Tanmay Garg

CS20BTECH11063

Data Science Analysis Assignment 6

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
import seaborn as sns
```

Q1

In [2]:
```
einstein_theory_value = 1.74
newtonian_theory_value = einstein_theory_value / 2
eddington_val = 1.61
eddington_error = 0.4
crommelin_val = 1.98
crommelin_error = 0.16

llh_eddington_einstein = stats.norm.pdf(eddington_val, loc=einstein_theory_value, scale=eddington_error)
llh_eddington_newtonian = stats.norm.pdf(eddington_val, loc=newtonian_theory_value, scale=eddington_error)
llh_crommelin_einstein = stats.norm.pdf(crommelin_val, loc=einstein_theory_value, scale=crommelin_error)
llh_crommelin_newtonian = stats.norm.pdf(crommelin_val, loc=newtonian_theory_value, scale=crommelin_error)

unnorm_einstein = llh_eddington_einstein * llh_crommelin_einstein
unnorm_newtonian = llh_eddington_newtonian * llh_crommelin_newtonian

norm_einstein = unnorm_einstein / (unnorm_einstein + unnorm_newtonian)
norm_newtonian = unnorm_newtonian / (unnorm_einstein + unnorm_newtonian)

print("Bayes Factor for Einstein vs Newtonian: ", norm_einstein / norm_newtonian)
print("Bayes Factor for Newtonian vs Einstein: ", norm_newtonian / norm_einstein)
```

```
Bayes Factor for Einstein vs Newtonian:  48164622958.34179
Bayes Factor for Newtonian vs Einstein:  2.076212661033209e-11
```

Q2

In [3]:
```
# # Read the data from the csv file
# df = pd.read_csv('q2.csv', sep=' ')
# df.drop('ID', axis=1, inplace=True)
# df.drop('sigmax', axis=1, inplace=True)
# # df.drop('sigma_y', axis=1, inplace=True)
# df.drop('rhoxy', axis=1, inplace=True)
# # remove first 4 rows
# df = df.iloc[4:]
# print(df.head())

# # Define the function to fit the data
# def curve_func(m, x, c):
#     return m * x + c

# # log likelihood function
# def log_likelihood(theta, x, y, yerr):
#     m, c = theta
#     model = curve_func(m, x, c)
#     sigma2 = yerr ** 2
#     return -0.5 * np.sum((y - model) ** 2 / sigma2 + np.log(sigma2))

# # log prior function
# def log_prior(theta):
#     m, c = theta
#     if -100 < m < 100 and -100 < c < 100:
#         return 0.0
#     return -np.inf

# # log posterior function
# def log_probability(theta, x, y, yerr):
#     lp = log_prior(theta)
#     if not np.isfinite(lp):
#         return -np.inf
#     return lp + log_likelihood(theta, x, y, yerr)

# # MCMC sampling
# import emcee
# ndim, nwalkers = 2, 100
# pos = np.random.randn(nwalkers, ndim)
# nsteps = 5000
# sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=(df['x'], df['y'], df['sigmay']))
# sampler.run_mcmc(pos, nsteps, progress=True)

# # Plot MCMC results
# fig, axes = plt.subplots(2, figsize=(10, 7), sharex=True)
# labels = ["m", "c"]


# mcmc_samples = sampler.get_chain(flat=True)
# m_medain, c_median = np.median(mcmc_samples, axis=0)
```

In [4]:
```
# print(mcmc_samples.shape)
# m_sigma68, c_sigma68 = np.percentile(mcmc_samples, [16, 84], axis=0)
# m_sigma95, c_sigma95 = np.percentile(mcmc_samples, [2.5, 97.5], axis=0) - [m_medain, c_median]

# import corner

# fig = corner.corner(mcmc_samples, labels=labels,
#                       quantiles=[0.16, 0.5, 0.84], show_titles=True,
#                       truths=[m_medain, c_median], title_kwargs={"fontsize": 12})
# fig.suptitle("MCMC results", fontsize=16)
# plt.show()
```

In [5]:
```
# import corner

# fig = corner.corner(mcmc_samples, labels=columns,
#                       quantiles=[0.16, 0.5, 0.84], show_titles=True,
#                       )
# plt.show()
```

In [6]:
```
# import numpy as np
# import pandas as pd
# import matplotlib.pyplot as plt

# # Read the data from the csv file
# df = pd.read_csv('q2.csv', sep=' ')
# df.drop('ID', axis=1, inplace=True)
# df.drop('sigmax', axis=1, inplace=True)
# # df.drop('sigma_y', axis=1, inplace=True)
# df.drop('rhoxy', axis=1, inplace=True)
# # remove first 4 rows
# df = df.iloc[4:]
# print(df.head())

# # Extract the columns as numpy arrays
# x = df['x'].values
# y = df['y'].values
# sigma_y = df['sigmay'].values

# # Define the design matrix
# A = np.vstack((x, np.ones_like(x))).T

# # Define the covariance matrix
# C = np.diag(sigma_y**2)

# # Solve for the parameters using linear algebra
# cov = np.linalg.inv(np.dot(A.T, np.linalg.solve(C, A)))
# m_fit, b_fit = np.dot(cov, np.dot(A.T, np.linalg.solve(C, y)))
# # Calculate the predicted values and residuals
# y_pred = m_fit * x + b_fit
# residuals = y - y_pred

# # Calculate the chi-square value and degrees of freedom
# chi2 = np.sum(residuals**2 / sigma_y**2)
# dof = len(x) - 2

# # Print the results
# print(f'm = {m_fit:.3f}')
# print(f'b = {b_fit:.3f}')
# print(f'chi^2 = {chi2:.3f}')
# print(f'dof = {dof}')

# # Calculate the confidence intervals
# alpha = 0.32
# m_sigma68, b_sigma68 = np.sqrt(np.diag(cov)) * np.sqrt(chi2/dof) * np.sqrt(alpha/(1-alpha))
# m_sigma95, b_sigma95 = np.sqrt(np.diag(cov)) * np.sqrt(chi2/dof) * np.sqrt(0.05)

# # Plot the data and the best-fit line
# fig, ax = plt.subplots()
# ax.errorbar(x, y, yerr=sigma_y, fmt='o')
# ax.plot(x, m_fit*x + b_fit, '-')
# ax.set_xlabel('x')
# ax.set_ylabel('y')
# ax.set_title('Best-fit straight line')
# plt.show()

# # Print the confidence intervals
# print(f'm = {m_fit:.3f} + {m_sigma68:.3f} - {m_sigma68:.3f} (68% CI) + {m_sigma95:.3f} - {m_sigma95:.3f} (95% CI)')
# print(f'b = {b_fit:.3f} + {b_sigma68:.3f} - {b_sigma68:.3f} (68% CI) + {b_sigma95:.3f} - {b_sigma95:.3f} (95% CI)')
```

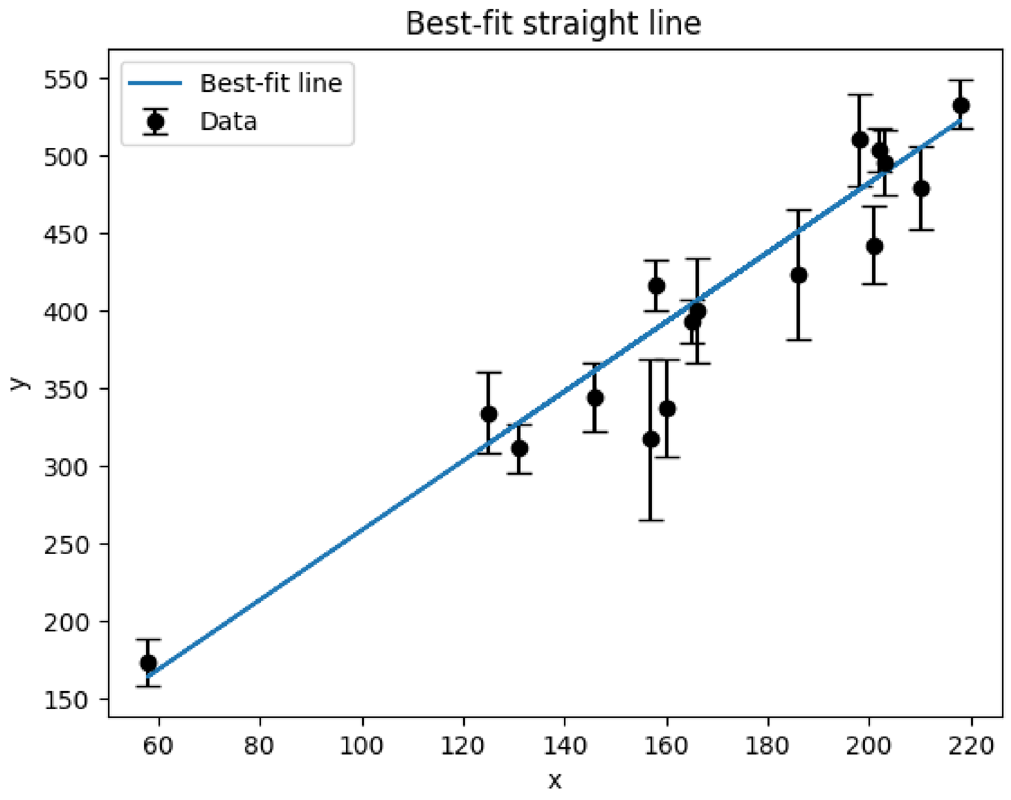
In [11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import emcee

# Read the data from the csv file
df = pd.read_csv('q2.csv', sep=' ')
df.drop('ID', axis=1, inplace=True)
df.drop('sigmax', axis=1, inplace=True)
# df.drop('sigma_y', axis=1, inplace=True)
df.drop('rhoxy', axis=1, inplace=True)
# remove first 4 rows
df = df.iloc[4:]
print(df.head())

# Fit the data using method in paper
Y = df['y'].to_numpy()
X = df['x'].to_numpy()
```

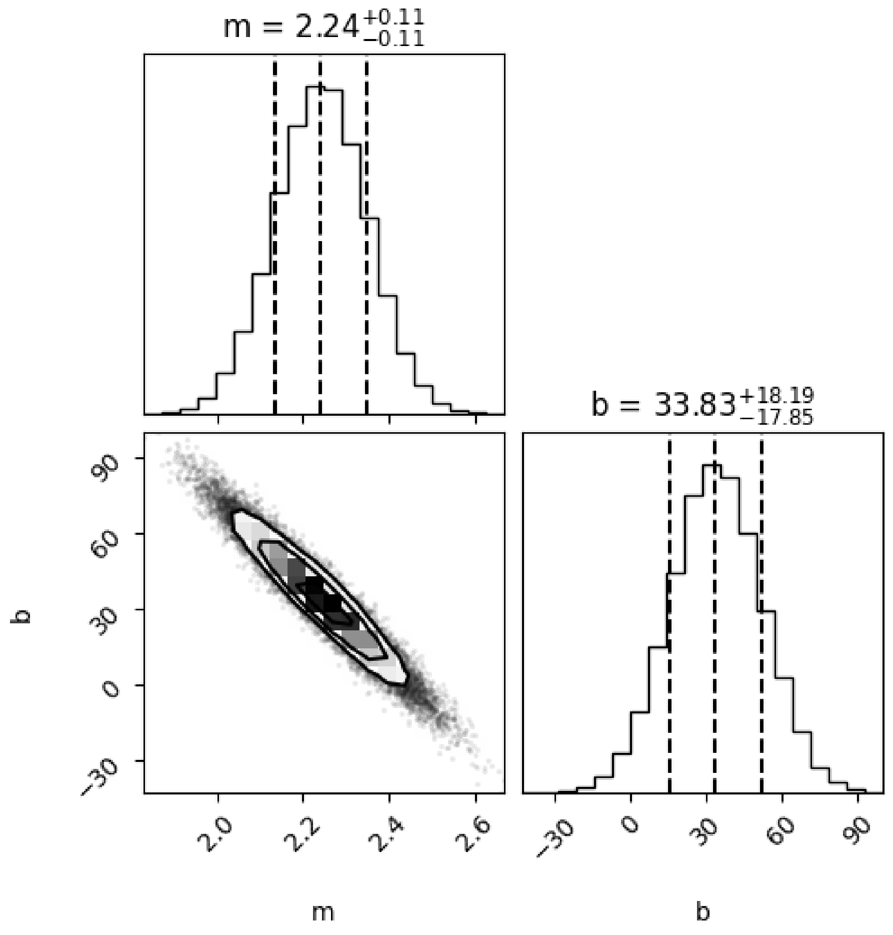
```python
X = np.concatenate((np.ones((len(X), 1)), X.reshape(-1, 1)), axis=1)
C = np.diag(df['sigmay'].to_numpy() ** 2)
best_fit_val = np.linalg.inv(X.T @ np.linalg.inv(C) @ X) @ (X.T @ np.linalg.inv(C) @ Y)
m_fit = best_fit_val[1]
b_fit = best_fit_val[0]
print(f'm = {m_fit:.3f}')
print(f'b = {b_fit:.3f}')


# Extract the columns as numpy arrays
x = df['x'].values
y = df['y'].values
sigma_y = df['sigmay'].values

# Define the log likelihood function
def ln_likelihood(theta, x, y, sigma_y):
    m, b = theta
    y_pred = m * x + b
    chi2 = np.sum((y - y_pred)**2 / sigma_y**2)
    return -0.5 * chi2

# Define the log prior function
def ln_prior(theta):
    m, b = theta
    if -30 < m < 30 and -100 < b < 100:
        return 0.0
    return -np.inf

# Define the log probability function
def ln_prob(theta, x, y, sigma_y):
    lp = ln_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + ln_likelihood(theta, x, y, sigma_y)

# Set up the initial positions and number of walkers
ndim = 2
nwalkers = 100
# set pos to be a small random perturbation of the best-fit values
pos = np.array([m_fit, b_fit]) + 1e-4 * np.random.randn(nwalkers, ndim)
# pos = pos + 1e-4 * np.random.randn(nwalkers, ndim)

# Set up the sampler
sampler = emcee.EnsembleSampler(nwalkers, ndim, ln_prob, args=(x, y, sigma_y))

# Run the sampler and discard the burn-in samples
nburn = 1000
nsteps = 5000
sampler.run_mcmc(pos, nsteps, progress=True)
samples = sampler.chain[:, nburn:, :].reshape(-1, ndim)
```

```
# Calculate the confidence intervals
m_median, b_median = np.median(samples, axis=0)
m_sigma68, b_sigma68 = np.percentile(samples, [16, 84], axis=0)
m_sigma95, b_sigma95 = np.percentile(samples, [2.5, 97.5], axis=0)

# Print the results
print(f'm = {m_median:.3f} + {m_sigma68[1] - m_median:.3f} - {m_median - m_sigma68[0]:.3f} (68% CI) + {m_sigma95[1] - m_median:.3f} - {m_median - m_sigma95[(
print(f'b = {b_median:.3f} + {b_sigma68[1] - b_median:.3f} - {b_median - b_sigma68[0]:.3f} (68% CI) + {b_sigma95[1] - b_median:.3f} - {b_median - b_sigma95[(

# Plot the data and the best-fit line
fig, ax = plt.subplots()
ax.errorbar(x, y, yerr=sigma_y, fmt='ok', ecolor='black', lw=1.5, capsize=5, label='Data')
ax.plot(x, m_median*x + b_median, '-', label='Best-fit line')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
ax.set_title('Best-fit straight line')
plt.show()

# # Plot MCMC chains
# fig, axes = plt.subplots(2, 1, figsize=(10, 7), sharex=True)
# labels = ['m', 'b']
# for i in range(ndim):
#     axes[i].plot(sampler.chain[:, :, i].T, 'k', alpha=0.3)
#     axes[i].set_xlim(0, len(sampler.chain[0]))
#     axes[i].set_ylabel(labels[i])
#     axes[i].yaxis.set_label_coords(-0.1, 0.5)
# axes[-1].set_xlabel('Step number')
# plt.show()

# use the corner package to plot the 2D posterior distributions
import corner
flat_samples = sampler.get_chain(discard=nburn, thin=15, flat=True)
fig = corner.corner(data=flat_samples, labels=['m', 'b'],
                    quantiles=[0.16, 0.5, 0.84], show_titles=True,
                    title_kwargs={"fontsize": 12})
plt.show()

# plt.figure(figsize=(15, 5))
# plt.scatter(flat_samples[:, 0], flat_samples[:, 1], s=1)
# plt.xlabel('m')
# plt.ylabel('b')
# plt.xlim(1.5,3)
# plt.ylim(-50,100)
# plt.show()
```

```
     x    y  sigmay
4  203  495      21
5   58  173      15
6  210  479      27
7  202  504      14
8  198  510      30
m = 2.240
b = 34.048
```

```
100%|██████████| 5000/5000 [00:17<00:00, 286.82it/s]
m = 2.241 + 13.734 - 0.107 (68% CI) + -3.841 - 0.210 (95% CI)
b = 33.847 + 18.168 - 31.500 (68% CI) + 35.518 - 31.397 (95% CI)
```





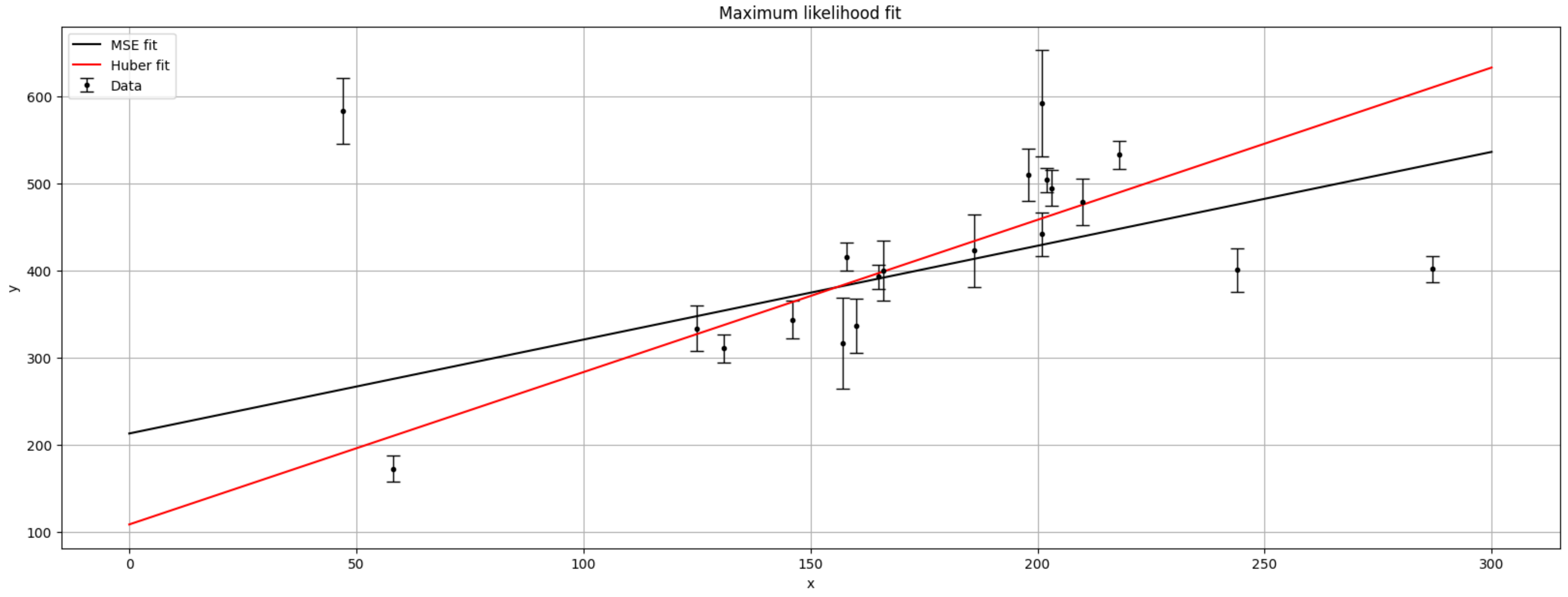
Q3

```
In [8]: from scipy import optimize

        # Read the data from the csv file
        df = pd.read_csv('q2.csv', sep=' ')
        df.drop('ID', axis=1, inplace=True)
        df.drop('sigmax', axis=1, inplace=True)
        # df.drop('sigma_y', axis=1, inplace=True)
        df.drop('rhoxy', axis=1, inplace=True)
        print(df.head())
        x = df['x'].values
        y = df['y'].values
        sigma_y = df['sigmay'].values

        def mse_loss(theta, x=x, y=y, sigma_y=sigma_y):
            dy = y - (theta[0] + theta[1] * x)
```

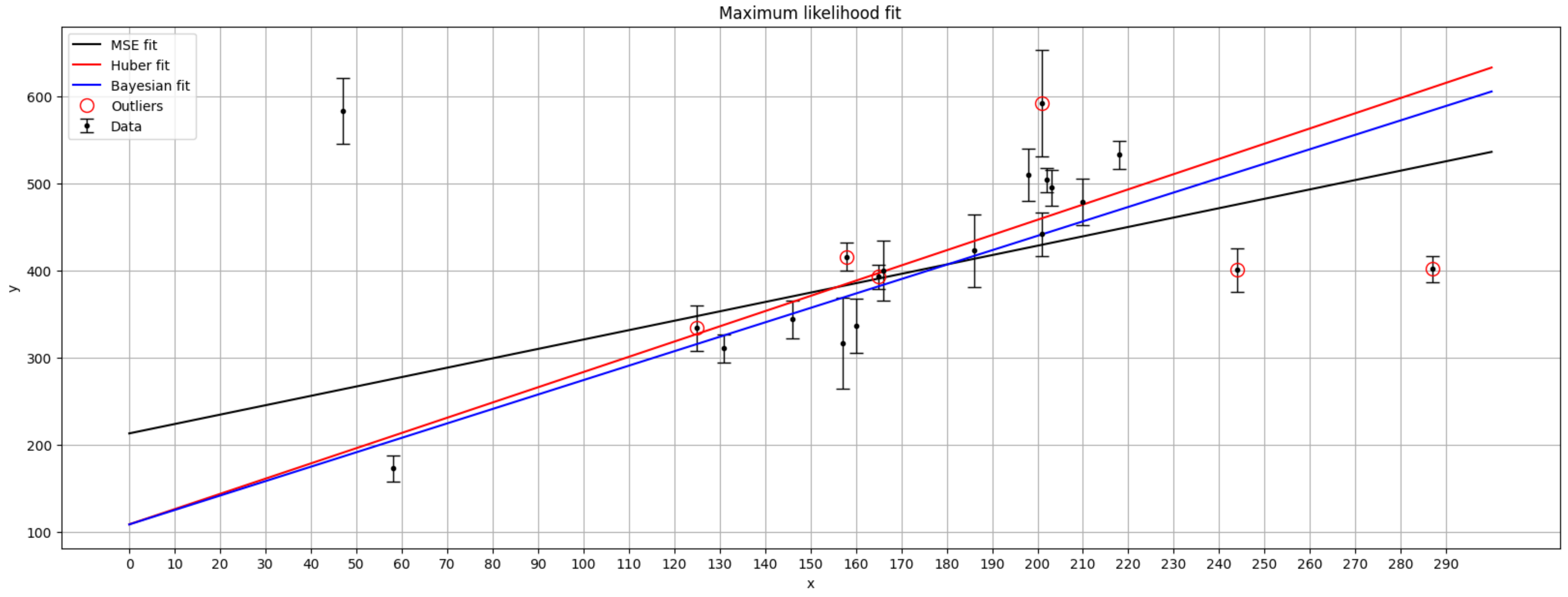
```python
    return np.sum(0.5 * (dy / sigma_y) ** 2)

def huber_loss(t, c=2):
    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, sigma_y=sigma_y, c=3):
    return huber_loss((y - theta[0] - theta[1] * x) / sigma_y, c).sum()

# Find the best-fit parameters using the least-squares method
theta1 = optimize.fmin(mse_loss, [0, 0], disp=False)

# Find the best-fit parameters using the Huber loss function
theta2 = optimize.fmin(total_huber_loss, [0, 0], disp=False)

# Print the results of the fits
print(f'MSE fit: m = {theta1[1]:.3f}, b = {theta1[0]:.3f}')
print(f'Huber fit: m = {theta2[1]:.3f}, b = {theta2[0]:.3f}')

xfit = np.linspace(0, np.round(x.max(), -2), 1000)
plt.figure(figsize=(20, 7))
plt.errorbar(x, y, yerr=sigma_y, fmt='.k', capsize=5, lw=1, label='Data')
plt.plot(xfit, theta1[1] * xfit + theta1[0], '-k', label='MSE fit')
plt.plot(xfit, theta2[1] * xfit + theta2[0], '-r', label='Huber fit')
plt.title('Maximum likelihood fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()

# Bayesian Marginalization
def log_prior(theta):
    # m, b = theta
    if 0 < all(theta[2:]) < 1:
        return 0.0
    else:
        return -np.inf

def log_likelihood(theta, x, y, sigma_y, sigmaB):
    dy = y - (theta[0] + theta[1] * x)
    clipped_dat = np.clip(theta[2:], 0, 1)
    log_l1 = np.log(clipped_dat) - 0.5 * np.log(2 * np.pi * sigma_y ** 2) - 0.5 * (dy / sigma_y) ** 2
    log_l2 = np.log(1 - clipped_dat) - 0.5 * np.log(2 * np.pi * sigmaB ** 2) - 0.5 * (dy / sigmaB) ** 2
    return np.sum(np.logaddexp(log_l1, log_l2))

def log_probability(theta, x, y, sigma_y, sigmaB):
    lp = log_prior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + log_likelihood(theta, x, y, sigma_y, sigmaB)
```

```python
ndim = 2 + len(x)
nwalkers = 100
nburn = 1000
nsteps = 15000

pos = np.zeros((nwalkers, ndim))
pos[:, :2] = np.random.normal(theta2, 1, (nwalkers, 2))
pos[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_probability, args=[x, y, sigma_y, 10])
sampler.run_mcmc(pos, nsteps, progress=True)
samples = sampler.get_chain(discard=nburn, thin=15, flat=True)
# plt.figure(figsize=(15, 5))
# plt.plot(samples[:, 0], samples[:, 1], 'k.', alpha=0.1)
# plt.xlabel('m')
# plt.ylabel('b')
# plt.show()

theta3 = np.mean(samples[:, :2], axis=0)
g = np.mean(samples[:, 2:], axis=0)
outliers = (g < 0.499)
plt.figure(figsize=(20, 7))
plt.errorbar(x, y, yerr=sigma_y, fmt='.k', capsize=5, lw=1, label='Data')
plt.plot(xfit, theta1[1] * xfit + theta1[0], '-k', label='MSE fit')
plt.plot(xfit, theta2[1] * xfit + theta2[0], '-r', label='Huber fit')
plt.plot(xfit, theta3[1] * xfit + theta3[0], '-b', label='Bayesian fit')
plt.plot(x[outliers], y[outliers], 'ro', ms=10, mfc='none', mec='red', label='Outliers')
plt.title('Maximum likelihood fit')
plt.xlabel('x')
plt.ylabel('y')
plt.xticks(np.arange(0, 300, 10))
# plt.yticks(np.arange(100, 630, 10))
plt.legend()
plt.grid()
```

```
     x    y  sigmay
0  201  592      61
1  244  401      25
2   47  583      38
3  287  402      15
4  203  495      21
MSE fit: m = 1.077, b = 213.274
Huber fit: m = 1.747, b = 108.948
```



```
  0%|          | 0/15000 [00:00<?, ?it/s]/home/kali1tanmay/.local/lib/python3.10/site-packages/emcee/moves/red_blue.py:99: RuntimeWarning: invalid value enc
ountered in double_scalars
  lnpdiff = f + nlp - state.log_prob[j]
100%|██████████| 15000/15000 [00:25<00:00, 577.16it/s]
```


Maximum likelihood fit
MSE fit
Huber fit
Bayesian fit
Outliers
Data
y
x
100
200
300
400
500
600
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290