Lab Report

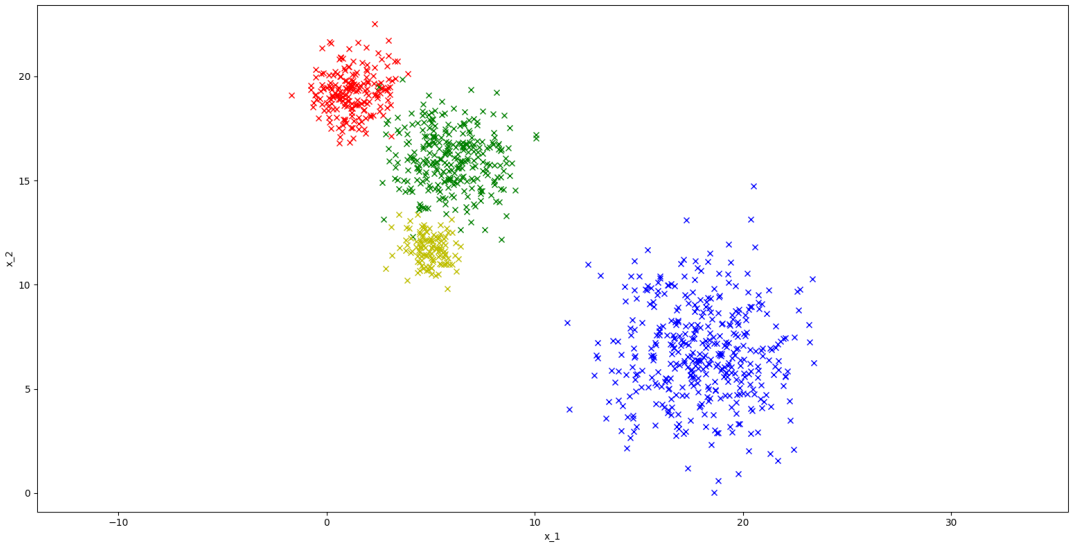
Date: 2 February 2018

LIBIN N GEORGE

111501015

INITIAL MODEL

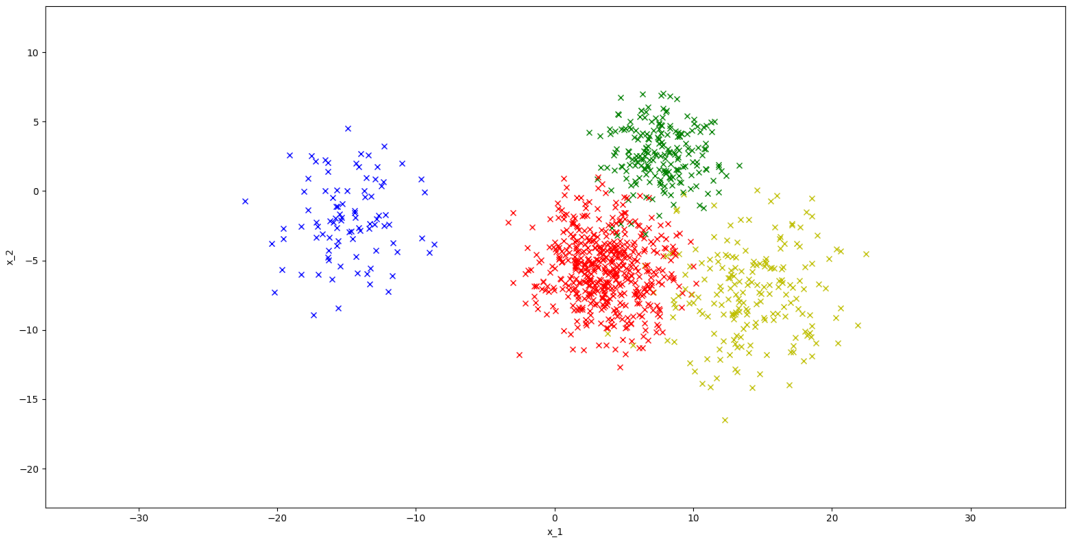
For generating points for a Gaussian Mixture having three clusters closer cluster and one cluster which have larger variance and also away from other clusters. The first model was a simple Gaussian Mixture Model which generates mean for four clusters and generate the points.

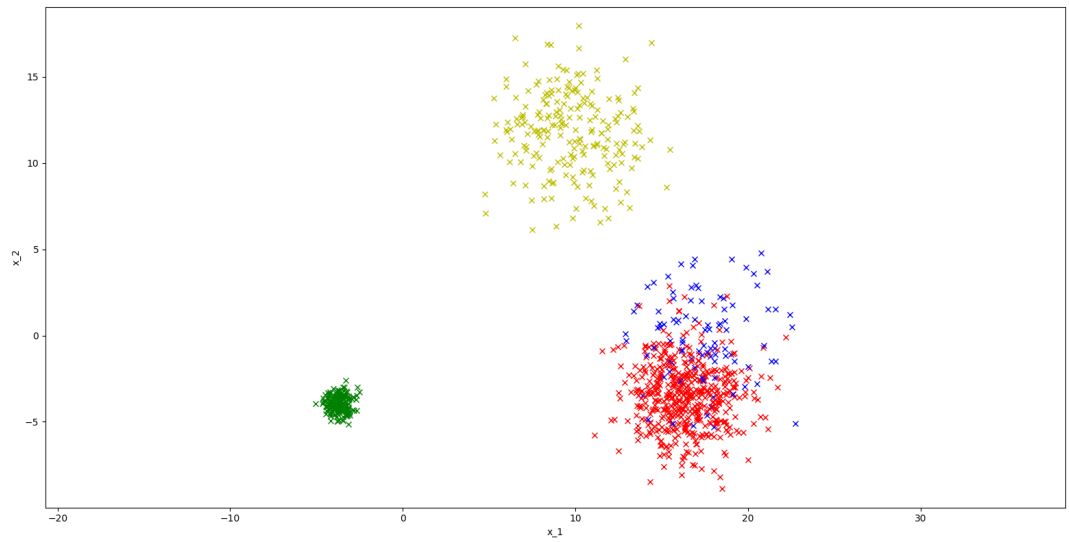


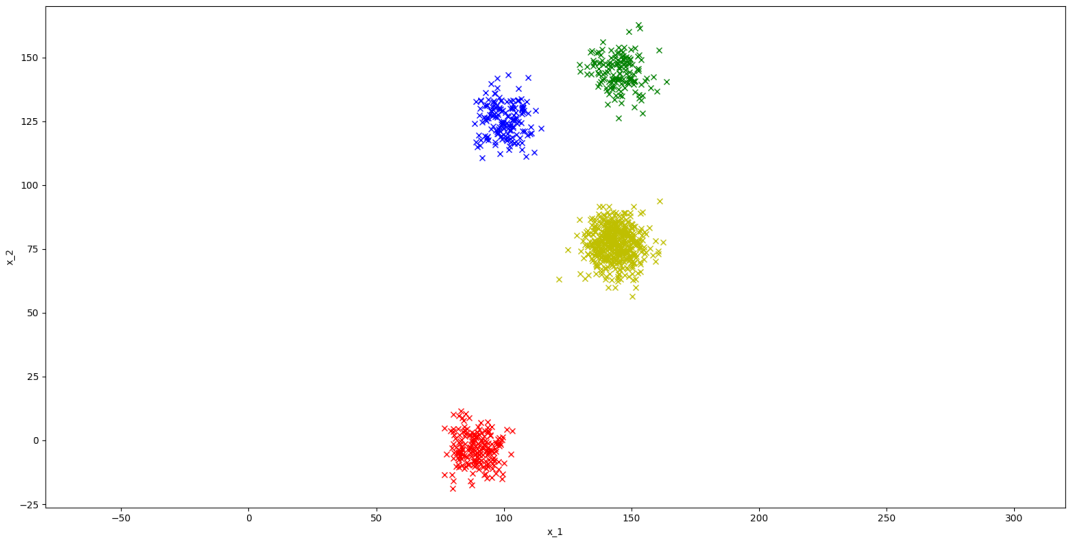
Results for Simple Gaussian Mixture model

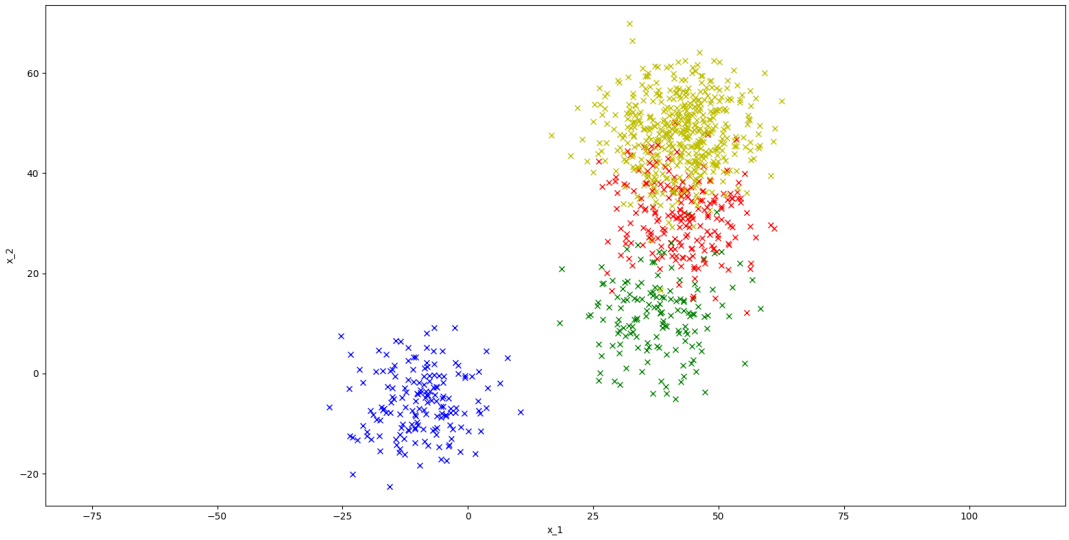
```
import numpy as np
import matplotlib.pyplot as plt

def generate_mu_sigma(mu1, sigma1 ,mu2, sigma2, clusters):
    mu_x = np.random.normal(mu1, sigma1, clusters)
    mu_y = np.random.normal(mu1, sigma1, clusters)
```

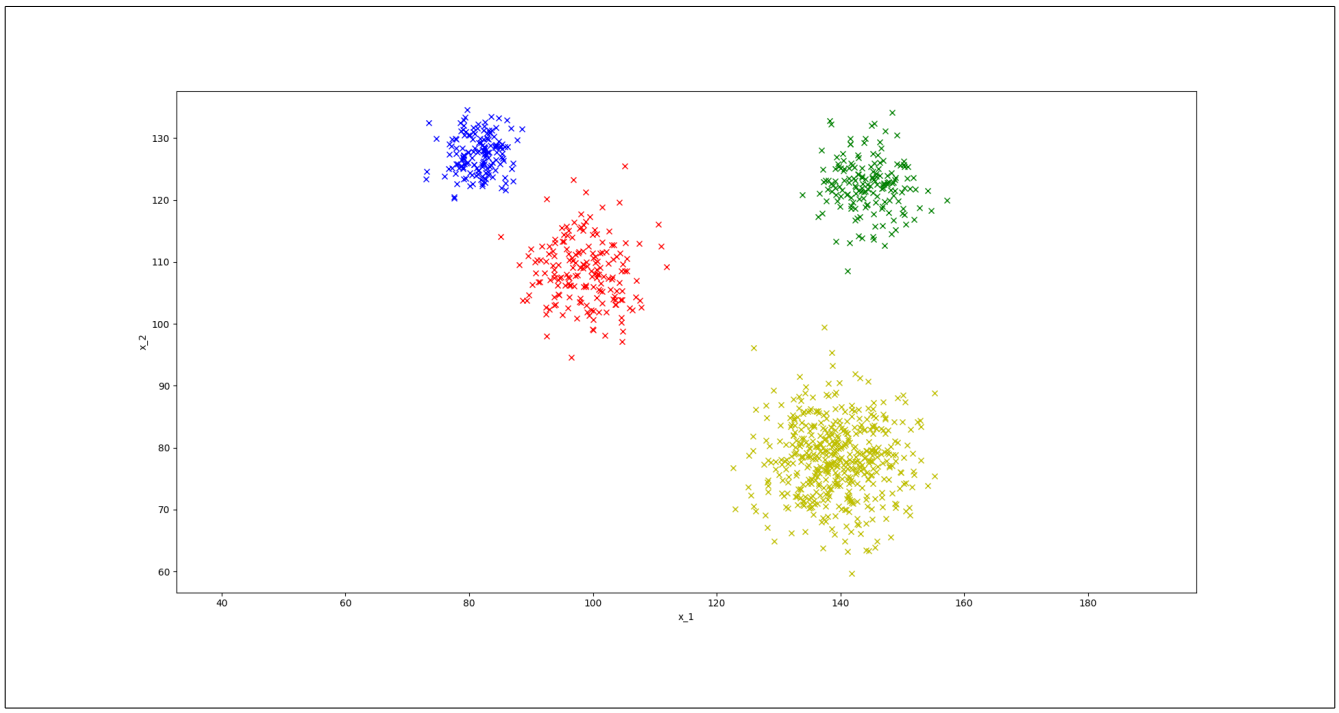
```
    sigma_x = np.random.normal(mu2, sigma2, clusters)
    sigma_y = np.random.normal(mu2, sigma2, clusters)
    mu = []
    sigma = []
    for i in range(clusters):
        mu.append([mu_x[i], mu_y[i]])
        sigma.append([[sigma_x[i], 0], [0, sigma_x[i]]])
    return (mu, sigma)
print generate_mu_sigma(1, 0.5, 1, 1.0, 4)

def generate_z(theta):
    z = []
    z_end=0
    for i in theta:
        z.append(z_end + i)
        z_end = z_end + i
    x  = np.random.random_sample()
    sample = 0
    for i in z:
        if x < i:
            break
        sample += 1
    return sample

print generate_z([0.1, 0.2, 0.5, 0.2])

def generate_x(theta, mu1, mu2, sigma1, sigma2, clusters, num_points = 1000):
    mu, sigma = generate_mu_sigma(mu1, sigma1, mu2, sigma2, clusters)
    xs = []
    ys = []
    color = ['b', 'g', 'r', 'y']
    for i in range(num_points):
        z = generate_z(theta)
        x, y = np.random.multivariate_normal(mu[z], sigma[z], 1).T
        plt.plot(x, y, 'x', c=color[z])
        xs.append(x)
        ys.append(y)
    # plt.plot(xs, ys, 'x')
    plt.axis('equal')
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    # plt.title('lamda = ' + str(lamda) + 'sigma = ' + str(sigma))
    plt.show()

print generate_x([0.1, 0.2, 0.5, 0.2], 1.0, 5.0, 10.0, 2.0, 4)
```


10
5
0
−5
−10
−15
−20
x_2
−30
−20
−10
0
10
20
30
x_1


15
10
5
0
−5
x_2
−20
−10
0
10
20
30
x_1

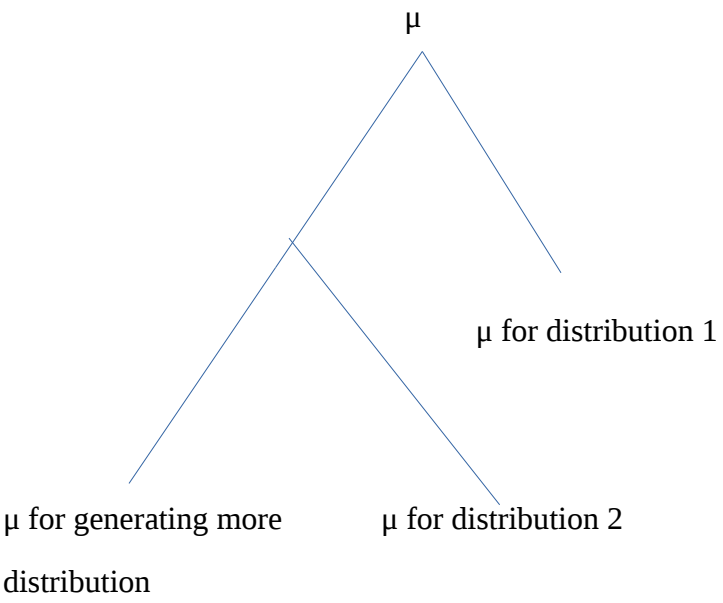

150
125
100
75
50
25
0
−25
x_2
−50
0
50
100
150
200
250
300
x_1


60
40
20
0
−20
x_2
−75
−50
−25
0
25
50
75
100
x_1



Modification of Gaussian Mixture model for Better Results

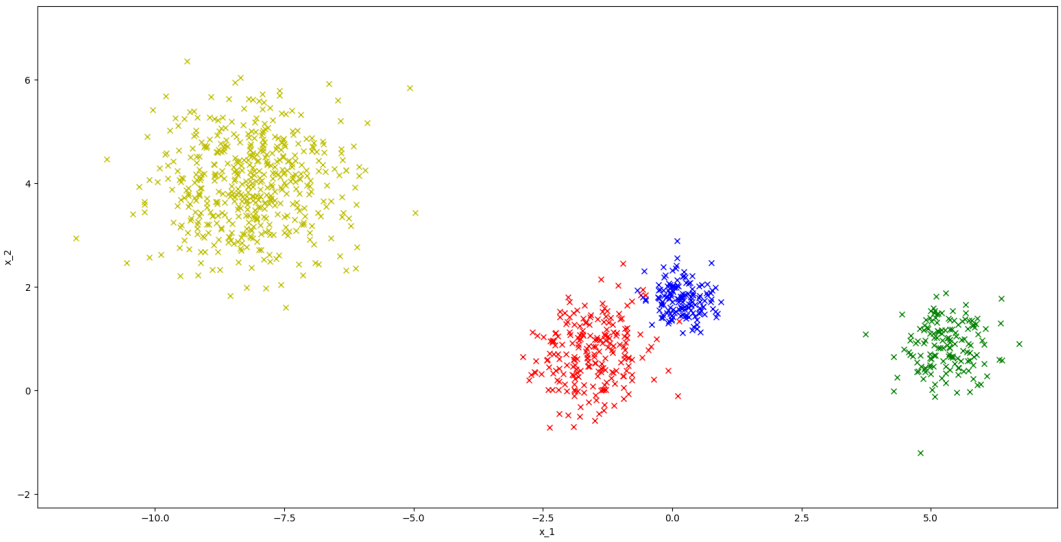
For generating points for a Gaussian Mixture having three clusters closer cluster and one cluster which have larger variance and also away from other clusters. From the given Dataset we can see that one cluster is away from other three clusters. So we have to generate mean for different distributions such that distributions comes closer as more clusters are generated. That is, we have to sample two points (mean for individual distributions) from a given Gaussian distribution and recursively sample another two points with one of the already generated mean (points). Thus we get means of the distribution get closer and closer as more means are generated.

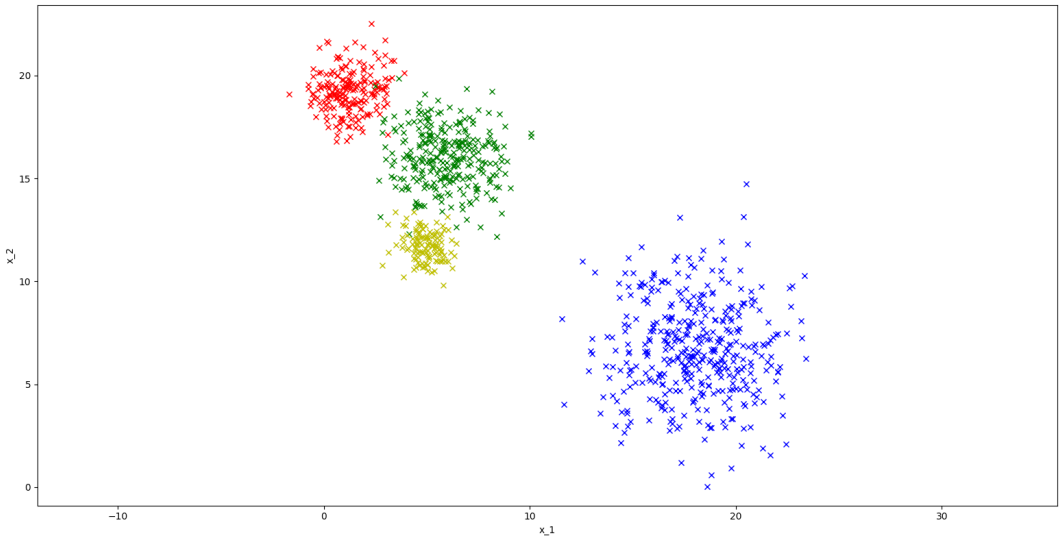


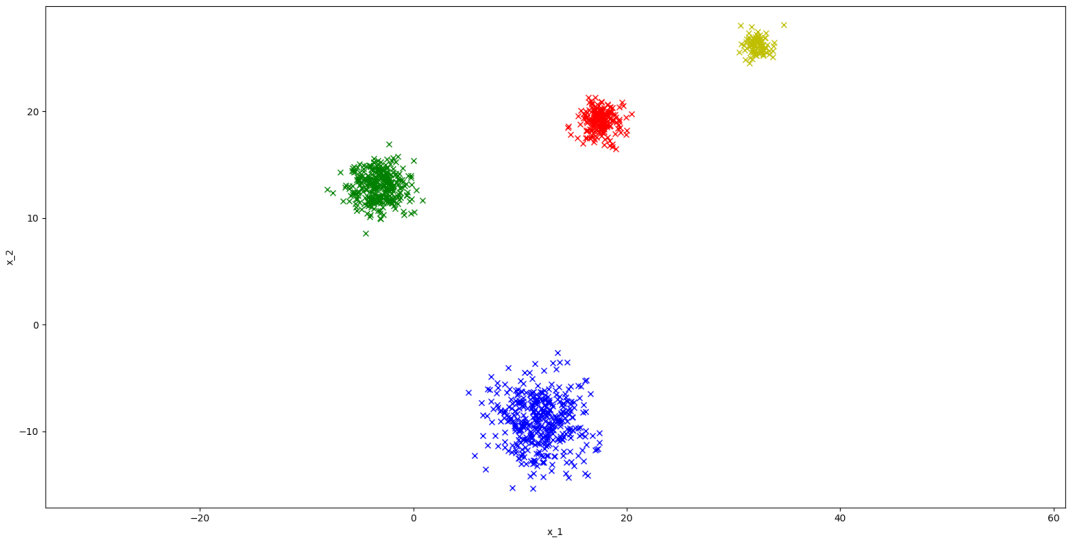
Results of Modified Gaussian Mixture Method

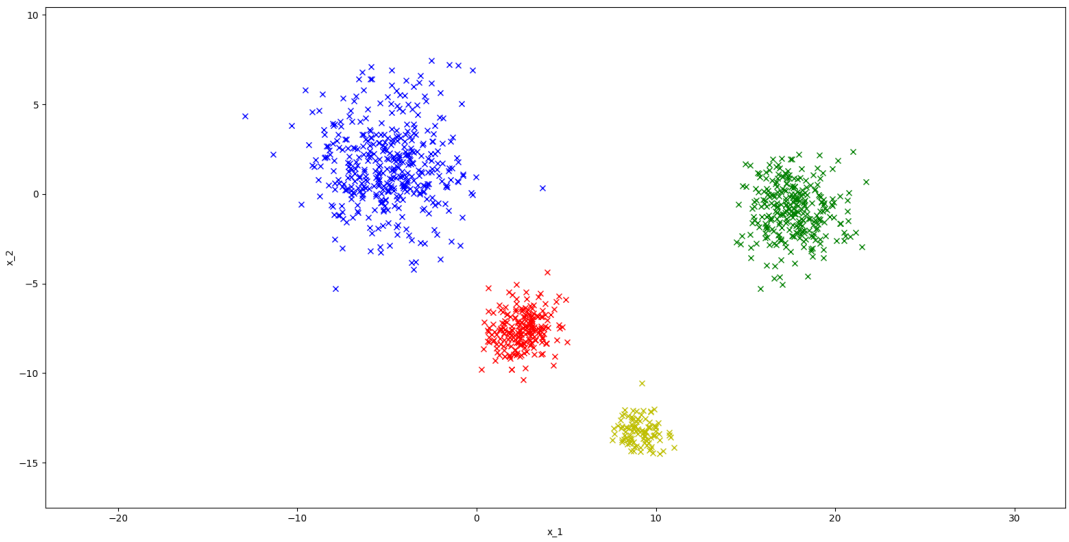
```
import numpy as np
import matplotlib.pyplot as plt

def generate_means(mu1, sigma1 , clusters):
    mu_out = []
    sigma_out = []
    for i in range(clusters-1):
        mu_1, mu1 = np.random.multivariate_normal(mu1, sigma1, 2)
        print mu_1
        sigma1 = np.array(sigma1)*0.5
        mu_out.append(mu_1)
    mu_out.append(mu1)
    return mu_out

print generate_means([2,2], [[10, 0],[0, 10]], 4)

def generate_variances(mu1, sigma1 , clusters):
    sigma_out = []
    for i in range(clusters-1):
        var_1, mu1 = np.random.normal(mu1, sigma1, 2)
        sigma_out.append([[var_1, 0], [0, var_1]])
    sigma_out.append([[mu1, 0], [0, mu1]])
```

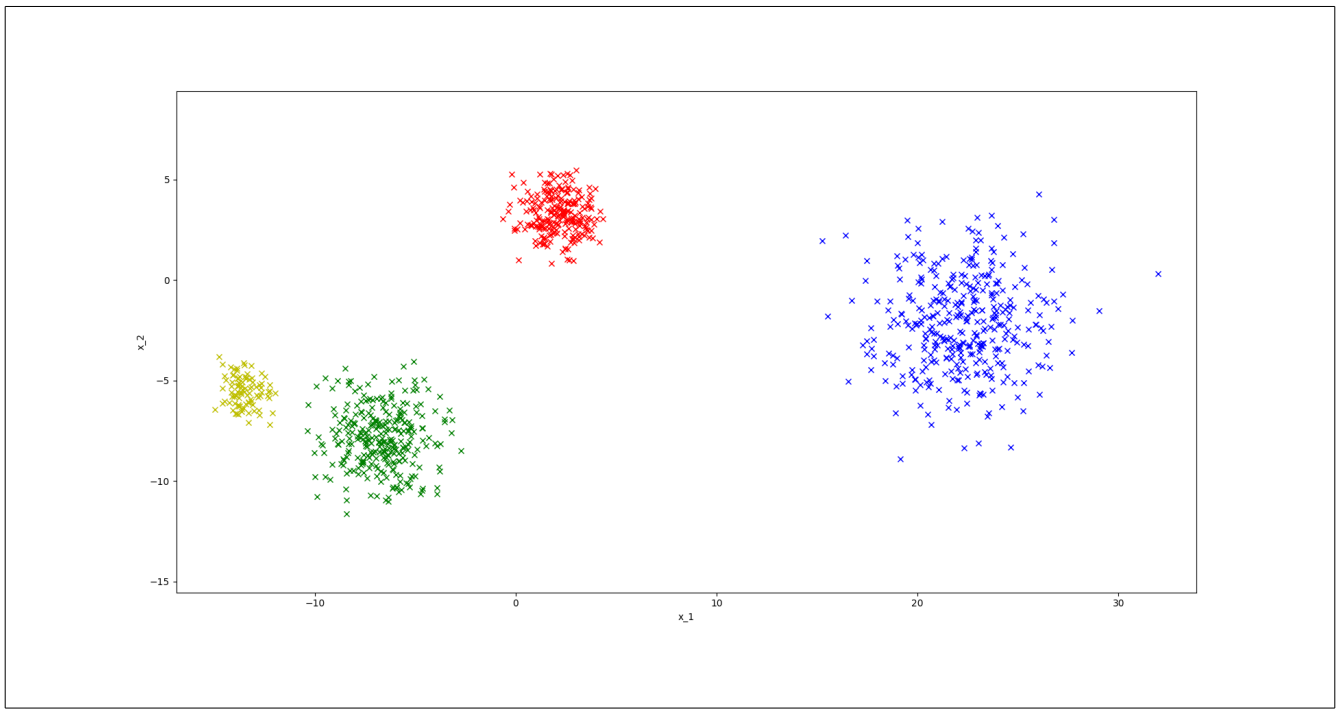
```
    return sigma_out
print generate_variances(5.0, 10.0, 4)

def generate_z(theta):
    z = []
    z_end=0
    for i in theta:
        z.append(z_end + i)
        z_end = z_end + i
    x  = np.random.random_sample()
    sample = 0
    for i in z:
        if x < i:
            break
        sample += 1
    return sample

print generate_z([0.15, 0.15, 0.2, 0.5])

def generate_x(theta, mu1, mu2, sigma1, sigma2, clusters, num_points = 1000):
    mu = generate_means(mu1, sigma1, clusters)
    print mu
    sigma = [[[5, 0],[0, 5]], [[2, 0],[0, 2]], [[1, 0],[0, 1]], [[0.5, 0],[0, 0.5]]]
    xs = []
    ys = []
    color = ['b', 'g', 'r', 'y']
    for i in range(num_points):
        z = generate_z(theta)
        x, y = np.random.multivariate_normal(mu[z], sigma[z], 1).T
        plt.plot(x, y, 'x', c=color[z])
        xs.append(x)
        ys.append(y)
    plt.axis('equal')
    plt.xlabel('x_1')
    plt.ylabel('x_2')
    plt.show()

print generate_x([0.4, 0.3, 0.2, 0.1], [5,5] , 1.0 , [[100.0, 0], [0, 100.0]], 4.0, 4)
```


x_2
x_1
6
4
2
0
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0


x_2
x_1
20
15
10
5
0
−10
0
10
20
30


x_2
x_1
20
10
0
−10
−20
0
20
40
60


x_2
x_1
10
5
0
−5
−10
−15
−20
−10
0
10
20
30


5
0
−5
−10
−15
x_2
−10
0
10
20
30
x_1