Assignment 6

Pushkal Mishra
EE20BTECH11042

Importing Libraries

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

Question 1

[2]:
```
t_einstein = 1.74
t_newtonian = t_einstein / 2

t_eddington = 1.61
t_eddington_error = 0.4

t_crommelin = 1.98
t_crommelin_error = 0.16

# Calculating probability densities from Eddington's team-
einstein_eddin_pdf = stats.norm(t_einstein, t_eddington_error).pdf(t_eddington)
newton_eddin_pdf = stats.norm(t_newtonian, t_eddington_error).pdf(t_eddington)

# Calculating Bayes factor from Eddington's measurements-
eddin_bayes_factor = einstein_eddin_pdf / newton_eddin_pdf

# Calculating probability densities from Crommelin's team-
einstein_cromm_pdf = stats.norm(t_einstein, t_crommelin_error).pdf(t_crommelin)
newton_cromm_pdf = stats.norm(t_newtonian, t_crommelin_error).pdf(t_crommelin)

# Calculating Bayes factor from Crommelin's measurements-
cromm_bayes_factor = einstein_cromm_pdf / newton_cromm_pdf

print(f"\nBayes factor computed using both measurements: {eddin_bayes_factor *␣
↪cromm_bayes_factor}")
```

```
print(f"With a very Decisive strength of evidence, the above value suggests that␣
↪Einstein's predictions are strongly \nsupported when compared to Newton's␣
↪predictions from the given data.\n")

print(f"Bayes factor computed from Eddington's measurements only:␣
↪{eddin_bayes_factor}")
print(f"Bayes factor computed from Crommelin's measurements only:␣
↪{cromm_bayes_factor}\n")

print(f"From individual data, Einstein's predictions are still strongly␣
↪supported when compared to Newton's predictions")
print("with Eddington's measurements giving Substantial and Crommelin's␣
↪measurements giving Decisive strength of evidence.")

print(f"\nHere we consider Newtonian gravity as the Null-Hypothesis.")
```

```
Bayes factor computed using both measurements: 48164622958.3418
With a very Decisive strength of evidence, the above value suggests that
Einstein's predictions are strongly
supported when compared to Newton's predictions from the given data.

Bayes factor computed from Eddington's measurements only: 5.25109958796716
Bayes factor computed from Crommelin's measurements only: 9172292802.960836

From individual data, Einstein's predictions are still strongly supported when
compared to Newton's predictions
with Eddington's measurements giving Substantial and Crommelin's measurements
giving Decisive strength of evidence.

Here we consider Newtonian gravity as the Null-Hypothesis.
```

Question 2

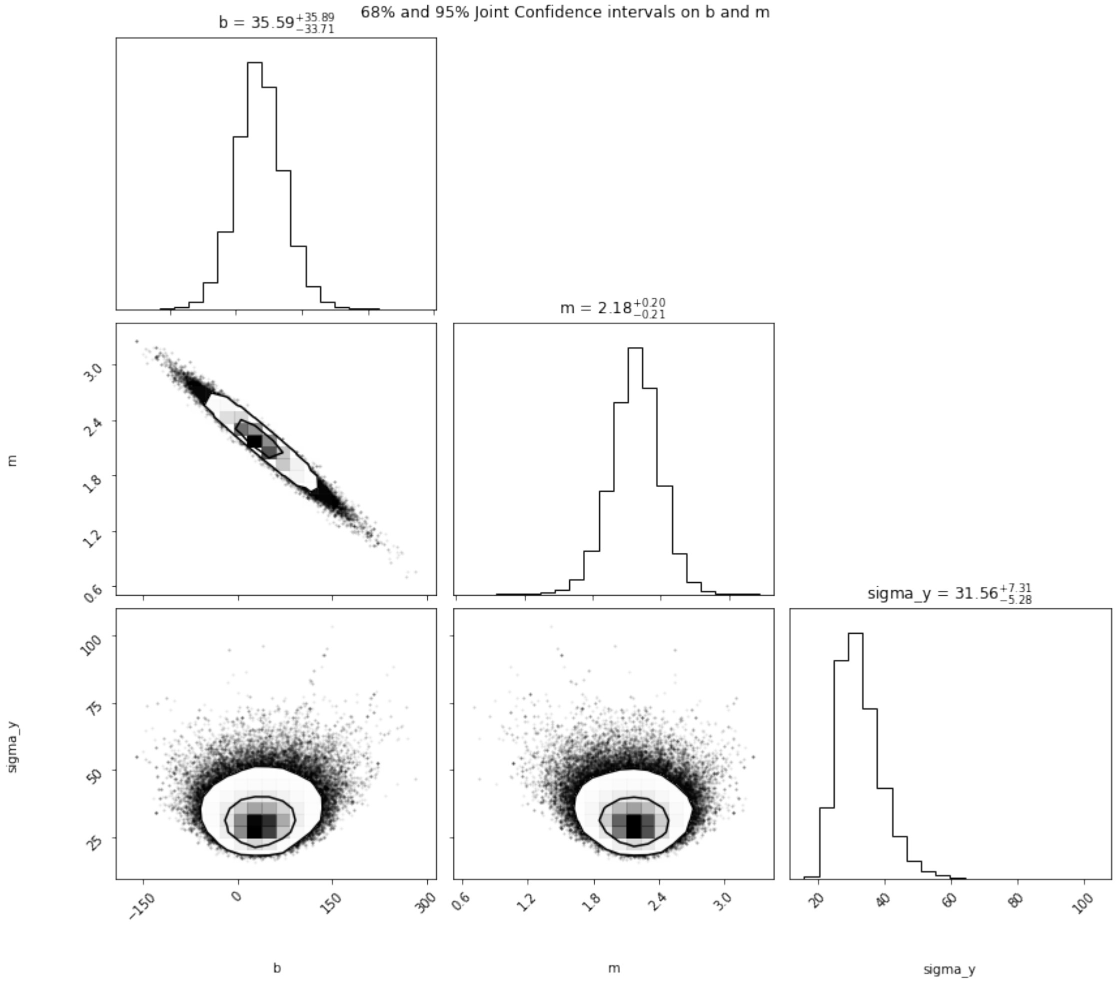
[3]:
```
# Functions from JVDPs blog on performing Bayesian analysis
def log_prior(theta):
    alpha, beta, sigma = theta
    if sigma < 0:
        return -np.inf  # log(0)
    else:
        return -1.5 * np.log(1 + beta ** 2) - np.log(sigma)

def log_likelihood(theta, x, y):
    alpha, beta, sigma = theta
    y_model = alpha + beta * x
```

```
    return -0.5 * np.sum(np.log(2 * np.pi * sigma ** 2) + (y - y_model) ** 2 /␣
 ↪sigma ** 2)

def log_posterior(theta, x, y):
    log_pr = log_prior(theta)
    if np.isfinite(log_pr):
        return log_pr + log_likelihood(theta, x, y)
    return -np.inf
```

[4]:
```
array = np.loadtxt("q2_data.csv", delimiter = " ", dtype = str)

x = []
y = []
sigma_y = []

for lst in array:
    x.append(float(lst[1]))
    y.append(float(lst[2]))
    sigma_y.append(float(lst[3]))

x = np.array(x)
y = np.array(y)
sigma_y = np.array(sigma_y)

n_params = 3
n_walkers = 100
n_burn = 1000
n_steps = 5000

np.random.seed(11042)

initial_guesses = np.random.random([n_walkers, n_params])
mcmc_sampler = emcee.EnsembleSampler(n_walkers, n_params, log_posterior, args =␣
 ↪(x, y))

dump = mcmc_sampler.run_mcmc(initial_guesses, n_steps, progress = True)
```

```
100%|| 5000/5000 [00:13<00:00,
378.84it/s]
```

[5]:
```
mcmc_samples = mcmc_sampler.get_chain(discard = n_burn, flat=True)

fig = plt.figure(figsize = (13, 11))
figure = corner.corner(mcmc_samples, levels = [0.68, 0.95], labels = ["b", "m",␣
 ↪"sigma_y"], fig = fig, show_titles = True)
plt.suptitle("68% and 95% Joint Confidence intervals on b and m")
plt.show()
```



68% and 95% Joint Confidence intervals on b and m
b = 35.59$^{+35.89}_{-33.71}$
m = 2.18$^{+0.20}_{-0.21}$
sigma_y = 31.56$^{+7.31}_{-5.28}$
m
sigma_y
b
m
sigma_y

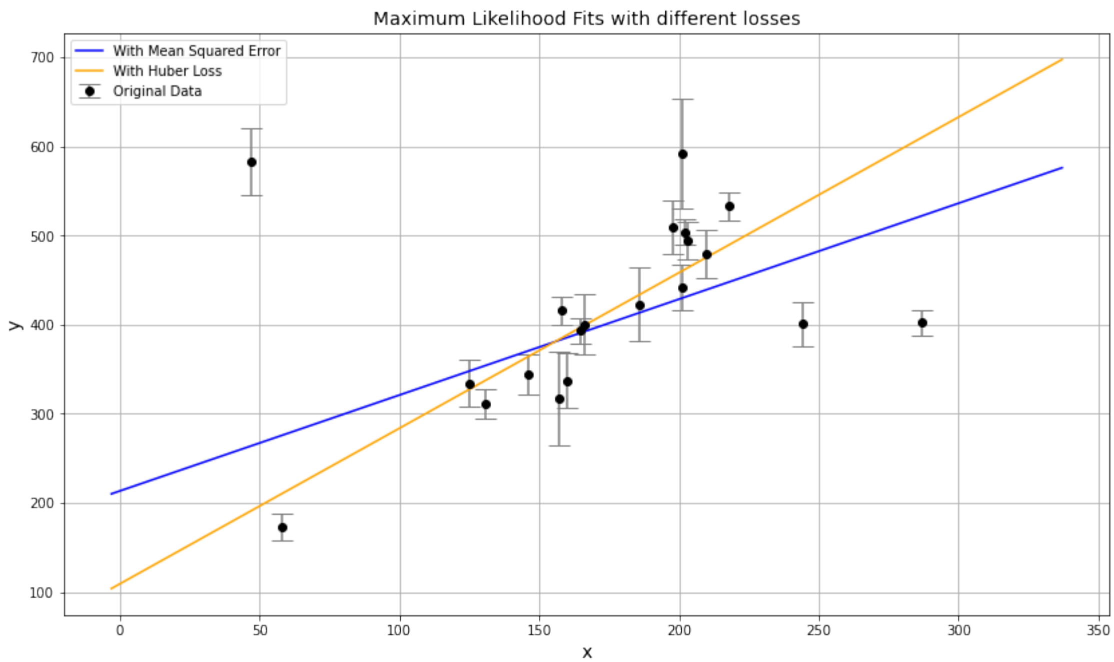
Question 3

[6]:
```
array = np.loadtxt("q3_data.csv", delimiter = " ", dtype = str)

x = []
y = []
sigma_y = []
i = 0
for lst in array:
    x.append(float(lst[0]))
    y.append(float(lst[1]))
    sigma_y.append(float(lst[2]))

x = np.array(x)
y = np.array(y)
sigma_y = np.array(sigma_y)
```

[7]:
```
def squared_loss(theta, x = x, y = y, e = sigma_y):
    dy = y - theta[0] - theta[1] * x
    return np.sum(0.5 * (dy / e) ** 2)

def huber_loss(t, c = 3):
    return ((abs(t) < c) * 0.5 * t ** 2
            + (abs(t) >= c) * -c * (0.5 * c - abs(t)))

def total_huber_loss(theta, x = x, y = y, e = sigma_y, c = 3):
    return huber_loss((y - theta[0] - theta[1] * x) / e, c).sum()

def log_prior(theta):
    if (all(theta[2:] > 0) and all(theta[2:] < 1)):
        return 0
    else:
        return -np.inf

def log_likelihood(theta, x, y, e, sigma_B):
    dy = y - theta[0] - theta[1] * x
    g = np.clip(theta[2:], 0, 1)
    logL1 = np.log(g) - 0.5 * np.log(2 * np.pi * e ** 2) - 0.5 * (dy / e) ** 2
    logL2 = np.log(1 - g) - 0.5 * np.log(2 * np.pi * sigma_B ** 2) - 0.5 * (dy /␣
↪sigma_B) ** 2
    return np.sum(np.logaddexp(logL1, logL2))

def log_posterior(theta, x, y, e, sigma_B):
    return log_prior(theta) + log_likelihood(theta, x, y, e, sigma_B)
```

[8]:
```
theta1_mse = optimize.fmin(squared_loss, [0, 0], disp = False)
theta2_huber = optimize.fmin(total_huber_loss, [0, 0], disp = False)

x_test = np.linspace(min(x) - 50, max(x) + 50, 10000)
```

```
y_test_mse = theta1_mse[0] + theta1_mse[1] * x_test
y_test_huber = theta2_huber[0] + theta2_huber[1] * x_test

fig = plt.figure(figsize = (14, 8))
plt.errorbar(x, y, sigma_y, fmt = "ok", lw = 1.5, capsize = 8, ecolor = "gray",␣
 ↪label = "Original Data")
plt.plot(x_test, y_test_mse, color = "blue", label = "With Mean Squared Error")
plt.plot(x_test, y_test_huber, color = "orange", label = "With Huber Loss")
plt.title("Maximum Likelihood Fits with different losses", size = 14)
plt.xlabel("x", size = 14)
plt.ylabel("y", size = 14)
plt.legend()
plt.grid()
plt.show()
```



[9]:
```
ndim = 2 + len(x)
nwalkers = 75
nburn = 10000
nsteps = 15000

np.random.seed(0)
starting_guesses = np.zeros((nwalkers, ndim))
starting_guesses[:, :2] = np.random.normal(theta2_huber, 1, (nwalkers, 2))
starting_guesses[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))
```

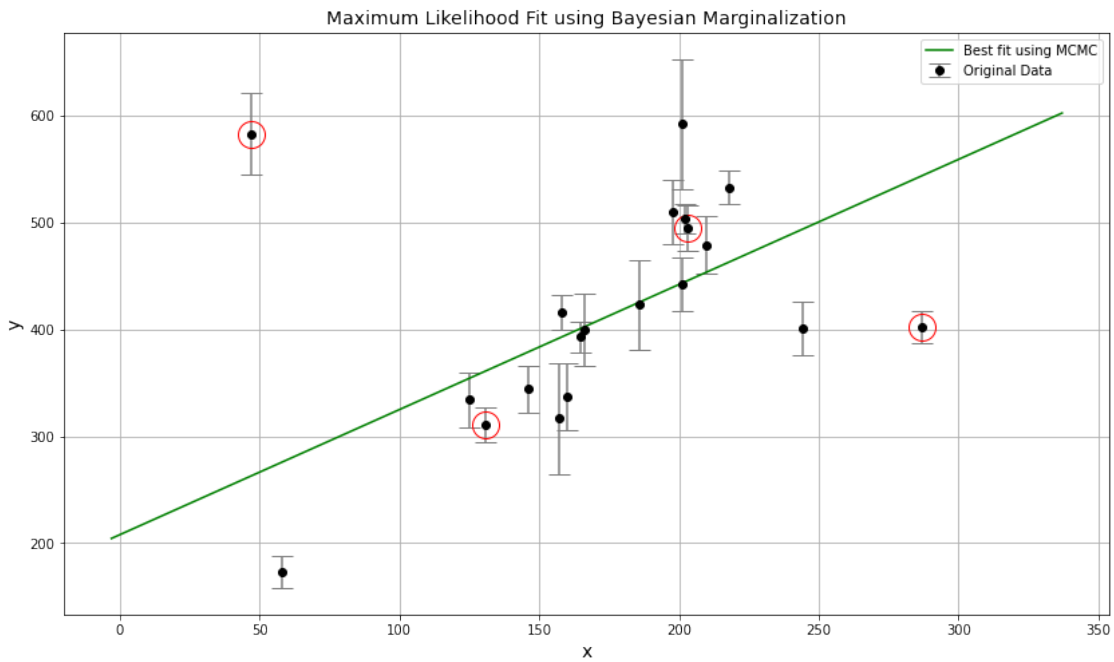
```
mcmc_sampler = emcee.EnsembleSampler(nwalkers, ndim, log_posterior, args=[x, y,␣
↪sigma_y, 50])
%time mcmc_sampler.run_mcmc(starting_guesses, nsteps, progress = True)
sample = mcmc_sampler.chain
sample = mcmc_sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

```
  0%|                                                      | 0/15000
[00:00<?, ?it/s]/var/folders/9l/0v1lmq9n7vv9hrrfhyb6bd4w0000gn/T/ipykernel_44885
/3498753377.py:22: RuntimeWarning: divide by zero encountered in log
  logL2 = np.log(1 - g) - 0.5 * np.log(2 * np.pi * sigma_B ** 2) - 0.5 * (dy /
sigma_B) ** 2
/var/folders/9l/0v1lmq9n7vv9hrrfhyb6bd4w0000gn/T/ipykernel_44885/3498753377.py:2
1: RuntimeWarning: divide by zero encountered in log
  logL1 = np.log(g) - 0.5 * np.log(2 * np.pi * e ** 2) - 0.5 * (dy / e) ** 2
100%|| 15000/15000 [00:58<00:00,
255.21it/s]
```

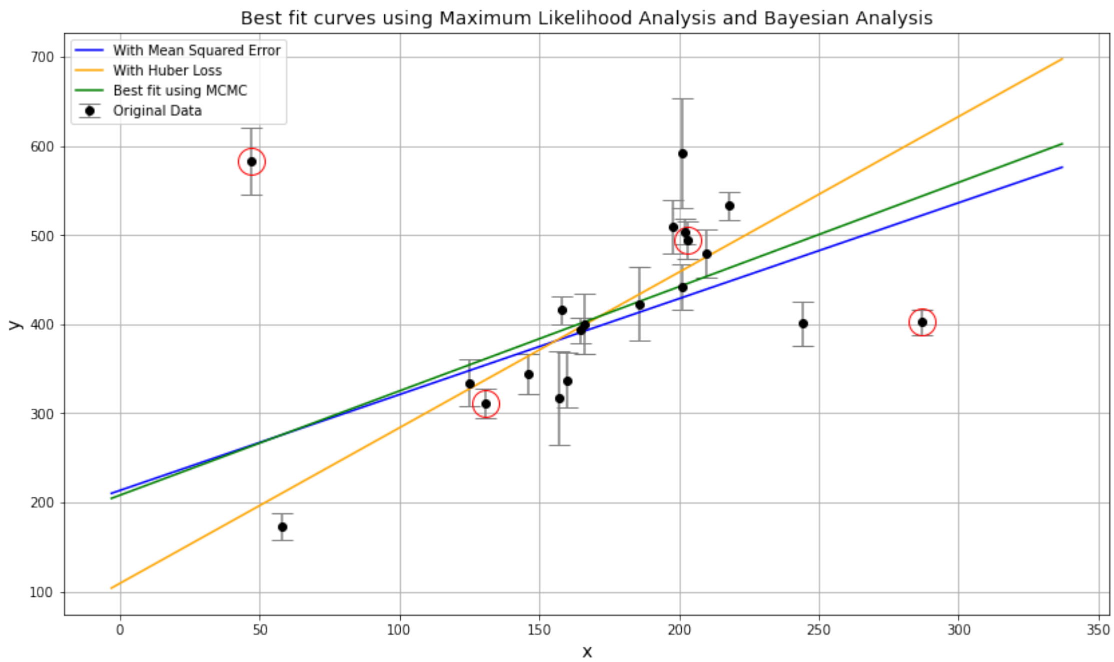
```
CPU times: user 58.5 s, sys: 782 ms, total: 59.3 s
Wall time: 58.9 s
```

[10]:

```
theta3_mcmc = np.mean(sample[:, :2], 0)
g = np.mean(sample[:, 2:], 0)
# outliers_cutoff = (sample[:, 2] + sample[:, 3]).mean() / 2
outliers_cutoff = 0.39
outliers = (g < outliers_cutoff)

y_test_mcmc = theta3_mcmc[0] + theta3_mcmc[1] * x_test

fig = plt.figure(figsize = (14, 8))
plt.errorbar(x, y, sigma_y, fmt = "ok", lw = 1.5, capsize = 8, ecolor = "gray",␣
↪label = "Original Data")
plt.plot(x_test, y_test_mcmc, color = "green", label = "Best fit using MCMC")
plt.plot(x[outliers], y[outliers], "ro", ms = 20, mfc = "none", mec = "red")
plt.title("Maximum Likelihood Fit using Bayesian Marginalization", size = 14)
plt.xlabel("x", size = 14)
plt.ylabel("y", size = 14)
plt.legend()
plt.grid()
plt.show()
```



[11]:
```
fig = plt.figure(figsize = (14, 8))
plt.errorbar(x, y, sigma_y, fmt = "ok", lw = 1.5, capsize = 8, ecolor = "gray",␣
 ↪label = "Original Data")
plt.plot(x_test, y_test_mse, color = "blue", label = "With Mean Squared Error")
plt.plot(x_test, y_test_huber, color = "orange", label = "With Huber Loss")
plt.plot(x_test, y_test_mcmc, color = "green", label = "Best fit using MCMC")
plt.plot(x[outliers], y[outliers], "ro", ms = 20, mfc = "none", mec = "red")
plt.title("Best fit curves using Maximum Likelihood Analysis and Bayesian␣
 ↪Analysis", size = 14)
plt.xlabel("x", size = 14)
plt.ylabel("y", size = 14)
plt.legend()
plt.grid()
plt.show()
```


Best fit curves using Maximum Likelihood Analysis and Bayesian Analysis
With Mean Squared Error
With Huber Loss
Best fit using MCMC
Original Data
y
x
100
200
300
400
500
600
700
0
50
100
150
200
250
300
350