Data Science Analysis Assignment - 7

In [2]:

```
#importing required libraries
import pandas as pd
import numpy as np
import emcee
import matplotlib.pyplot as plt
from sklearn.neighbors import KernelDensity
from astroML.plotting import plot_mcmc
import nestle
```

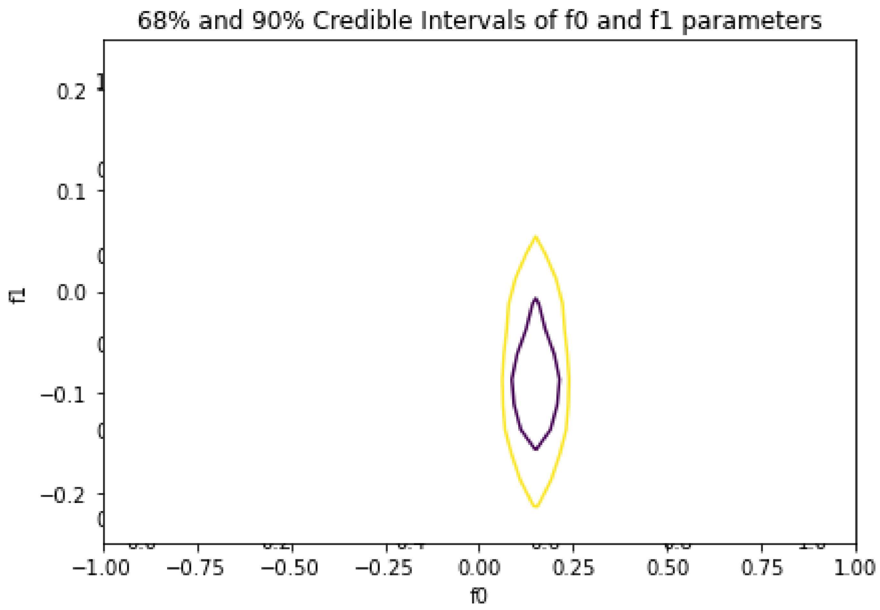
Q1.

In [14]:

```
#importing data from csv file
data = pd.read_csv("D:\CLASSES\SEM 4\Data Science Analysis\A7\data1.csv",sep="\s+")

#writin the data into arrays
z = data['z']
fgas = data['fgas']
fgas_err = data['fgas_error']

#defining required functions
def log_prior(theta):
    f0, f1, sigma = theta
    if sigma > 0 and (0 < f0 < 0.5) and (-0.5 < f1 < 0.5):
        return -1.5 * np.log(1 + (f0*f1) ** 2) - np.log(sigma)
    else:
        return -np.inf

def log_likelihood(theta, x, y):
    f0, f1, sigma = theta
    y_model = f0 + f0*f1 * 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):
    return log_prior(theta) + log_likelihood(theta, x, y)

#MCMC parameters
```

```python
num_dim = 3
num_burn_period = 1000
num_steps = 10000
num_walkers = 50

#initial guess
initial_guess = np.random.random((num_walkers,num_dim))

#begining MCMC algorithm
sampler = emcee.EnsembleSampler(num_walkers, num_dim, log_posterior,args=[z,fgas])
sampler.run_mcmc(initial_guess, num_steps, progress = True)
sample = sampler.chain[:, num_burn_period:, :].reshape(-1, num_dim)

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

#printing the parameters
print("The estimated value of the parameter f0 is %s" %(best_theta[0]))
print("The estimated value of the parameter f1 is %s" %(best_theta[1]))

#plotting
fig = plt.figure(figsize = (6,4))
plt.grid()
ax = plot_mcmc(sample.T[:2], fig=fig, limits=[(-1, 1), (-0.25, 0.25)], levels=[0.68, 0.90], labels=["f0", "f1"])
plt.title("68% and 90% Credible Intervals of f0 and f1 parameters");
```

```
  0%|          | 0/10000 [00:00<?, ?it/s]C:\Users\Swetha\anaconda3\lib\site-packages\emcee\moves\red_blue.py:99: RuntimeWarning: i
nvalid value encountered in double_scalars
  lnpdiff = f + nlp - state.log_prob[j]
100%|██████████| 10000/10000 [09:27<00:00, 17.61it/s]
The estimated value of the parameter f0 is 0.20723426425141844
The estimated value of the parameter f1 is -0.02617451364882282
```



Q2.

In [36]:

```
data = np.array([[ 0.42, 0.72, 0.  , 0.3 , 0.15,
                  0.09, 0.19, 0.35, 0.4 , 0.54,
                  0.42, 0.69, 0.2 , 0.88, 0.03,
                  0.67, 0.42, 0.56, 0.14, 0.2 ],
                 [ 0.33, 0.41, -0.22, 0.01, -0.05,
                  -0.05, -0.12, 0.26, 0.29, 0.39,
                  0.31, 0.42, -0.01, 0.58, -0.2 ,
                  0.52, 0.15, 0.32, -0.13, -0.09 ],
                 [ 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 functions
def polynomial_fit(theta, x):
    """Polynomial model of degree (len(theta) - 1)"""
    return sum(t * x ** n for (n, t) in enumerate(theta))
```

```python
def log_prior(theta):
    return 200 * theta - 100

def log_likelihood(theta, data=data):
    x, y, sigma_y = data
    yM = polynomial_fit(theta, x)
    return -0.5 * np.sum(np.log(2 * np.pi * sigma_y ** 2) + (y - yM) ** 2 / sigma_y ** 2)

#calculating bayesian evidenence for linear and quadrartic model
np.random.seed(1)

linear = nestle.sample(log_likelihood, log_prior, 2)
quadratic = nestle.sample(log_likelihood, log_prior, 3)

#printing the values
print("--Linear Model--")
print(linear.summary())

print("\n--Quadratic Model--")
print(quadratic.summary())

print("\nLog-evidence value for Linear Model:    %s" %(linear.logz))
print("Log-evidence value for Quadratic Model: %s" %(quadratic.logz))
```

```
--Linear Model--
niter: 1590
ncall: 2714
nsamples: 1690
logz:  6.981 +/-  0.375
h: 14.089

--Quadratic Model--
niter: 2102
ncall: 3892
nsamples: 2202
logz:  2.706 +/-  0.432
h: 18.644

Log-evidence value for Linear Model:    6.981327447725231
Log-evidence value for Quadratic Model: 2.7057266014815293
```

JVDP's 5th blog article tells the Log-Evidence values for

Linear Model : 46942613.34886921

Quadratic Model : 111116773.89368105

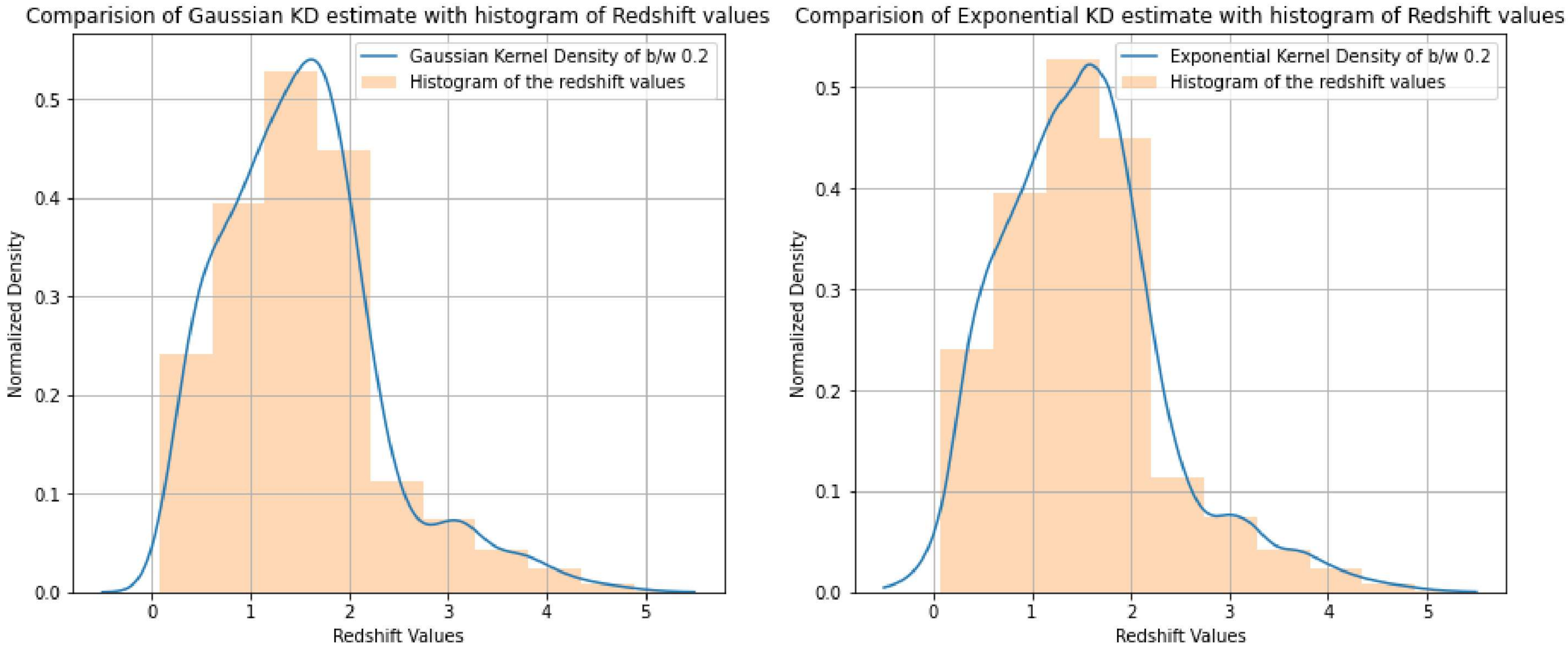
We can see that these values do not match the values we obtained from the nested sampling by "nestle".

Q3.

In [9]:

```
df = pd.read_csv("D:\CLASSES\SEM 4\Data Science Analysis\A7\data2.csv", sep="\s+",usecols=['z'])

redshift = df['z'].values
redshift.shape = (redshift.size,1)

x = np.linspace(-0.5,5.5,len(redshift))
x.shape = (x.size,1)

#KDE estimates
#gaussian
kde1 = KernelDensity(kernel = 'gaussian',bandwidth = 0.2)
kde1.fit(redshift)
den_g = kde1.score_samples(x)

#exponetial
kde2 = KernelDensity(kernel = 'exponential',bandwidth = 0.2)
kde2.fit(redshift)
den_e = kde2.score_samples(x)

#plotting
fig, ax = plt.subplots(1,2,figsize=(15,6))

#gaussian
ax[0].plot(x[:,0],np.exp(den_g),label = "Gaussian Kernel Density of b/w 0.2")
ax[0].hist(redshift, alpha = 0.3, density = True, label = "Histogram of the redshift values")

ax[0].set_title('Comparision of Gaussian KD estimate with histogram of Redshift values')
ax[0].set_xlabel('Redshift Values')
ax[0].set_ylabel('Normalized Density')

ax[0].legend()
ax[0].grid()

#exponential
ax[1].plot(x[:,0],np.exp(den_e),label = "Exponential Kernel Density of b/w 0.2")
ax[1].hist(redshift, alpha = 0.3, density = True, label = "Histogram of the redshift values")
```

```
ax[1].set_title('Comparision of Exponential KD estimate with histogram of Redshift values')
ax[1].set_xlabel('Redshift Values')
ax[1].set_ylabel('Normalized Density')

ax[1].legend()
ax[1].grid()
```



In [13]:

```
fig = plt.figure(figsize = (8,6))
plt.grid()

plt.plot(x[:, 0], np.exp(den_g), label="Gaussian Kernel Density")
plt.plot(x[:, 0], np.exp(den_e), label="Exponential Kernel Density")

plt.title("Comparison between the Gaussian KD and Exponential KD")
plt.xlabel("Redshift values")
plt.ylabel("Normalized values")
```

```
plt.legend()
plt.show()
```