Assignment-3

Name:- Ashwin Kumar

Roll – CE21BTECH11008

Q1.

Installing the library

```
pip install numpy matplotlib astroML
```

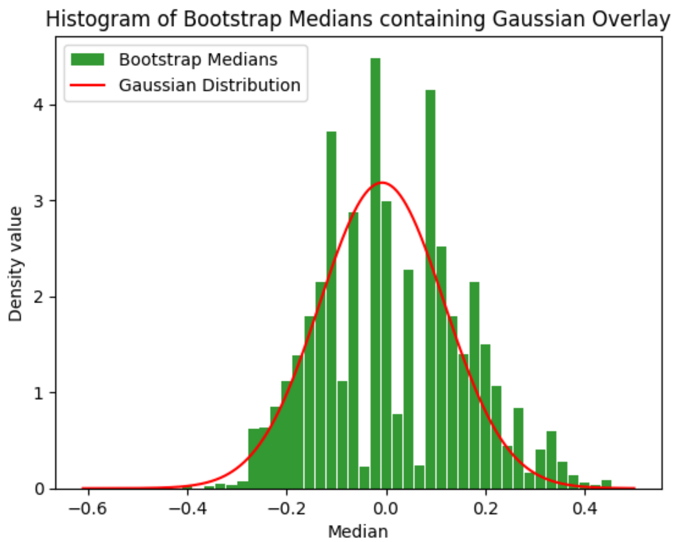
Importing the library

```
import numpy as np
import matplotlib.pyplot as plt
from astroML.stats import median_sigmaG
from scipy.stats import norm
```

Code

```
# central estimates and parameter of the samples
mean = 0
std_dev = 1
sample_size = 100  #large enough to give good estimate
number_of_boot_samples = 10000

# Generate a random sample from the Gaussian distribution
data_sample = np.random.normal(mean, std_dev, sample_size)


# mu2_bootstrap = astroML.resample.bootstrap(data,n,sigmaG,kwargs=dict(axis=1))
#not able to use this function here so going with conventional method to find the
medians

#making function that is taking two paramete first onw is data sampe and second one is
number od bootstrap sample to be generated
def generate_bootstrap_medians(datasets, num_samples):
    medians = np.empty(num_samples) #making the array name m
    for i in range(num_samples):
        bootstrap_sample = np.random.choice(datasets, size=len(datasets),
replace=True)
        medians[i] = np.median(bootstrap_sample)  #for each bootstrap sample that we
have genrated finding the median
    return medians


# Generate bootstrap medians
bootstrap_medians = generate_bootstrap_medians(data_sample, number_of_boot_samples)

# standard deviation as specified i.e root(pi/(2*n))
median_std_dev = np.sqrt(np.pi/(2*sample_size))
print(median_std_dev)

median_std_dev1=median_sigmaG(bootstrap_medians)
print(median_std_dev1)

# Plot histogram of bootstrap medians
plt.hist(bootstrap_medians, bins=50, density=True, alpha=0.8, color='green',
label='Bootstrap Medians',rwidth=0.9)

# Overlay Gaussian distribution on top of the histogram
```

```
x = np.linspace(min(bootstrap_medians),max(bootstrap_medians),1000)

#methodology 1
# y = (1 / (median_std_dev * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x -
np.median(bootstrap_medians)) / median_std_dev) ** 2)

#methodology 2
y = norm.pdf(x, np.median(bootstrap_medians), median_std_dev)
plt.plot(x, y, color='red', label='Gaussian Distribution')

plt.xlabel('Median')
plt.ylabel('Density value')
plt.title('Histogram of Bootstrap Medians containing Gaussian Overlay')
plt.legend()
plt.show()
```



Q2.

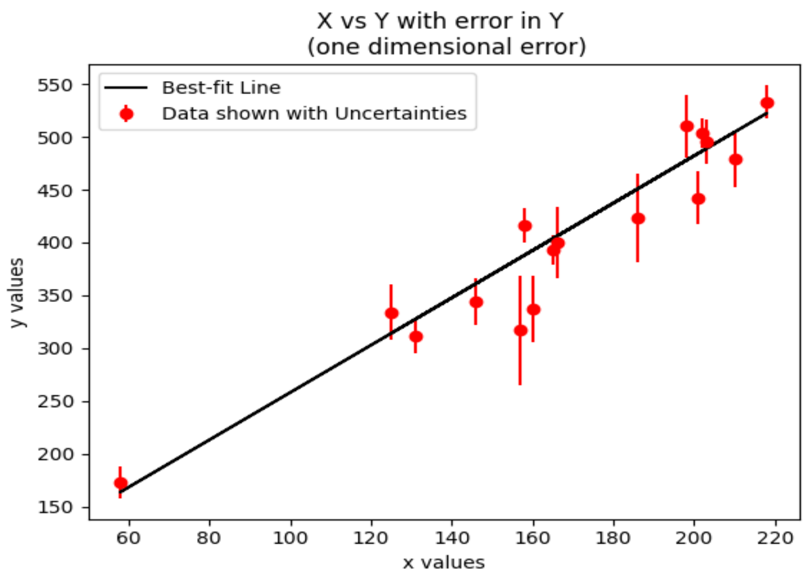
Importing the library

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd
```

code

```
# Assuming your data1 has the name data2_q2.txt with address /content/data2_q2.txt
data1 = pd.read_csv('/content/data2_q2.txt',delimiter=' ')  #space to denote the
separation so delimiter = ' '

#preprocessing the data

#removing the σx ρxy from the data
columns_to_drop = ['σx' , 'ρxy' ,'Unnamed: 6']
data1 = data1.drop(columns=columns_to_drop)

# print(data1)  #data without removing first four rows

#rempving the first four rows
data1 = data1.iloc[4:]
#changing its index to 1 again for better processing in the future
data1.reset_index(drop=True, inplace=True)

# print(data1) #data with removing first four rows

#Extracting x, y, and σy values from the data---
x=data1['x']
y=data1['y']
σy=data1['σy']

# Define the linear function

# first paremeter is x coordinate followed by slope and at the last intercept is there
def linear_function(x, m, c):
    return m * x + c

# Perform the curve fit using scipy's curve_fit with method='lm'
params, covariance = curve_fit(linear_function, x, y, sigma=σy, method='lm')

# Extract the best-fit parameters
bestfit_slope, bestfit_incpt = params

# Calculate the uncertainties in the parameters from the covariance matrix
sigma_m, sigma_b = np.sqrt(np.diag(covariance))

# Plot the data and the best-fit line with uncertainties

#fmt ='o' showing that points in error bar represneted as circle
plt.errorbar(x, y, yerr=σy, fmt='o', label='Data shown with
Uncertainties',color='red')
plt.plot(x, linear_function(np.array(x), bestfit_slope, bestfit_incpt), label='Best-
fit Line',color='k')

#adding labels and legends of the plots
plt.title('X vs Y with error in Y \n (one dimensional error)')
```

```
plt.xlabel('x values')
plt.ylabel('y values')
plt.legend()
plt.show()

# Display the best-fit parameters and their uncertainties
print(f" \nslope of the best fit line : {bestfit_slope} ± {sigma_m}")
print(f"intercept of best fit line : {bestfit_incpt} ± {sigma_b}")
```



Q 3.

Importing the library

```
import pandas as pd
import numpy as np
from scipy import stats
```

code

```
#These are the graph made to chose 50 data points and this is luminousity vs time
graph and
#time is only free parameter. degree of freedom = 50-1 = 49
# reduced chi square =
# overestimated - 0.24          correct error- 0.96           underestimated error-
3.84         incorrect model- 2.85
chisquare_overestimate = 0.24*49
chisquare_correct = 0.96*49
chisquare_underestimate = 3.84*49
chisquare_incorrect = 2.85*49
v=49 # (50-1)
pvalue_overestimate = 1-stats.chi2(v).cdf(chisquare_overestimate)
pvalue_correct = 1-stats.chi2(v).cdf(chisquare_correct)
```

```
pvalue_underestimate = 1-stats.chi2(v).cdf(chisquare_underestimate)
pvalue_incorrect = 1-stats.chi2(v).cdf(chisquare_incorrect)

print(f"p value for overestimat = {pvalue_overestimate} 'p value for correct ='
{pvalue_correct} 'p value for underestimate ='{pvalue_underestimate } 'p value for
incorrect =' { pvalue_incorrect} " )

#result
#p value for overestimat = 0.9999999917009567 'p value for correct ='
0.5529264339960218 'p value for underestimate ='0.0 'p value for incorrect ='
1.2107292945984227e-10
```