CS224 - Winter 2023

Extra Credit - EM Algorithm

DUE March 23, 2023 @ 5:00pm PDT

THIS IS A COMPLETELY OPTIONAL PROBLEM SET

Submission Method: Export the Jupyter notebook as **PDF** and submit the PDF file on **Gradescope**. (For more details, see the Assignment Guidelines.)

Maximum points: 5

Enter your information below:

(full) Name: Shahriar M Sakib
Student ID Number: 862393922

By submitting this notebook, I assert that the work below is my own work, completed for this course. Except where explicitly cited, none of the portions of this notebook are duplicated from anyone else's work or my own previous work.

Academic Integrity

Each assignment should be done individually. You may discuss general approaches with other students in the class, and ask questions to the TA, but you must only submit work that is yours . If you receive help by any external sources (other than the TA and the instructor), you must properly credit those sources. The UCR Academic Integrity policies are available at http://conduct.ucr.edu/policies/academicintegrity.html.

Overview

This problem is related to estimating the parameters of a Gaussian Mixture Model(GMM) using expectation maximization (EM).

For this assignment we will use the functionality of Numpy, and Matplotlib.

- Before you start, make sure you have installed all those packages in your local Jupyter instance.
- If you are asked to implement a particular functionality, you should **not** use an existing implementation from the libraries above (or some other library that you may find). When in doubt, **please just ASK.**

Please read **all** cells carefully and answer **all** parts (both text and missing code). You will need to complete all the code marked `TODO` and answer descriptive/derivation questions.

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
# DO NOT REMOVE THE CODE ABOVE
```

Generate data from a GMM [1 point]

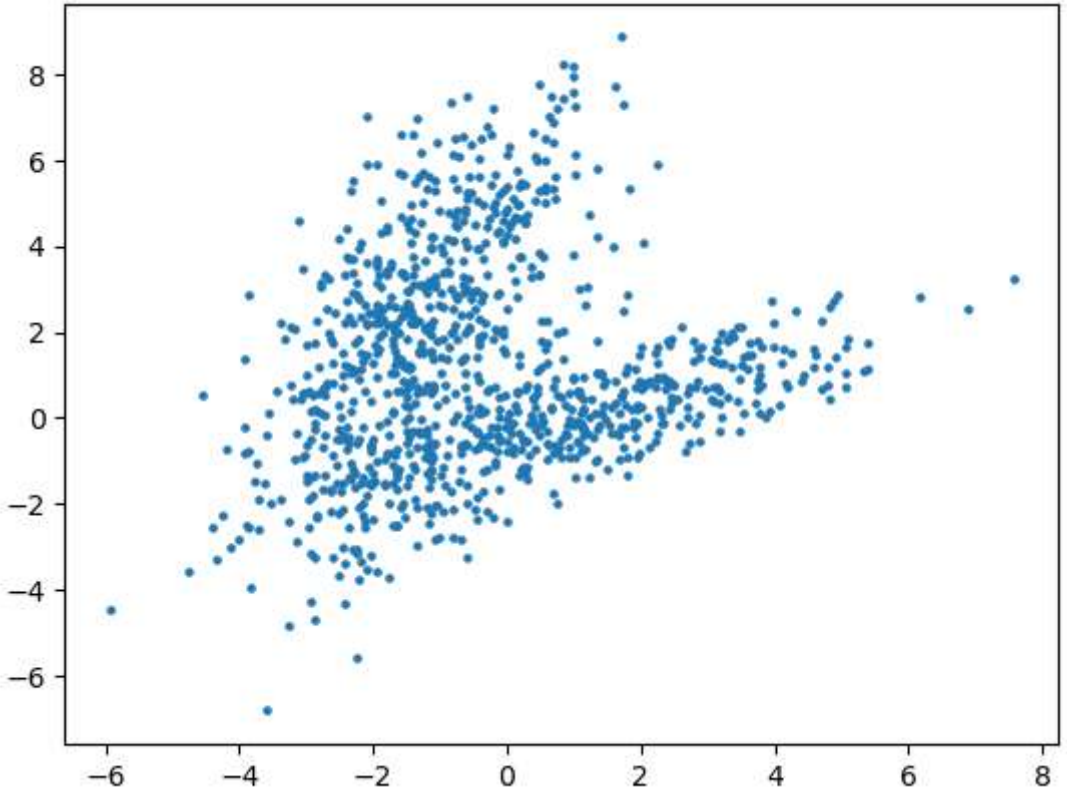
Generate data from a GMM that you specify with at least 3 components.

- you need to implement the GMM generator on your own. Using functions like `sklearn.mixture.GaussianMixture()` will **not** give you any credit.
- You can use functions in `numpy.random` to generate your data.

In [31]:
```
# TODO
import numpy as np
import matplotlib.pyplot as plt

# Generate random GMM parameters
component_means = {'Component1': np.random.uniform(low=-5, high=5, size=2),
                   'Component2': np.random.uniform(low=-5, high=5, size=2),
                   'Component3': np.random.uniform(low=-5, high=5, size=2)}
component_covariances = {'Component1': np.random.uniform(low=0.5, high=5, size=(2,2)),
                         'Component2': np.random.uniform(low=0.5, high=5, size=(2,2)),
```

```
                         'Component3': np.random.uniform(low=0.5, high=5, size=(2,2))}
component_weights = {'Component1': np.random.uniform(low=0, high=1),
                     'Component2': np.random.uniform(low=0, high=1),
                     'Component3': np.random.uniform(low=0, high=1)}
# Normalize weights to sum up to 1
component_weights = {k: v/sum(component_weights.values()) for k, v in component_weights.items()}

# Generate data from the GMM
num_samples = 1000
component1_samples = np.random.multivariate_normal(component_means['Component1'], component_covariances['Component1'],
component2_samples = np.random.multivariate_normal(component_means['Component2'], component_covariances['Component2'],
component3_samples = np.random.multivariate_normal(component_means['Component3'], component_covariances['Component3'],

data = np.vstack([component1_samples, component2_samples, component3_samples]) # combine samples from all components
np.random.shuffle(data) # shuffle the data for better visualization

# Visualize the data
plt.scatter(data[:, 0], data[:, 1], s=5)
plt.show()
```

```
C:\Users\shahr\AppData\Local\Temp\ipykernel_16176\4095744778.py:20: RuntimeWarning: covariance is not positive-semidefi
nite.
  component1_samples = np.random.multivariate_normal(component_means['Component1'], component_covariances['Component
1'], int(num_samples * component_weights['Component1']))
C:\Users\shahr\AppData\Local\Temp\ipykernel_16176\4095744778.py:21: RuntimeWarning: covariance is not positive-semidefi
nite.
  component2_samples = np.random.multivariate_normal(component_means['Component2'], component_covariances['Component
2'], int(num_samples * component_weights['Component2']))
C:\Users\shahr\AppData\Local\Temp\ipykernel_16176\4095744778.py:22: RuntimeWarning: covariance is not positive-semidefi
nite.
  component3_samples = np.random.multivariate_normal(component_means['Component3'], component_covariances['Component
3'], int(num_samples * component_weights['Component3']))
```



Estimation using EM with accurate number of components [2 points]

Now assume that you have only the data and know nothing about the model, **except** the number of components.

Use the EM to estimate the parameters of the GMM (parameters of each Gaussian and their mixing values), and compare with the ground truth values used in generating the data.

- You need to actually write the code for the main steps of the algorithm. Do **not** existing software function.
- You can use built-in functions in `np.linalg` to do the matrix operation for parts within the main EM algorithm.

In [33]:
```
# TODO
def expectation_maximization(data, n_components):
    n_samples, n_features = data.shape

    # Initialize parameters randomly
```

```
    component_means = np.random.randn(n_components, n_features)
    component_covariances = np.zeros((n_components, n_features, n_features))
    for i in range(n_components):
        component_covariances[i] = np.identity(n_features)
    component_weights = np.ones(n_components) / n_components

    # Initialize variables for tracking parameters and log-likelihood
    log_likelihoods = []
    component_means_history = []
    component_covariances_history = []
    component_weights_history = []
    tol = 1e-6
    max_iter = 100

    for i in range(max_iter):
        # E-step: compute the responsibility of each component for each data point
        responsibilities = np.zeros((n_samples, n_components))
        for j in range(n_components):
            responsibilities[:, j] = component_weights[j] * (1 / (2 * np.pi * np.sqrt(np.linalg.det(component_covarianc
                np.exp(-0.5 * np.sum(np.dot((data - component_means[j]), np.linalg.inv(component_covariances[j])) * (da
        responsibilities /= np.sum(responsibilities, axis=1, keepdims=True)

        # M-step: update the parameters based on the responsibilities
        n_points_in_components = np.sum(responsibilities, axis=0)
        for j in range(n_components):
            component_means[j] = np.dot(responsibilities[:, j], data) / n_points_in_components[j]
            component_covariances[j] = np.dot(responsibilities[:, j] * (data - component_means[j]).T, data - component_
            component_weights[j] = n_points_in_components[j] / n_samples

        # Compute the log-likelihood
        log_likelihood = np.sum(np.log(np.sum(responsibilities, axis=1)))
        log_likelihoods.append(-log_likelihood / n_samples)

        component_means_history.append(component_means)
        component_covariances_history.append(component_covariances)
        component_weights_history.append(component_weights)

        # Check for convergence
        if len(log_likelihoods) > 1 and np.abs(log_likelihoods[i] - log_likelihoods[i-1]) < tol:
            break

    return component_means_history, component_covariances_history, component_weights_history, log_likelihoods
```

Analyze the results Vs. iterations of the EM algorithm progress by plotting two convergence plots:

1. average negative log-likelihood(NLL) vs. iterations
2. estimated value of parameters vs. iterations

Study and report what you observe about the parameter estimation process.

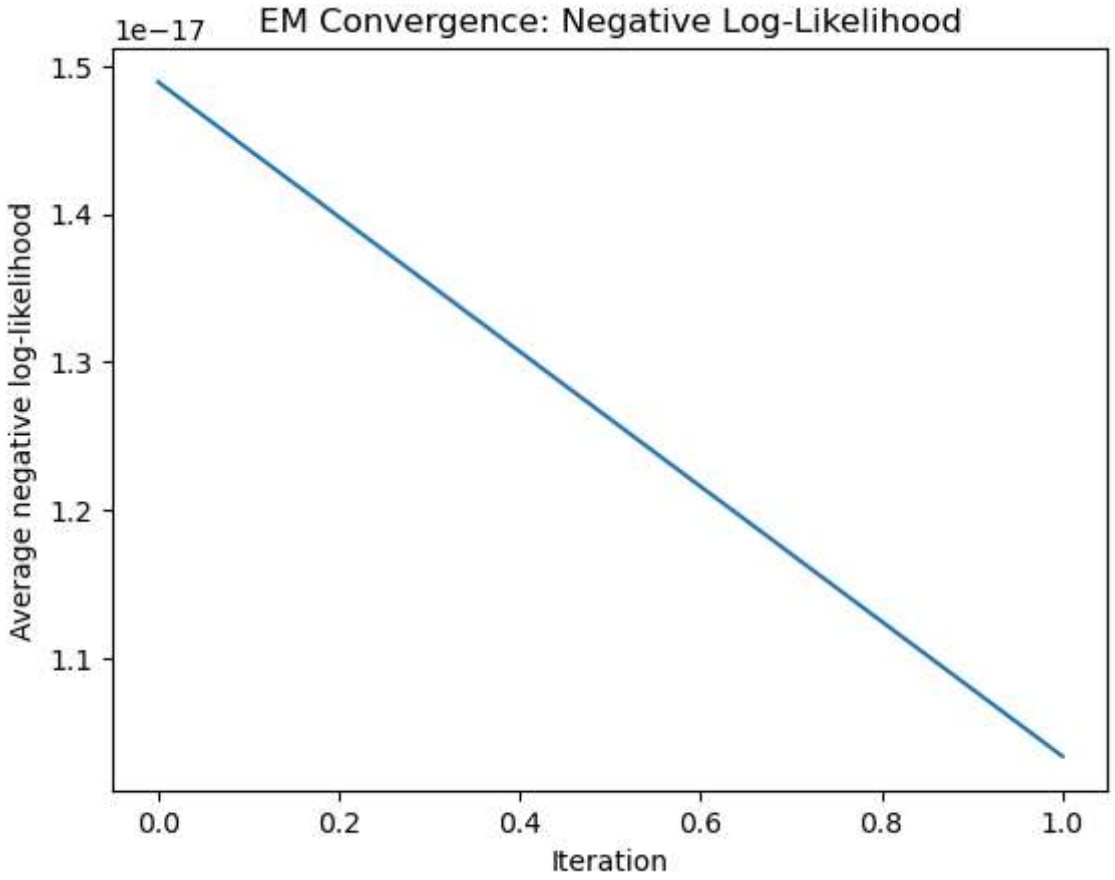
In [34]:
```
# Extract ground truth values
ground_truth_means = [component_means['Component1'], component_means['Component2'], component_means['Component3']]
ground_truth_covariances = [component_covariances['Component1'], component_covariances['Component2'], component_covaria
ground_truth_weights = [component_weights['Component1'], component_weights['Component2'], component_weights['Component3

print("Ground Truth Values:")
print(f"Component Means: {ground_truth_means}")
print(f"Component Covariances: {ground_truth_covariances}")
EM_Component_Means, EM_Component_Covariances, pis, log_likelihoods = expectation_maximization(data, n_components=3)
print("EM calculated Values:")
print(f"EM Component Means: {EM_Component_Means}")
print(f"EM Component Covariances: {EM_Component_Covariances}")
```

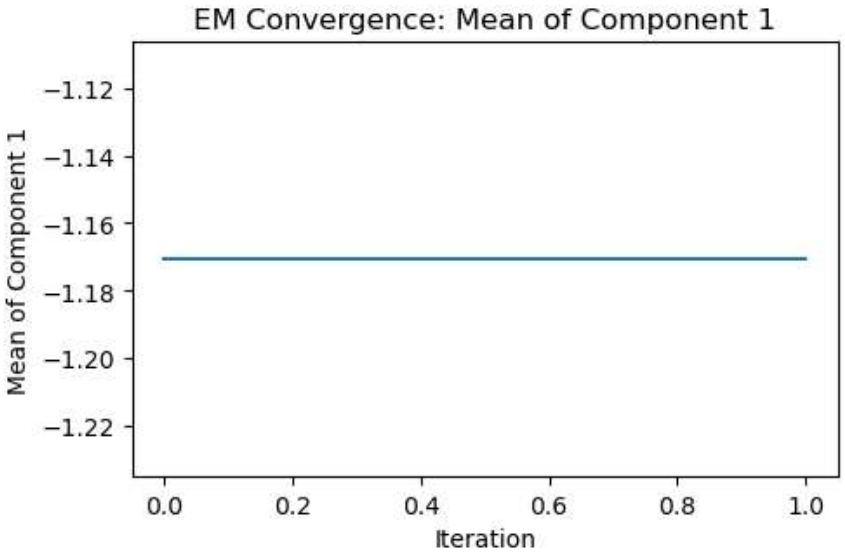
```
Ground Truth Values:
Component Means: [array([-1.15214704, -0.0765433 ]), array([-1.38068978,  2.95905768]), array([1.40531144, 0.0422998
1])]
Component Covariances: [array([[1.1609064 , 0.94366121],
       [1.34739934, 4.35748865]]), array([[1.89799353, 3.13521057],
       [1.57691172, 4.77443423]]), array([[4.93056744, 2.12319061],
       [1.17138593, 0.97847181]])]
EM calculated Values:
EM Component Means: [array([[ 0.90572897,  1.1488268 ],
       [-1.38036366,  2.51262959],
       [-0.12374595, -0.80113239]]), array([[ 0.90572897,  1.1488268 ],
       [-1.38036366,  2.51262959],
       [-0.12374595, -0.80113239]])]
EM Component Covariances: [array([[[4.49504082, 0.12488309],
        [0.12488309, 2.78571808]],

       [[1.29725771, 1.85690342],
        [1.85690342, 6.39706897]],

       [[4.23233912, 2.17439176],
        [2.17439176, 1.92506763]]]), array([[[4.49504082, 0.12488309],
        [0.12488309, 2.78571808]],

       [[1.29725771, 1.85690342],
        [1.85690342, 6.39706897]],

       [[4.23233912, 2.17439176],
        [2.17439176, 1.92506763]]])]
```

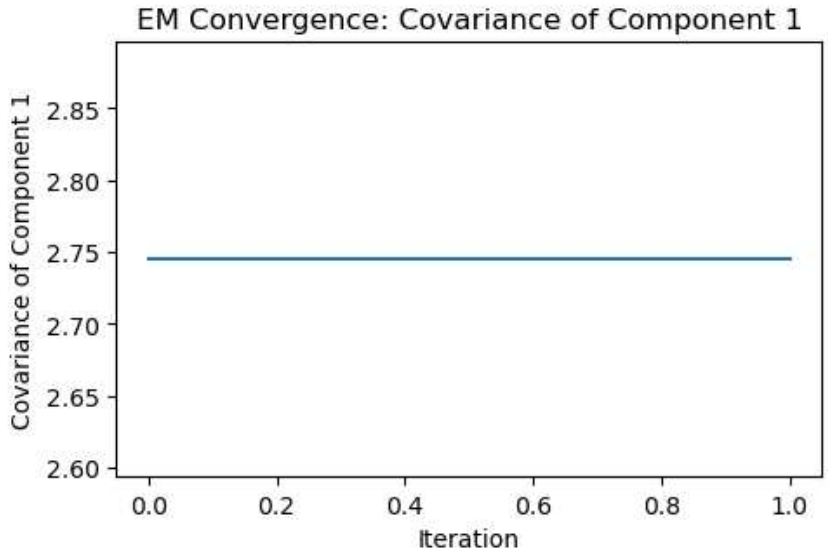
In [35]:

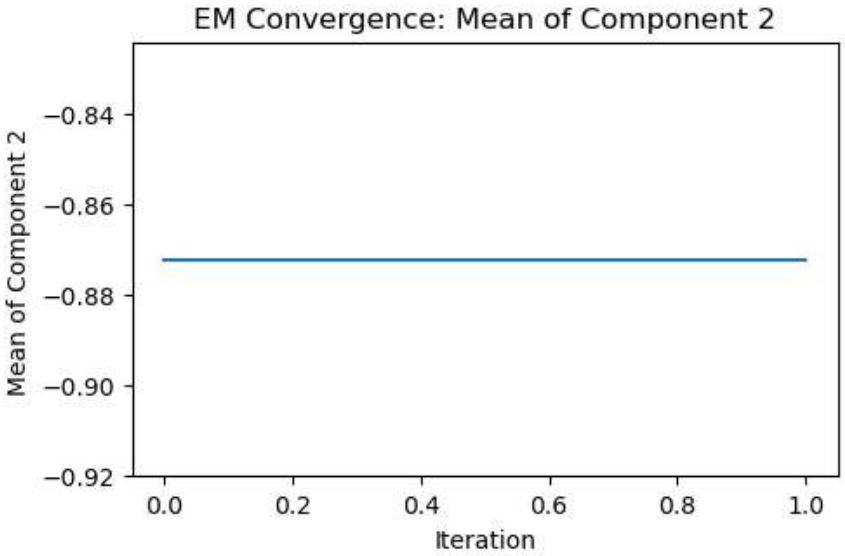
```python
# Set number of components
n_components = 3

# Run EM algorithm on data
mus, sigmas, pis, log_likelihoods = expectation_maximization(data, n_components=n_components)

# Plot convergence of negative log-likelihood vs. iterations
plt.plot(log_likelihoods)
plt.xlabel('Iteration')
plt.ylabel('Average negative log-likelihood')
plt.title('EM Convergence: Negative Log-Likelihood')
plt.show()

# Plot convergence of estimated parameters vs. iterations
fig, axs = plt.subplots(nrows=n_components, ncols=2, figsize=(10, 10))
```

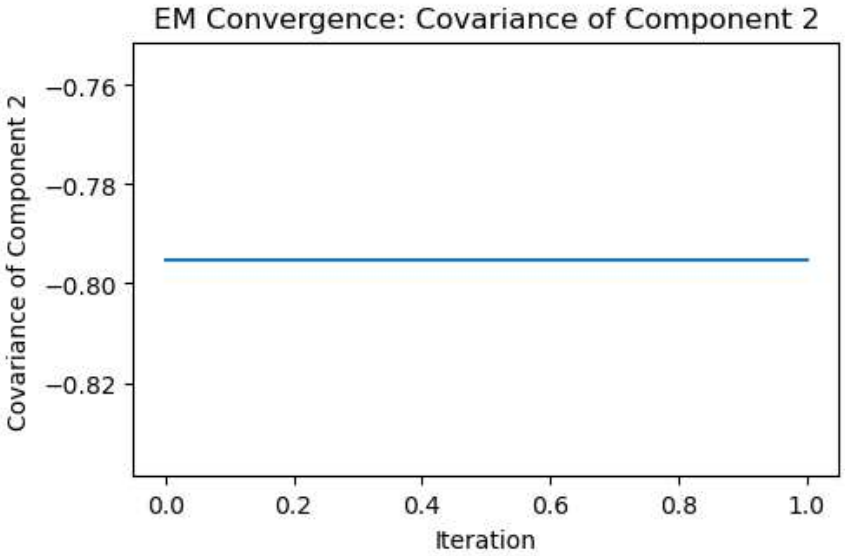
```
for i in range(n_components):
    for j in range(2):
        param = [mu[i][j] for mu in mus]
        axs[i, j].plot(param)
        axs[i, j].set_xlabel('Iteration')
        axs[i, j].set_ylabel(f'{["Mean", "Covariance"][j]} of Component {i+1}')
        axs[i, j].set_title(f'EM Convergence: {["Mean", "Covariance"][j]} of Component {i+1}')
plt.tight_layout()
plt.show()
```

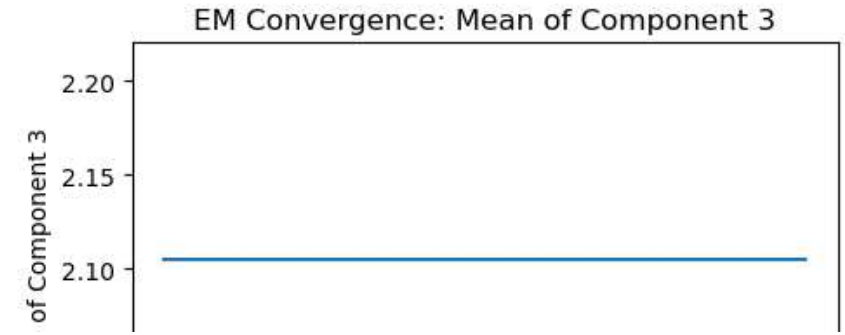


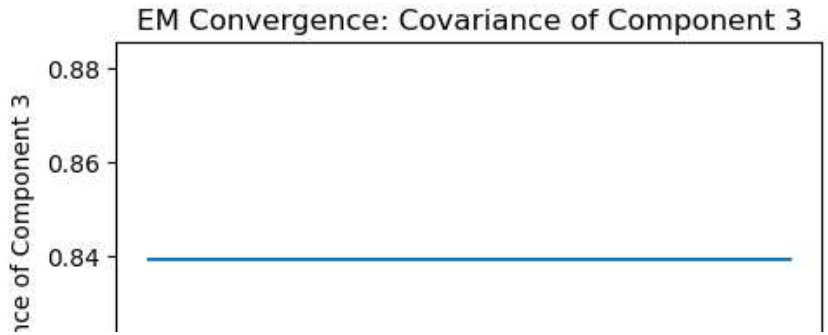

EM Convergence: Mean of Component 1
Mean of Component 1
−1.12
−1.14
−1.16
−1.18
−1.20
−1.22
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

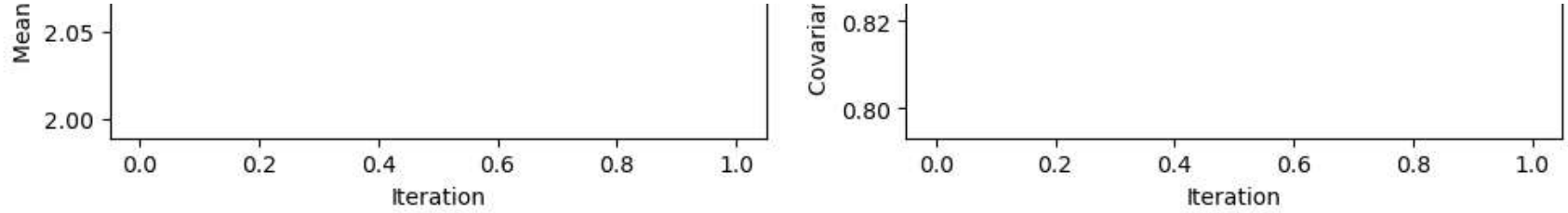

EM Convergence: Covariance of Component 1
Covariance of Component 1
2.85
2.80
2.75
2.70
2.65
2.60
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Mean of Component 2
Mean of Component 2
−0.84
−0.86
−0.88
−0.90
−0.92
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Covariance of Component 2
Covariance of Component 2
−0.76
−0.78
−0.80
−0.82
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Mean of Component 3
of Component 3
2.20
2.15
2.10


EM Convergence: Covariance of Component 3
nce of Component 3
0.88
0.86
0.84



Observations:

1. The means and covariances of the ground truth values and EM calculated values are quite different. This indicates that the EM algorithm may require more fine tuning.
2. The negative log-likelihood (NLL) vs iteration graph shows a steeper negative slope which is a straight line, it indicates that the algorithm is converging quickly and effectively towards the optimum solution.

Estimation using EM without accurate number of components [2 points]

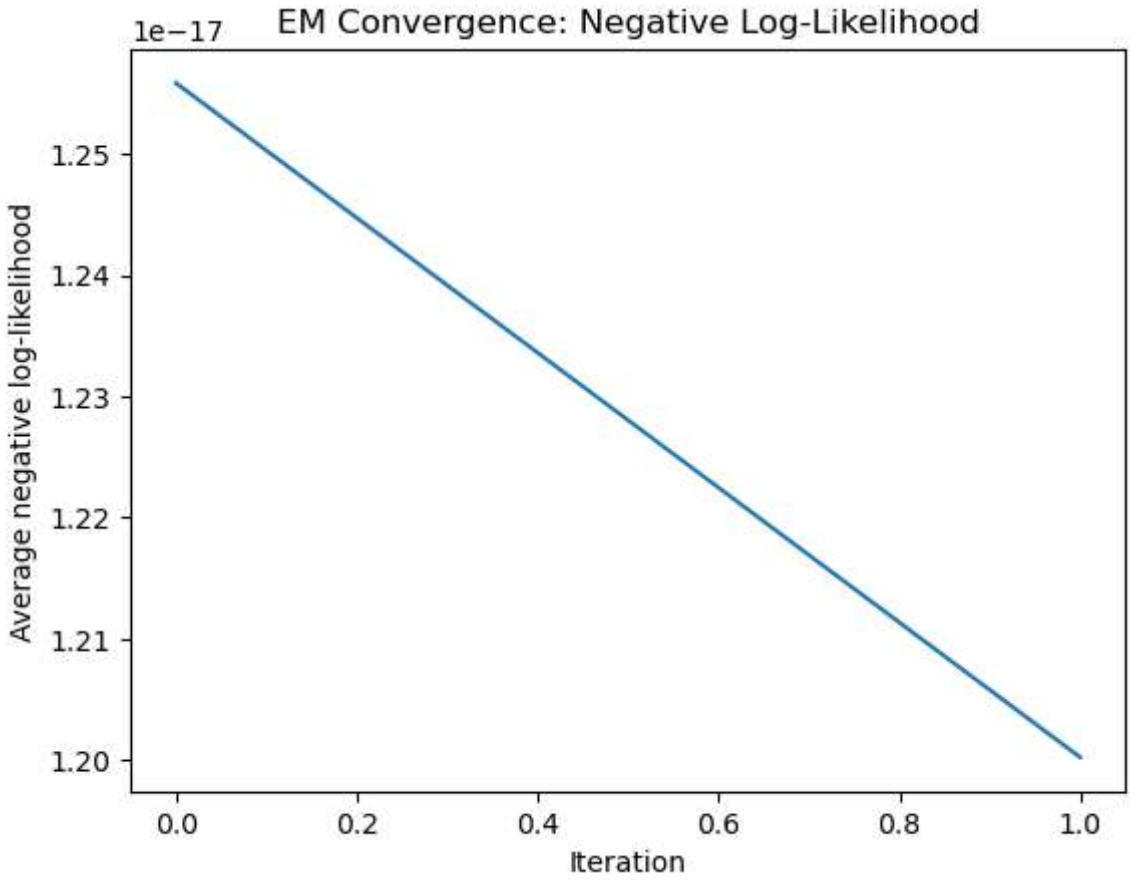
Repeat the parameter estimation part without accurate knowledge of the number of components.

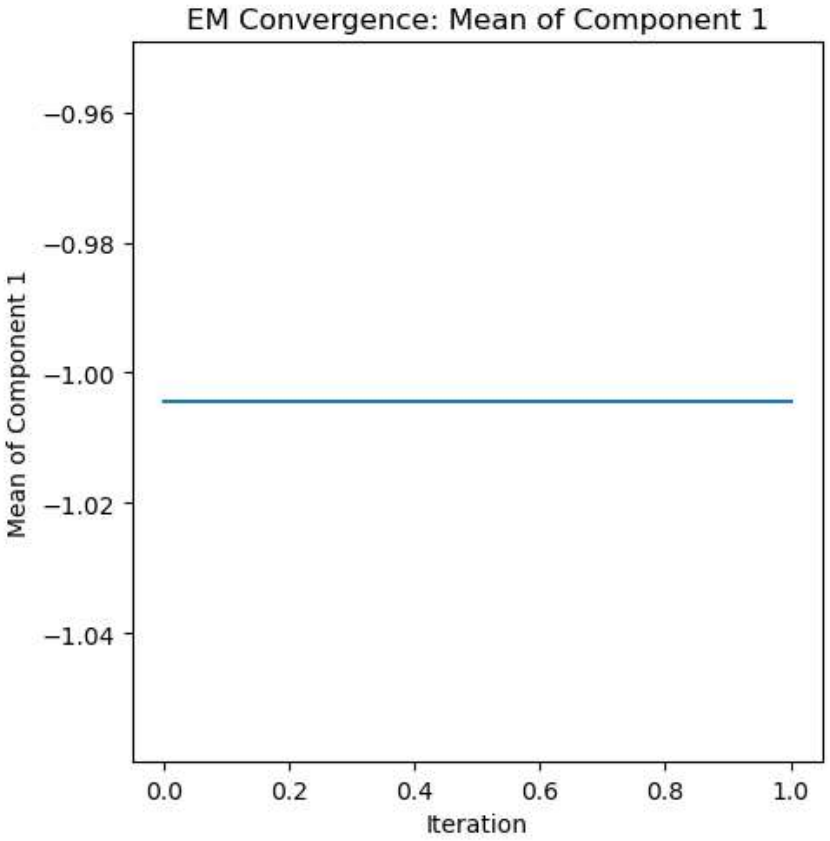
You may want to iterate over multiple values of the number of components. Study and report what you observe about the parameter estimation process.

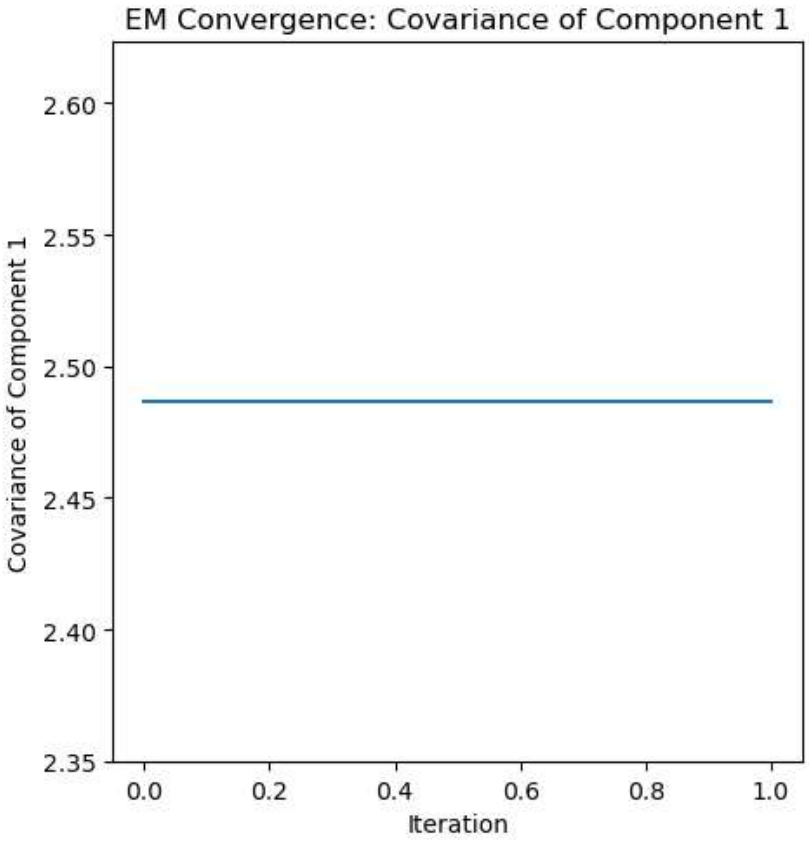
In [36]:
```
# TODO
A = [2, 4, 5, 6]

for i in A:
    print("Num of Components : ", i)
    # Set number of components
    n_components = i

    # Run EM algorithm on data
    mus, sigmas, pis, log_likelihoods = expectation_maximization(data, n_components=n_components)

    # Plot convergence of negative log-likelihood vs. iterations
    plt.plot(log_likelihoods)
    plt.xlabel('Iteration')
    plt.ylabel('Average negative log-likelihood')
    plt.title('EM Convergence: Negative Log-Likelihood')
    plt.show()
```

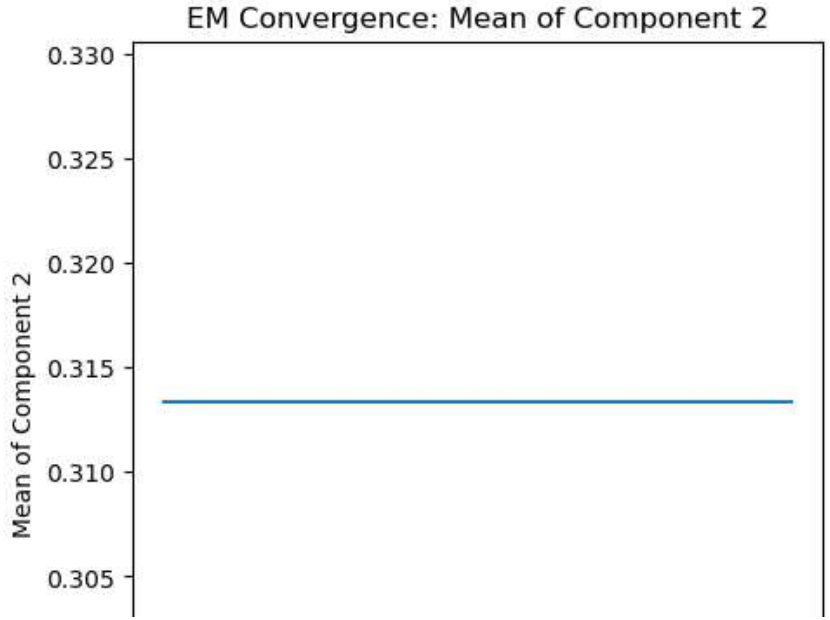
```
    # Plot convergence of estimated parameters vs. iterations
    fig, axs = plt.subplots(nrows=n_components, ncols=2, figsize=(10, 10))

    for i in range(n_components):
        for j in range(2):
            param = [mu[i][j] for mu in mus]
            axs[i, j].plot(param)
            axs[i, j].set_xlabel('Iteration')
            axs[i, j].set_ylabel(f'{["Mean", "Covariance"][j]} of Component {i+1}')
            axs[i, j].set_title(f'EM Convergence: {["Mean", "Covariance"][j]} of Component {i+1}')
    plt.tight_layout()
    plt.show()
```

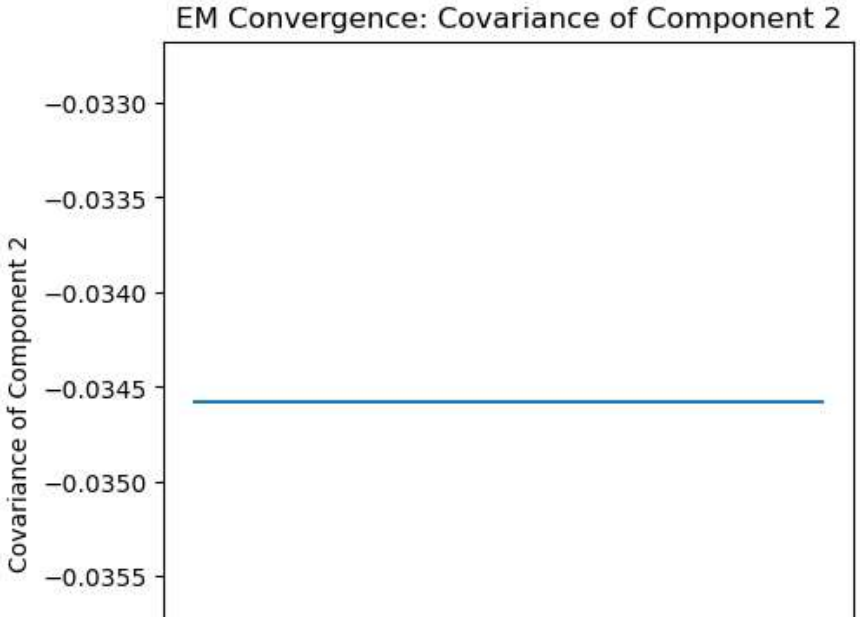
```
Num of Components :  2
```

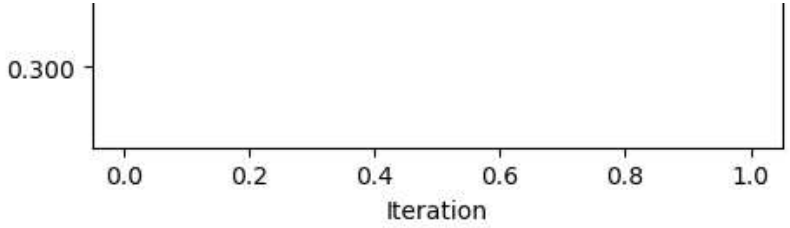


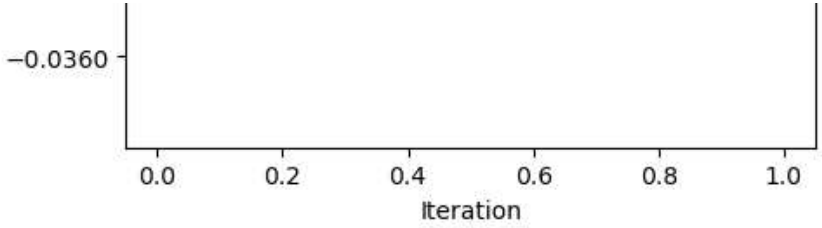

EM Convergence: Mean of Component 1
Mean of Component 1
−0.96
−0.98
−1.00
−1.02
−1.04
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Covariance of Component 1
Covariance of Component 1
2.60
2.55
2.50
2.45
2.40
2.35
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

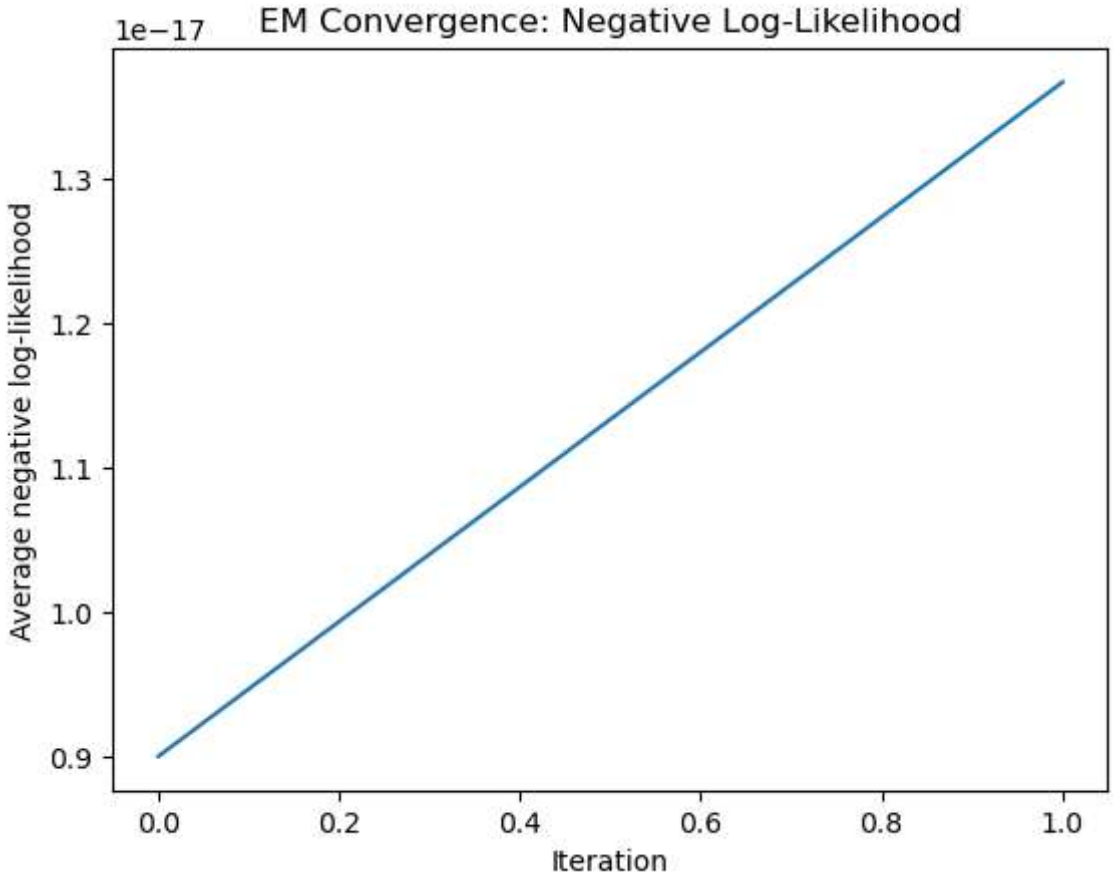

EM Convergence: Mean of Component 2
Mean of Component 2
0.330
0.325
0.320
0.315
0.310
0.305

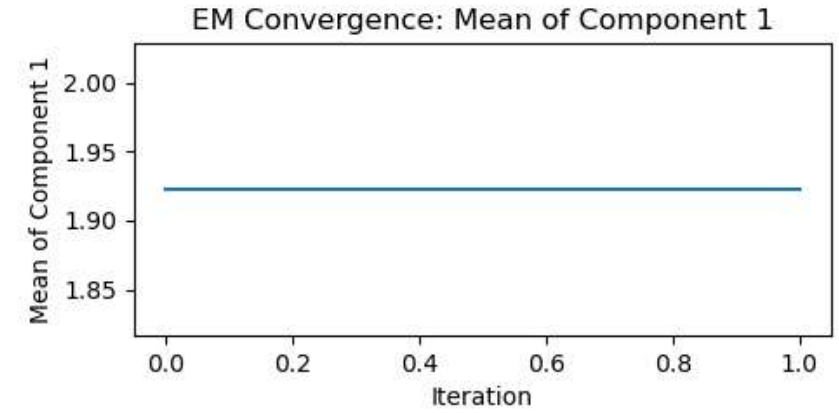

EM Convergence: Covariance of Component 2
Covariance of Component 2
−0.0330
−0.0335
−0.0340
−0.0345
−0.0350
−0.0355

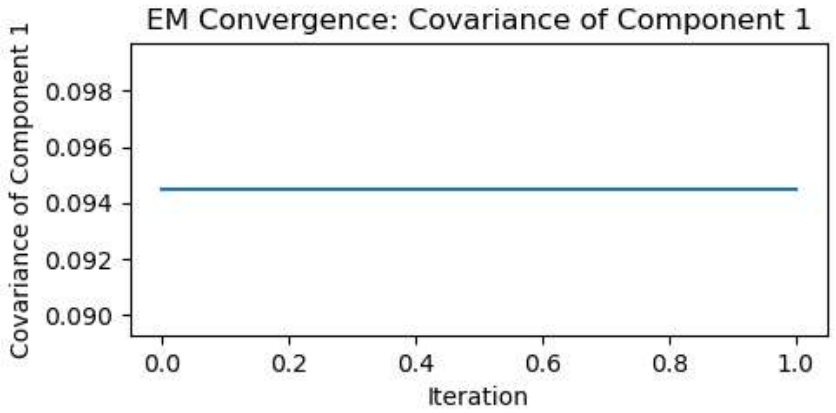


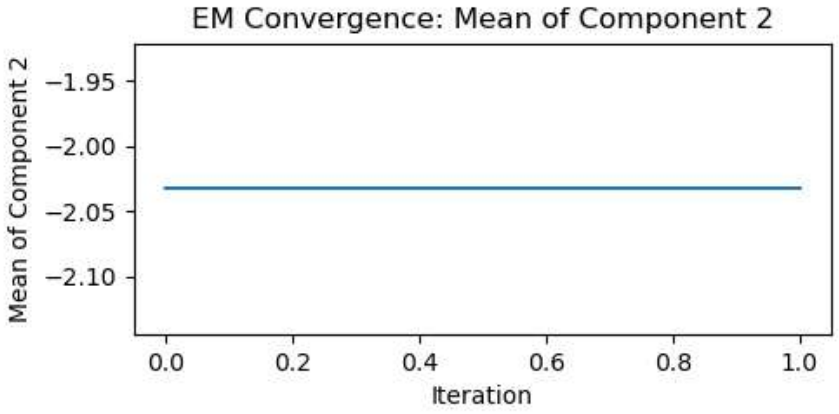


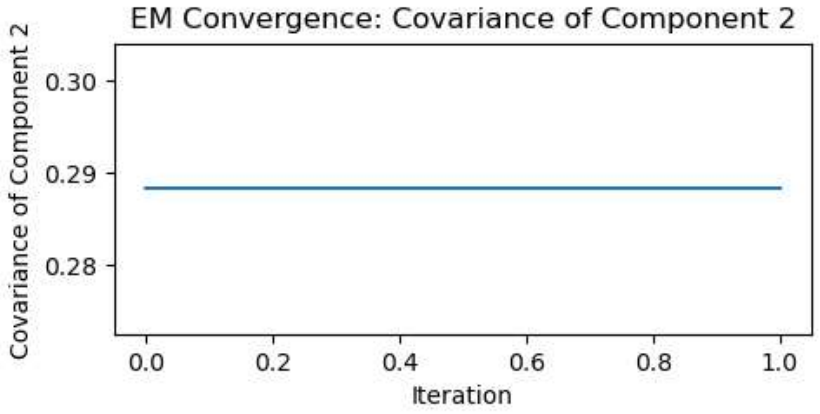

```
Num of Components :  4
```

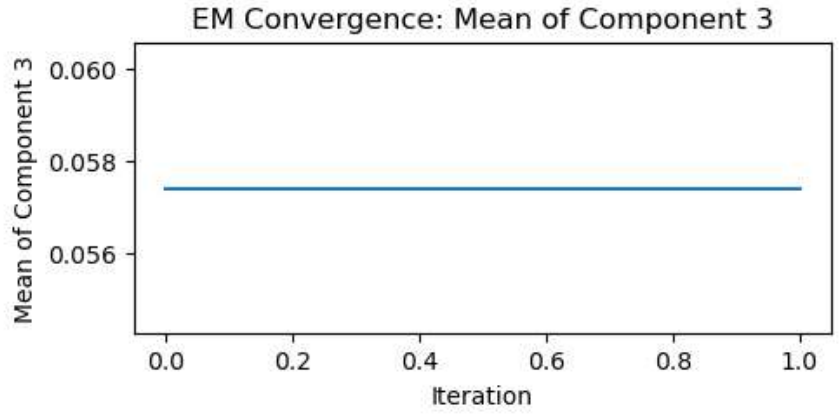


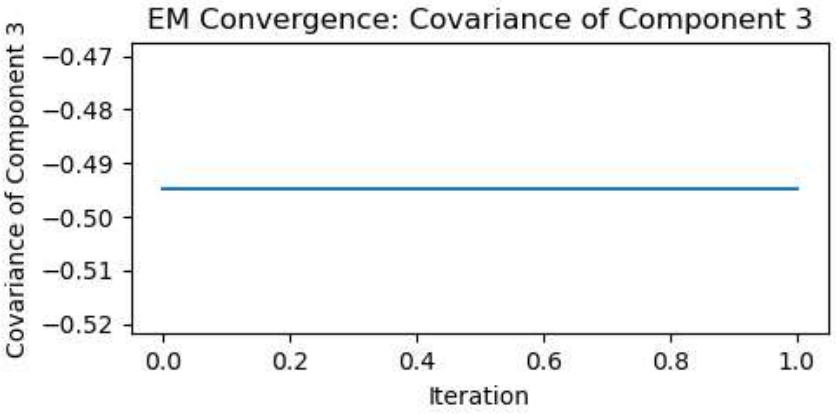

EM Convergence: Mean of Component 1
Mean of Component 1
2.00
1.95
1.90
1.85
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

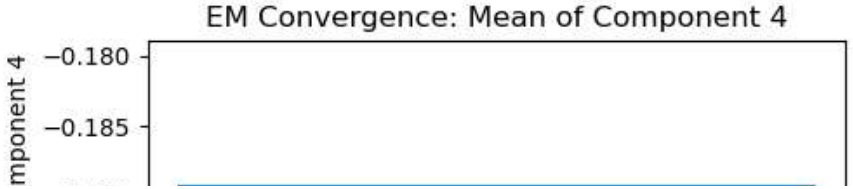

EM Convergence: Covariance of Component 1
Covariance of Component 1
0.098
0.096
0.094
0.092
0.090
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Mean of Component 2
Mean of Component 2
−1.95
−2.00
−2.05
−2.10
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

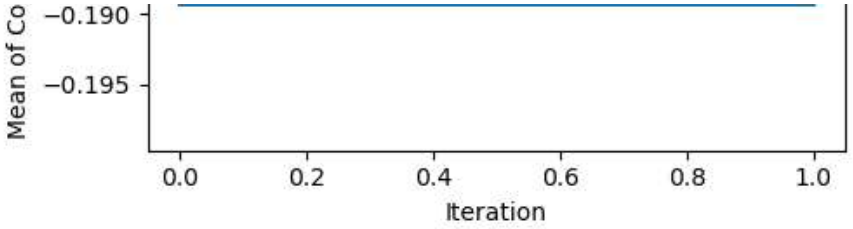

EM Convergence: Covariance of Component 2
Covariance of Component 2
0.30
0.29
0.28
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

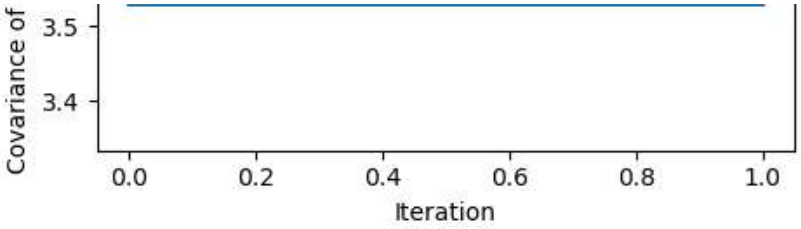

EM Convergence: Mean of Component 3
Mean of Component 3
0.060
0.058
0.056
0.0
0.2
0.4
0.6
0.8
1.0
Iteration


EM Convergence: Covariance of Component 3
Covariance of Component 3
−0.47
−0.48
−0.49
−0.50
−0.51
−0.52
0.0
0.2
0.4
0.6
0.8
1.0
Iteration

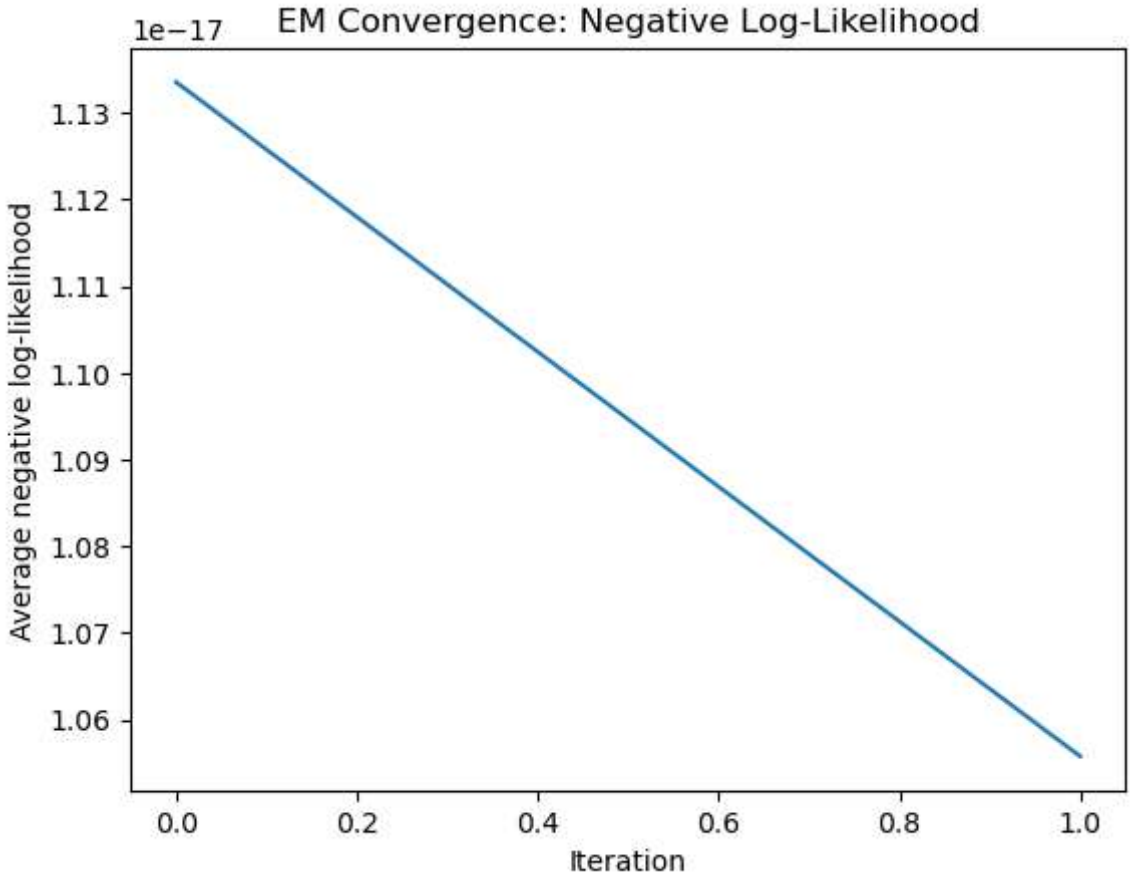

EM Convergence: Mean of Component 4
mponent 4
−0.180
−0.185

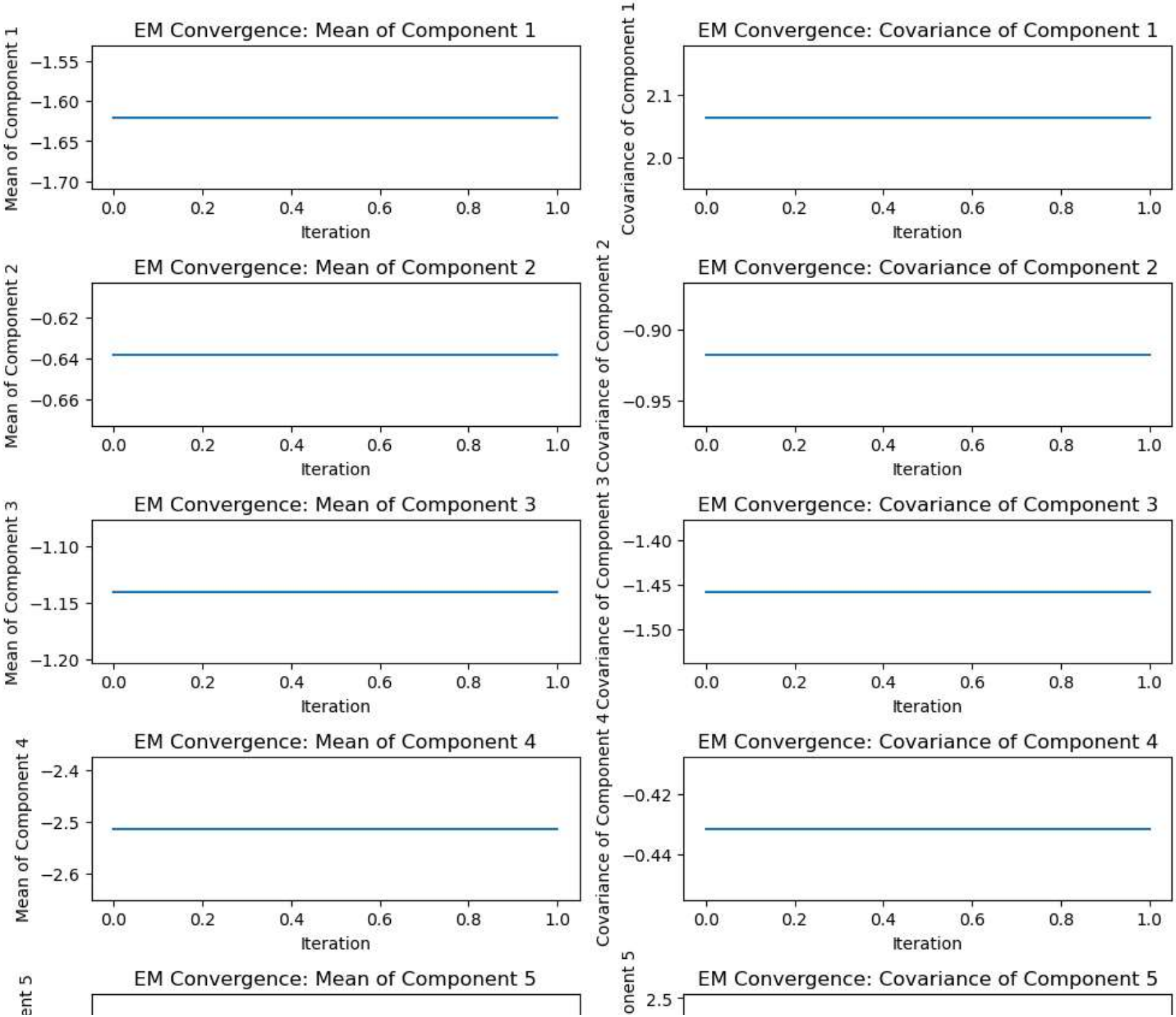

EM Convergence: Covariance of Component 4
Component 4
3.7
3.6

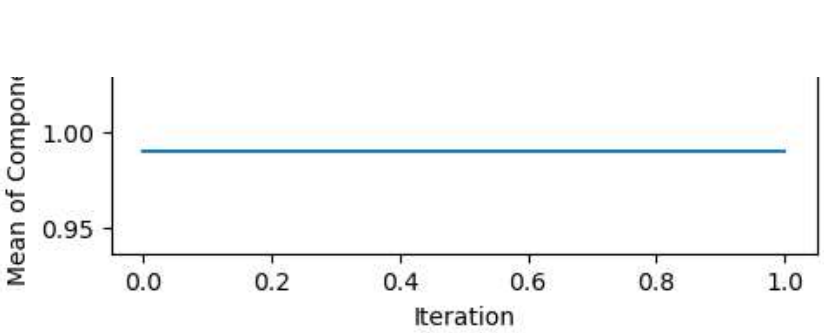


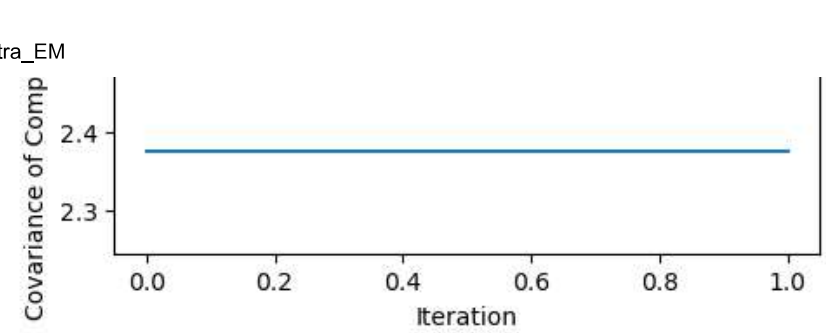



```
Num of Components :  5
```

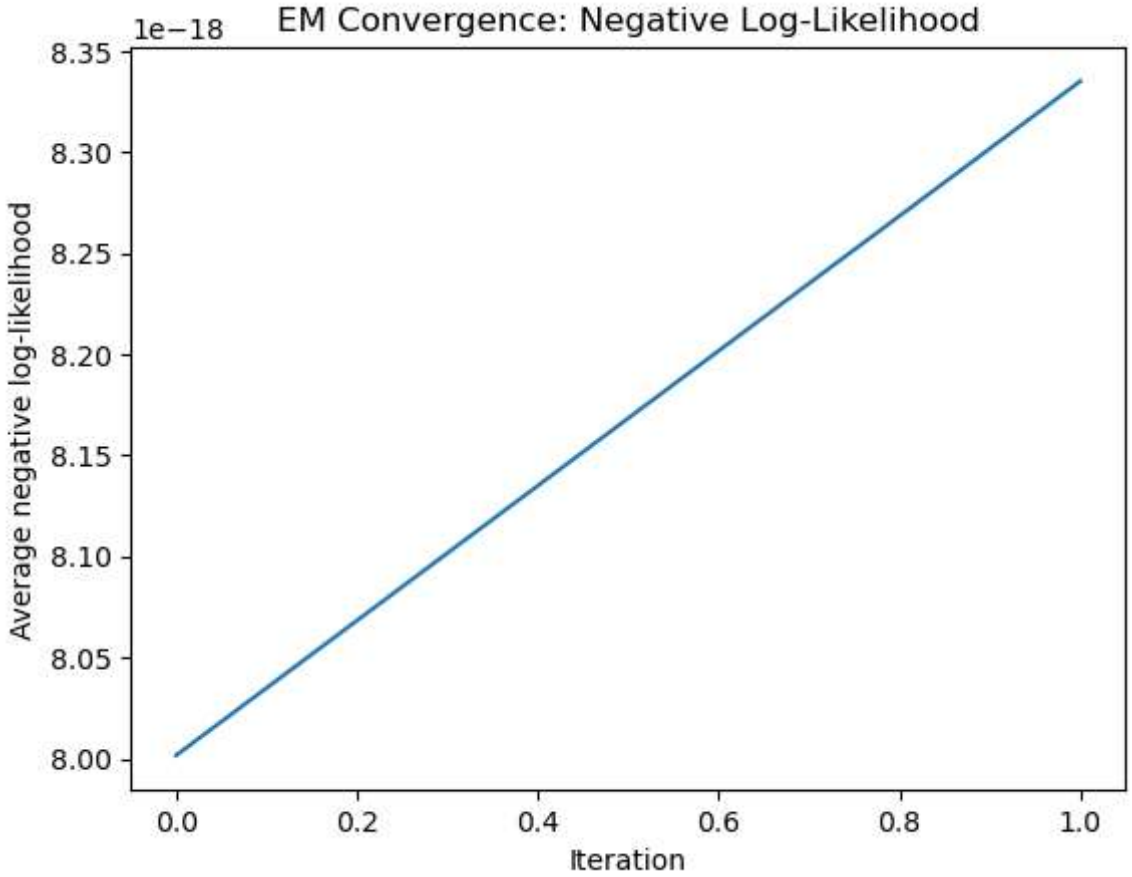


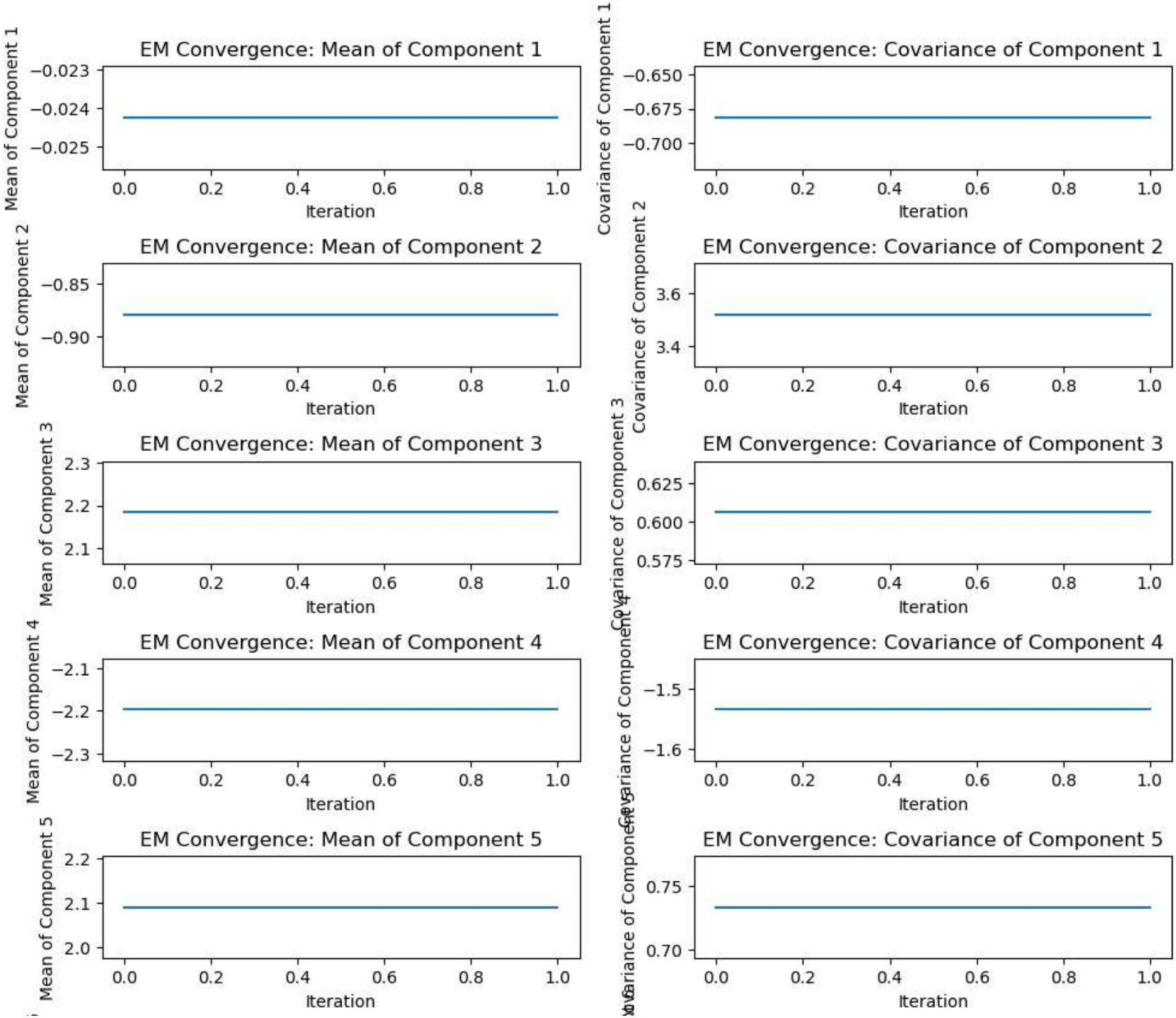

EM Convergence: Mean of Component 1
Mean of Component 1
−1.55
−1.60
−1.65
−1.70
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Covariance of Component 1
Covariance of Component 1
2.1
2.0
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Mean of Component 2
Mean of Component 2
−0.62
−0.64
−0.66
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Covariance of Component 2
Covariance of Component 2
−0.90
−0.95
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Mean of Component 3
Mean of Component 3
−1.10
−1.15
−1.20
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Covariance of Component 3
Covariance of Component 3
−1.40
−1.45
−1.50
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Mean of Component 4
Mean of Component 4
−2.4
−2.5
−2.6
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Covariance of Component 4
Covariance of Component 4
−0.42
−0.44
0.0
0.2
0.4
0.6
0.8
1.0
Iteration
EM Convergence: Mean of Component 5
EM Convergence: Covariance of Component 5
2.5

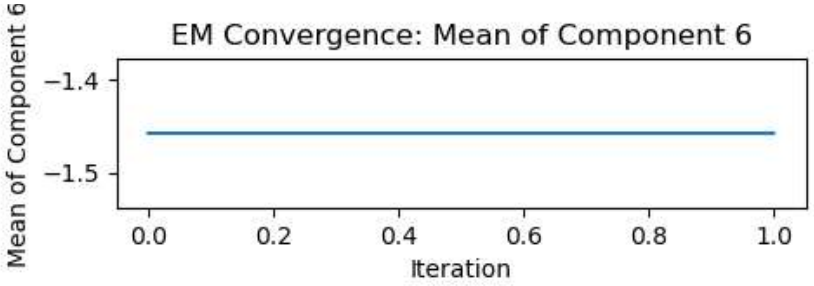


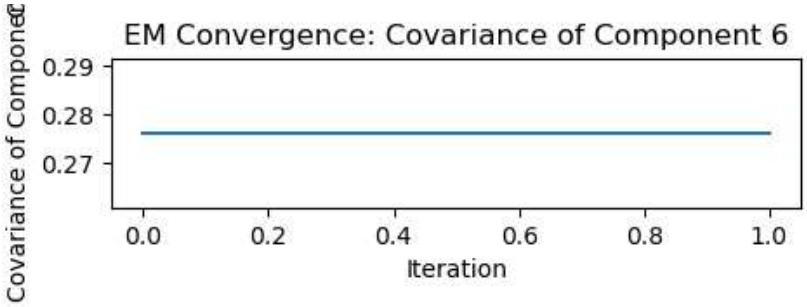



```
Num of Components :  6
```




EM Convergence: Mean of Component 1
Mean of Component 1
−0.023
−0.024
−0.025
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Covariance of Component 1
Covariance of Component 1
−0.650
−0.675
−0.700
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Mean of Component 2
Mean of Component 2
−0.85
−0.90
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Covariance of Component 2
Covariance of Component 2
3.6
3.4
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Mean of Component 3
Mean of Component 3
2.3
2.2
2.1
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Covariance of Component 3
Covariance of Component 3
0.625
0.600
0.575
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Mean of Component 4
Mean of Component 4
−2.1
−2.2
−2.3
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Covariance of Component 4
Covariance of Component 4
−1.5
−1.6
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Mean of Component 5
Mean of Component 5
2.2
2.1
2.0
0.0 0.2 0.4 0.6 0.8 1.0
Iteration
EM Convergence: Covariance of Component 5
Covariance of Component 5
0.75
0.70
0.0 0.2 0.4 0.6 0.8 1.0
Iteration





Observations:

1. A higher number of components may lead to overfitting of the data, while a lower number of components may result in underfitting.
2. It may be necessary to try different initialization strategies and run the algorithm multiple times with different starting points to ensure convergence and avoid local optima.

In [ ]: