#QUESTION\_1

import statistics as statistics

from uncertainties import ufloat

from uncertainties.umath import \*

from cmath import pi

Eddington = ufloat(1.61, 0.40)

Crommelin = ufloat(1.98, 0.16)

ProbabilityEddingtonvalue = Eddington/pi

ProbabilityCrommelinvalue = Crommelin/pi

Priorodds = ProbabilityEddingtonvalue/ProbabilityCrommelinvalue

print('The value of prior odds = ', Priorodds)

PH1givenD\_Einstein = 1.74/pi

PH0givenD\_Newton = 0.87/pi

Posteriorodds = PH0givenD\_Newton/PH1givenD\_Einstein

print( 'The value of posterior odds, ', Posteriorodds)

Bayesfactor = Posteriorodds/Priorodds

print('the value of Bayes factor is ', Bayesfactor)

OUTPUT

The value of prior odds = 0.81+/-0.21

The value of posterior odds, 0.5

the value of Bayes factor is 0.61+/-0.16

#QUESTION\_2

import emcee

import matplotlib.pyplot as plt

import numpy as np

data = np.loadtxt('C:\\Users\\Heera Baiju\\Desktop\\Q2data.txt')

xdata = data[:,1]

ydata = data[:,2]

error = data[:,3]

def compute\_sigma\_level(trace1, trace2, nbins = 20):

    L, xbins, ybins = np.histogram2d(trace1, trace2, nbins)

    L[L == 0] = 1E-16

    shape = L.shape

    L = L.ravel()

    i\_sort = np.argsort(L)[::-1]

    i\_unsort = np.argsort(i\_sort)

    L\_cumsum = L[i\_sort].cumsum()

    L\_cumsum /= L\_cumsum[-1]

    xbins = 0.5 \* (xbins[1:] + xbins[:-1])

    ybins = 0.5 \* (ybins[1:] + ybins[:-1])

    return xbins, ybins, L\_cumsum[i\_unsort].reshape(shape)

def plot\_MCMC\_trace(ax, trace, scatter = False, \*\*kwargs):

    xbins, ybins, sigma = compute\_sigma\_level(trace[0], trace[1])

    ax.contour(xbins, ybins, sigma.T, levels = [0.683, 0.955], \*\*kwargs)

    if scatter:

        ax.plot(trace[0], trace[1], ',k', alpha = 0.1)

    ax.set\_xlabel('m')

    ax.set\_ylabel('b')

def plot\_MCMC\_results(trace, colors = 'k'):

    fig, ax = plt.subplots(1, 1, figsize = (8, 5))

    plt.title('68% and 95% joint confidence intervals on b and m')

    plot\_MCMC\_trace(ax, trace, True, colors = colors)

def log\_prior(theta):

    beta = theta

    return -1.5 \* np.log(1 + beta \*\* 2)

def log\_likelihood(theta, x, y):

    alpha, beta = theta

    y\_model = alpha + beta \* x

    return -0.5 \* np.sum(np.log(2 \* np.pi \* error \*\* 2) + (y - y\_model) \*\* 2 / error \*\*

2)

def log\_posterior(theta, x, y):

    return log\_prior(theta) + log\_likelihood(theta, x, y)

ndim = 2

nwalkers = 50

nburn = 1000

nsteps = 2000

np.random.seed(0)

starting\_guesses = np.random.random((nwalkers, ndim))

sampler = emcee.EnsembleSampler(nwalkers, ndim, log\_posterior, args = [xdata,

ydata])

sampler.run\_mcmc(starting\_guesses, nsteps)

emcee\_trace = sampler.chain[:, nburn:, :].reshape(-1, ndim).T

plot\_MCMC\_results(emcee\_trace)

plt.show()

OUTPUT

Chart

Description automatically generated

#QUESTION\_3

import emcee

import numpy as np

from scipy import optimize

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv("C:\\Users\\Heera Baiju\\Desktop\\Q3data.csv")

x = data['x'].values

y = data['y'].values

e = data['sigma\_y'].values

xfit = np.linspace(0, 300, 1000)

t = np.linspace(-20, 20)

def squared\_loss(theta, x = x, y = y, e = e):

    dy = y - theta[0] - theta[1] \* x

    return np.sum(0.5 \* (dy / e) \*\* 2)

theta1 = optimize.fmin(squared\_loss, [0, 0], disp = False)

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 = e, c = 3):

    return huber\_loss((y - theta[0] - theta[1] \* x) / e, c).sum()

theta2 = optimize.fmin(total\_huber\_loss, [0, 0], disp = False)

plt.errorbar(x, y, e, fmt = '.k', ecolor = 'gray')

plt.plot(xfit, theta1[0] + theta1[1] \* xfit, color = 'lightgray')

plt.plot(xfit, theta2[0] + theta2[1] \* xfit, color = 'black')

plt.title('Maximum Likelihood fit: Huber loss')

plt.show()

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)

ndim = 2 + len(x)

nwalkers = 50

nburn = 10000

nsteps = 15000

np.random.seed(4)

starting\_guesses = np.zeros((nwalkers, ndim))

starting\_guesses[:, :2] = np.random.normal(theta2, 1, (nwalkers, 2))

starting\_guesses[:, 2:] = np.random.normal(0.5, 0.1, (nwalkers, ndim - 2))

sampler = emcee.EnsembleSampler(nwalkers, ndim, log\_posterior, args=[x, y, e,

50])

sampler.run\_mcmc(starting\_guesses, nsteps)

sample = sampler.chain # shape = (nwalkers, nsteps, ndim)

sample = sampler.chain[:, nburn:, :].reshape(-1, ndim)

theta3 = np.mean(sample[:, :2], 0)

g = np.mean(sample[:, 2:], 0)

outliers = (g < 0.38)

# Plotting

plt.errorbar(x, y, e, fmt = '.k', ecolor = 'gray')

plt.plot(xfit, theta1[0] + theta1[1] \* xfit, color = 'lightgray')

plt.plot(xfit, theta2[0] + theta2[1] \* xfit, color = 'lightgray')

plt.plot(xfit, theta3[0] + theta3[1] \* xfit, color = 'black')

plt.plot(x[outliers], y[outliers], 'ro', ms=20, mfc = 'none', mec='red')

plt.title('Maximum Likelihood fit: Bayesian Marginalization')

plt.show()

OUTPUT

Chart, line chart

Description automatically generated

Chart

Description automatically generated