo = stats.norm.isf(pval_ligo)

print('Significance for LIGO: {}'.format(significance_ligo))

# pval_ligo = 2 * 10 ** (-7)
# Z_score = stats.norm.ppf(1 - pval_ligo/2)
# significance_ligo = Z_score
# print('Significance for LIGO: {}'.format(significance_ligo))

# Super K
chi2_val = 65.2
dof_val = 67
chi2_dof = 1 - stats.chi2.cdf(chi2_val, dof_val)
```

```
print('Goodness of Fit for Super K: {}'.format(chi2_dof))
```

```
Significance for Higgs Boson: [1.28155157 2.32634787 3.09023231 4.26489079 5.19933758 5.99780702]
Significance for LIGO: 5.068957749717791
Goodness of Fit for Super K: 0.5394901931099038
```