COMP 4331 Tut 7

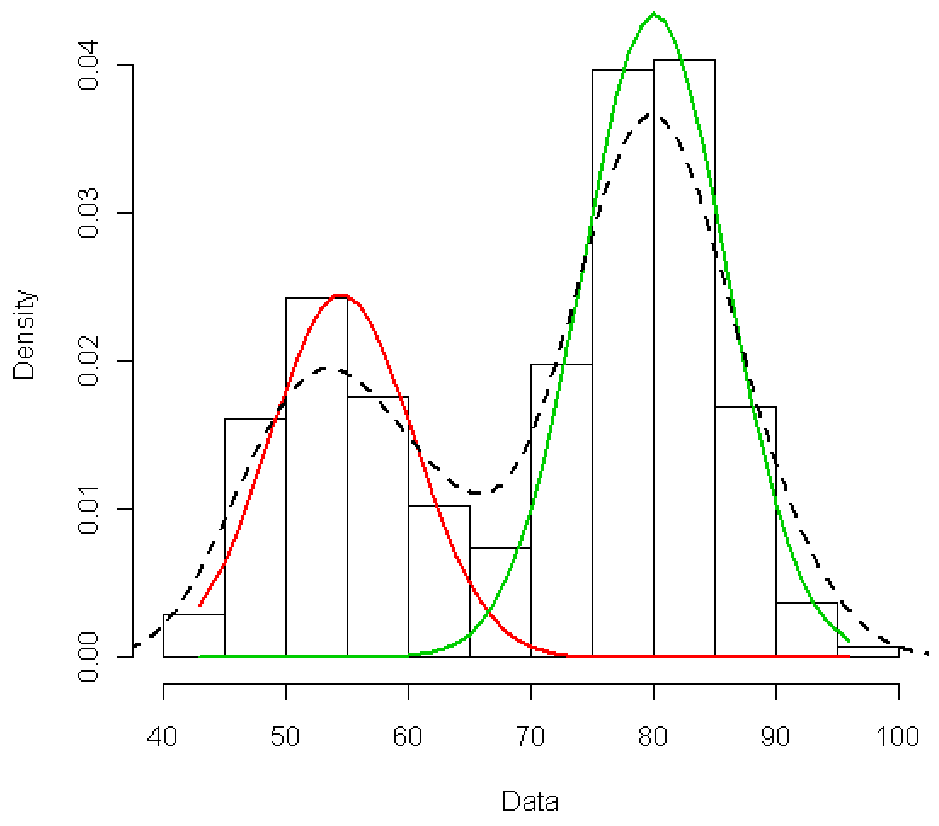
jiaxin Xie

Outline

- A demo of finite mixture model
- Generating data
- EM algorithm

finite mixture model

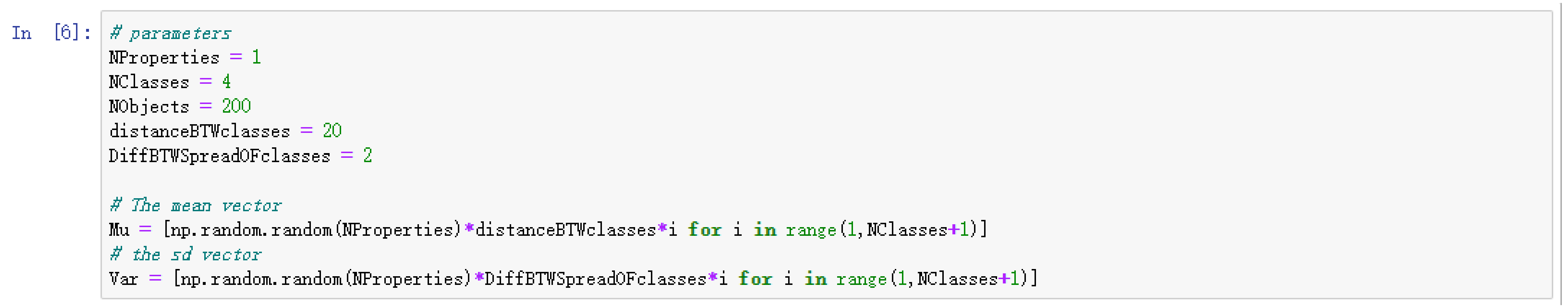
One dimensional Gaussian mixtures
Our data is just points along one dimensions, but coming from K Gaussian distributions. It means that we have one variable about one property, and we have several data points and we do not know to which cluster they belong, i.e. from which Gaussian distribution they are coming from. Here we use a Maximum likelihood estimate to infer the unknown classes of the data points.



finite mixture model

Generating data with known parameters
Here, we have just one property, 4 classes, and 200 data points. The data points are generated from 4 different Gaussian distributions with certain means and variance. Our objective is to infer the clusters or groups for data points. Thus we want to infer the parameters of the Gaussian distributions from which the data was generated, and to infer for each data point that to which cluster they belong.

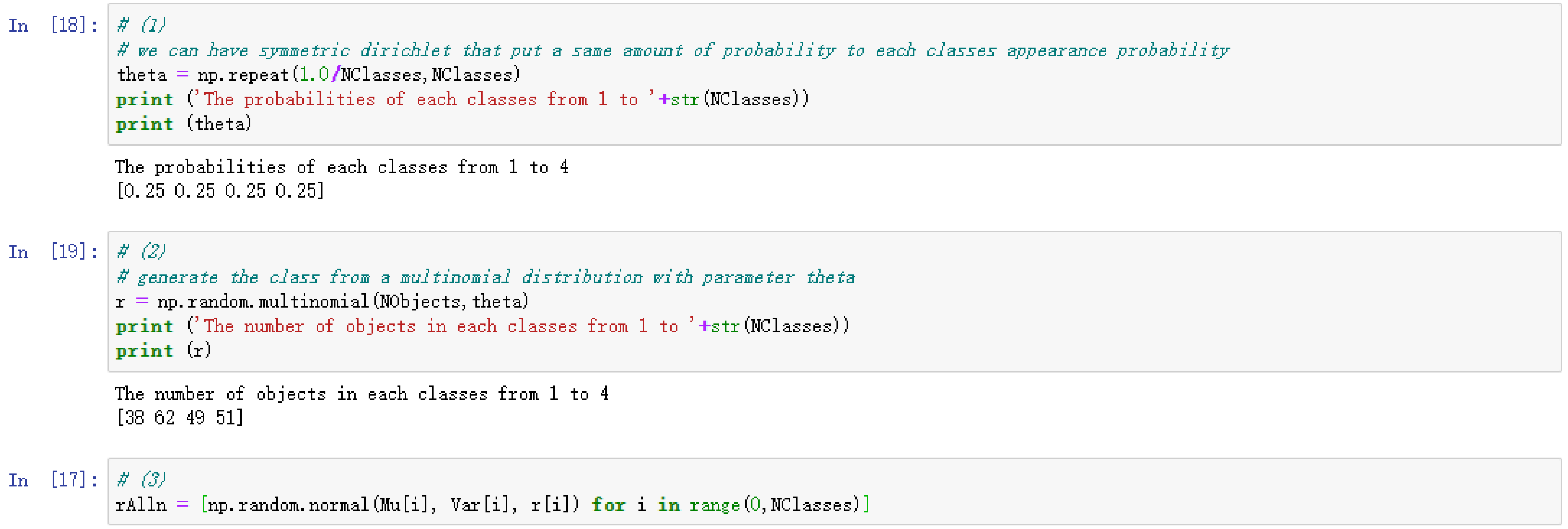
First, we set these known parameters: μ - vector of the means; Σ - the vector of the variances



```
In [6]: # parameters
NProperties = 1
NClasses = 4
NObjects = 200
distanceBTWclasses = 20
DiffBTWSpreadOFclasses = 2

# The mean vector
Mu = [np.random.random(NProperties)*distanceBTWclasses*i for i in range(1,NClasses+1)]
# the sd vector
Var = [np.random.random(NProperties)*DiffBTWSpreadOFclasses*i for i in range(1,NClasses+1)]
```

finite mixture model

In order to generate data from these classes along this one property we need to define a probability distribution over the classes. That will be a multinomial distribution, which have the parameter ω which is a vector of the probability of each class.
(1) We take samples from the dirichlet distribution. It as one parameter: α . This sample will be the ω values for the multinomial distribution;
(2) We generate the number of points in the classes with the parameter ω from the multinomial distribution;
(3) We generate the given number of points from our Gaussian distributions for each classes with the previously defined μ and Σ vectors.



In [18]:
```
# (1)
# we can have symmetric dirichlet that put a same amount of probability to each classes appearance probability
theta = np.repeat(1.0/NClasses,NClasses)
print ('The probabilities of each classes from 1 to '+str(NClasses))
print (theta)
```

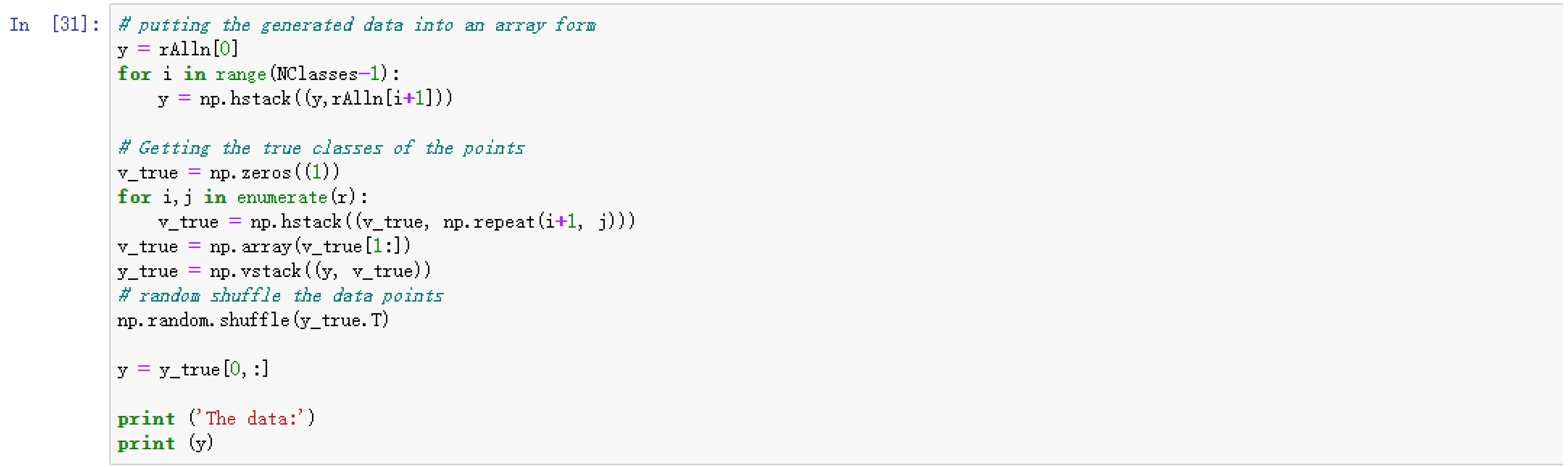
```
The probabilities of each classes from 1 to 4
[0.25 0.25 0.25 0.25]
```

In [19]:
```
# (2)
# generate the class from a multinomial distribution with parameter theta
r = np.random.multinomial(NObjects,theta)
print ('The number of objects in each classes from 1 to '+str(NClasses))
print (r)
```

```
The number of objects in each classes from 1 to 4
[38 62 49 51]
```

In [17]:
```
# (3)
rAlln = [np.random.normal(Mu[i], Var[i], r[i]) for i in range(0,NClasses)]
```

finite mixture model



In [31]:

```
# putting the generated data into an array form
y = rAlln[0]
for i in range(NClasses-1):
    y = np.hstack((y,rAlln[i+1]))

# Getting the true classes of the points
v_true = np.zeros((1))
for i,j in enumerate(r):
    v_true = np.hstack((v_true, np.repeat(i+1, j)))
v_true = np.array(v_true[1:])
y_true = np.vstack((y, v_true))
# random shuffle the data points
np.random.shuffle(y_true.T)

y = y_true[0,:]

print ('The data:')
print (y)
```

Inference with EM algorithm

The EM algorithm has two steps.

Repeat until converged:

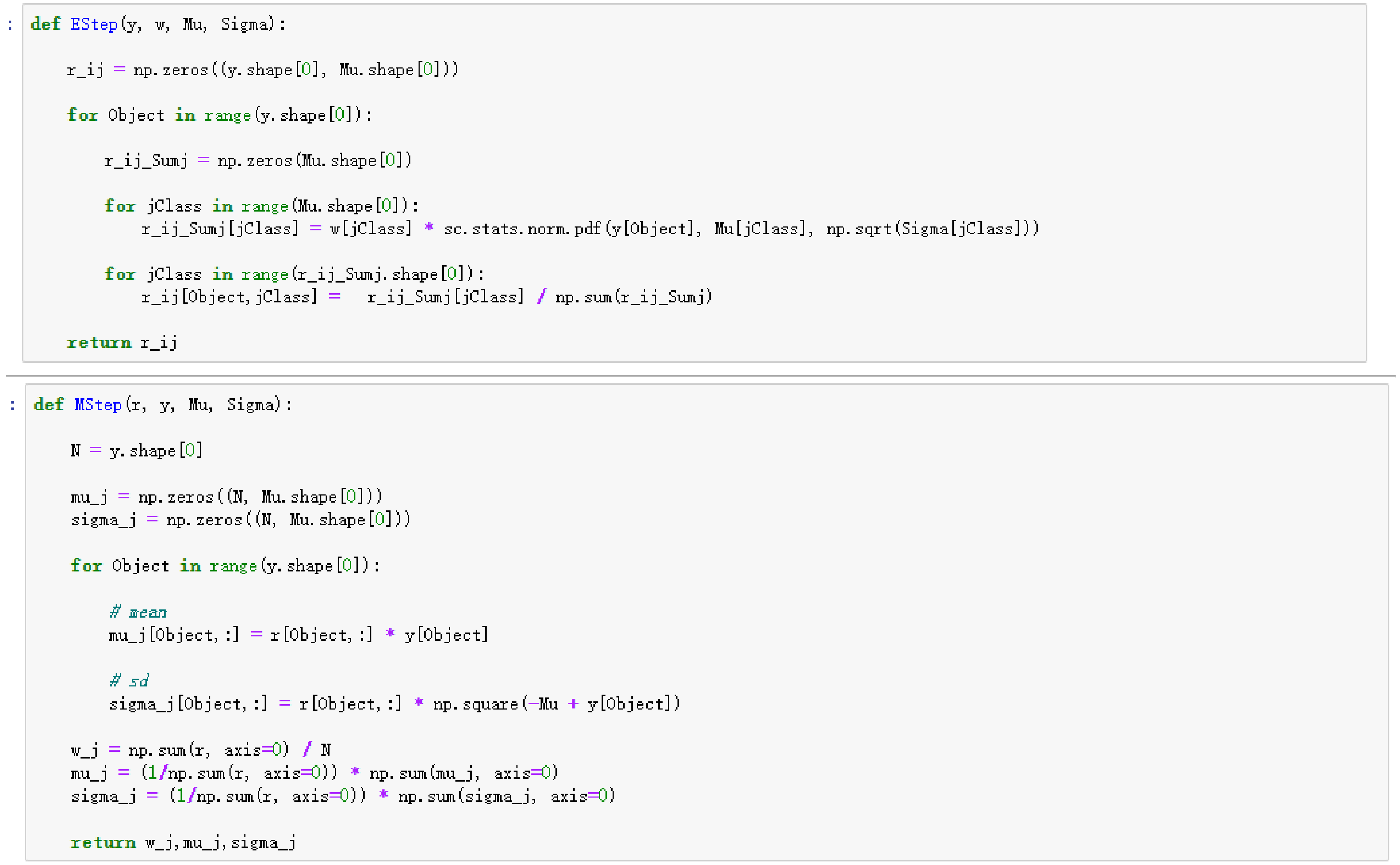
- E-step. Compute the expectations, or responsibilities $r_{j,i}$, of the latent variables υ_i, where j represents the classes, y the data points, and i the ith points:
$$r_{j,i} = P(\upsilon_i = j \mid y_i)$$
- M-step. Compute the maximum likelihood parameters given these responsibilities:
$$\theta \leftarrow \arg\max_{\theta} \sum_i \sum_j r_{j,i}(log P(\upsilon_i = j) + log P(y_i \mid \upsilon_i = j))$$

In our mixture of Gaussian the EM is the following.

Repeat until converged:

- E-step. Compute the expectations, or responsibilities $r_{j,i}$, of the latent variables υ_i, where j represents the classes, y the data points, and i the ith points:
$$r_{j,i} = \frac{\omega_j P(y_i \mid \mu_j, \Sigma_j)}{\sum_j \omega_j P(y_i \mid \mu_j, \Sigma_j)}$$
- M-step. Compute the maximum likelihood parameters given these responsibilities:
  - (1) The weigths for the mixing proportions, ω_j, where N is the number of data points: The estimate of the probability of a class as an outcome is the proportion of times it appears:
$$\omega_j \leftarrow \arg\max \sum_i \sum_j r_{j,i} log P(z_i = j) = \sum_i \sum_j r_{j,i} log \omega_j = \frac{1}{N} \sum_i r_{j,i}$$
  - (2) The means and the variances for the Gauss distributions, μ_j, σ_j : We want to maximize this term:
$$\arg\max \sum_i r_{j,i} log P(y_i \mid \upsilon_i = j) = \sum_i r_{j,i} log Gaussian(y_i; \mu_j, \sigma_j)$$
$$\mu_j \leftarrow \frac{1}{\sum_i r_{j,i}} \sum_i r_{j,i} y_i$$
$$\sigma_j \leftarrow \frac{1}{\sum_i r_{j,i}} \sum_i r_{j,i}(y_i - \mu_j)^2$$

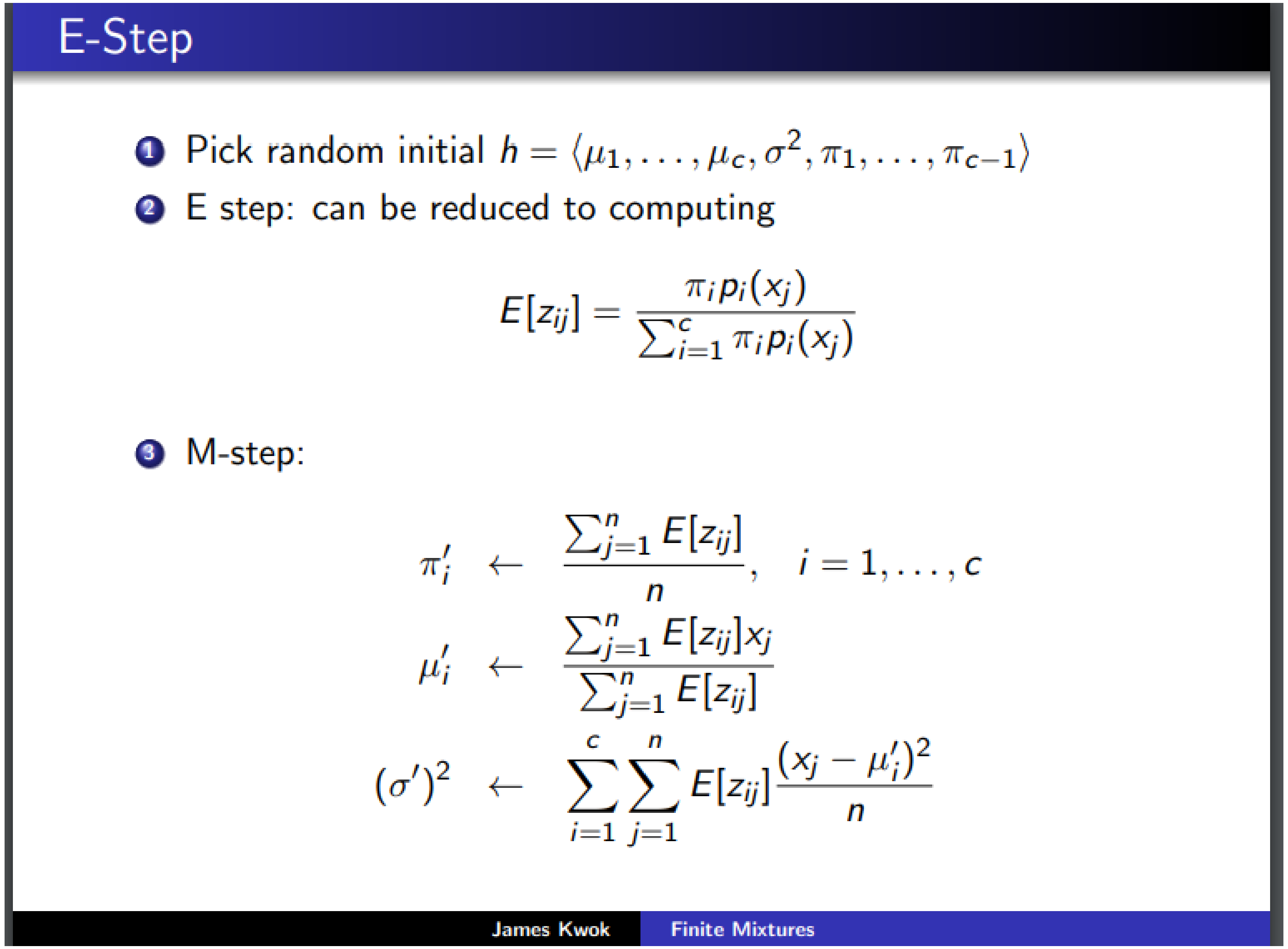
EM algorithm



```
def EStep(y, w, Mu, Sigma):

    r_ij = np.zeros((y.shape[0], Mu.shape[0]))

    for Object in range(y.shape[0]):

        r_ij_Sumj = np.zeros(Mu.shape[0])

        for jClass in range(Mu.shape[0]):
            r_ij_Sumj[jClass] = w[jClass] * sc.stats.norm.pdf(y[Object], Mu[jClass], np.sqrt(Sigma[jClass]))

        for jClass in range(r_ij_Sumj.shape[0]):
            r_ij[Object,jClass] =   r_ij_Sumj[jClass] / np.sum(r_ij_Sumj)

    return r_ij
```

```
def MStep(r, y, Mu, Sigma):

    N = y.shape[0]

    mu_j = np.zeros((N, Mu.shape[0]))
    sigma_j = np.zeros((N, Mu.shape[0]))

    for Object in range(y.shape[0]):

        # mean
        mu_j[Object,:] = r[Object,:] * y[Object]

        # sd
        sigma_j[Object,:] = r[Object,:] * np.square(-Mu + y[Object])

    w_j = np.sum(r, axis=0) / N
    mu_j = (1/np.sum(r, axis=0)) * np.sum(mu_j, axis=0)
    sigma_j = (1/np.sum(r, axis=0)) * np.sum(sigma_j, axis=0)

    return w_j,mu_j,sigma_j
```

Any difference?



E-Step

1. Pick random initial $h = \langle \mu_1, \dots, \mu_c, \sigma^2, \pi_1, \dots, \pi_{c-1} \rangle$
2. E step: can be reduced to computing

$$E[z_{ij}] = \frac{\pi_i p_i(x_j)}{\sum_{i=1}^{c} \pi_i p_i(x_j)}$$

3. M-step:

$$\begin{aligned}
\pi_i' &\leftarrow \frac{\sum_{j=1}^{n} E[z_{ij}]}{n}, \quad i = 1, \dots, c \\
\mu_i' &\leftarrow \frac{\sum_{j=1}^{n} E[z_{ij}] x_j}{\sum_{j=1}^{n} E[z_{ij}]} \\
(\sigma')^2 &\leftarrow \sum_{i=1}^{c} \sum_{j=1}^{n} E[z_{ij}] \frac{(x_j - \mu_i')^2}{n}
\end{aligned}$$