1. Gibbs Sampling with a Gaussian Mixture Model

May 2, 2017

Use conjugate priors in a Gaussian mixture model for clustering. Put a Dirichlet prior on the cluster probabilities, a multivariate Normal on the cluster means, and independent Gamma distributions on the variances of each feature.

1 a) Implement a Gibbs sampler to generate samples from the posterior distribution of cluster memberships.

First, I will use K-means to initialize the clusters intelligently.

```
In [1]: import numpy
        import math
        import random

        def Kmeans(data, K):
            """ Initializes the clusters intelligently using K-means. Copied from homework #4. '

            numpy.random.seed(1)
            # randomly generate K means
            means = []
            for k in range(K):
                meank = []
                # for each attribute, generate a mean between the min and max value of that attr
                for col in range(data.shape[1]):
                    meank.append(random.uniform(numpy.amin(data[:, col]), numpy.amax(data[:, col
                # add this mean to the list
                means.append(meank)
            means = numpy.array(means)

            # store whether the cluster assignments have been changed in a given iteration
            changed = True

            # while the cluster assignments are still changing, assign values to their nearest c
            while changed:
                # set changed to False for this round
                changed = False
```

```
            # calculate the Euclidian distance from each instance to each cluster mean
            dists = numpy.empty((data.shape[0], K))
            for k in range(K):
                dists[:, k] = numpy.apply_along_axis(lambda x: math.sqrt(numpy.sum((x-means[

            # find the nearest cluster center for each instance
            clusters = numpy.argmin(dists, axis = 1)

            # calculate the new mean of each cluster
            for k in range(K):
                clusterk = 1*(clusters == k)
                # if this cluster doesn't have any instances in it, try again
                if numpy.sum(clusterk) == 0:
                    return Kmeans(data, K)
                # loop through each attribute
                for col in range(data.shape[1]):
                    meanka = numpy.sum(data[:, col]*clusterk)/numpy.sum(clusterk)
                    # if the new mean of this cluster is different from the old mean, turn c
                    if meanka != means[k, col]:
                        changed = True
                    # add the new mean to the array
                    means[k, col] = meanka

        # once the cluster assignments stop changing, return the assignments and the means
        return clusters, means
```

Now I can use clusters generated by K-means to start Gibbs sampling.

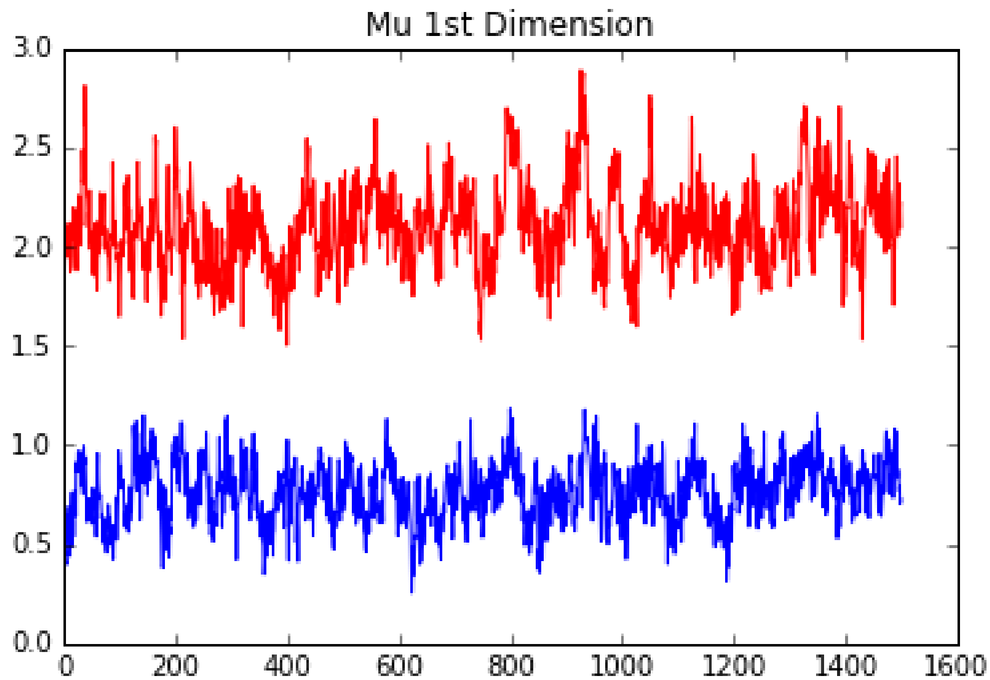
```
In [2]: import scipy.stats
        import matplotlib.pyplot as plt

        def GibbsSample(data, K, iters, prior_alpha, prior_mean, prior_variance, prior_a, prior_
            """ Iteratively samples cluster assignments and parameter values to approximate samp

            numpy.random.seed(1)

            # run K-means on the data
            clusters, means = Kmeans(data, K)

            # an empty array of cluster assignments to hold the results of each iteration
            Allclusters = numpy.empty((iters+1, data.shape[0]))
            # and put the initial clusters from K-means in it
            Allclusters[0, :] = clusters
            # an empty array of cluster proportions
            pis = numpy.empty((iters, K))
            # cluster means
            mus = numpy.empty((iters, K, data.shape[1]))
            # and cluster variances
```

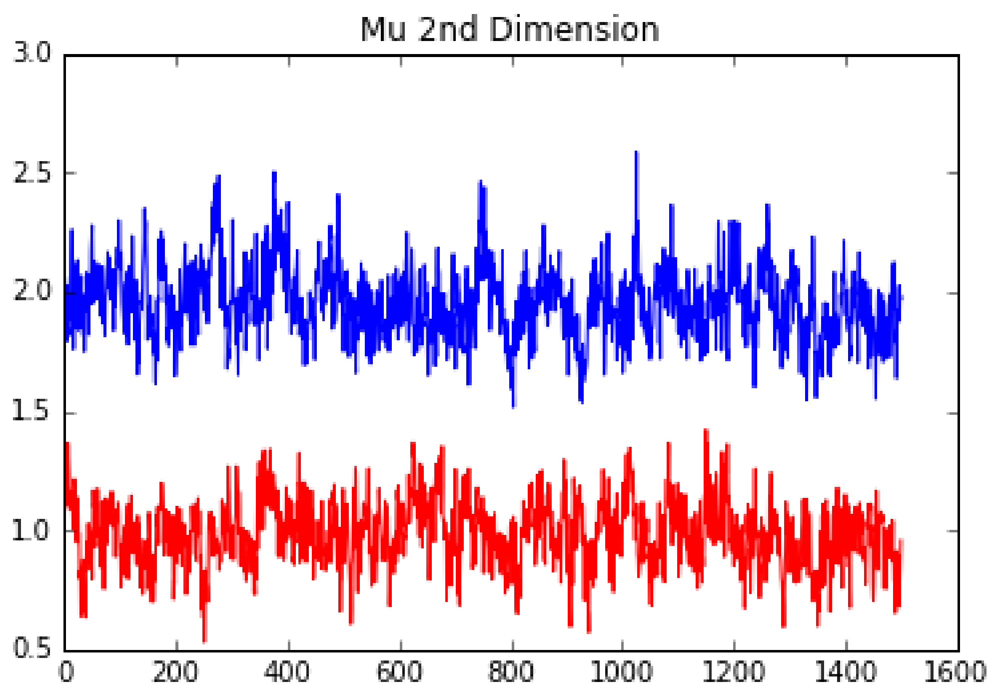
```
sigmas = numpy.empty((iters, K, data.shape[1]))
# also save the log likelihood
likelihood = numpy.zeros(iters)

# now iteratively sample for the designated number of iterations
for i in range(iters):
    # create a list of alpha parameters on the Dirichlet so those can be handled tog
    Dirichlet_alpha = numpy.empty(K)

    # update the other parameters for each cluster separately
    for k in range(K):
        # the number of instances in this cluster
        nk = numpy.sum(1*(Allclusters[i, :] == k))
        # if a cluster is empty or only has one instance in it, sample from the pric
        if nk <= 5:
            mus[i, k, :] = numpy.random.multivariate_normal(prior_mean*numpy.ones(da
                                                            prior_variance*numpy.ide
            for d in range(data.shape[1]):
                sigmas[i, k, d] = numpy.random.gamma(prior_a, 1/prior_b)
        # otherwise update the parameters and sample from the posterior
        else:
            # the variance among instances in this cluster
            hk = numpy.linalg.inv(numpy.diag(numpy.var(data[Allclusters[i, :] == k,

            # update the prior parameters to posteriors based on the current cluster
            # first the parameters on mu
            Normal_variance = numpy.linalg.inv(numpy.identity(hk.shape[0])*(prior_va
            Normal_mean = numpy.dot(prior_variance*prior_mean + nk*numpy.dot(hk,
                                    numpy.mean(data[Allclusters[i, :] == k, :], axis
            # now I can draw a value of mu
            mus[i, k, :] = numpy.random.multivariate_normal(Normal_mean, Normal_vari

            # update the parameters on the variance
            Gamma_a = prior_a + nk/2
            Gamma_b = prior_b + numpy.sum(numpy.apply_along_axis(lambda d: (d - mus[
            # sample from the posterior on sigma
            for d in range(data.shape[1]):
                sigmas[i, k, d] = numpy.random.gamma(Gamma_a, 1/Gamma_b[d])

        # update the parameters on pi
        Dirichlet_alpha[k] = prior_alpha + nk

    # sample from the posterior on pi for all of the clusters at once
    pis[i, :] = numpy.random.dirichlet(Dirichlet_alpha)

    # use the new parameter values to update the cluster assignments
    for n in range(data.shape[0]):
        qn = numpy.empty(K)
```

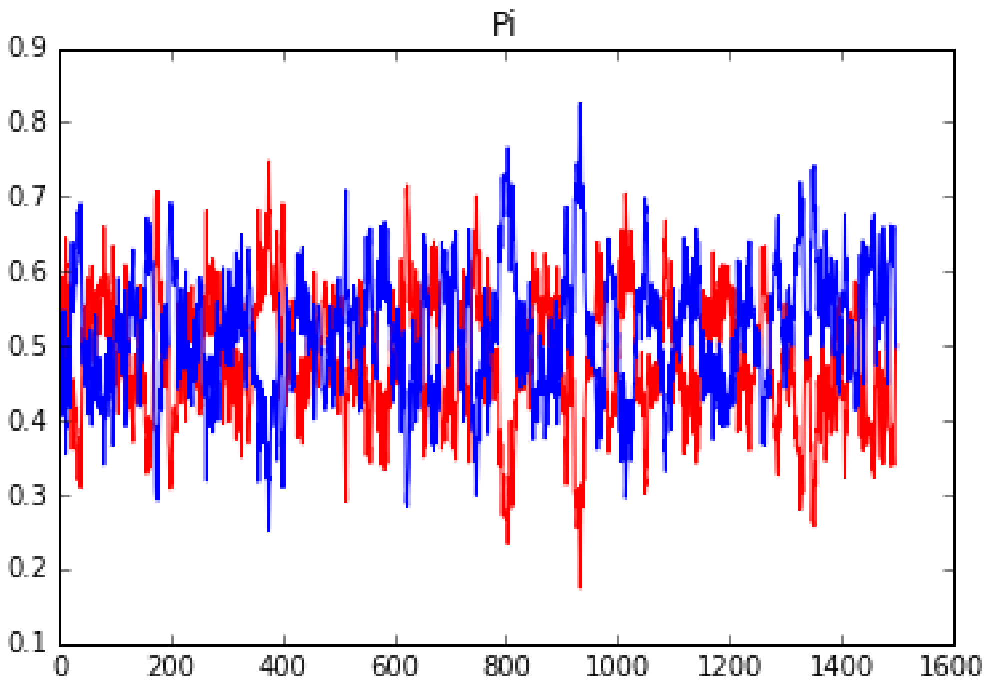
```
                for k in range(K):
                    qn[k] = pis[i, k]*scipy.stats.multivariate_normal.pdf(data[n, :], mean =
                qn = qn/numpy.sum(qn)
                z = int(numpy.random.choice(numpy.arange(K), p = qn))
                Allclusters[i+1, n] = z
                likelihood[i] += numpy.log(qn[z])

        return Allclusters, pis, mus, sigmas, likelihood
```

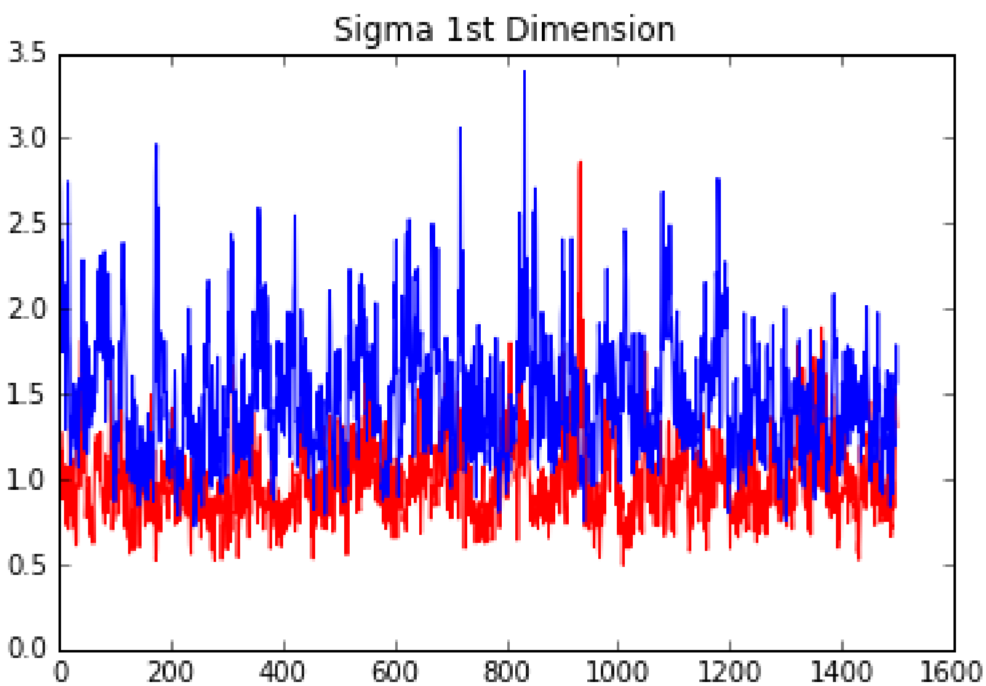
Now I'm going to load some of the data sets we used in homework 4 and cluster those.

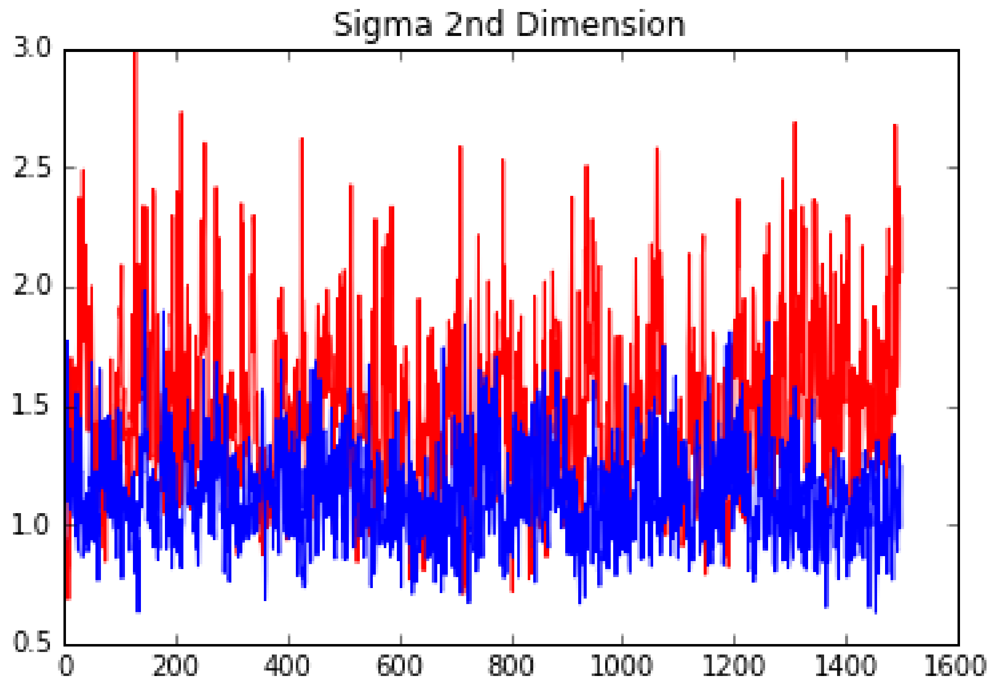
```
In [11]: import matplotlib.pyplot as plt

         S1data = numpy.loadtxt("S1train.csv", delimiter = ",", usecols = (1, 2))
         S1clusters, S1pi, S1mu, S1sigma, S1likelihood = GibbsSample(S1data, 2, 1500, 10, 1.5, .
         plt.plot(numpy.arange(1500), S1mu[:, 0, 0], "r", numpy.arange(1500), S1mu[:, 1, 0], "b"
         plt.title("Mu 1st Dimension")
         plt.show()
         plt.plot(numpy.arange(1500), S1mu[:, 0, 1], "r", numpy.arange(1500), S1mu[:, 1, 1], "b"
         plt.title("Mu 2nd Dimension")
         plt.show()
         plt.plot(numpy.arange(1500), S1pi[:, 0], "r", numpy.arange(1500), S1pi[:, 1], "b")
         plt.title("Pi")
         plt.show()
         plt.plot(numpy.arange(1500), S1sigma[:, 0, 0], "r", numpy.arange(1500), S1sigma[:, 1, 0
         plt.title("Sigma 1st Dimension")
         plt.show()
         plt.plot(numpy.arange(1500), S1sigma[:, 0, 1], "r", numpy.arange(1500), S1sigma[:, 1, 1
         plt.title("Sigma 2nd Dimension")
         plt.show()

         # plot the clusters
         plt.scatter(S1data[:, 0], S1data[:, 1], c = numpy.mean(S1clusters[-1000:, :], axis = 0)
         plt.title("Average Cluster Membership after Convergence")
         plt.colorbar()
         plt.show()
```

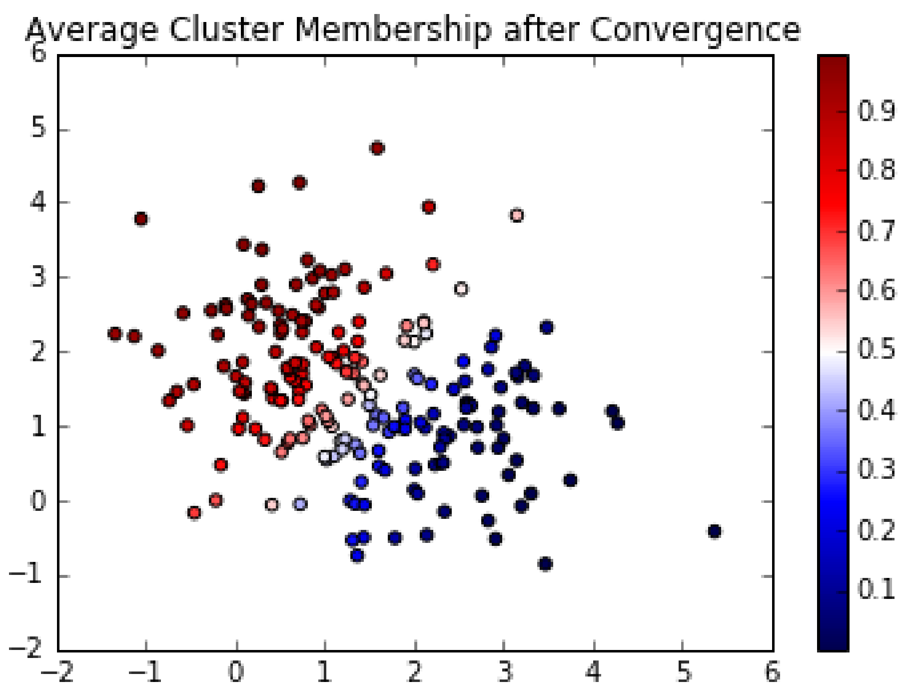

Mu 1st Dimension
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400
1600


Mu 2nd Dimension
3.0
2.5
2.0
1.5
1.0
0.5
0
200
400
600
800
1000
1200
1400
1600


Pi
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
200
400
600
800
1000
1200
1400
1600

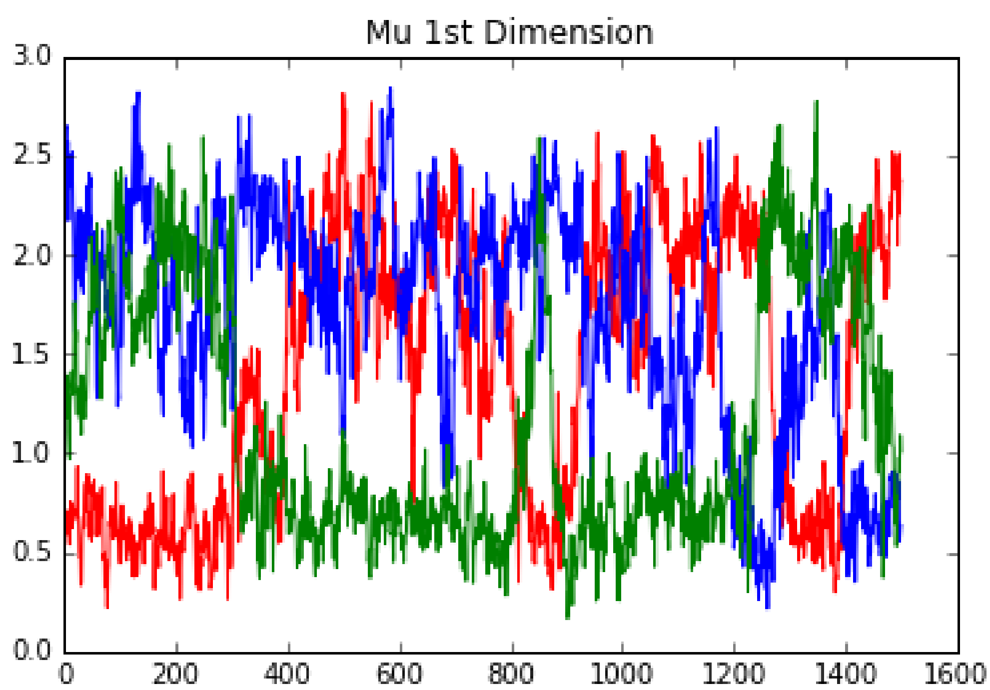

Sigma 1st Dimension
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400
1600

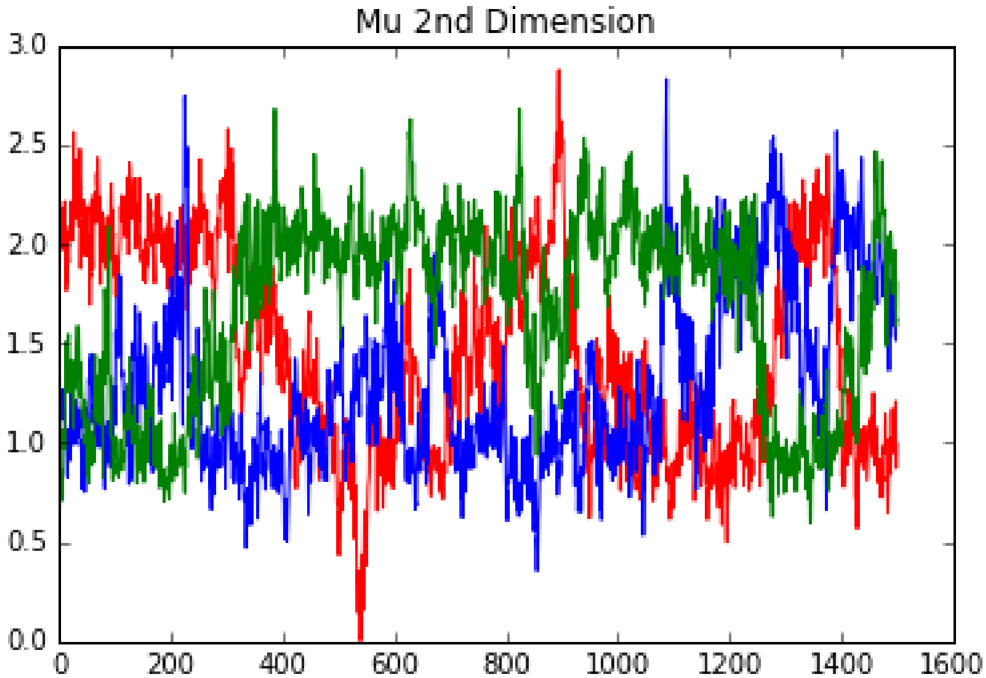


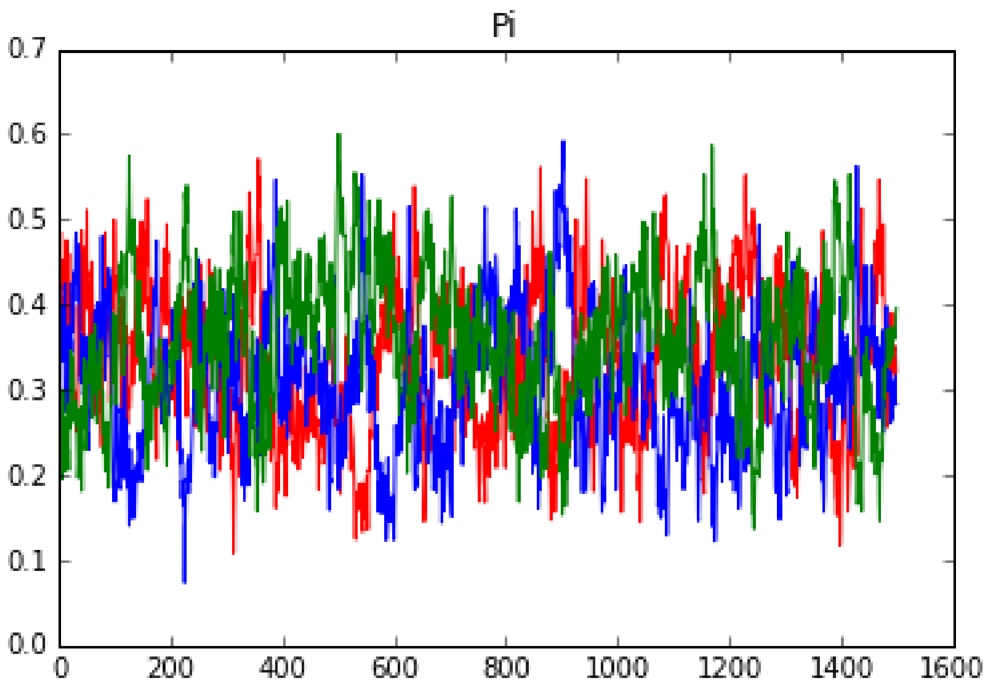


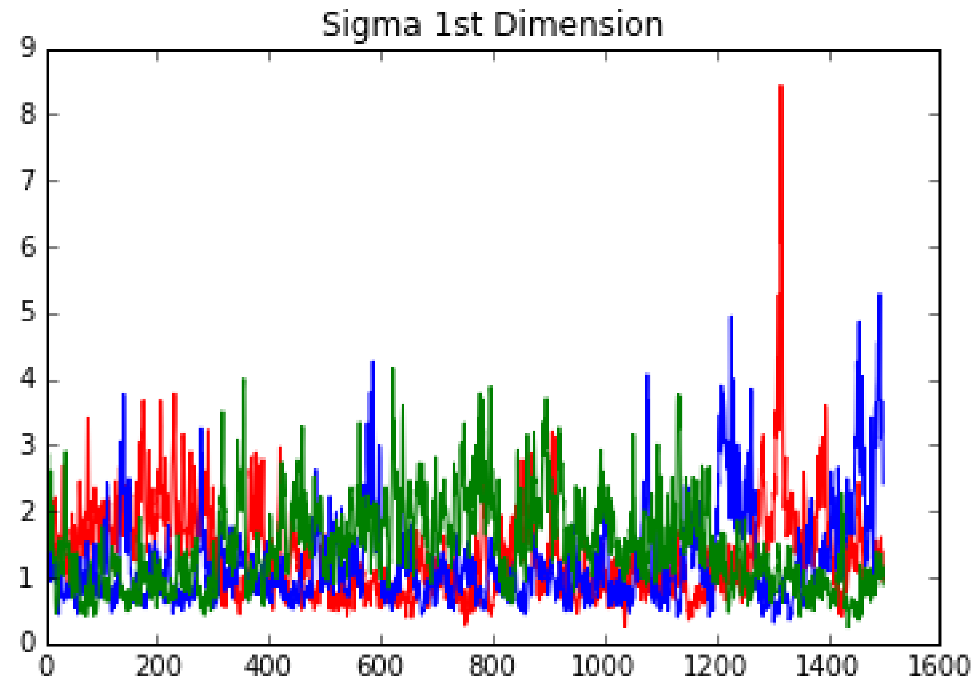

```
In [5]: # first synthetic data set with three clusters
        S1clusters3, S1pi3, S1mu3, S1sigma3, S1likelihood3 = GibbsSample(S1data, 3, 1500, 10, 1.
```

```python
plt.plot(numpy.arange(1500), S1mu3[:, 0, 0], "r", numpy.arange(1500), S1mu3[:, 1, 0], "b
        numpy.arange(1500), S1mu3[:, 2, 0], "g")
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1mu3[:, 0, 1], "r", numpy.arange(1500), S1mu3[:, 1, 1], "b
       numpy.arange(1500), S1mu3[:, 2, 1], "g")
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1pi3[:, 0], "r", numpy.arange(1500), S1pi3[:, 1], "b",
        numpy.arange(1500), S1pi3[:, 2], "g")
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), S1sigma3[:, 0, 0], "r", numpy.arange(1500), S1sigma3[:, 1,
        numpy.arange(1500), S1sigma3[:, 2, 0], "g")
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1sigma3[:, 0, 1], "r", numpy.arange(1500), S1sigma3[:, 1,
        numpy.arange(1500), S1sigma3[:, 2, 1], "g")
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
plt.scatter(S1data[:, 0], S1data[:, 1], c = numpy.mean(S1clusters3[-1000:, :], axis = 0)
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

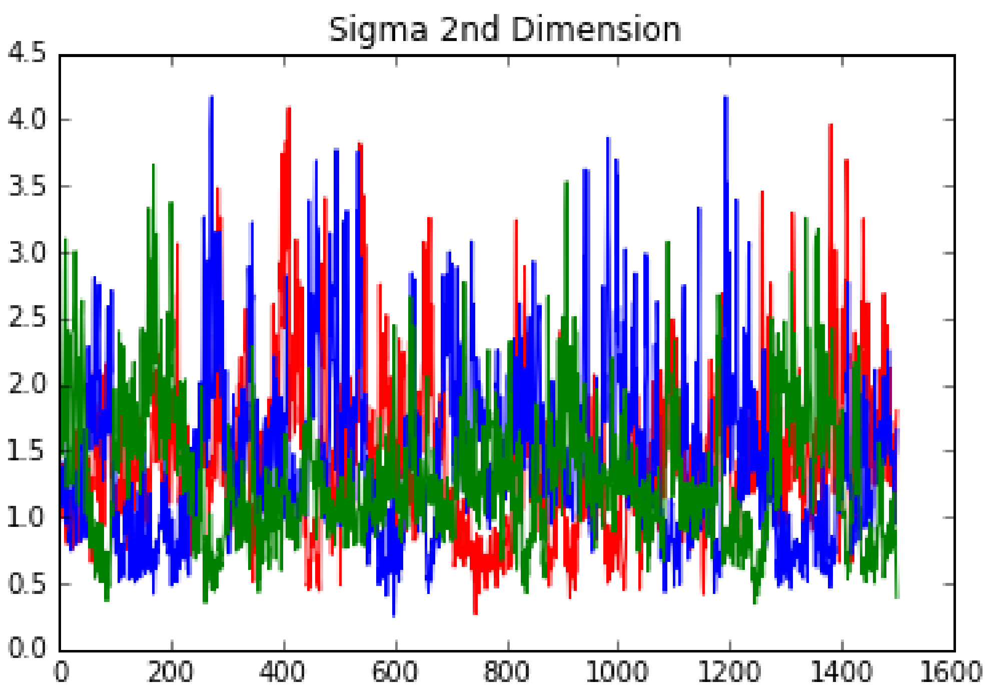



Mu 2nd Dimension
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400
1600

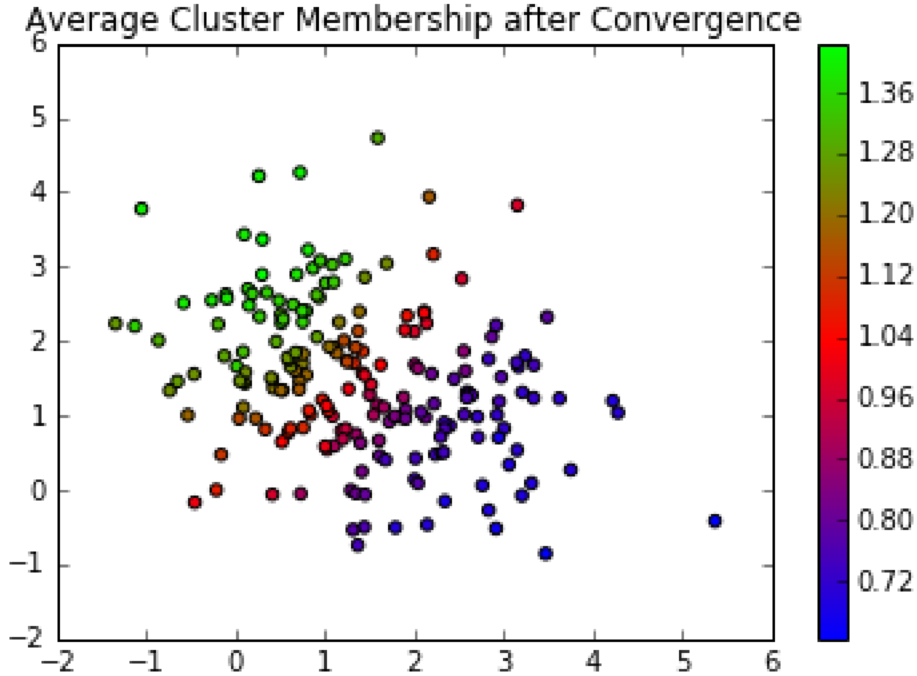

Pi
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000
1200
1400
1600



Sigma 1st Dimension
9
8
7
6
5
4
3
2
1
0
0
200
400
600
800
1000
1200
1400
1600

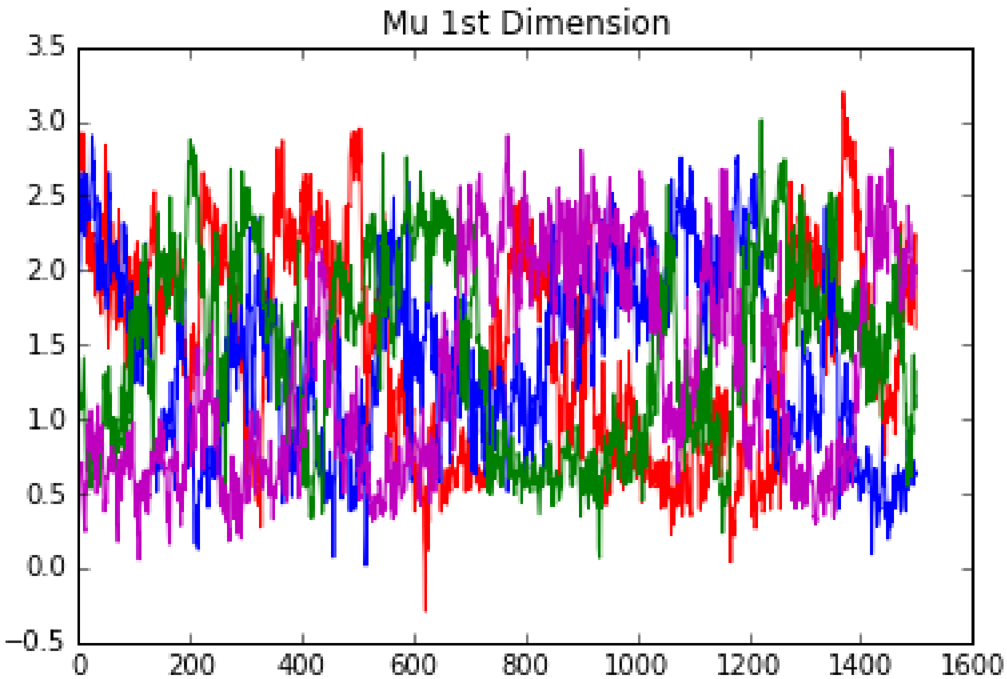


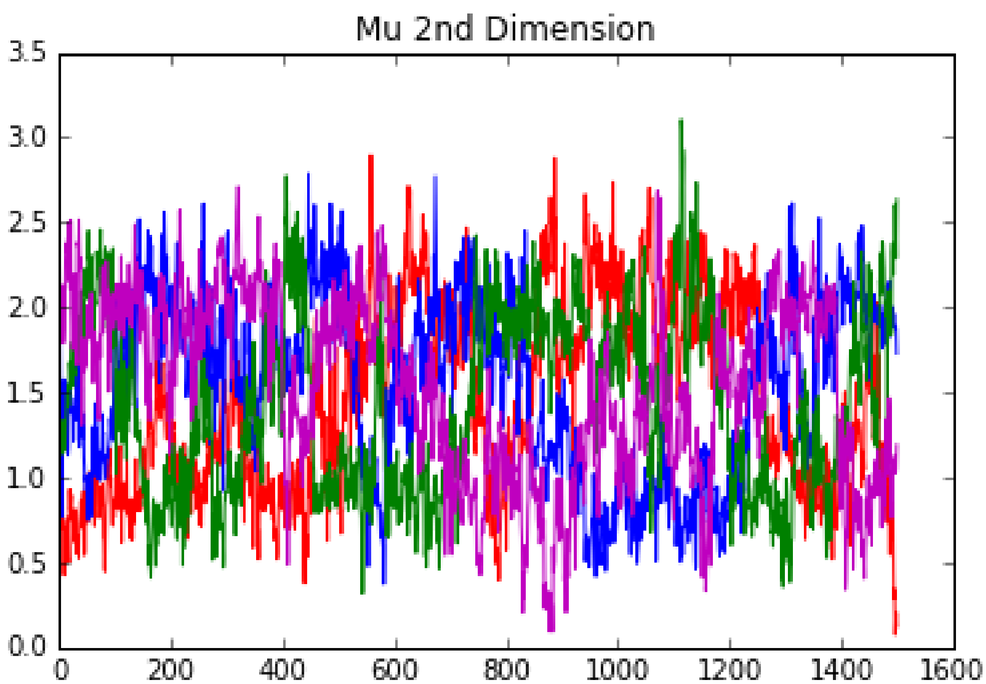
Sigma 2nd Dimension
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400
1600

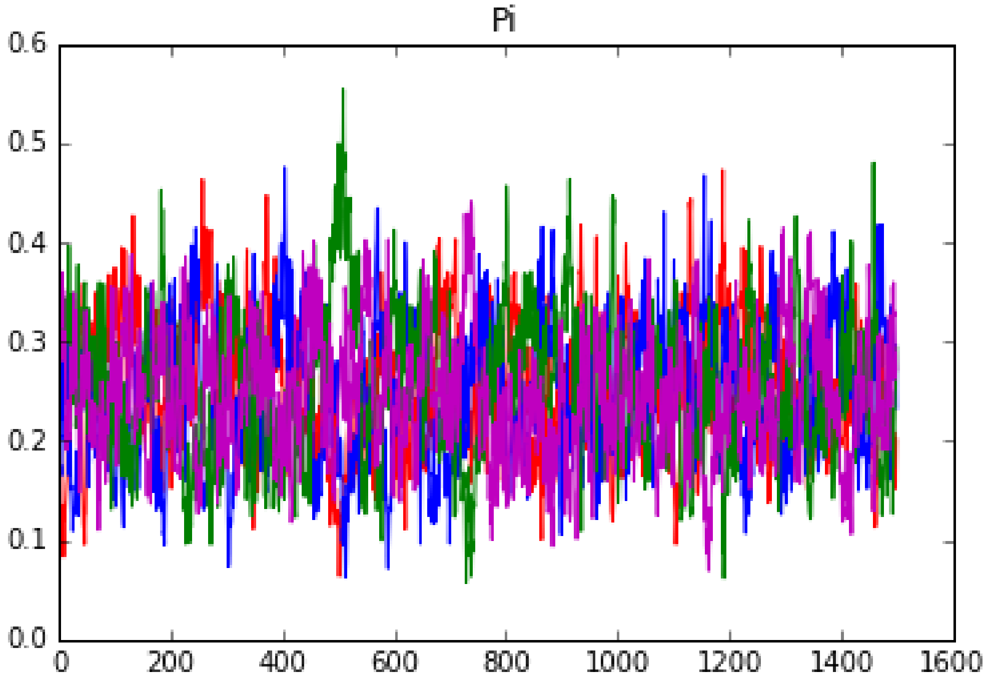


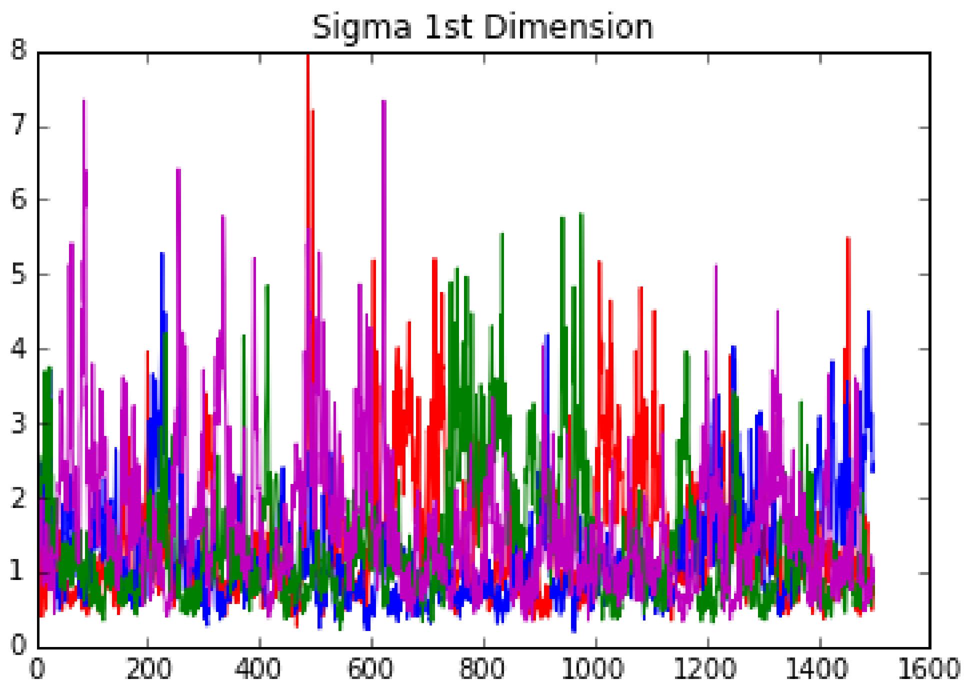
In [6]:
```
# first synthetic data set with four clusters
S1clusters4, S1pi4, S1mu4, S1sigma4, S1likelihood4 = GibbsSample(S1data, 4, 1500, 10, 1.
plt.plot(numpy.arange(1500), S1mu4[:, 0, 0], "r", numpy.arange(1500), S1mu4[:, 1, 0], "b
         numpy.arange(1500), S1mu4[:, 2, 0], "g", numpy.arange(1500), S1mu4[:, 3, 0], "m
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1mu4[:, 0, 1], "r", numpy.arange(1500), S1mu4[:, 1, 1], "b
         numpy.arange(1500), S1mu4[:, 2, 1], "g", numpy.arange(1500), S1mu4[:, 3, 1], "m
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1pi4[:, 0], "r", numpy.arange(1500), S1pi4[:, 1], "b",
         numpy.arange(1500), S1pi4[:, 2], "g", numpy.arange(1500), S1pi4[:, 3], "m")
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), S1sigma4[:, 0, 0], "r", numpy.arange(1500), S1sigma4[:, 1,
         numpy.arange(1500), S1sigma4[:, 2, 0], "g", numpy.arange(1500), S1sigma4[:, 3,
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S1sigma4[:, 0, 1], "r", numpy.arange(1500), S1sigma4[:, 1,
         numpy.arange(1500), S1sigma4[:, 2, 1], "g", numpy.arange(1500), S1sigma4[:, 3,
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
```

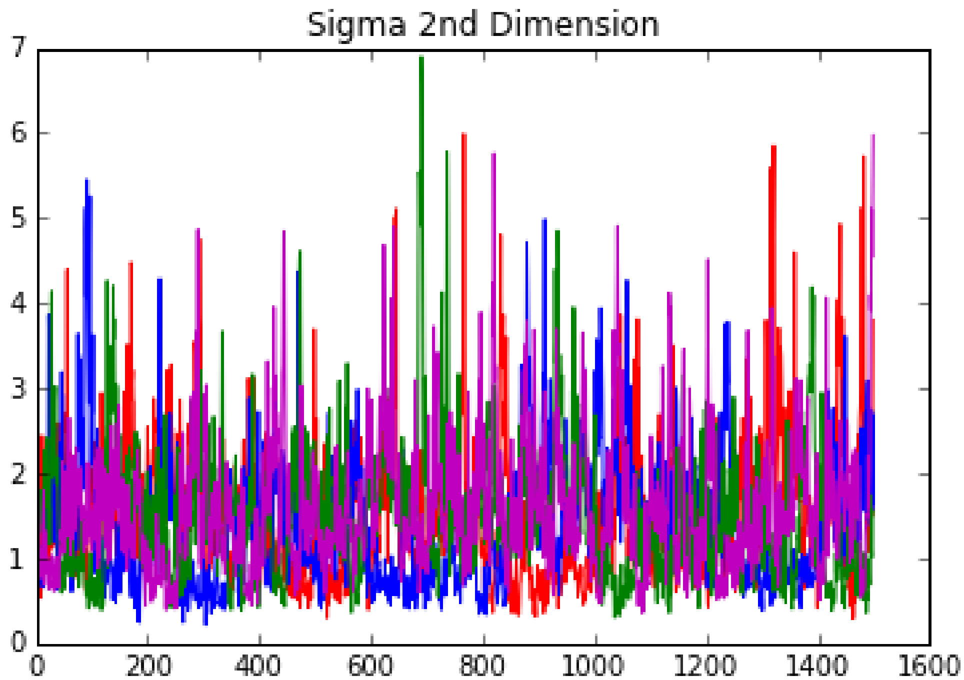
```
plt.scatter(S1data[:, 0], S1data[:, 1], c = numpy.mean(S1clusters4[-1000:, :], axis = 0)
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

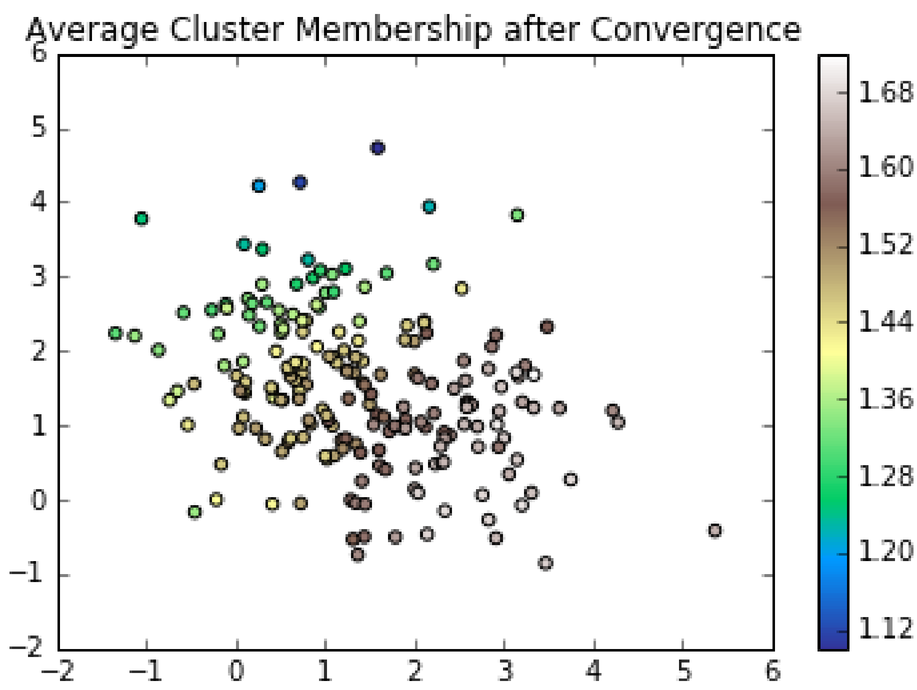





Pi
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000
1200
1400
1600

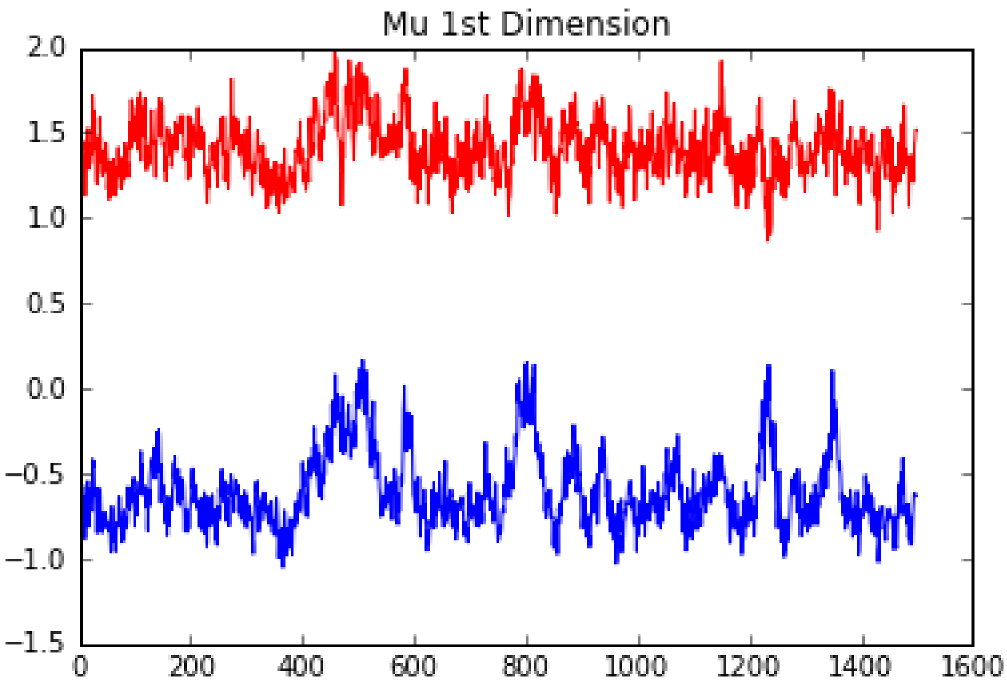

Sigma 1st Dimension
0
1
2
3
4
5
6
7
8
0
200
400
600
800
1000
1200
1400
1600

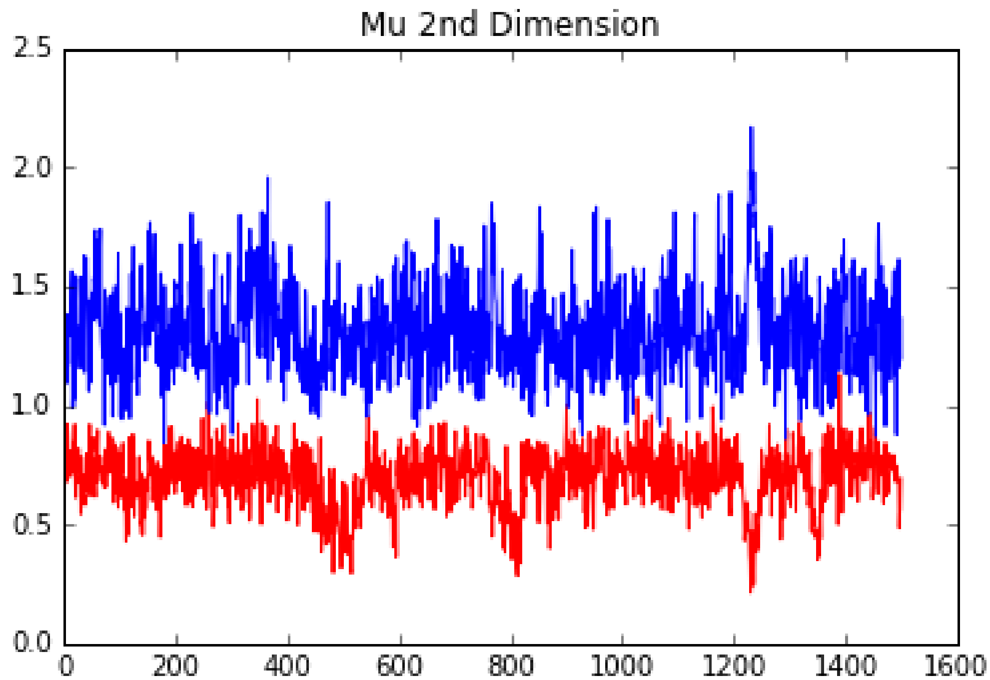


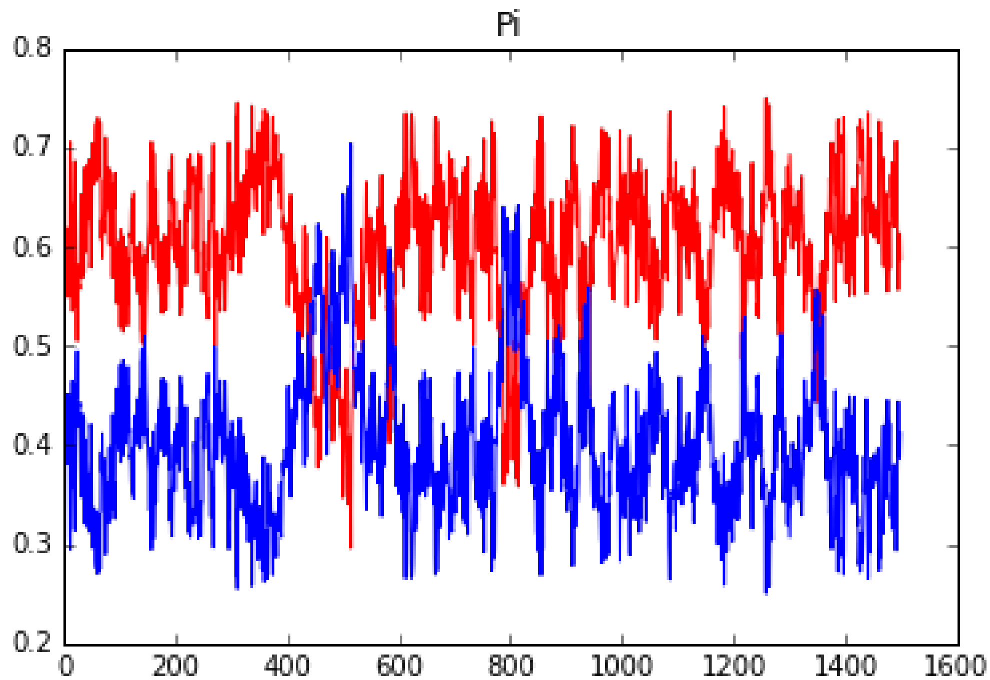


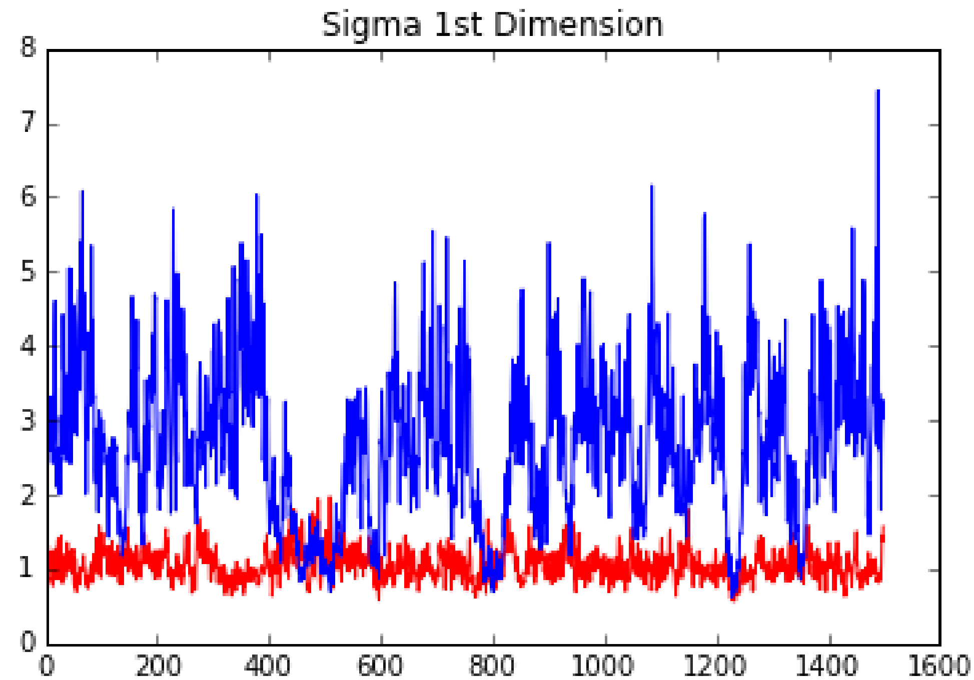

```
In [19]: # second synthetic data set
         S2data = numpy.loadtxt("S2train.csv", delimiter = ",", usecols = (1, 2))
```

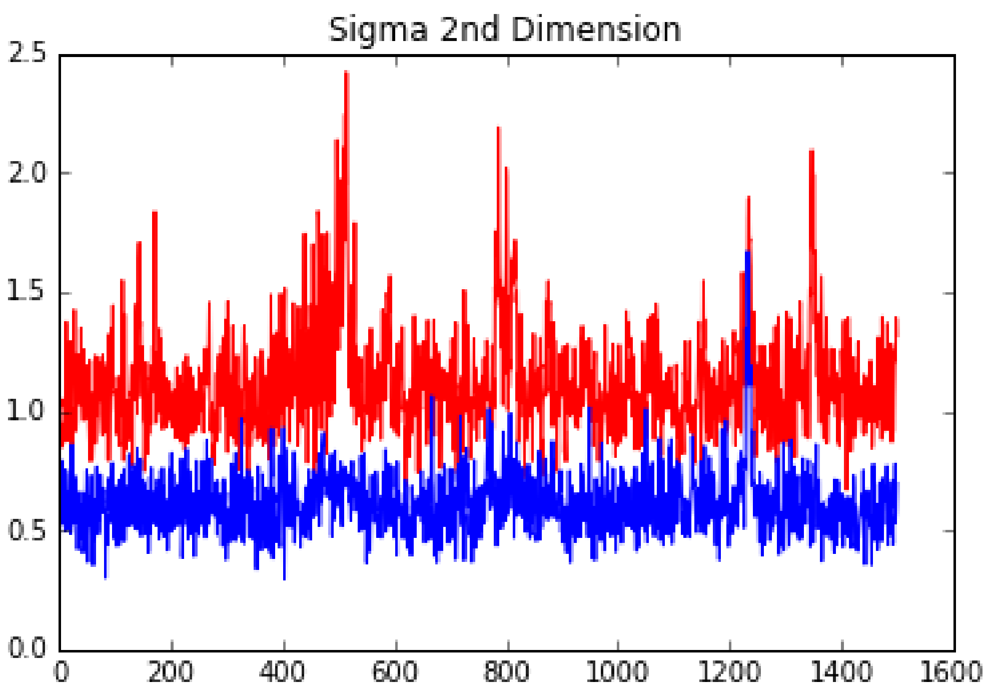
```
S2clusters, S2pi, S2mu, S2sigma, S2likelihood = GibbsSample(S2data, 2, 1500, 20, 1.5, .
plt.plot(numpy.arange(1500), S2mu[:, 0, 0], "r", numpy.arange(1500), S2mu[:, 1, 0], "b"
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S2mu[:, 0, 1], "r", numpy.arange(1500), S2mu[:, 1, 1], "b"
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), S2pi[:, 0], "r", numpy.arange(1500), S2pi[:, 1], "b")
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), S2sigma[:, 0, 0], "r", numpy.arange(1500), S2sigma[:, 1, 0
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), S2sigma[:, 0, 1], "r", numpy.arange(1500), S2sigma[:, 1, 1
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
plt.scatter(S2data[:, 0], S2data[:, 1], c = numpy.mean(S2clusters[-1000:, :], axis = 0)
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

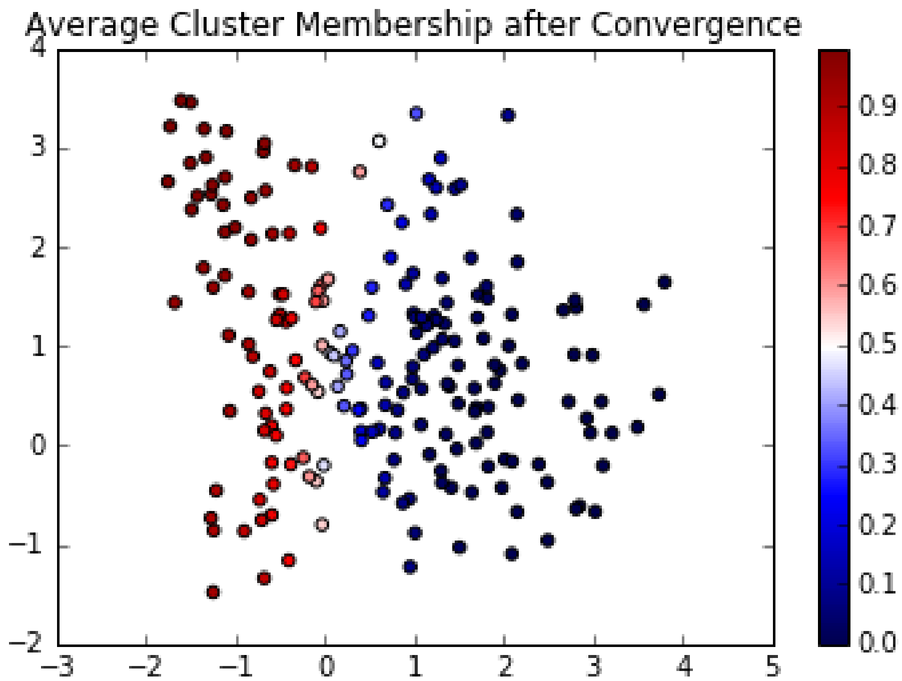



Mu 2nd Dimension
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
1200
1400
1600

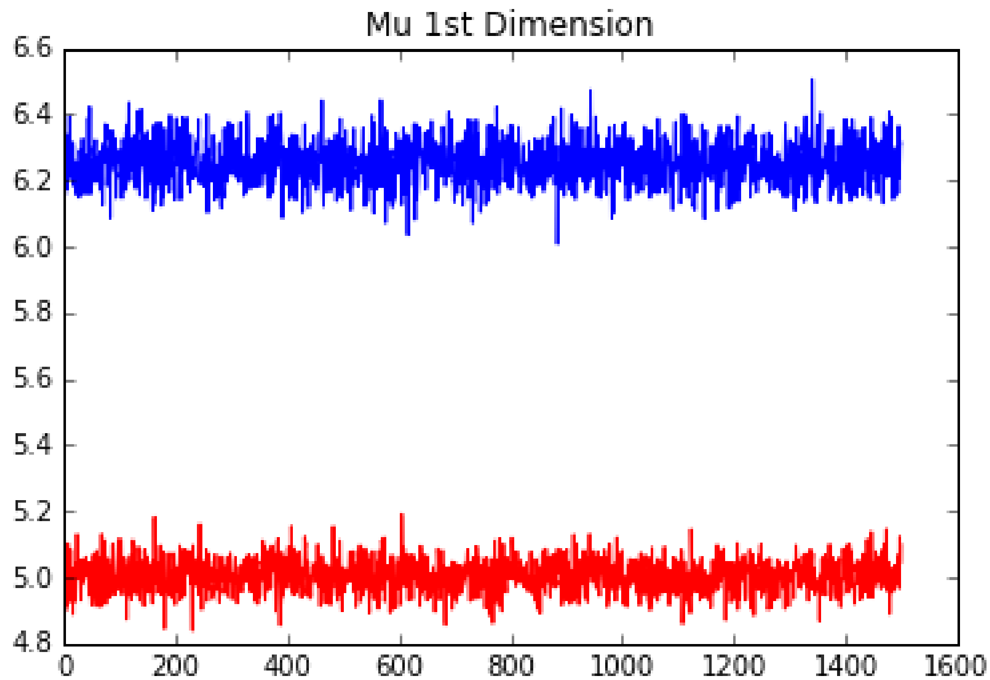

Pi
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000
1200
1400
1600

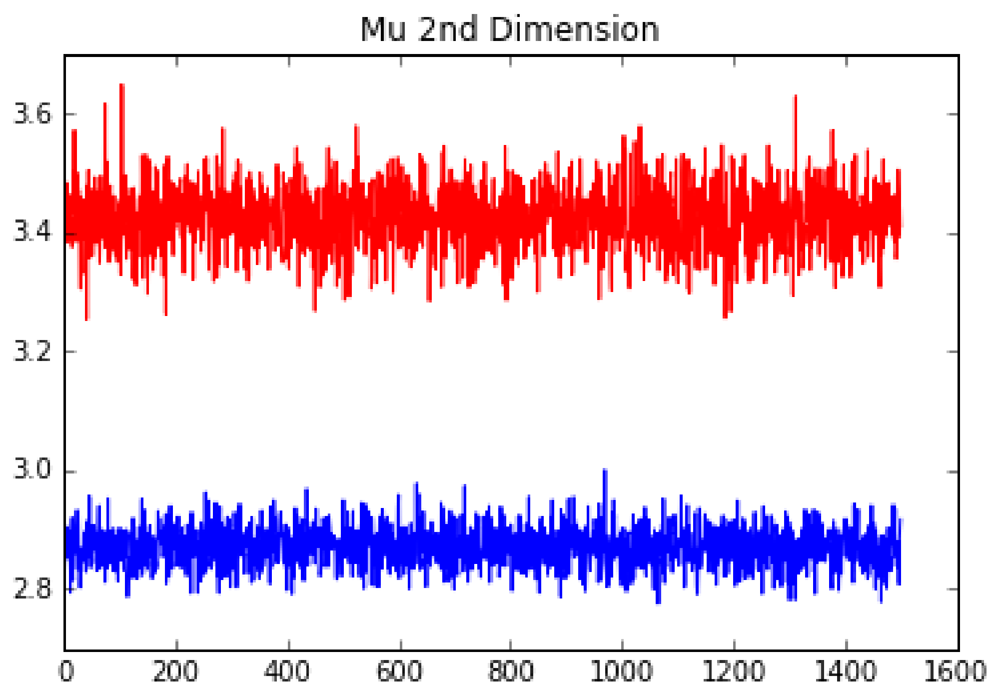

Sigma 1st Dimension
0
1
2
3
4
5
6
7
8
0
200
400
600
800
1000
1200
1400
1600

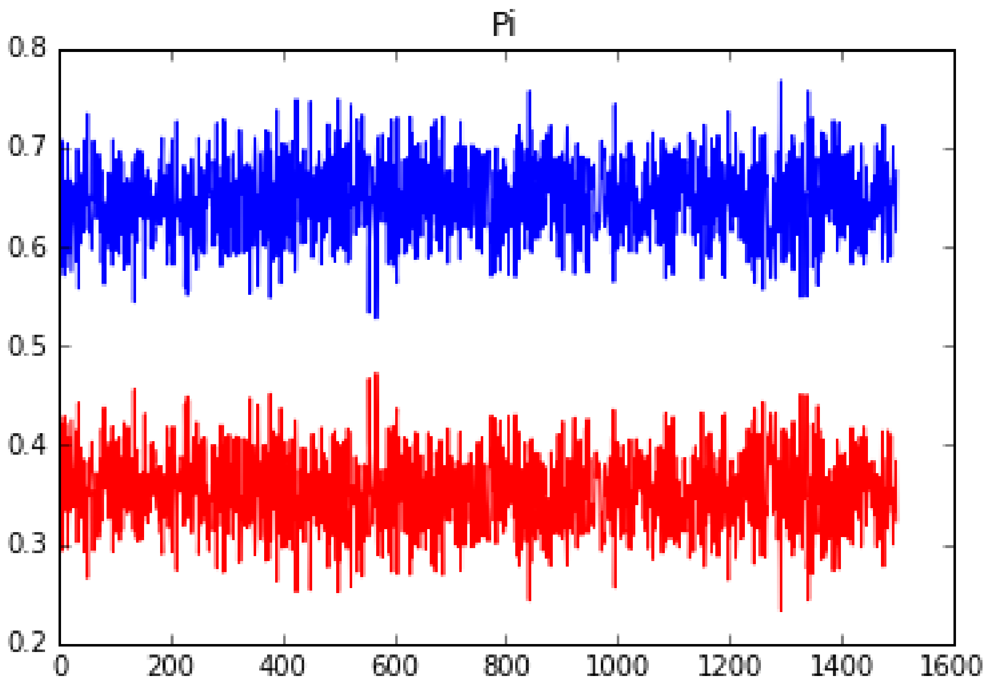

Sigma 2nd Dimension
0.0
0.5
1.0
1.5
2.0
2.5
0
200
400
600
800
1000
1200
1400
1600

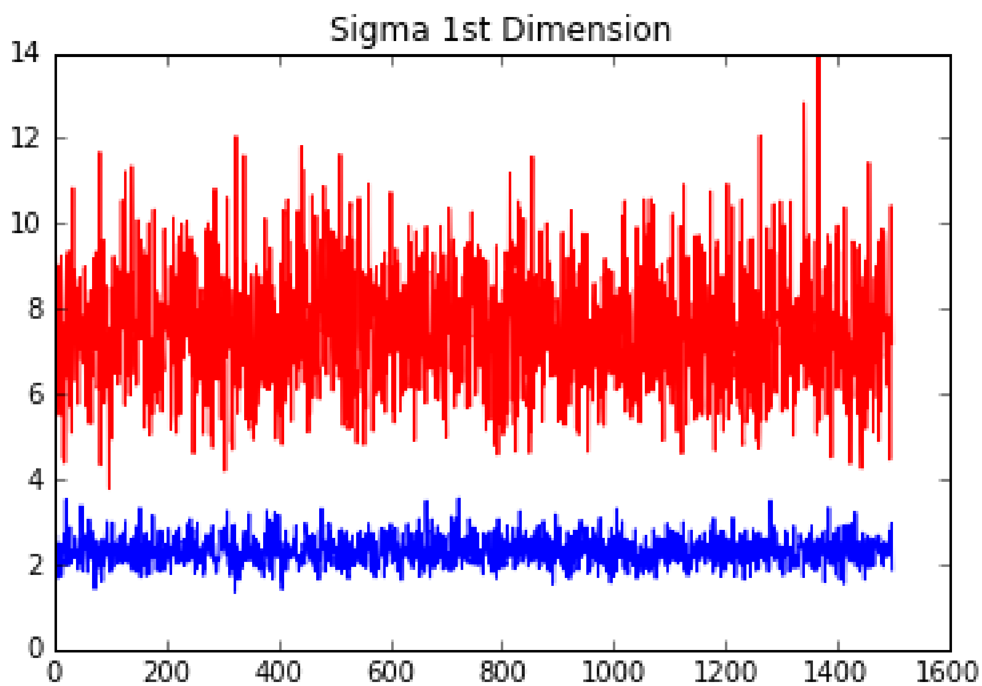


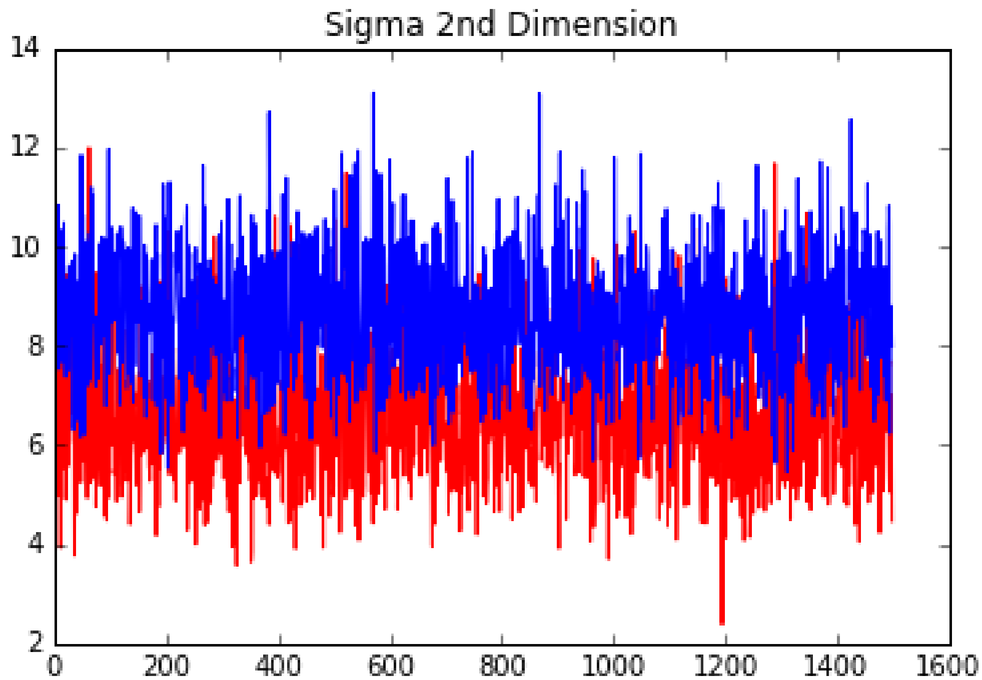
In [20]:
```
# Iris data set with two clusters
irisdata = numpy.loadtxt("iris_features.csv", delimiter = ",", skiprows = 1)
irisclusters2, irispi2, irismu2, irissigma2, irislikelihood2 = GibbsSample(irisdata, 2,
plt.plot(numpy.arange(1500), irismu2[:, 0, 0], "r", numpy.arange(1500), irismu2[:, 1, 0
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irismu2[:, 0, 1], "r", numpy.arange(1500), irismu2[:, 1, 1
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), irispi2[:, 0], "r", numpy.arange(1500), irispi2[:, 1], "b"
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), irissigma2[:, 0, 0], "r", numpy.arange(1500), irissigma2[:
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irissigma2[:, 0, 1], "r", numpy.arange(1500), irissigma2[:
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
plt.scatter(irisdata[:, 0], irisdata[:, 1], c = numpy.mean(irisclusters2[-1000:, :], ax
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

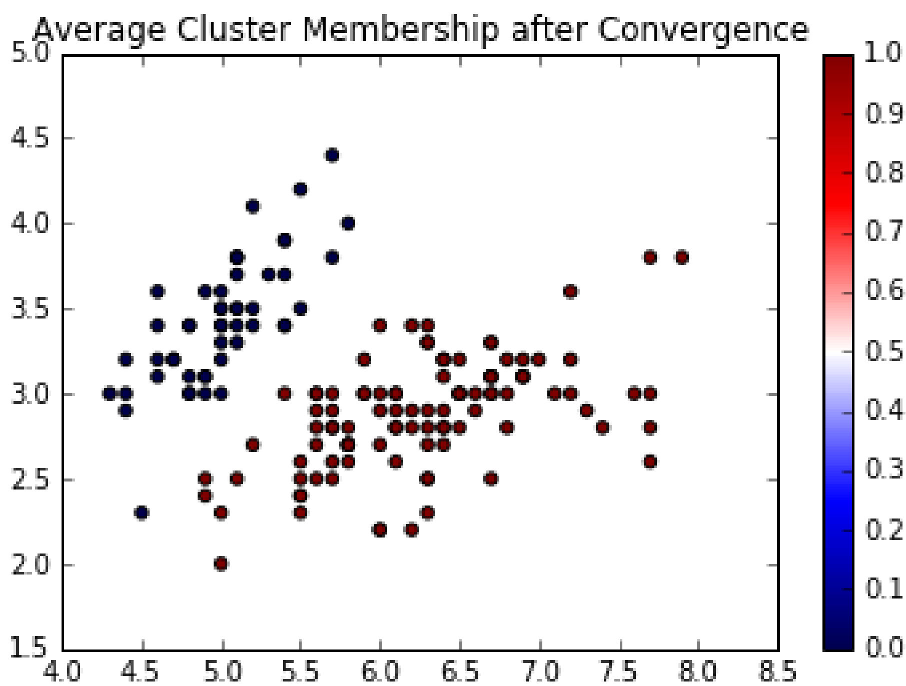

Mu 1st Dimension
6.6
6.4
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
0
200
400
600
800
1000
1200
1400
1600


Mu 2nd Dimension
3.6
3.4
3.2
3.0
2.8
0
200
400
600
800
1000
1200
1400
1600


Pi
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000
1200
1400
1600

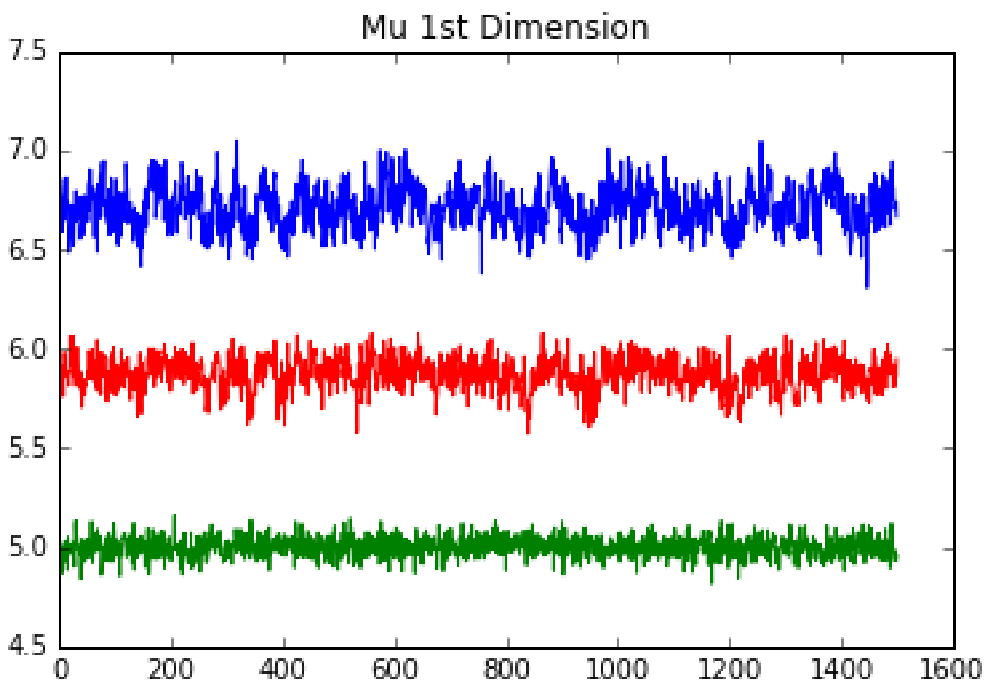

Sigma 1st Dimension
14
12
10
8
6
4
2
0
0
200
400
600
800
1000
1200
1400
1600

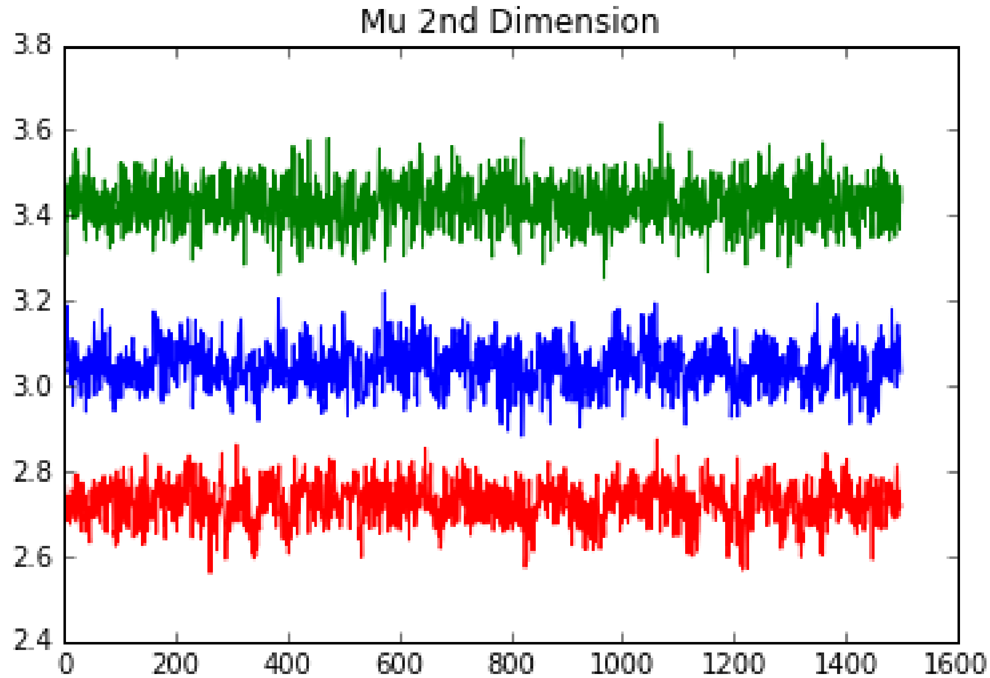


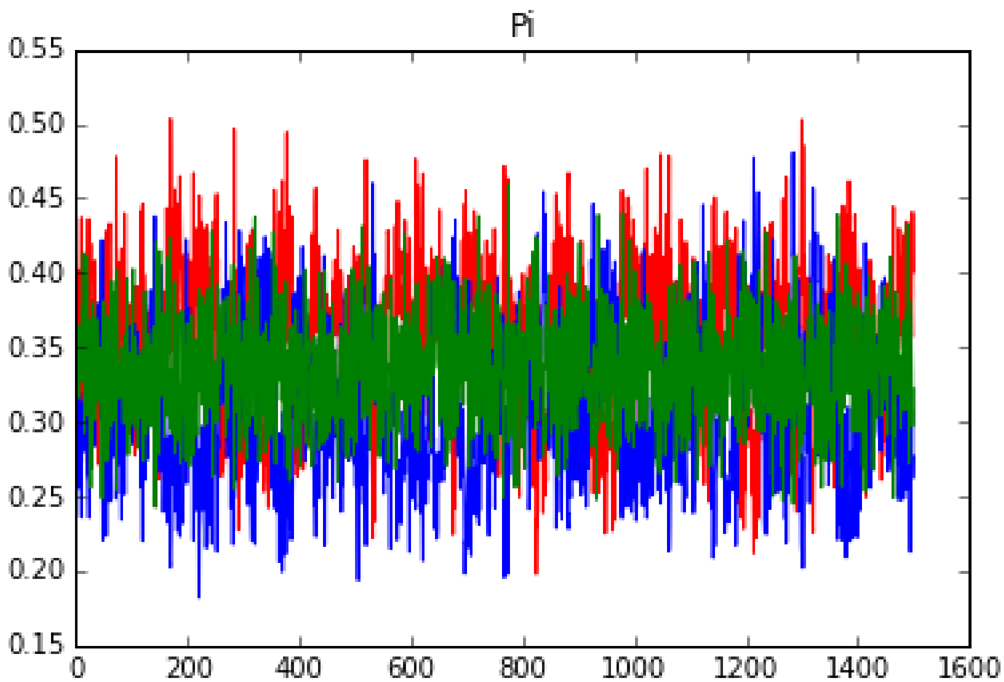


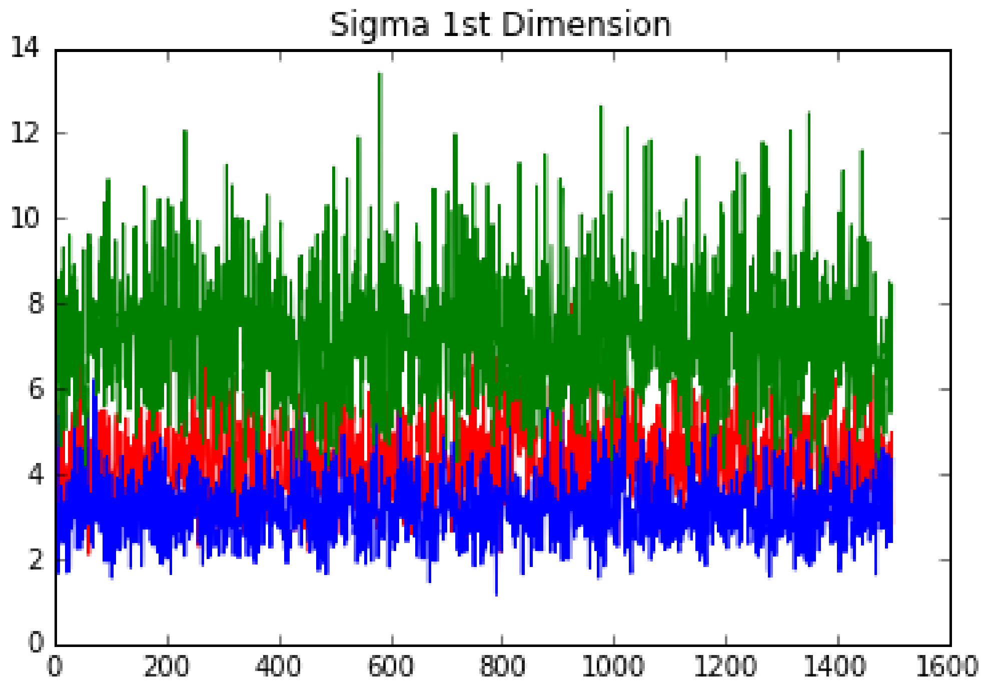
In [23]:
```
# Iris data set with three clusters
irisclusters3, irispi3, irismu3, irissigma3, irislikelihood3 = GibbsSample(irisdata, 3,
```

```
plt.plot(numpy.arange(1500), irismu3[:, 0, 0], "r", numpy.arange(1500), irismu3[:, 1, 0
         numpy.arange(1500), irismu3[:, 2, 0], "g")
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irismu3[:, 0, 1], "r", numpy.arange(1500), irismu3[:, 1, 1
         numpy.arange(1500), irismu3[:, 2, 1], "g")
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), irispi3[:, 0], "r", numpy.arange(1500), irispi3[:, 1], "b"
         numpy.arange(1500), irispi3[:, 2], "g")
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), irissigma3[:, 0, 0], "r", numpy.arange(1500), irissigma3[:
         numpy.arange(1500), irissigma3[:, 2, 0], "g")
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irissigma3[:, 0, 1], "r", numpy.arange(1500), irissigma3[:
         numpy.arange(1500), irissigma3[:, 2, 1], "g")
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
plt.scatter(irisdata[:, 0], irisdata[:, 1], c = numpy.mean(irisclusters3[-1000:, :], ax
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

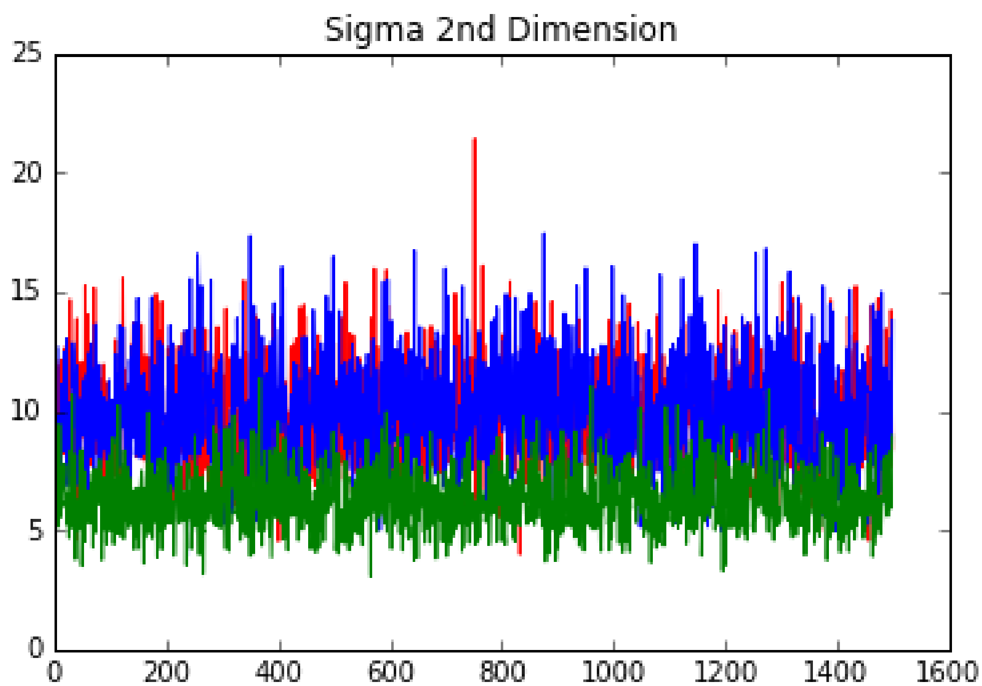



Mu 2nd Dimension
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
0
200
400
600
800
1000
1200
1400
1600

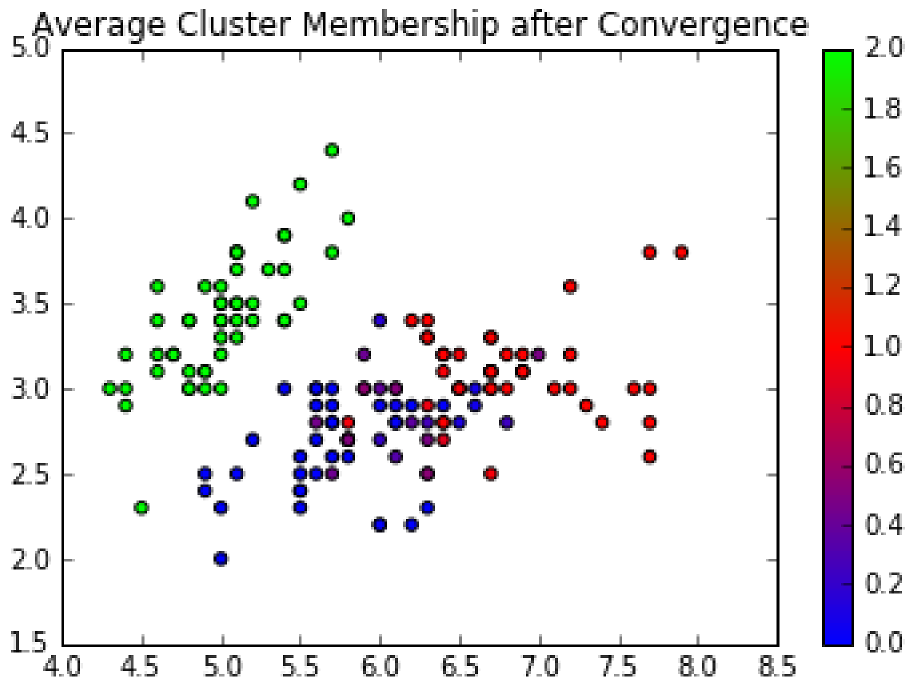

Pi
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
200
400
600
800
1000
1200
1400
1600



Sigma 1st Dimension
0
2
4
6
8
10
12
14
0
200
400
600
800
1000
1200
1400
1600

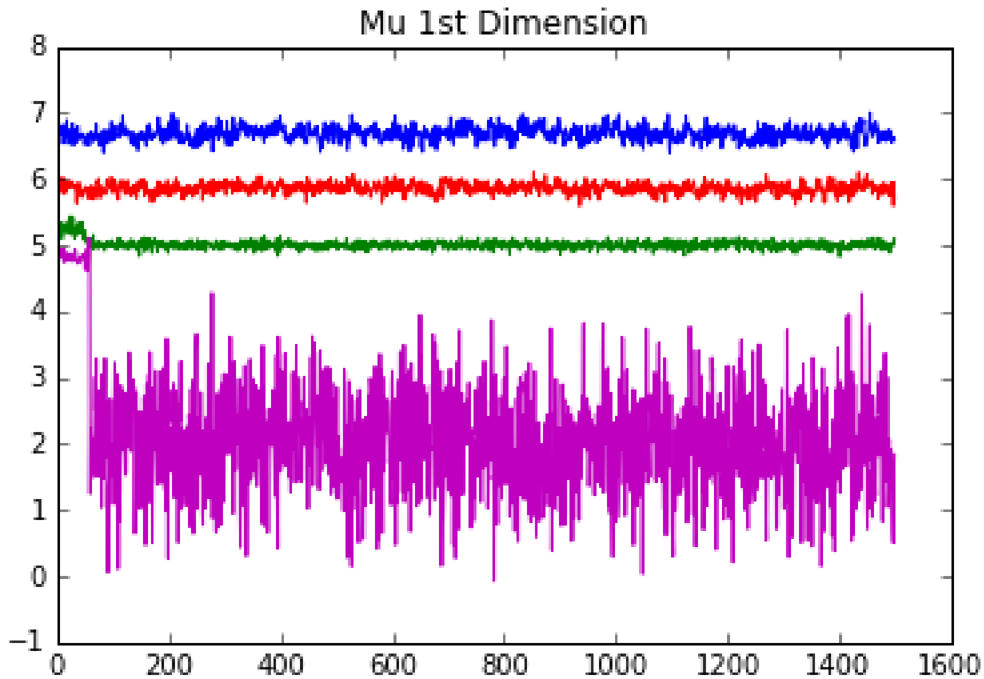


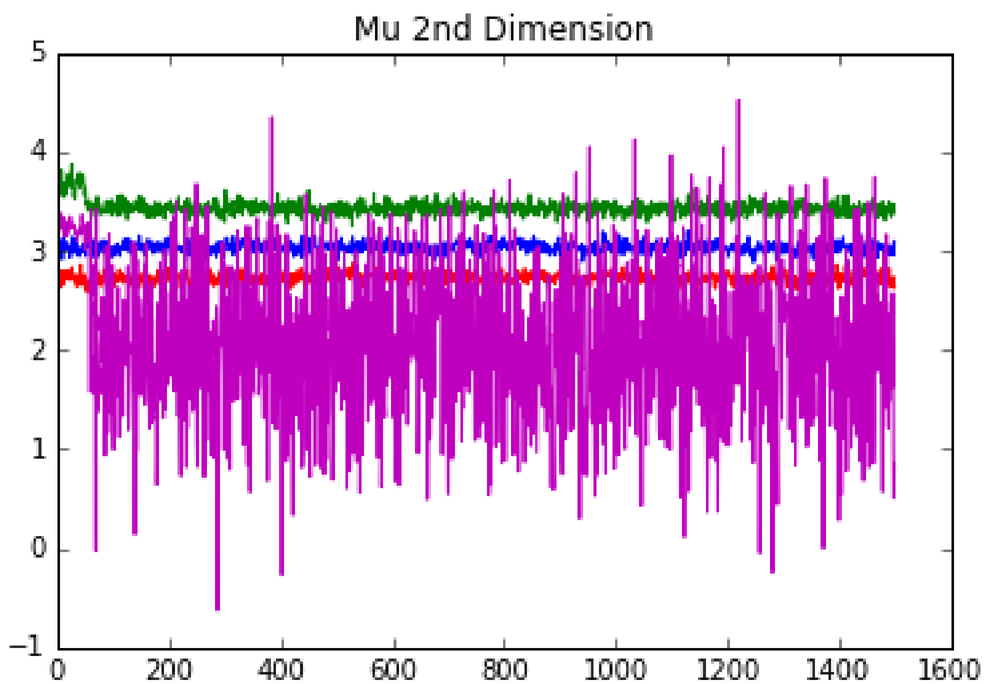
Sigma 2nd Dimension
0
5
10
15
20
25
0
200
400
600
800
1000
1200
1400
1600

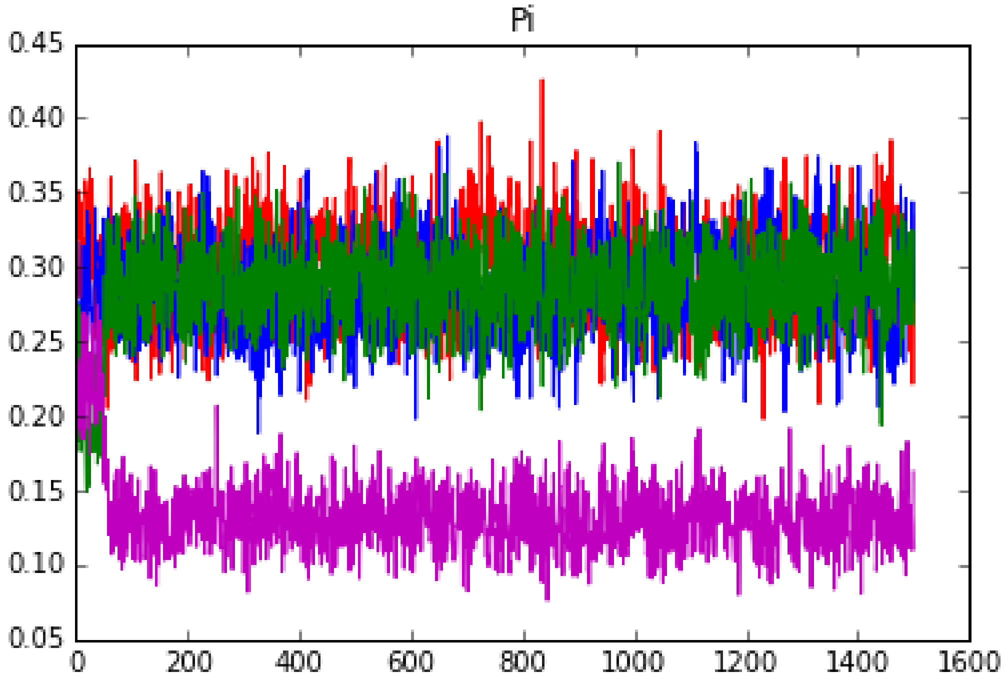


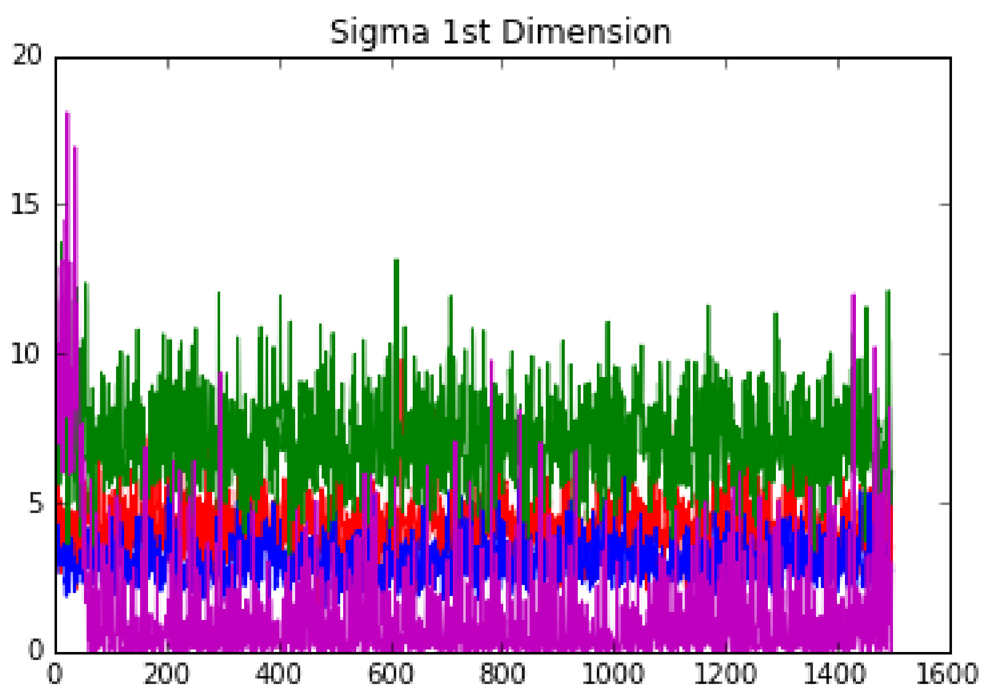
In [27]:
```
# Iris data set with four clusters
irisclusters4, irispi4, irismu4, irissigma4, irislikelihood4 = GibbsSample(irisdata, 4,
plt.plot(numpy.arange(1500), irismu4[:, 0, 0], "r", numpy.arange(1500), irismu4[:, 1, 0
         numpy.arange(1500), irismu4[:, 2, 0], "g", numpy.arange(1500), irismu4[:, 3, 0
plt.title("Mu 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irismu4[:, 0, 1], "r", numpy.arange(1500), irismu4[:, 1, 1
         numpy.arange(1500), irismu4[:, 2, 1], "g", numpy.arange(1500), irismu4[:, 3, 1
plt.title("Mu 2nd Dimension")
plt.show()
plt.plot(numpy.arange(1500), irispi4[:, 0], "r", numpy.arange(1500), irispi4[:, 1], "b"
         numpy.arange(1500), irispi4[:, 2], "g", numpy.arange(1500), irispi4[:, 3], "m"
plt.title("Pi")
plt.show()
plt.plot(numpy.arange(1500), irissigma4[:, 0, 0], "r", numpy.arange(1500), irissigma4[:
         numpy.arange(1500), irissigma4[:, 2, 0], "g", numpy.arange(1500), irissigma4[:
plt.title("Sigma 1st Dimension")
plt.show()
plt.plot(numpy.arange(1500), irissigma4[:, 0, 1], "r", numpy.arange(1500), irissigma4[:
         numpy.arange(1500), irissigma4[:, 2, 1], "g", numpy.arange(1500), irissigma4[:
plt.title("Sigma 2nd Dimension")
plt.show()

# plot the clusters
```

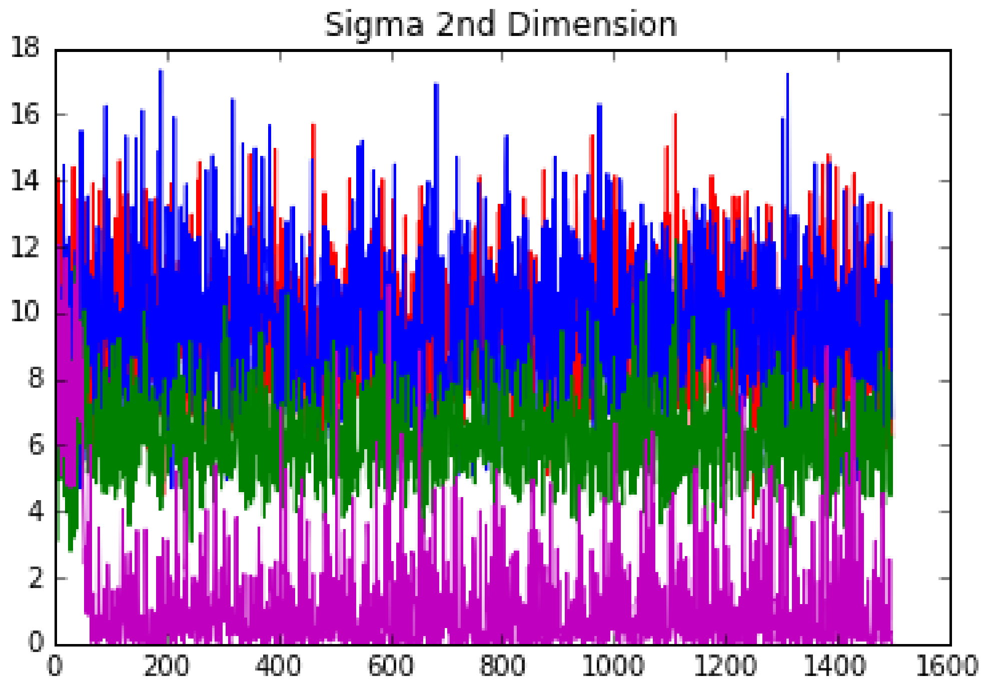
```
plt.scatter(irisdata[:, 0], irisdata[:, 1], c = numpy.mean(irisclusters2[-1000:, :], ax
plt.title("Average Cluster Membership after Convergence")
plt.colorbar()
plt.show()
```

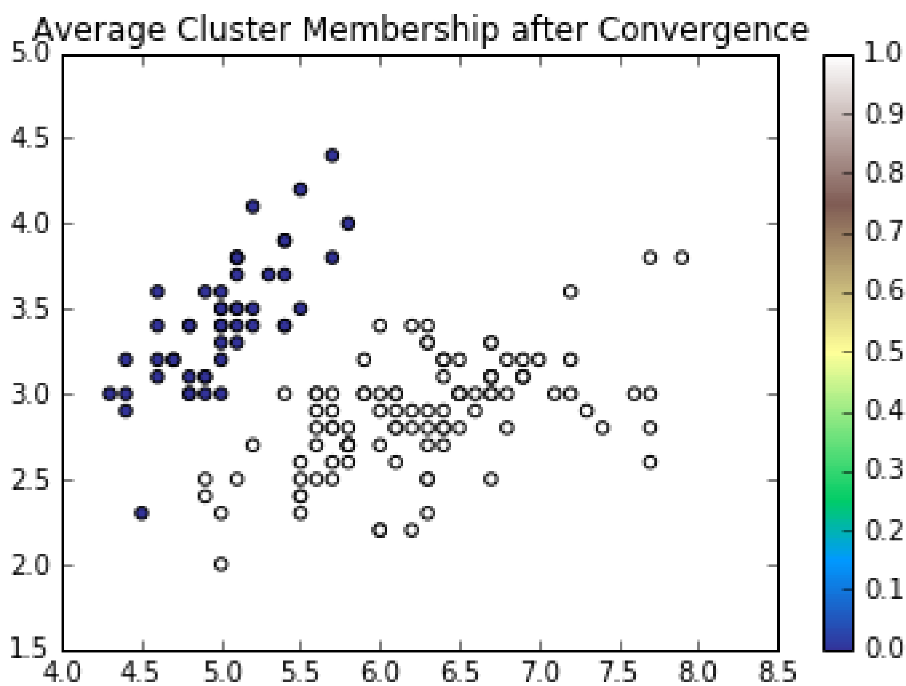





Pi
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
200
400
600
800
1000
1200
1400
1600


Sigma 1st Dimension
20
15
10
5
0
0
200
400
600
800
1000
1200
1400
1600


Sigma 2nd Dimension
18
16
14
12
10
8
6
4
2
0
0
200
400
600
800
1000
1200
1400
1600

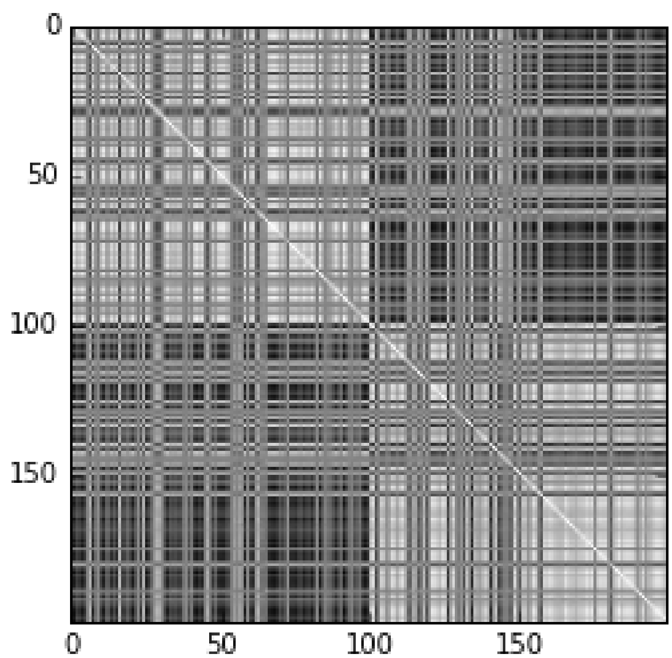

Average Cluster Membership after Convergence
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

2 b) Estimate the pairwise co-clustering matrix.

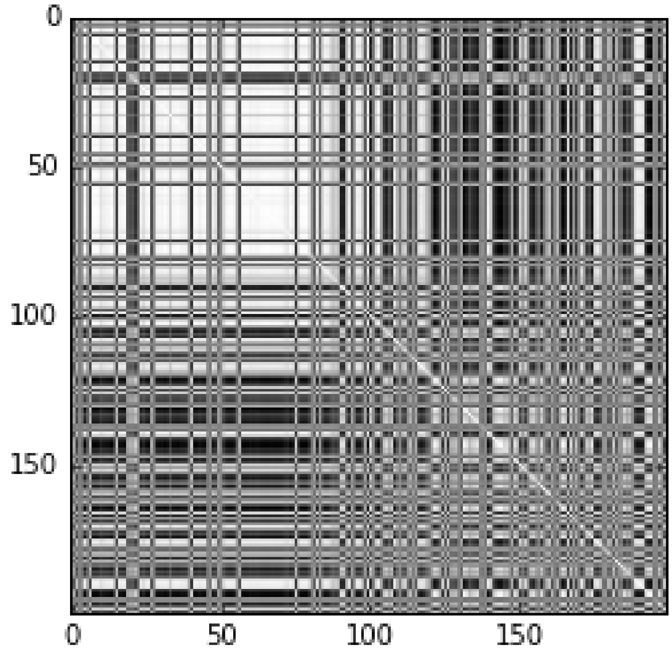
```
In [36]: def coclustering(clusters):
             """ Returns the pairwise co-clustering matrix given a list of clusters sampled for

             # create the empty matrix
             coclusters = numpy.empty((clusters.shape[1], clusters.shape[1]))

             # loop through each pair of points
             for i in range(clusters.shape[1]):
                 for j in range(clusters.shape[1]):
                     coclusters[i, j] = numpy.sum(1*(clusters[-1000:, i] == clusters[-1000:, j])

             return coclusters
```

Now I can display the coclustering matricies for each of the data sets clustered above.

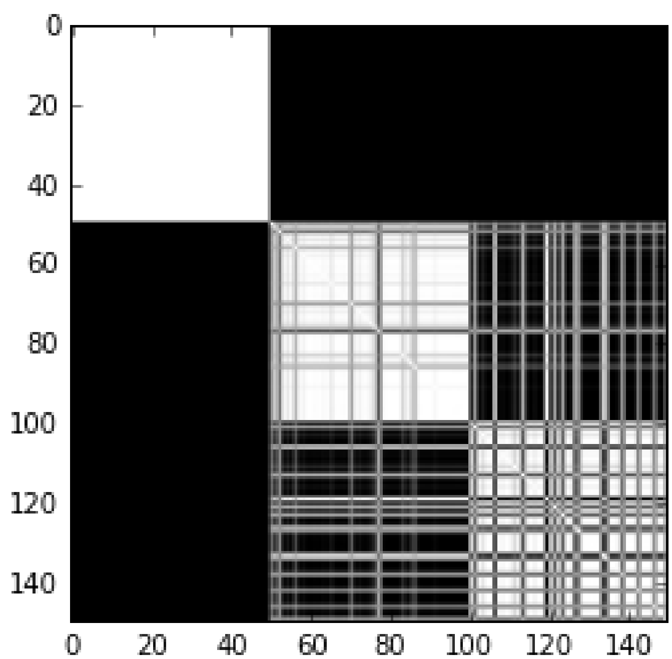
```
In [37]: # 1st synthetic data set
         coclusters = coclustering(S1clusters)
         plt.imshow(coclusters, cmap = "gray")
         plt.show()
```




```
In [38]: # 2nd synthetic data set
         coclusters = coclustering(S2clusters)
         plt.imshow(coclusters, cmap = "gray")
         plt.show()
```




```
In [39]: # iris data set
         coclusters = coclustering(irisclusters3)
         plt.imshow(coclusters, cmap = "gray")
         plt.show()
```



3 c) Estimate the log marginal likelihood of the training data.

```
In [16]: def logmarginalL(likelihood):
             """ Given a list of log likelihoods of each iteration, returns the overal log margi

             return (numpy.log(numpy.sum(numpy.exp(likelihood[-1000:] - numpy.mean(likelihood[-1
                     + numpy.mean(likelihood[-1000:]) + numpy.log(1/(1000+1)))
```

```
In [17]: # likelihood of the 1st synthetic data set
         print(logmarginalL(S1likelihood))
```

```
-45.6905648251
```

```
In [21]: # likelihood of the 2nd synthetic data set
         print(logmarginalL(S2likelihood))
```

```
-26.0345570935
```

```
In [22]: # likelihood of the iris data set with 2 clusters
         print(logmarginalL(irislikelihood2))
```

```
-0.00100444031679
```

```
In [25]: # likelihood of the iris data set with 3 clusters
         print(logmarginalL(irislikelihood3))
```

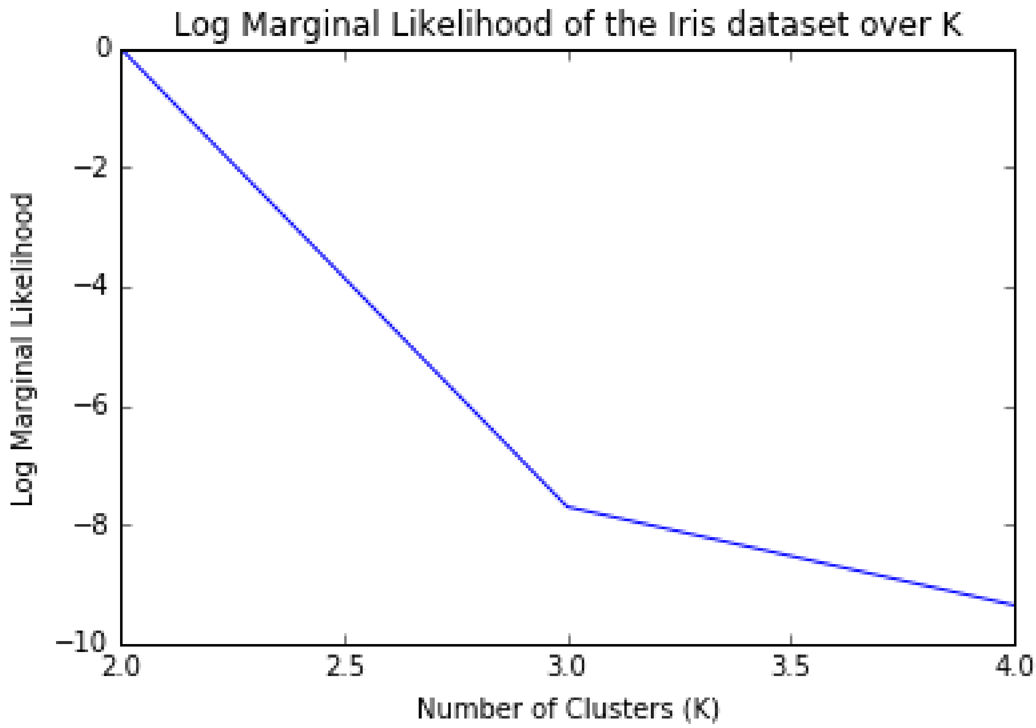
```
-7.70411275245
```

```
In [28]: # likelihood of the iris data set with 4 clusters
         print(logmarginalL(irislikelihood4))
```

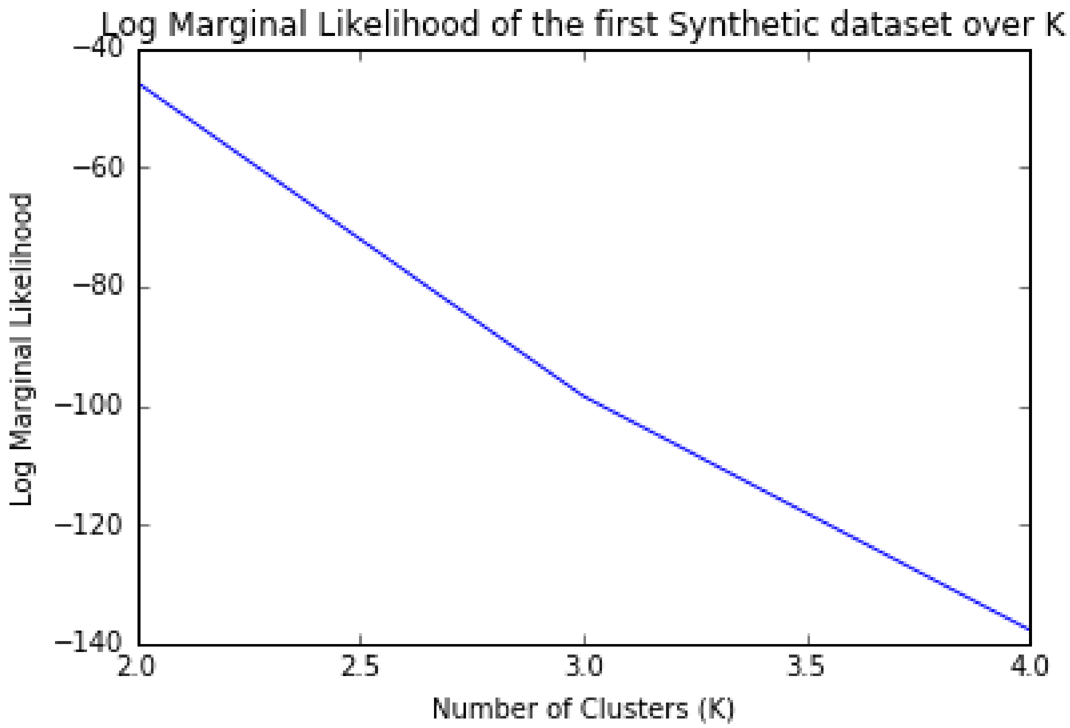
```
-9.35665644078
```

4 d) Plot and compare the results log-likelihood for different values of K for a few data sets. Does this metric tend to select a good value of K?

```
In [29]: plt.plot(range(2, 5), [logmarginalL(irislikelihood2), logmarginalL(irislikelihood3), lo
                 "b")
         plt.title("Log Marginal Likelihood of the Iris dataset over K")
         plt.xlabel("Number of Clusters (K)")
         plt.ylabel("Log Marginal Likelihood")
         plt.show()
```



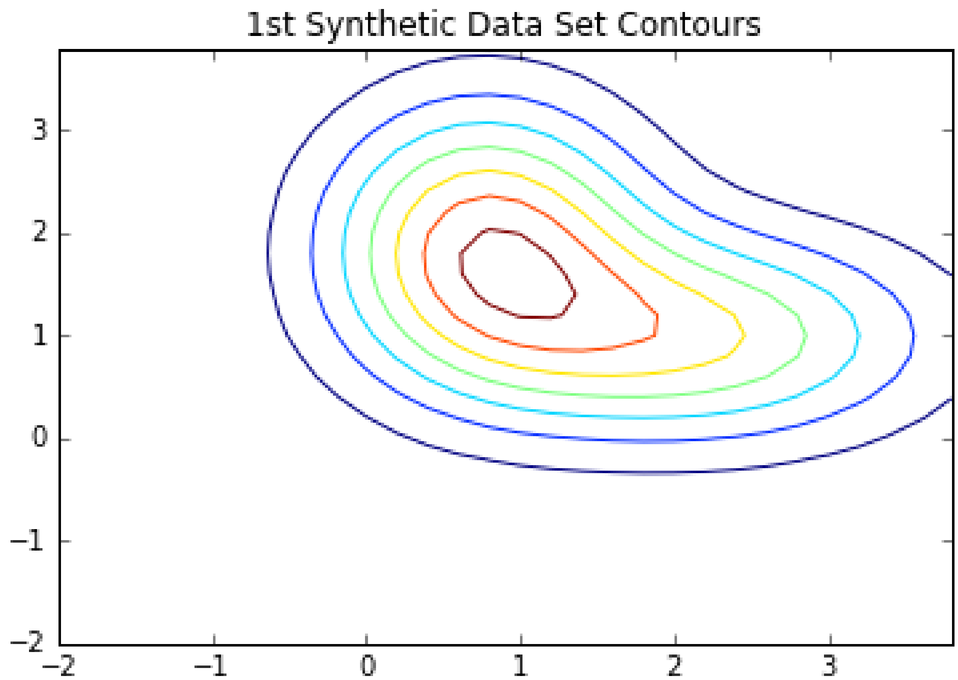

```
In [24]: plt.plot(range(2, 5), [logmarginalL(S1likelihood), logmarginalL(S1likelihood3), logmarg
                 "b")
         plt.title("Log Marginal Likelihood of the first Synthetic dataset over K")
         plt.xlabel("Number of Clusters (K)")
         plt.ylabel("Log Marginal Likelihood")
         plt.show()
```



It seems that log marginal likelihood prefers lower values of K. This may be because, as I have found, the more clusters there are, the less stable they are.

5 e) Plot contours of the marginal density function.

```
In [30]: # for convenience, I'll use matplotlib's bivariate normal function
         import matplotlib.mlab as mlab

         # make the x and y coordinates of the grid
         X, Y = numpy.meshgrid(numpy.arange(-2, 4, .2), numpy.arange(-2, 4, .2))
         Z = numpy.zeros((30, 30))
         # and the values at those points
         for k in range(2):
             Z += numpy.mean(S1pi[-1000:, k])*mlab.bivariate_normal(
                 X, Y, sigmax = numpy.mean(1/S1sigma[-1000:, k, 0]), sigmay = numpy.mean(1/S1sig
                 mux = numpy.mean(S1mu[-1000:, k, 0]), muy = numpy.mean(S1mu[-1000:, k, 1]))

         plt.contour(X, Y, Z)
         plt.title("1st Synthetic Data Set Contours")
         plt.show()
```



For this data set, I put a prior on the mean with mean 1.5 and variance .5 and a prior on the variance with both a and b equal to 1. This reflects the prior parameters on the mean fairly well, with the center of the contour around 1.5 in both dimensions. This is probably in part due to the fact that I decided on the prior mean of 1.5 based on a glace at the data. The variance appears to be closer to 1.5 or 2 than my prior guess of 1 suggests.