Assignment-7

Q1

```
#-------------------------------------------------------------
# Importing all libraries
import emcee
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#-------------------------------------------------------------
# Defining the data
data = pd.read_csv('E:\Data Science Analysis\Assignment-7\SPT.csv')
xdata = data['#z'].values
ydata = data['fgas'].values
e = data['fgas_error'].values

#-------------------------------------------------------------
# Defining the required functions for computing sigma level
def compute_sigma_level(trace1, trace2, nbins = 20):
    # From a set of traces, bin by number of standard deviations
    L, xbins, ybins = np.histogram2d(trace1, trace2, nbins)
    L[L == 0] = 1E-16
    shape = L.shape
    L = L.ravel()

    # Obtain the indices to sort and unsort the flattened array
    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)
```

```
#------------------------------------------------------------
# Defining the required functions for plot MCMC trace
def plot_MCMC_trace(ax, trace, scatter = False, **kwargs):
    # Plot traces and contours
    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')

#------------------------------------------------------------
# Defining the required functions for plot MCMC results
def plot_MCMC_results(trace, colors = 'k'):
    # Plot both the trace and the model together
    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)

#------------------------------------------------------------
# Defining the required functions for log prior
def log_prior(theta):
    beta = theta
    return -1.5 * np.log(1 + beta ** 2)

#------------------------------------------------------------
# Defining the required functions for log likelihood
def log_likelihood(theta, x, y):
    alpha, beta = theta
    y_model = alpha + beta * x
    return -0.5 * np.sum(np.log(2 * np.pi * e ** 2) + (y - y_model) ** 2 / e **
2)

#------------------------------------------------------------
# Defining the required functions for log posterior
def log_posterior(theta, x, y):
    return log_prior(theta) + log_likelihood(theta, x, y)


ndim = 2  # Number of parameters in the model
nwalkers = 50  # Number of MCMC walkers
nburn = 1000  # "Burn-in" period to let chains stabilize
nsteps = 2000  # Number of MCMC steps to take

# Set theta near the maximum likelihood
```

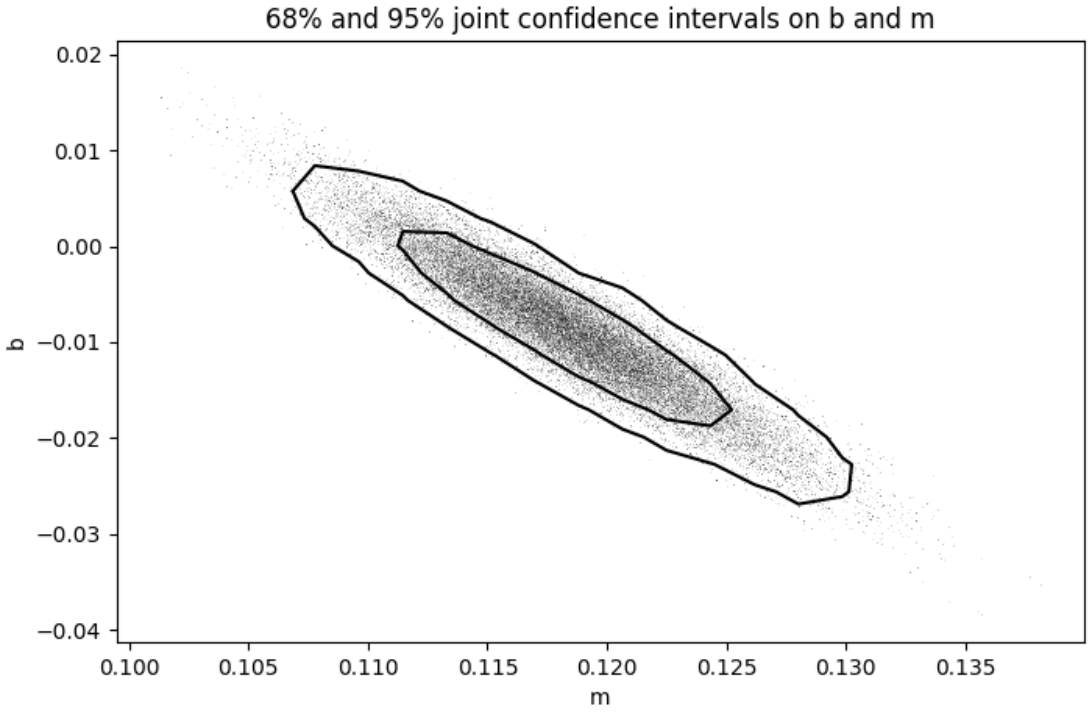
```
np.random.seed(0)
starting_guesses = np.random.random((nwalkers, ndim))

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



Q2

```
#------------------------------------------------------------
# Importing all libraries
import nestle
import numpy as np
from scipy import stats

global data, x, y, sigma_y

#------------------------------------------------------------
# Define the data array
data = np.array([[ 0.417022004703, 0.720324493442, 0.000114374817345,
0.302332572632,
                   0.146755890817, 0.0923385947688, 0.186260211378,
0.345560727043,
                   0.396767474231, 0.538816734003, 0.419194514403,
0.685219500397,
                   0.204452249732, 0.878117436391, 0.0273875931979,
0.670467510178,
                   0.417304802367, 0.558689828446, 0.140386938595, 0.198101489085
],
                 [ 0.121328306045, 0.849527236006, -1.01701405804, -
0.391715712054,
                  -0.680729552205, -0.748514873007, -0.702848628623, -
0.0749939588554,
                   0.041118449128, 0.418206374739, 0.104198664639, 0.7715919786,
                  -0.561583800669, 1.43374816145, -0.971263541306,
0.843497249235,
                  -0.0604131723596, 0.389838628615, -0.768234900293, -
0.649073386002 ],
                 [ 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

#------------------------------------------------------------
# Defining the required functions for polynomial fitting
def polynomial_fit(theta, x):
    # Polynomial model of degree (len(theta) - 1)
```

```
    return sum(t * x ** n for (n, t) in enumerate(theta))

#-------------------------------------------------------------
# Defining the required logL function
def logL(theta):
    # Gaussian log-likelihood of the model at theta
    y_fit = polynomial_fit(theta, x)
    return sum(stats.norm.logpdf(*args) for args in zip(y, y_fit, sigma_y))

#-------------------------------------------------------------
# Defining the required prior transform function
def prior_transform(x):
    return 10.0 * x - 5.0

#-------------------------------------------------------------
# Run nested sampling
Linear = nestle.sample(logL, prior_transform, 2)
Quadratic = nestle.sample(logL, prior_transform, 2)

#-------------------------------------------------------------
# Print the Bayesian evidence
print("Bayesian evidence for the linear model: ", Linear.logz)
print("Bayesian evidence for the quadratic model: ", Quadratic.logz)
```

Output

Bayesian evidence for the linear model: 13.45095553633228

Bayesian evidence for the quadratic model: 13.027698766948637

Q3

```
#-------------------------------------------------------------
# Importing all libraries
import csv
import numpy as np
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
from sklearn.neighbors import KernelDensity

#-------------------------------------------------------------
# Converting the dat file to csv file
# Read SDSS_quasar.dat to a list of lists
datContent = [i.strip().split() for i in open("E:\Data Science
Analysis\Assignment-7\SDSS_quasar.dat").readlines()]

#-------------------------------------------------------------
# Write it as a new CSV file
with open("./SDSS_quasar.csv", "w") as f:
    writer = csv.writer(f)
    writer.writerows(datContent)

#-------------------------------------------------------------
# Selecting the required column
data = pd.read_csv('SDSS_quasar.csv', usecols=['z'])
data = data.values
t = np.linspace(-0.5, 5.5, 100)

#-------------------------------------------------------------
# Computing the KDE values
kde1 = KernelDensity(kernel='gaussian', bandwidth=0.2).fit(data)
kde1 = kde1.score_samples(t.reshape(-1,1))

kde2 = KernelDensity(kernel='exponential', bandwidth=0.2).fit(data)
kde2 = kde2.score_samples(t.reshape(-1,1))
dist = norm(np.mean(data), np.std(data)).pdf(t.reshape(-1,1))

plt.plot(t, np.exp(kde1), label='gaussian kernel')
plt.plot(t, np.exp(kde2), label='exponential kernel')
plt.fill(t.reshape(-1,1), dist, fc='black', alpha=0.2,label='input distribution')
```

```
plt.title('Kernerl Density estimation')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.show()
```

Output