TONGJI UNIVERSITY

同濟大學

《模式识别》

高斯混合概型参数估计的EM算法

学　　院　电子与信息工程学院

专　　业　计算机科学与技术

学　　号　**2130750**

姓　　名　伍　谦

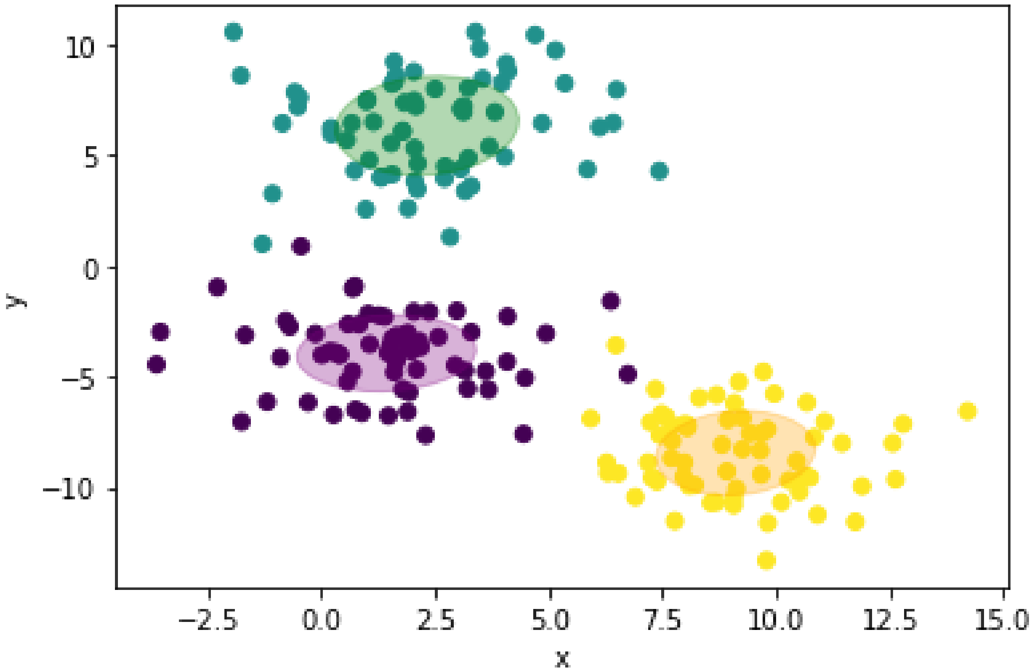
日　　期　**2021-10-20**

1. 前言

本文使用了 EM 算法对高斯混合概率模型进行参数估计。

2. 数据集介绍

使用了 sklearn.datasets 包中的 make_blobs 方法生成 200 个样本点，每个样本具有两维特征，共有三类。



3. 基本步骤

3.1 明确需要估计的参数

在该任务中需要估计的参数有：三个高斯分布的均值μ_1,μ_2,μ_3以及协方差σ_1，σ_2，σ_3，高斯混合模型中各类的先验概率p_1, p_2, p_3, 其中需要满足 p1+p2+p3=1。

3.2 初始化需要估计的参数

3.2.1 先验概率 p

```
import numpy as np
p=np.random.rand(classes)  #根据类别数随机生成p
p=p/p.sum()  #归一化，保证p之和为1
```

根据类别数随机生成先验概率，随后经过归一化来保证它们之和为1

3.2.2 高斯分布均值μ

```
means=np.random.rand(classes,dimention)  # shape [3,2]
```

根据类别数和特征维度生成均值，均值的尺寸为[类别数，维度]

3.2.3 高斯分布协方差σ

```
covs=np.empty((classes,dimention,dimention))
for i in range(classes):
    covs[i]=np.eye(dimention)*np.random.rand(1)*10
```

根据类别和均值生成协方差，协方差尺寸为[类别数，维度，维度]

3.3 使用 EM 算法进行一次参数更新的完整过程

3.3.1 计算每个样本属于某个类别的后验概率

$$T_{j,i}=\frac{p_j^{(t)}N(Y_i|\mu_j^{(t)},\sigma_j^{(t)})}{\sum_{j=1}^{c}p_j^{(t)}N(Y_i|\mu_j^{(t)},\sigma_j^{(t)})}$$

```
posterior= density * p #shape [2000,3]
posterior= posterior/posterior.sum(axis=1,keepdims=True)  # 归一化,使得各样本属于每个类别的概率之和为1
```

3.3.2 更新先验概率 p

$$p_j^{(t+1)}=\frac{1}{n}\sum_{i=1}^{n}T_{j,i}^{(t)}$$

```
p_hat=posterior.sum(axis=0)
p_hat=p_hat/num_of_data
```

3.3.3 更新均值μ

$$u_j^{(t+1)}=\frac{\sum_{i=1}^{n}T_{j,i}^{(t)}Y_i}{\sum_{i=1}^{n}T_{j,i}^{(t)}}$$

```
mean_hat=np.matmul(data.T,posterior).T
mean_hat=np.divide(mean_hat,np.sum(posterior,axis=0,keepdims=True).T)
```

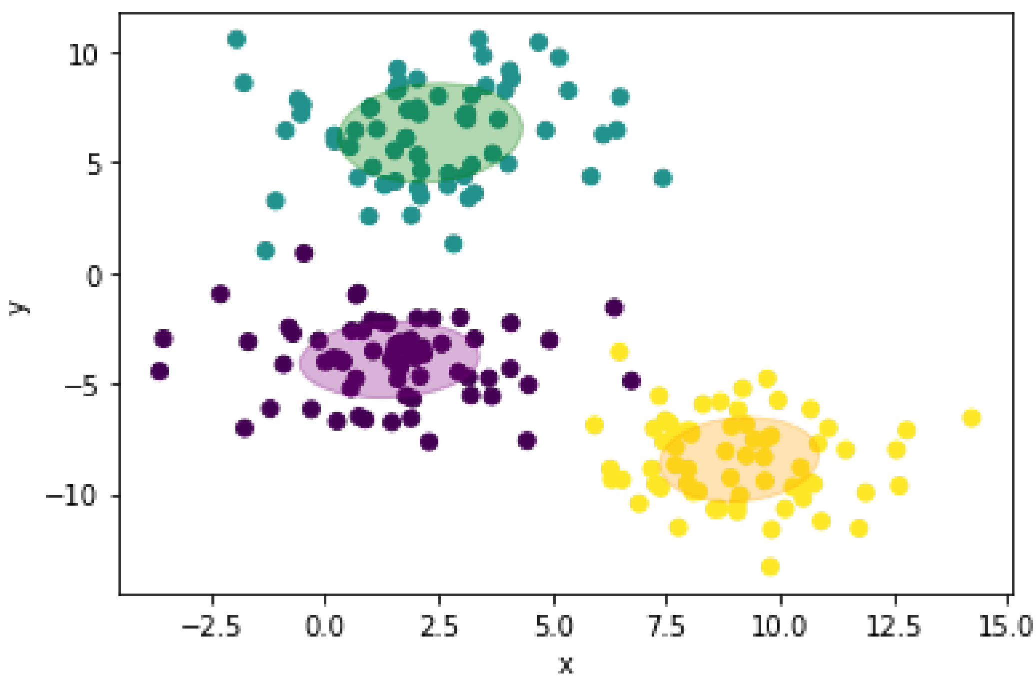
3.3.4 更新协方差σ

$$\sigma_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)} (Y_i - \mu_j^{t+1})(Y_i - \mu_j^{t+1})^T}{\sum_{i=1}^{n} T_{j,i}^{(t)}}$$

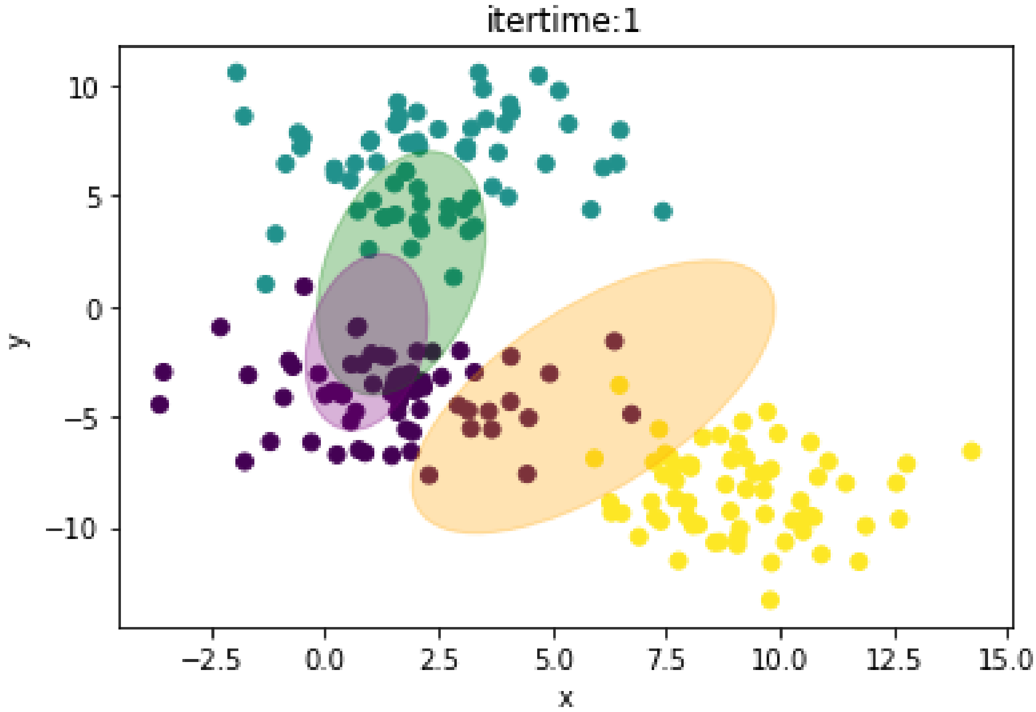
```
cov_hat=np.empty(covs.shape)
posterior_sum=np.sum(posterior,axis=0)
for i in range(classes):
    tmp=data-mean_hat[i]
    cov_hat[i]=np.dot(tmp.T*posterior[:,i],tmp)/posterior_sum[i]
```

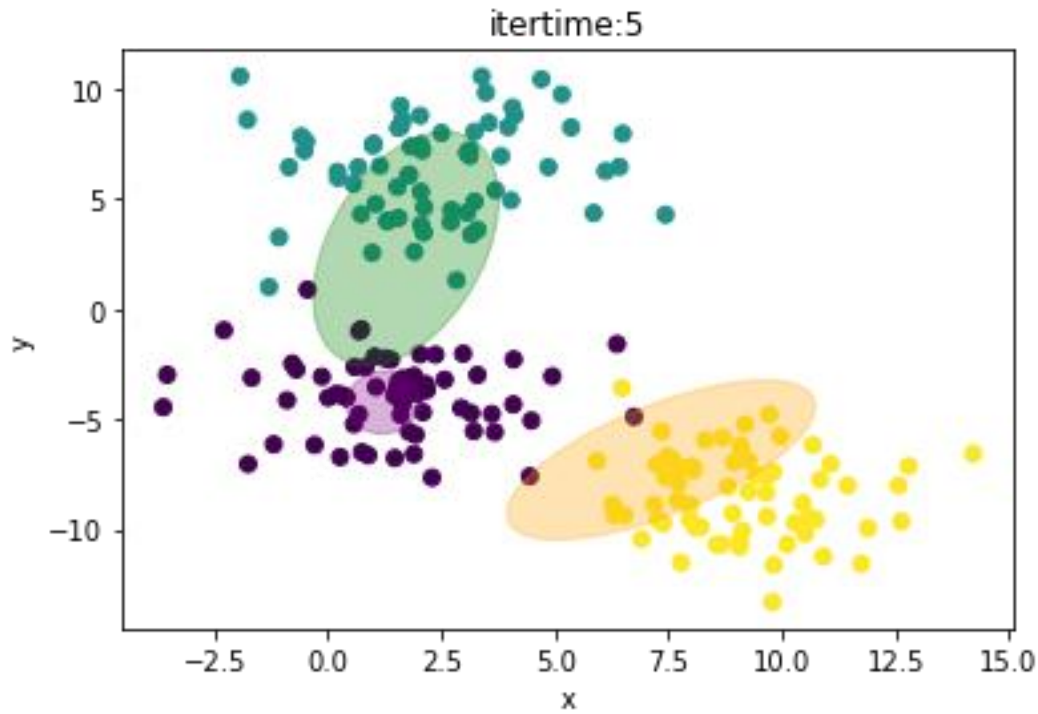
实验结果

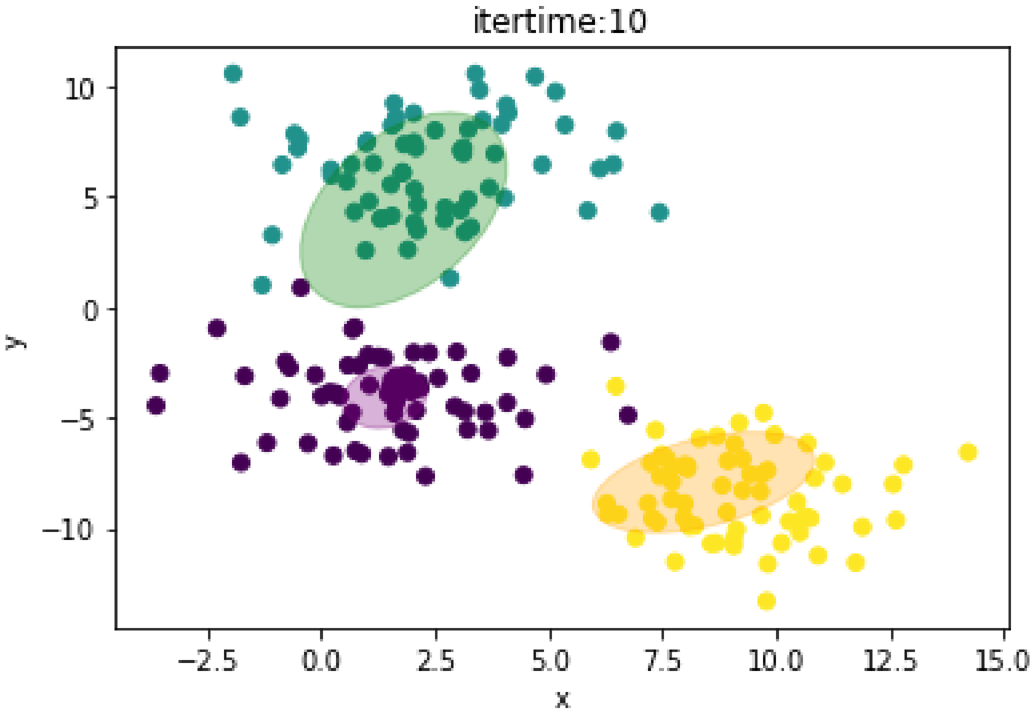
原始数据

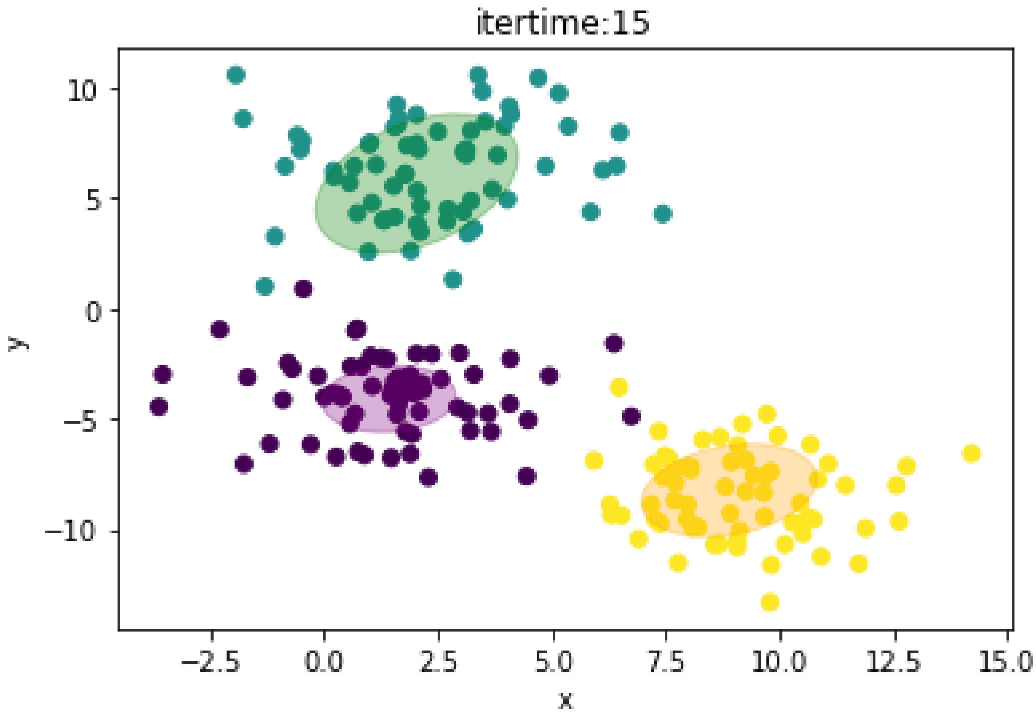


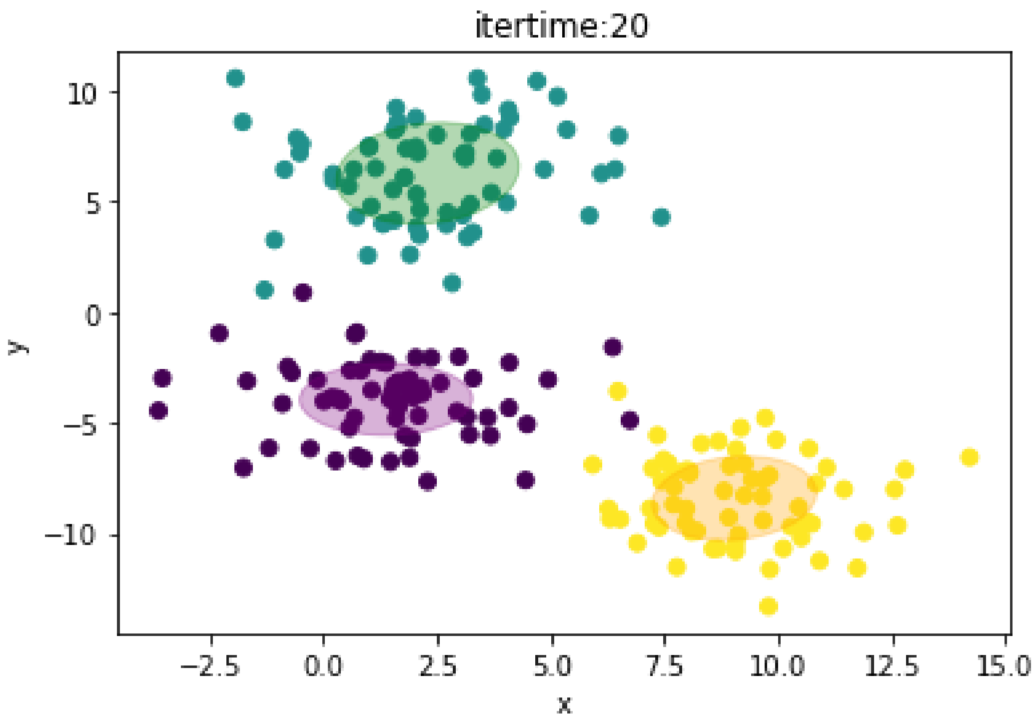
EM 过程

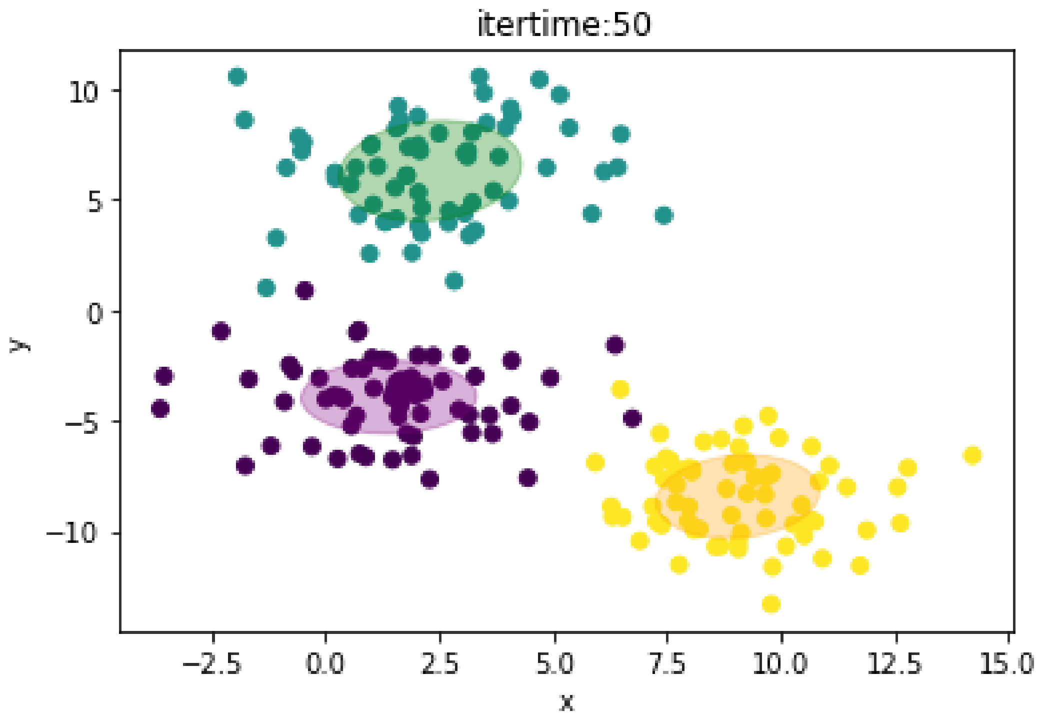





itertime:10
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:15
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:20
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:50
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

代码附录

EM+GMM

1.生成数据

In [1]:

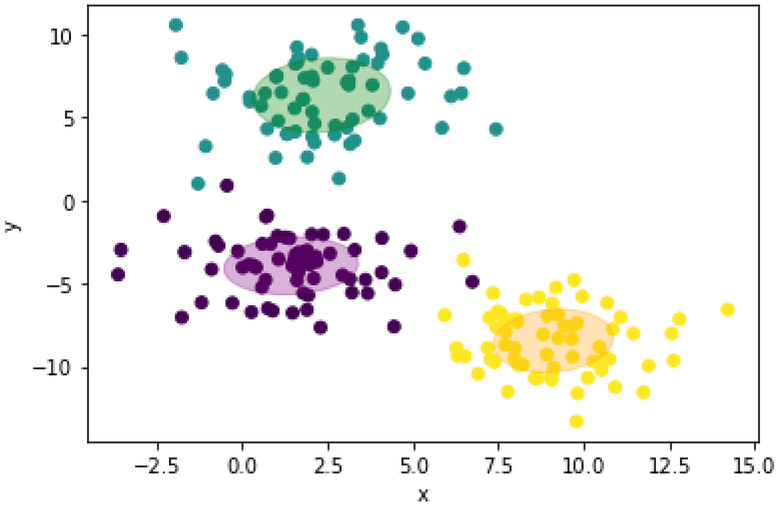
```
# 这个函数用来绘制正态置信椭圆
def make_ellipses(mean, cov, ax, confidence=1, alpha=0.3, color="red", eigv=True, arrow_color_list=None):
    """
    多元正态分布
    mean: 均值
    cov: 协方差矩阵
    ax: 画布的Axes对象
    confidence: 置信椭圆置信率 # 置信区间, 95%: 5.991  99%: 9.21  90%: 4.605
    alpha: 椭圆透明度
    eigv: 是否画特征向量
    arrow_color_list: 箭头颜色列表
    """
    import matplotlib as mpl
    lambda_, v = np.linalg.eig(cov)    # 计算特征值lambda_和特征向量v
    sqrt_lambda = np.sqrt(np.abs(lambda_))    # 存在负的特征值, 无法开方，取绝对值
    s = confidence
    width = 2 * np.sqrt(s) * sqrt_lambda[0]    # 计算椭圆的两倍长轴
    height = 2 * np.sqrt(s) * sqrt_lambda[1]   # 计算椭圆的两倍短轴
    angle = np.rad2deg(np.arccos(v[0, 0]))    # 计算椭圆的旋转角度
    ell = mpl.patches.Ellipse(xy=mean, width=width, height=height, angle=angle, color=color)    # 绘制椭圆
    ax.add_artist(ell)
    ell.set_alpha(alpha)
```

In [2]:

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot
import numpy as np
num_of_data=200#样本总数
dimention=2 #特征维度
classes=3   #样本类别
data, label_GT = make_blobs(n_samples=num_of_data, n_features=dimention, centers=classes,cluster_std=[2,2,2])
print(data.shape)
label_None=[0]*num_of_data
# pyplot.rcParams["figure.figsize"] = (8.0, 8.0)
fig, ax = pyplot.subplots()
ax.set_xlabel("x")
ax.set_ylabel("y")
# pyplot.scatter(data[:, 0], data[:, 1], c=label_None)
pyplot.scatter(data[:, 0], data[:, 1], c=label_GT)
data_0=data[np.where(label_GT==0)]
data_1=data[np.where(label_GT==1)]
data_2=data[np.where(label_GT==2)]
```

```
make_ellipses(np.mean(data_0,axis=0),np.cov(data_0.T),ax,color="purple")
make_ellipses(np.mean(data_1,axis=0),np.cov(data_1.T),ax,color="green")
make_ellipses(np.mean(data_2,axis=0),np.cov(data_2.T),ax,color="orange")
pyplot.show()
```

(200, 2)



2.需要估计的参数

三个类高斯分布的均值μ_1,μ_2,μ_3以及协方差$\sigma_1,\sigma_2,\sigma_3$

GMM中的隶属度p_1,p_2,p_3,其中需要满足$\sum_{i=1}^{3} p_i = 1$

3.初始化需要更新的参数

1.先验概率

In [43]:

```
import numpy as np
p=np.random.rand(classes)  #根据类别数随机生成p
p=p/p.sum()  #归一化，保证p之和为1
print(p.shape)
p
```

(3,)

Out[43]:

array([0.63236134, 0.04672559, 0.32091308])

2.高斯分布均值μ

μ_1,μ_2,μ_3

In [44]:

```
means=np.random.rand(classes,dimention)  # shape [3,2]
print(means.shape)
means
```

```
(3, 2)
```

Out[44]:

```
array([[0.75406162, 0.96695   ],
       [0.32071891, 0.67361435],
       [0.70194454, 0.66212949]])
```

3.高斯分布协方差σ

σ_1,σ_2,σ_3

In [45]:

```
covs=np.empty((classes,dimention,dimention))
for i in range(classes):
#     covs[i]=np.eye(dimention)*np.random.rand(1)*num_of_d
ata
    covs[i]=np.eye(dimention)*np.random.rand(1)*10
covs.shape
print(covs)
```

```
[[[5.4251931  0.        ]
  [0.         5.4251931 ]]

 [[3.45564757 0.        ]
  [0.         3.45564757]]

 [[8.23423343 0.        ]
  [0.         8.23423343]]]
```

一次参数更新的完整过程

In [46]:

```
density=np.empty((num_of_data,classes))   #[2000,3],用来保
存每个数据属于各个类别的后验概率 [样本数，类别数]
```

根据初始的均值和协方差，生成三个类别的高斯分布，并根据其概率密度函数计算条件概率 下面的norm就是二维高斯分布：$N(Y|\mu_j^{(t)}, \sigma_j^{(t)})$

In [47]:

```
from scipy import stats
norm1=stats.multivariate_normal(mean=means[0],cov=covs[0])
norm2=stats.multivariate_normal(mean=means[1],cov=covs[1])
norm3=stats.multivariate_normal(mean=means[2],cov=covs[2])
density[:,0]=norm1.pdf(data)
density[:,1]=norm2.pdf(data)
density[:,2]=norm3.pdf(data)
```

计算每个样本属于某个类别的后验概率 $p(x, w_i|\mu, \sigma) = p(x|w_i, \mu, \sigma) * p(w_i)$

$$T_{j,\ i} = \frac{p_j^{(t)} N(Y_i|\mu_j^{(t)}, \sigma_j^{(t)})}{\sum_{j=1}^{c} p_j^{(t)} N(Y_i|\mu_j^{(t)}, \sigma_j^{(t)})}$$

In [48]:

```
posterior= density * p #shape [2000,3]
posterior= posterior/posterior.sum(axis=1,keepdims=True)
# 归一化,使得各样本属于每个类别的概率之和为1
```

更新先验概率p

$$p_j^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n} T_{j,i}^{(t)}$$

In [49]:

```
p_hat=posterior.sum(axis=0)
p_hat=p_hat/num_of_data
p_hat
```

Out[49]:

```
array([0.33942861, 0.01187301, 0.64869838])
```

更新μ

$$u_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)} Y_i}{\sum_{i=1}^{n} T_{j,i}^{(t)}}$$

In [50]:

```
# print(posterior.shape)
# mean_hat=np.tensordot(posterior, data, axes=[0, 0])
# print(mean_hat)
mean_hat=np.matmul(data.T,posterior).T
# mean_hat=mean_hat/np.sum(posterior,axis=0)
mean_hat=np.divide(mean_hat,np.sum(posterior,axis=0,keepdi
ms=True).T)
mean_hat
```

Out[50]:

```
array([[ 1.92169983,  1.00954441],
       [ 1.04605817, -0.47266826],
       [ 5.54835235, -3.59526269]])
```

更新协方差σ

$$\sigma_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)} (Y_i - \mu_j^{t+1})(Y_i - \mu_j^{t+1})^T}{\sum_{i=1}^{n} T_{j,i}^{(t)}}$$

In [51]:

```
cov_hat=np.empty(covs.shape)
posterior_sum=np.sum(posterior,axis=0)
for i in range(classes):
    tmp=data-mean_hat[i]
    cov_hat[i]=np.dot(tmp.T*posterior[:,i],tmp)/posterior_
sum[i]
cov_hat
```

Out[51]:

```
array([[[ 4.4549339 ,  0.30095145],
        [ 0.30095145, 25.66500579]],

       [[ 2.32480724,  0.61603416],
        [ 0.61603416, 12.15803516]],
```

```
       [[ 16.49010491, -15.82663548],
        [-15.82663548,  43.79389101]]])
```

In [52]:

```
#更新参数
covs=cov_hat
means=mean_hat
p=p_hat
```

将参数更新过程封装成函数

In [53]:

```
def EM(data,p,means,covs):
    """
    data:样本，尺寸（num_of_data,dimention)
    p:各类别的先验概率,尺寸（classes,）
    mean: 各类别高斯分布的初始均值，尺寸(classes,dimentio
n），dimention是样本的特征维度
    cov：各类别高斯分布的初始协方差，尺寸（classes,dimenti
on,dimention)
    """
    from scipy import stats
    num_of_data,dimention=data.shape
    classes=p.shape[0]
    density=np.empty((num_of_data,classes))
    norm1=stats.multivariate_normal(means[0],covs[0])
    norm2=stats.multivariate_normal(means[1],covs[1])
    norm3=stats.multivariate_normal(means[2],covs[2])
    density[:,0]=norm1.pdf(data)
    density[:,1]=norm2.pdf(data)
    density[:,2]=norm3.pdf(data)
    posterior= density * p #shape [2000,3]
    posterior= posterior/posterior.sum(axis=1,keepdims=Tr
ue)  # 归一化,使得各样本属于每个类别的概率之和为1
    p_hat=posterior.sum(axis=0)
    p_hat=p_hat/num_of_data
    mean_hat=np.matmul(data.T,posterior).T
    mean_hat=np.divide(mean_hat,np.sum(posterior,axis=0,ke
epdims=True).T)
    cov_hat=np.empty(covs.shape)
    posterior_sum=np.sum(posterior,axis=0)
    for i in range(classes):
        tmp=data-mean_hat[i]
        cov_hat[i]=np.dot(tmp.T*posterior[:,i],tmp)/poster
ior_sum[i]

    #更新参数
    covs=cov_hat
    means=mean_hat
    p=p_hat
    return p,means,covs
```

In [54]:

```
def draw(data,label_GT,means,covs,iter_time):
    fig, ax = pyplot.subplots()
    ax.set_xlabel("x")
```

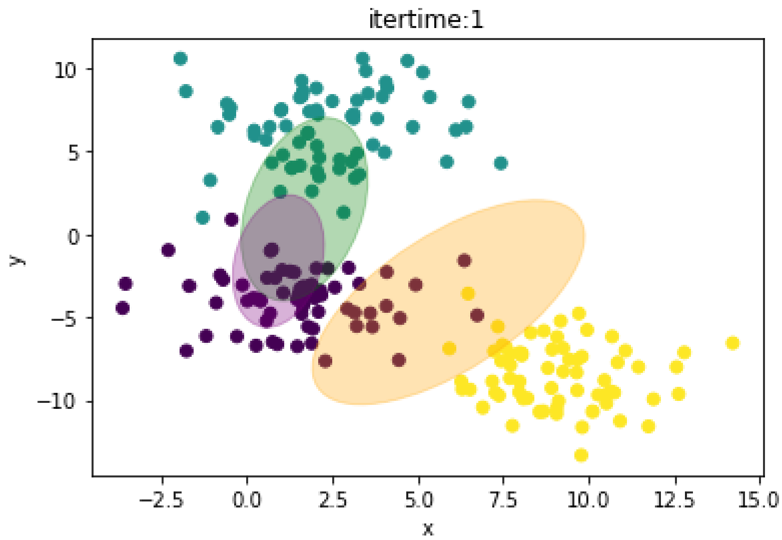
```
    ax.set_ylabel("y")
    # pyplot.scatter(data[:, 0], data[:, 1], c=label_None)
    pyplot.scatter(data[:, 0], data[:, 1], c=label_GT)
    make_ellipses(means[0],covs[0],ax,color="green")
    make_ellipses(means[1],covs[1],ax,color="purple")
    make_ellipses(means[2],covs[2],ax,color="orange")
    pyplot.title("itertime:{}".format(iter_time))
    pyplot.show()
```

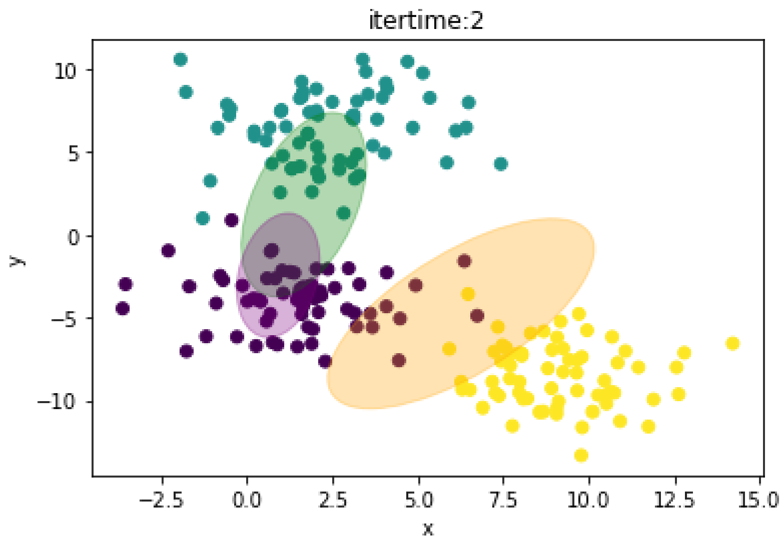
收敛过程

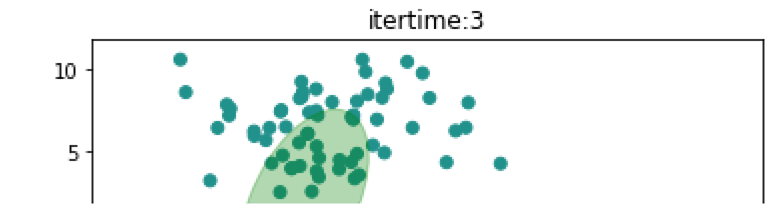
注意！如果这里圈的颜色和样本点的颜色不一致，是正常的，因为这是一个无监督过程，无法预先知道样本的类别号，只知道哪些样本应该归为一类。所以我无法给他们分配固定的颜色。

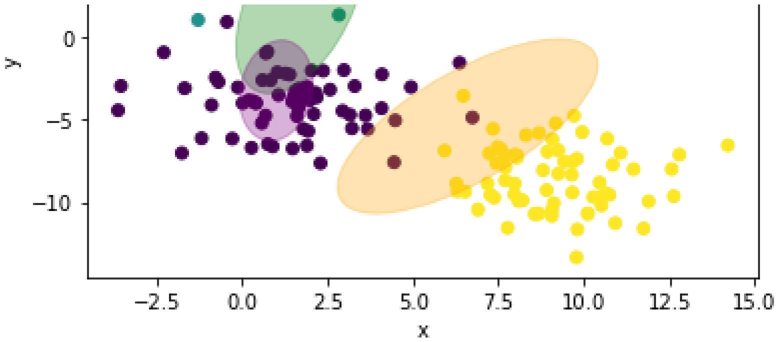
In [55]:

```
for i in range(50):
    p,means,covs=EM(data,p,means,covs)
    draw(data,label_GT,means,covs,i+1)
```

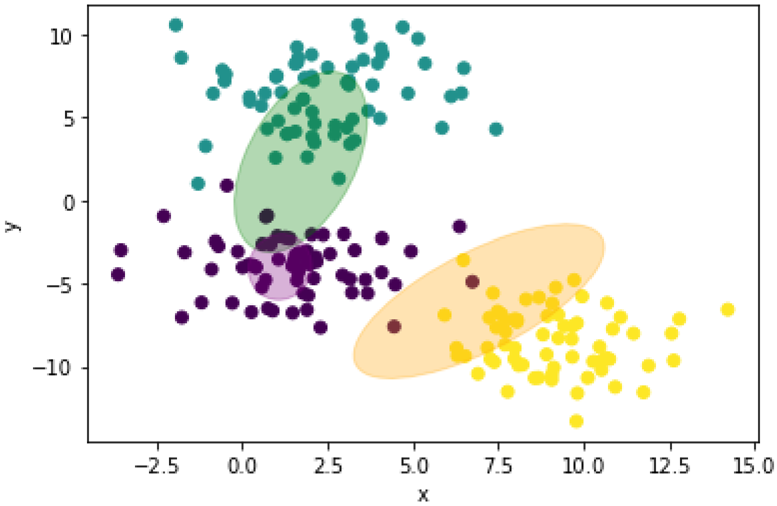




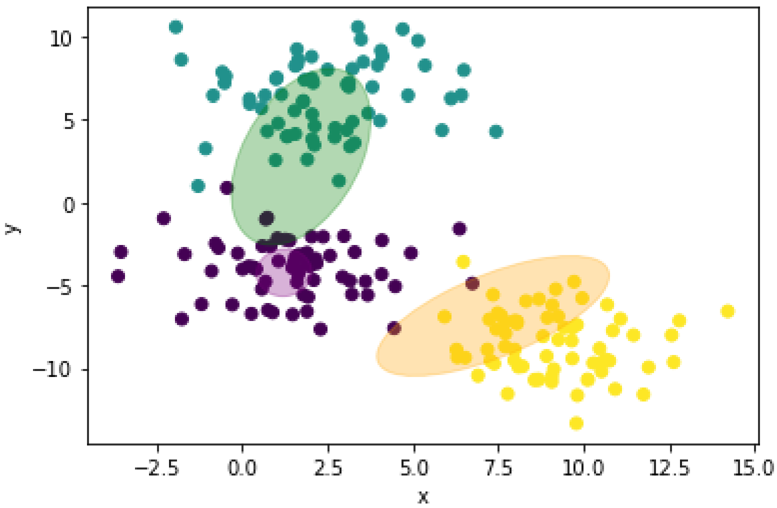




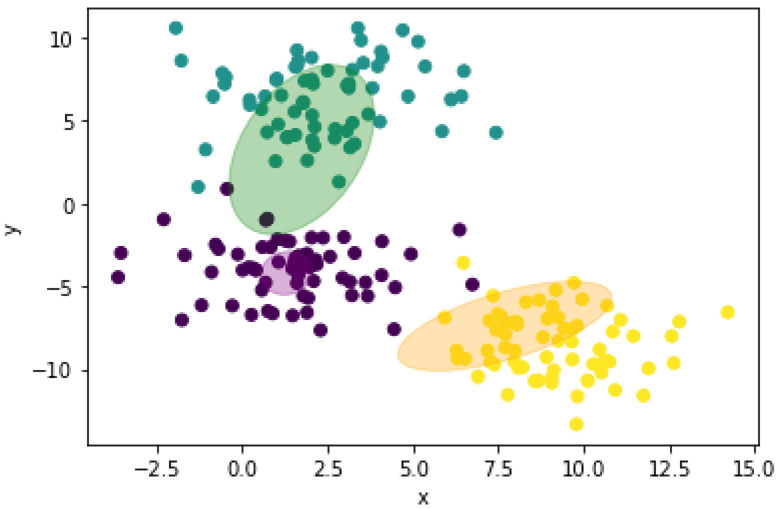
itertime:4



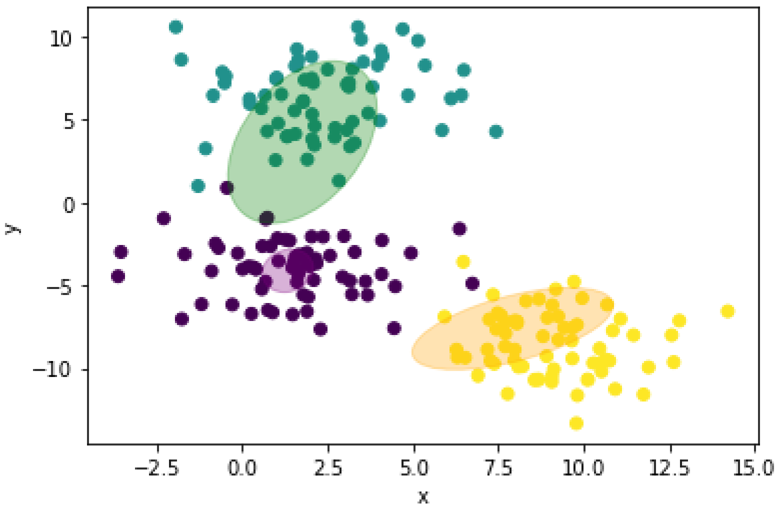
itertime:5



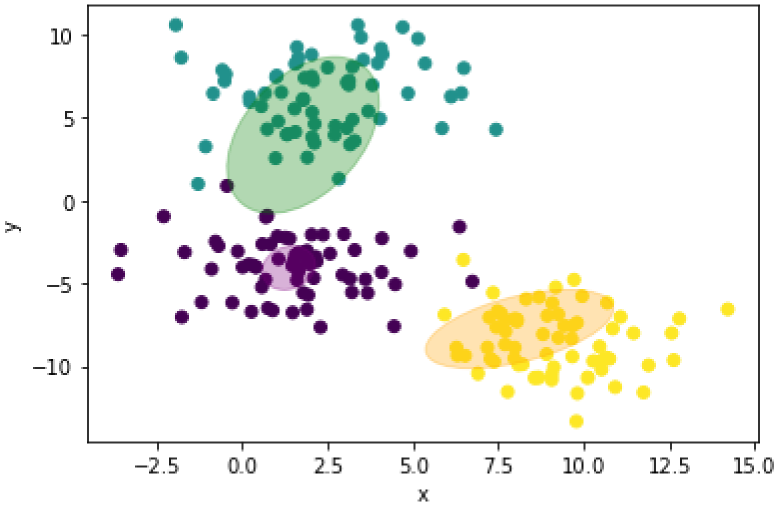
itertime:6



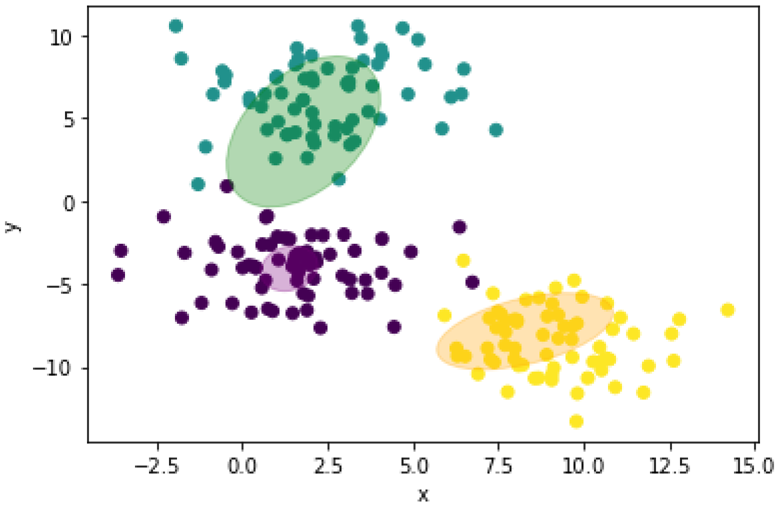
itertime:7



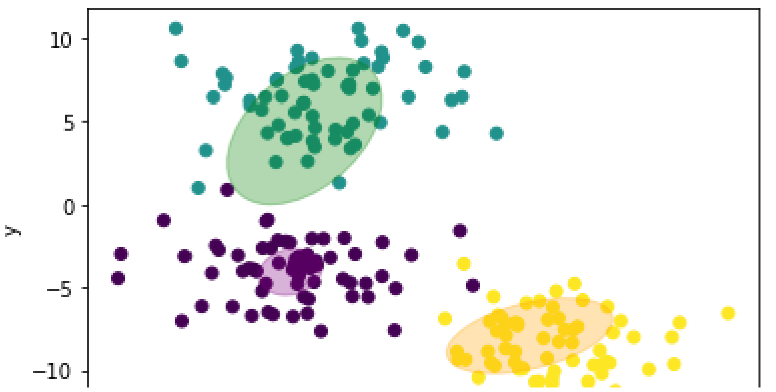
itertime:8

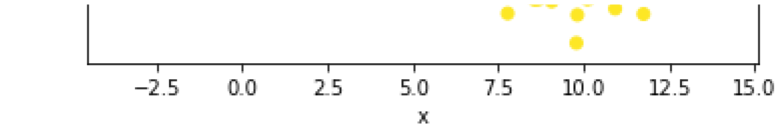


itertime:9

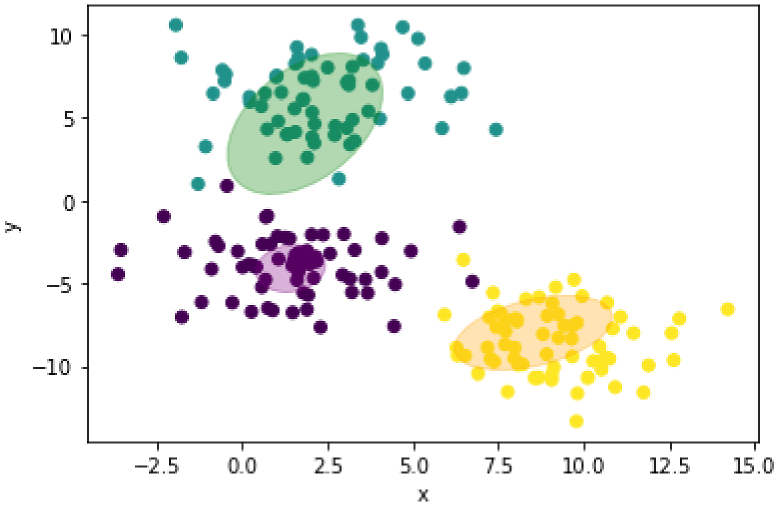


itertime:10

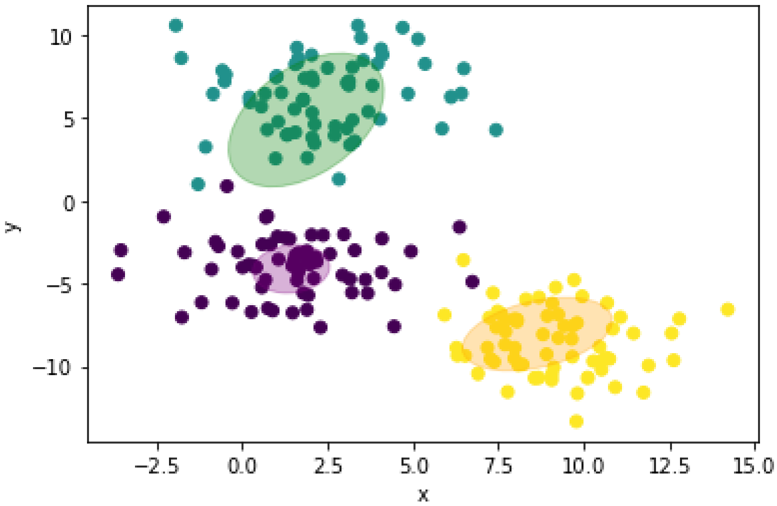




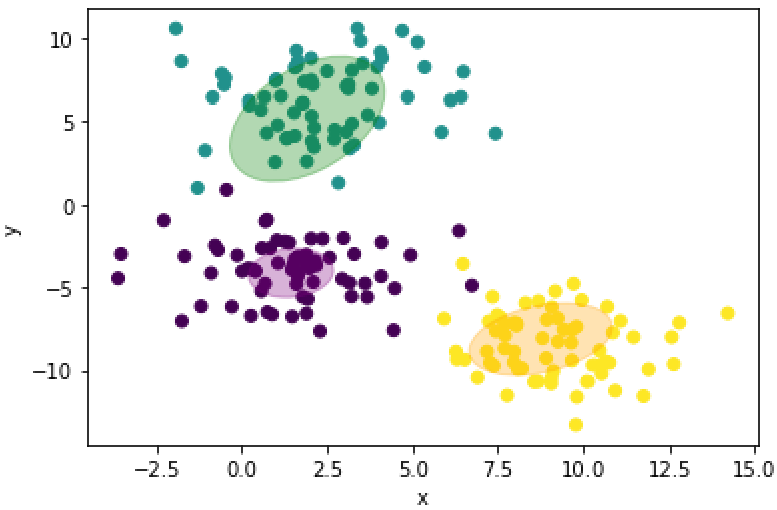
itertime:11



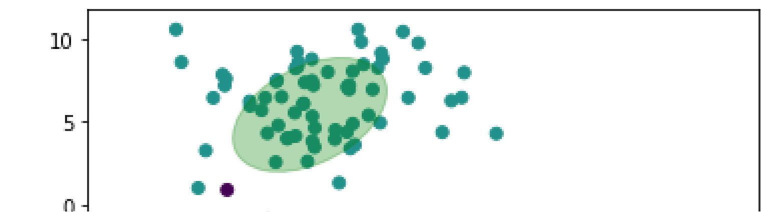
itertime:12

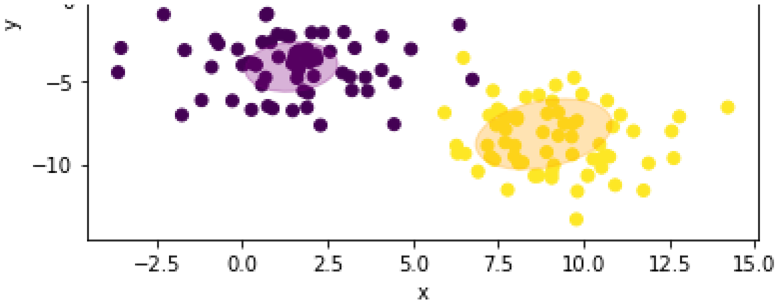


itertime:13

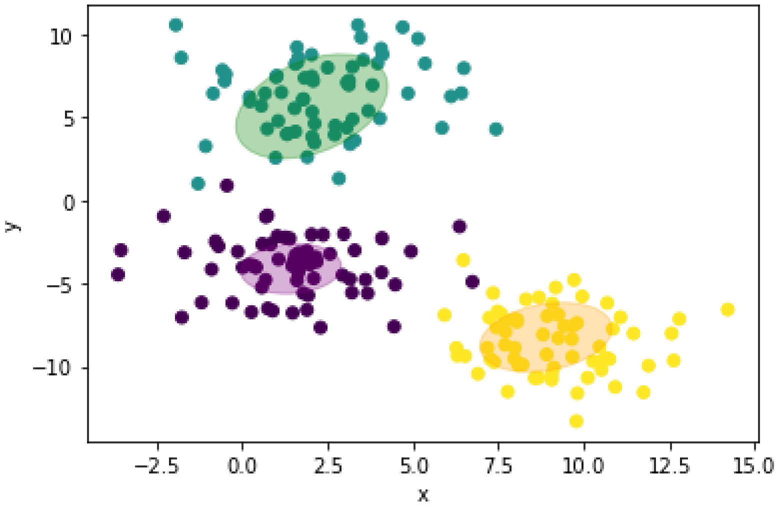


itertime:14

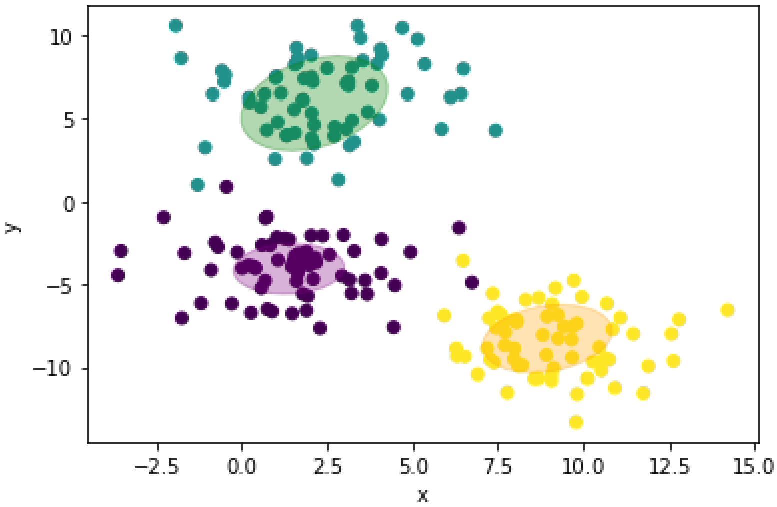




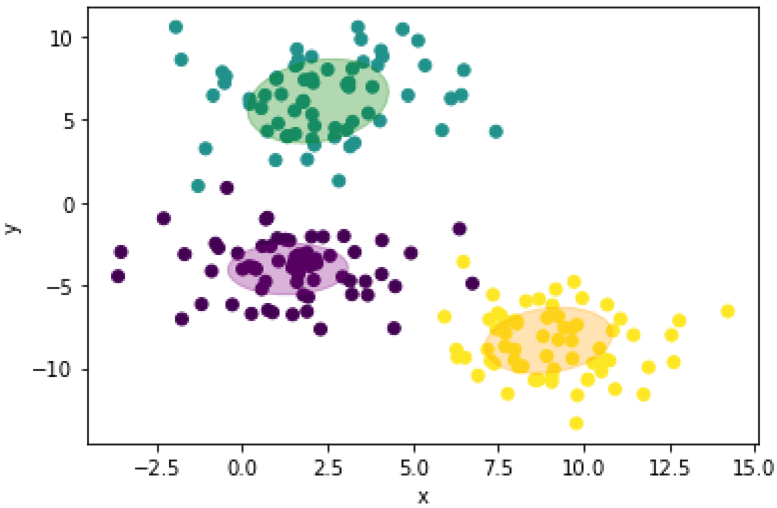
itertime:15



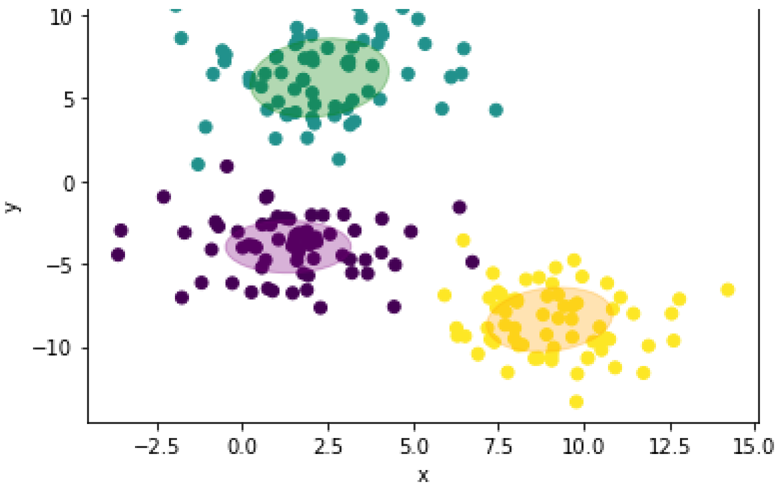
itertime:16



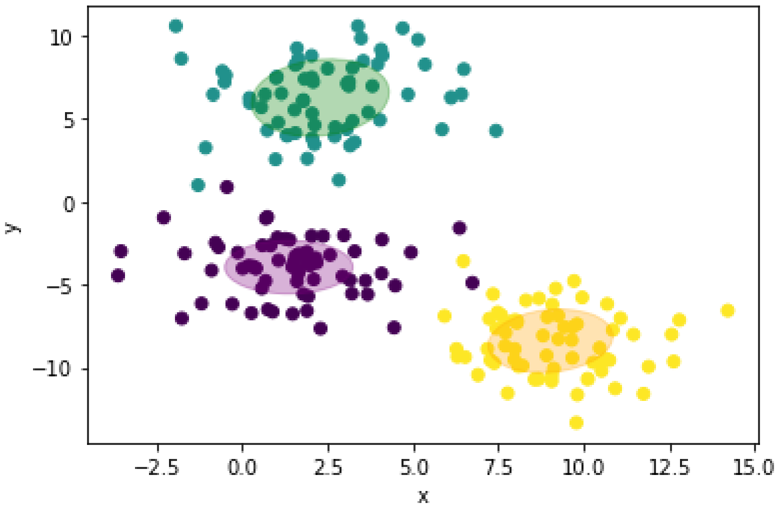
itertime:17



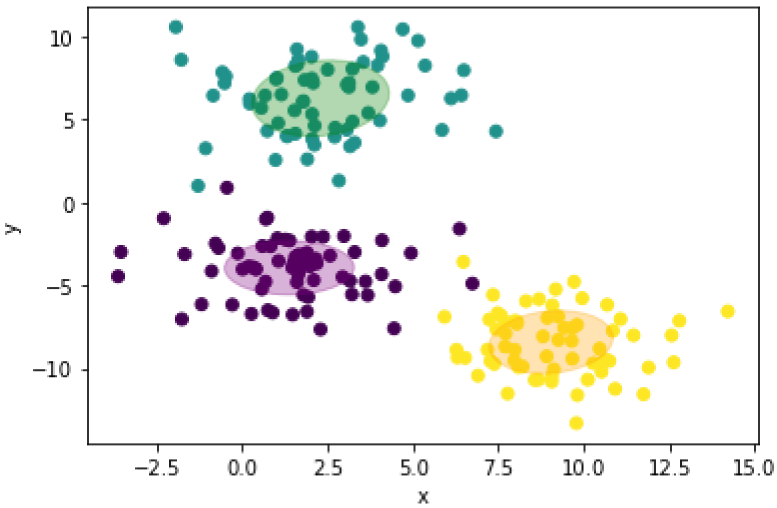
itertime:18



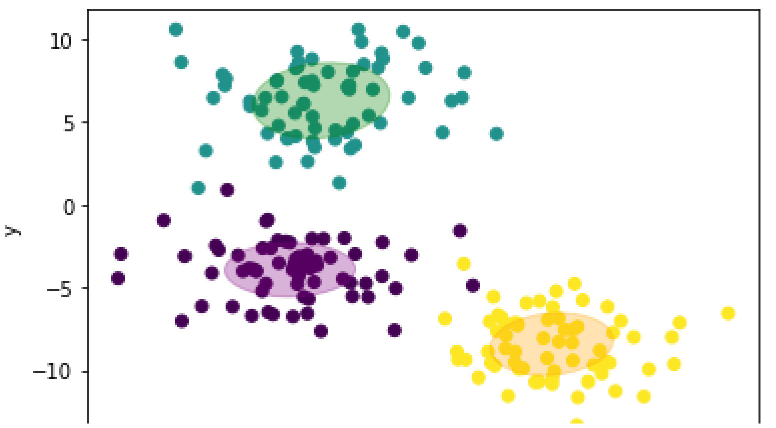
itertime:19

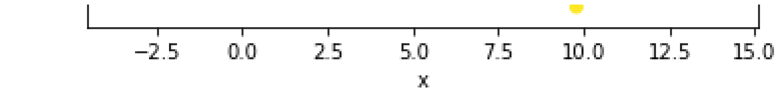


itertime:20

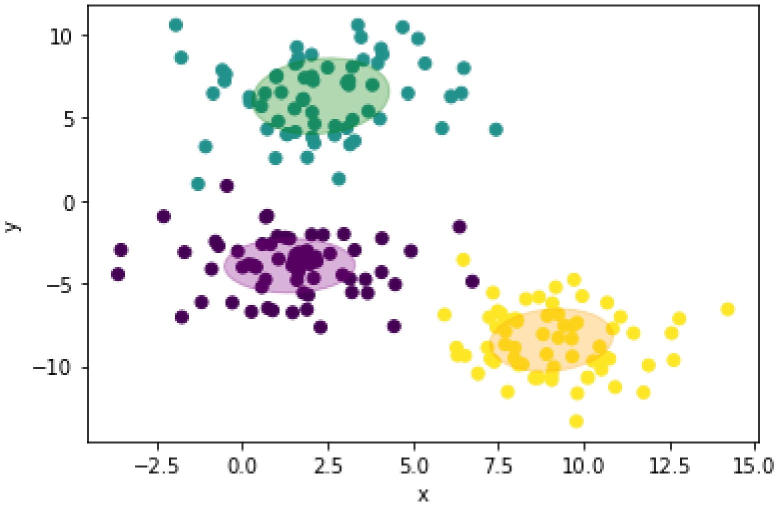


itertime:21

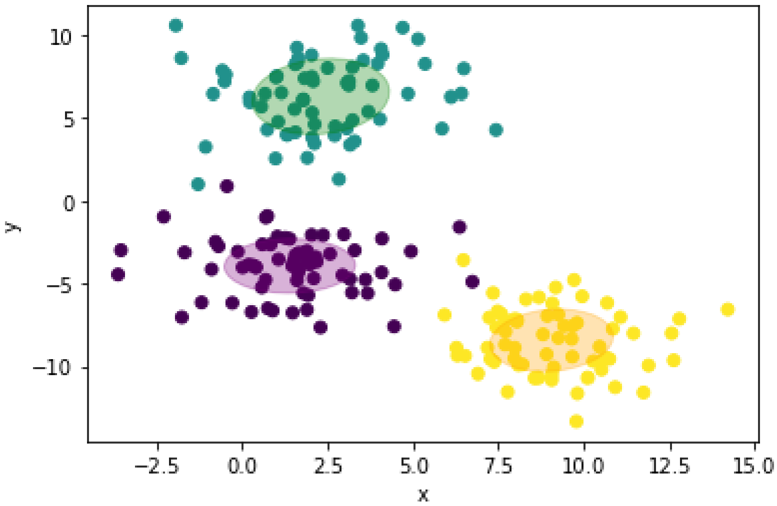




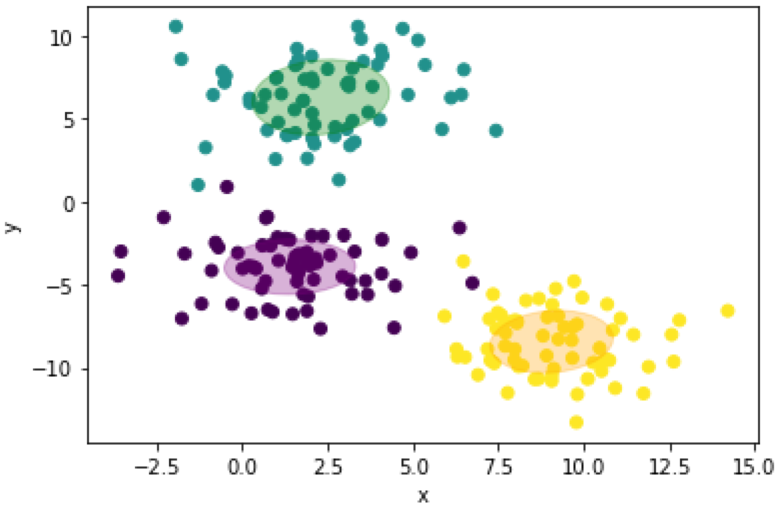
itertime:22



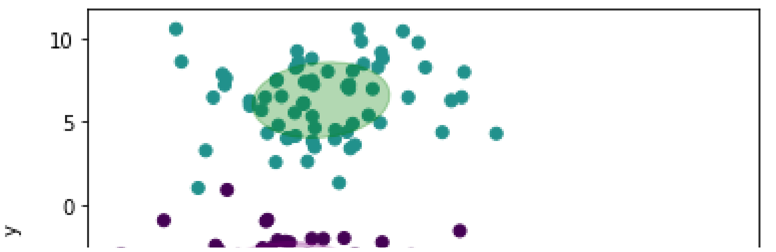
itertime:23

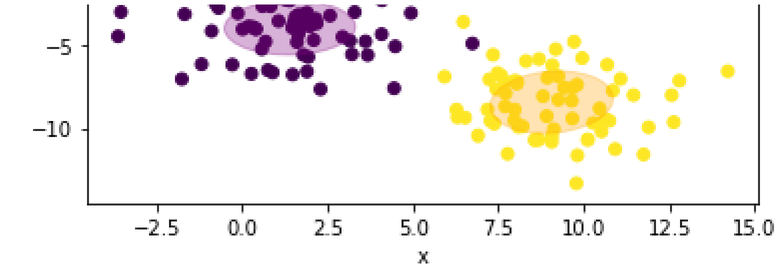


itertime:24

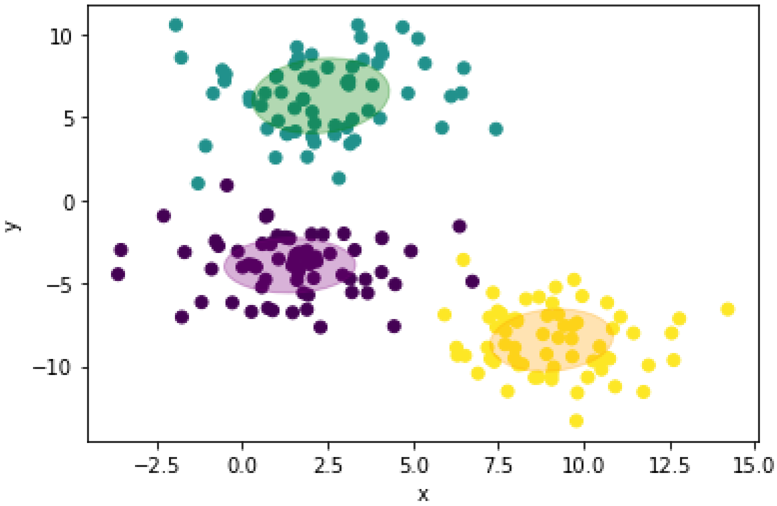


itertime:25

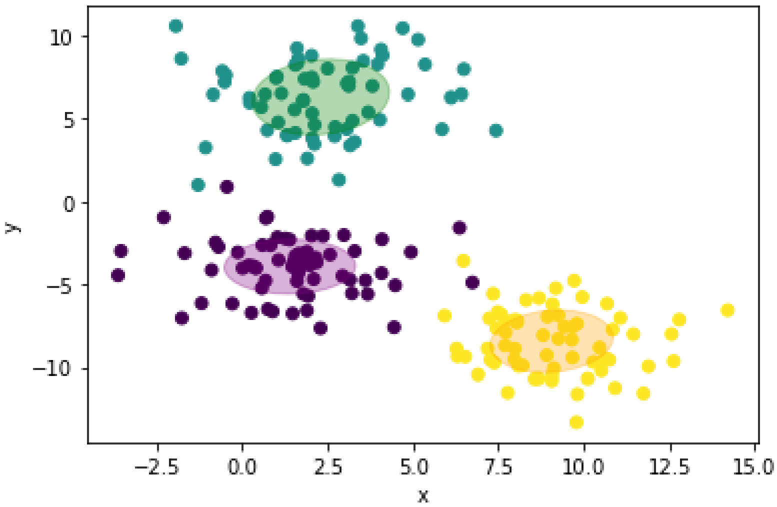




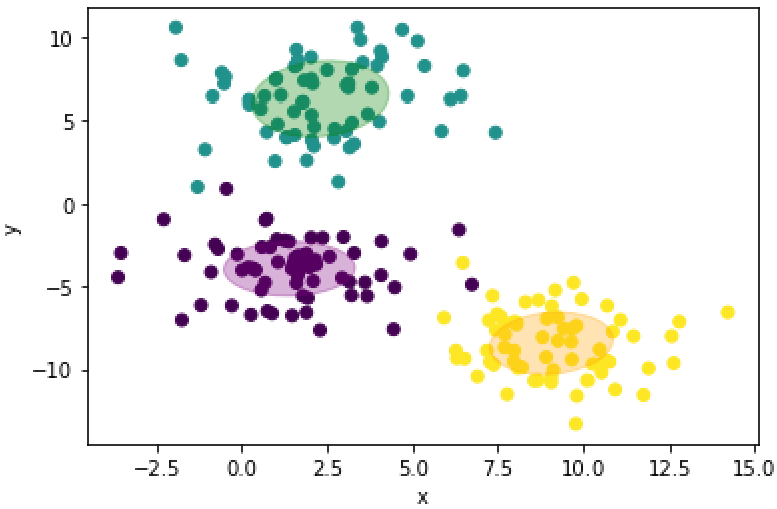
itertime:26



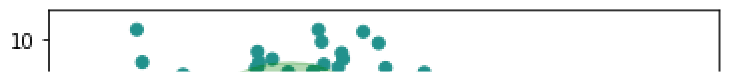
itertime:27

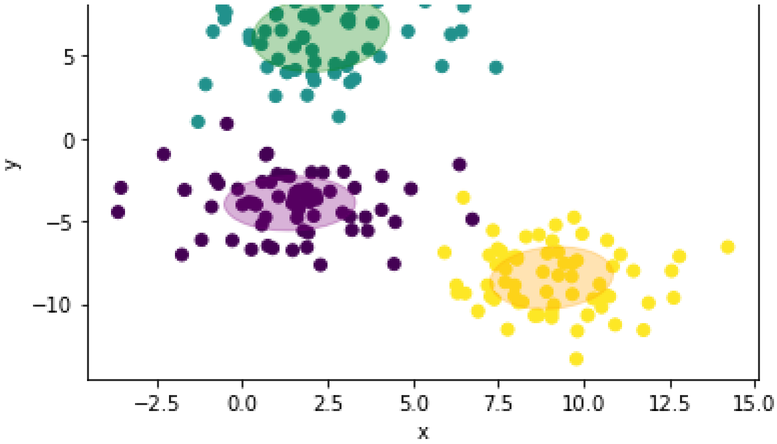


itertime:28

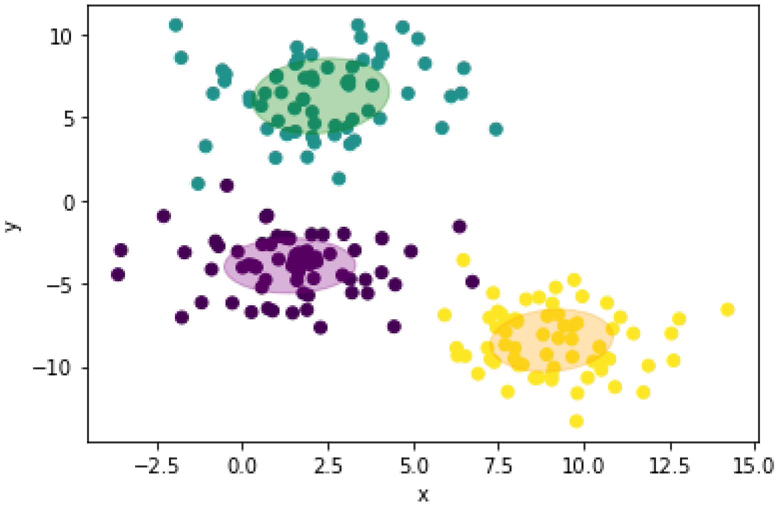


itertime:29

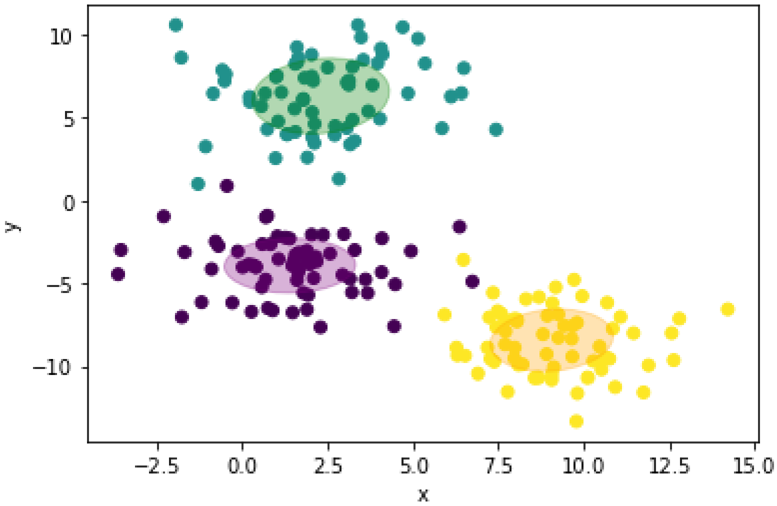




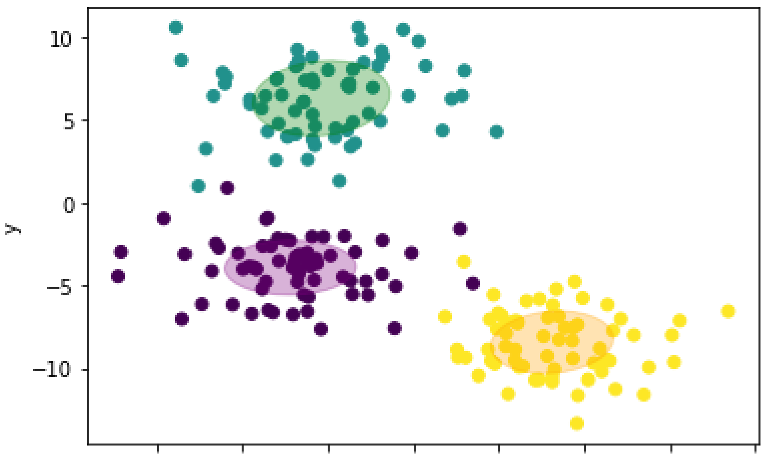
itertime:30



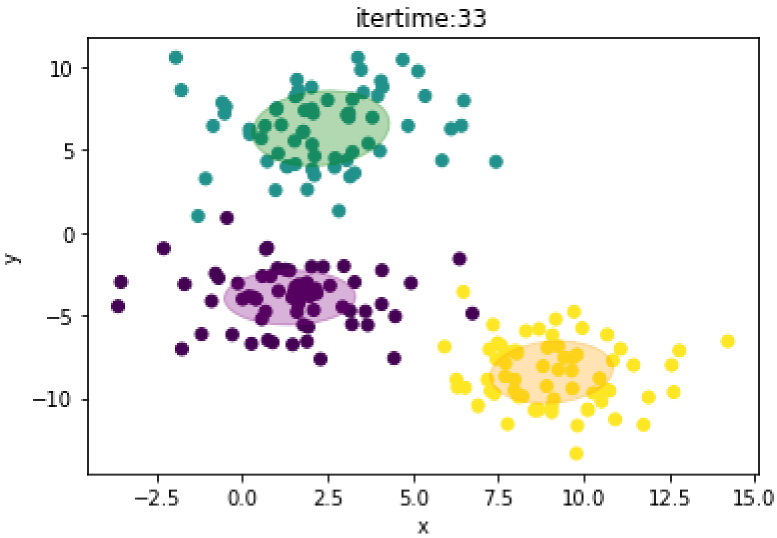
itertime:31

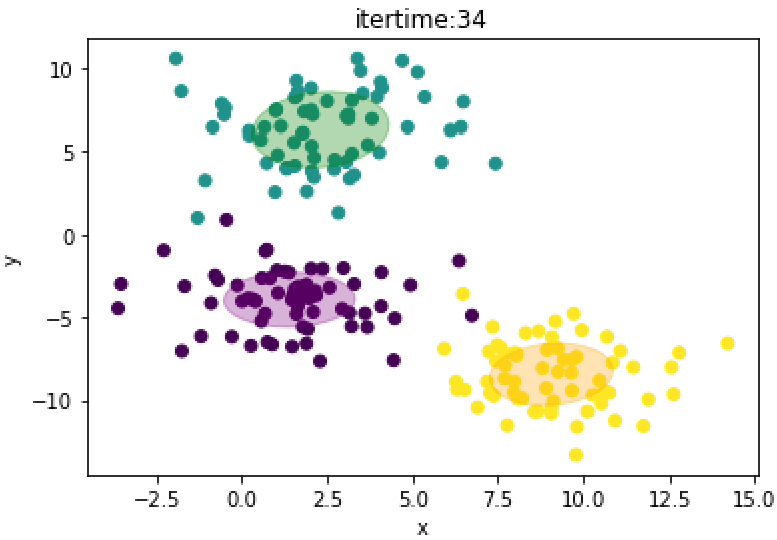


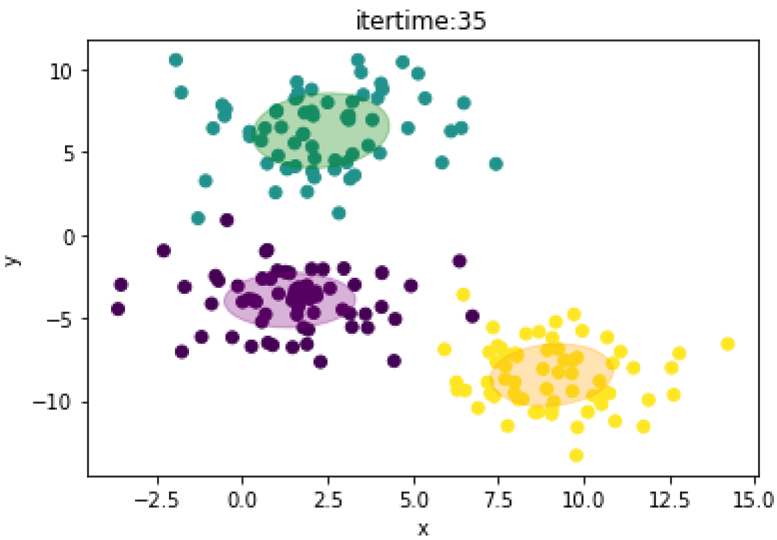
itertime:32

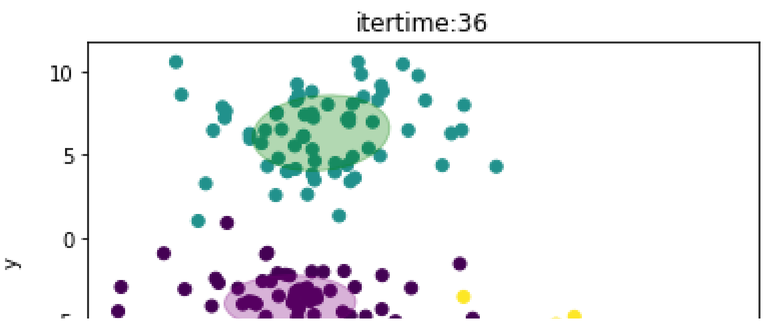


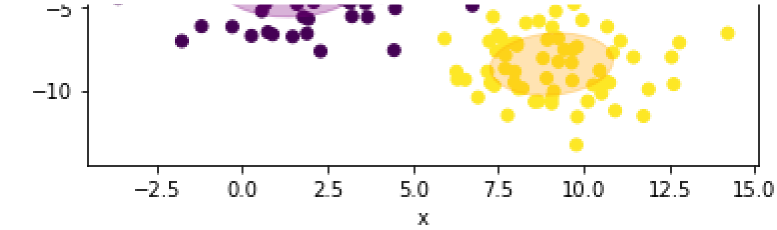

−2.5 0.0 2.5 5.0 7.5 10.0 12.5 15.0
x

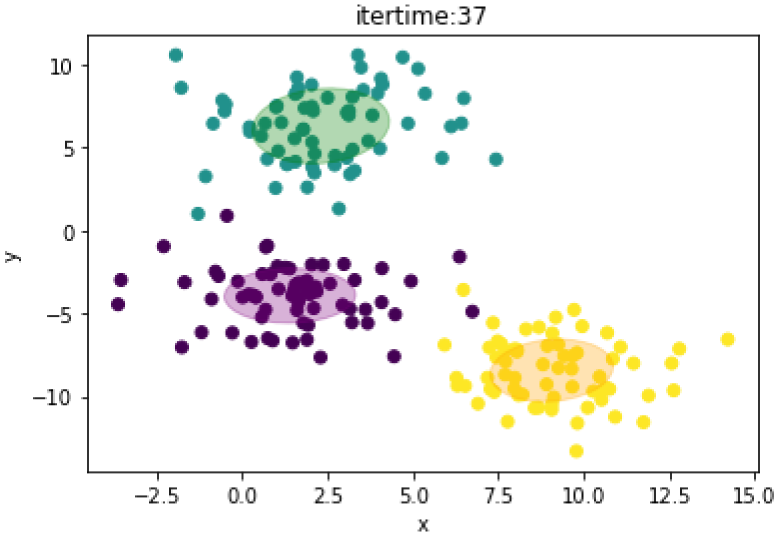

itertime:33
y
x

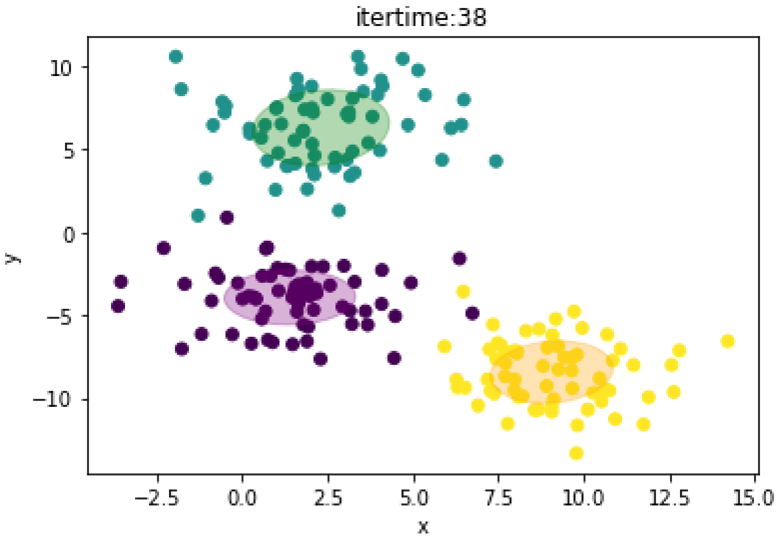

itertime:34
y
x

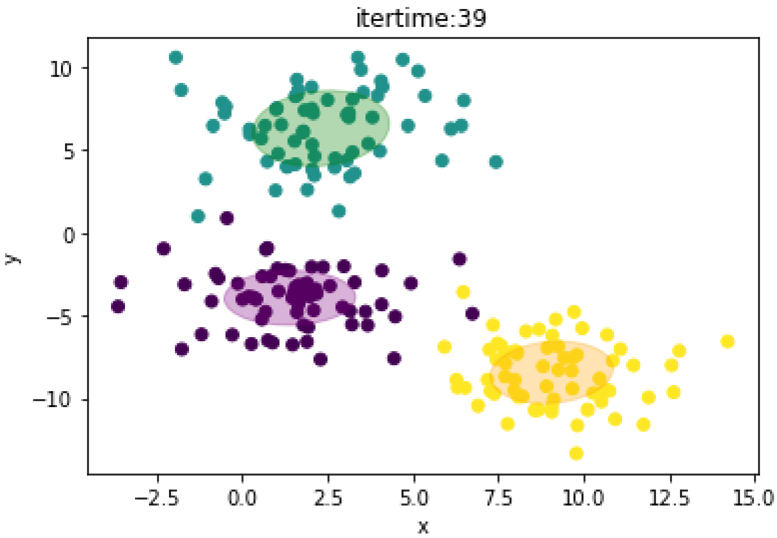

itertime:35
y
x

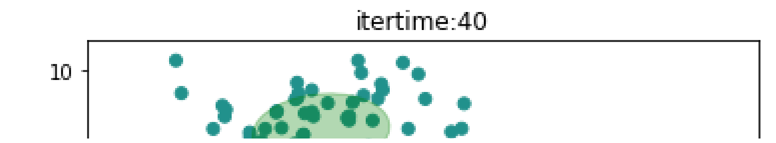

itertime:36
y

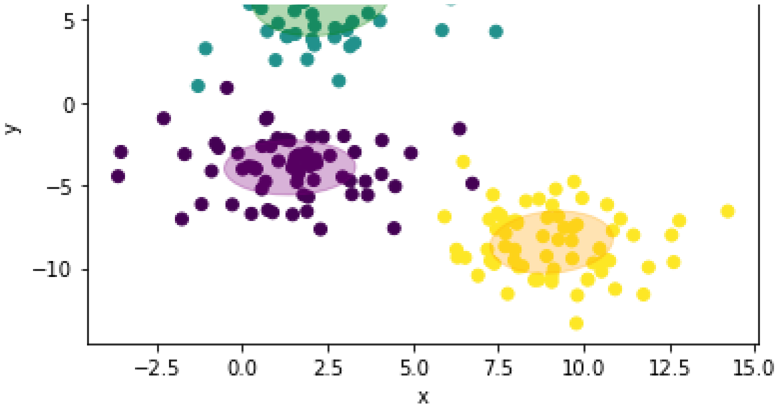

−5
−10
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

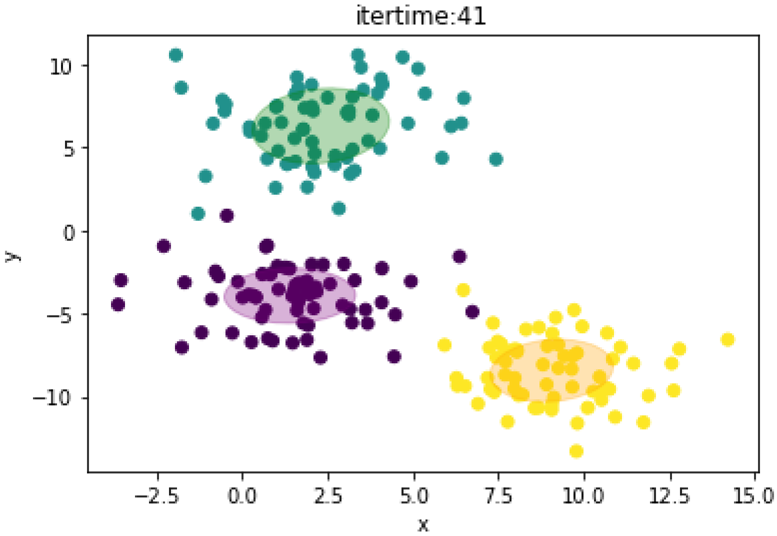

itertime:37
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

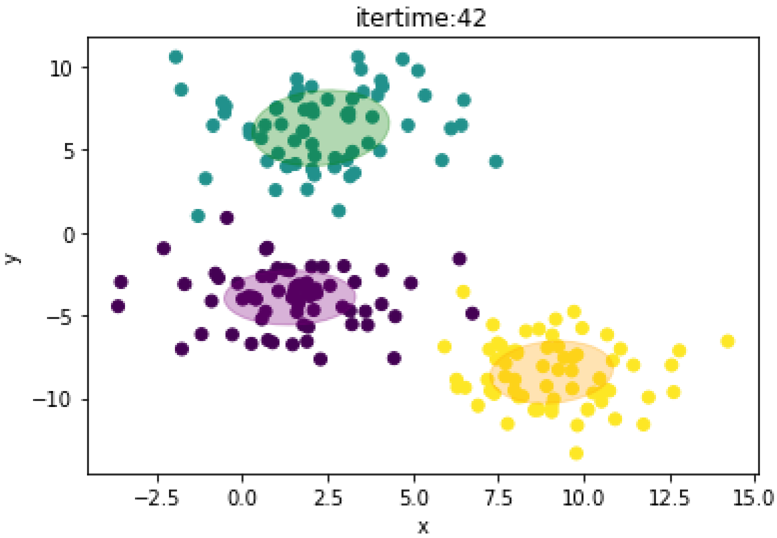

itertime:38
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

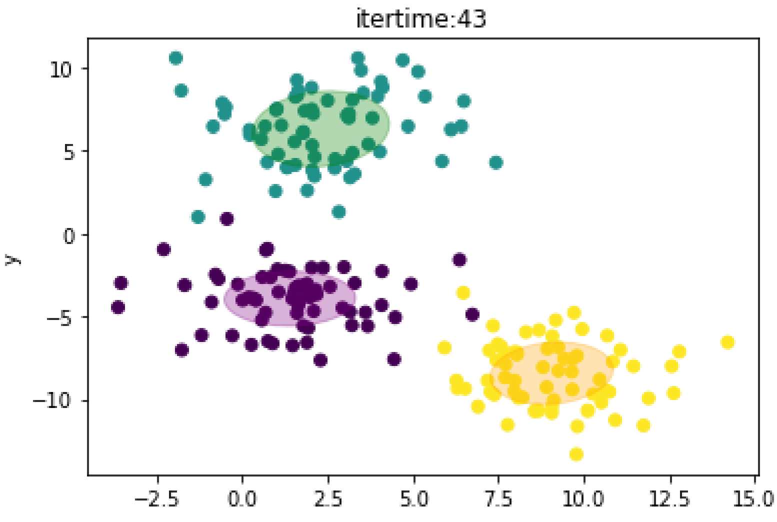

itertime:39
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x

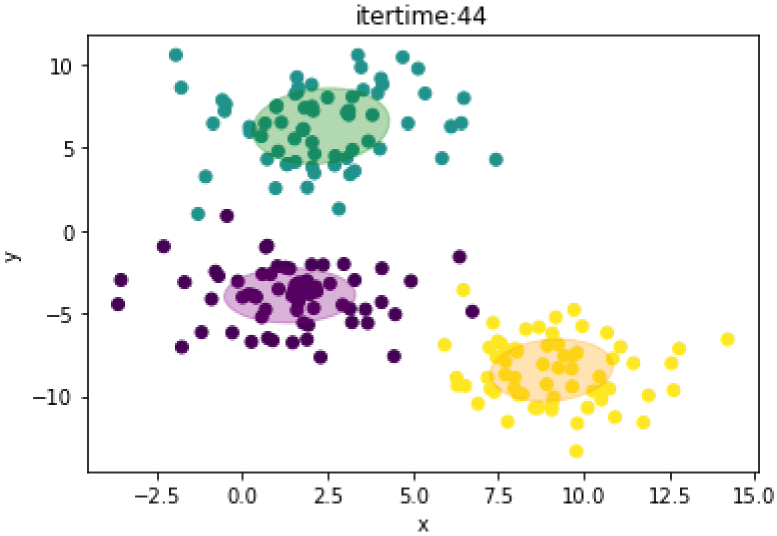

itertime:40
10

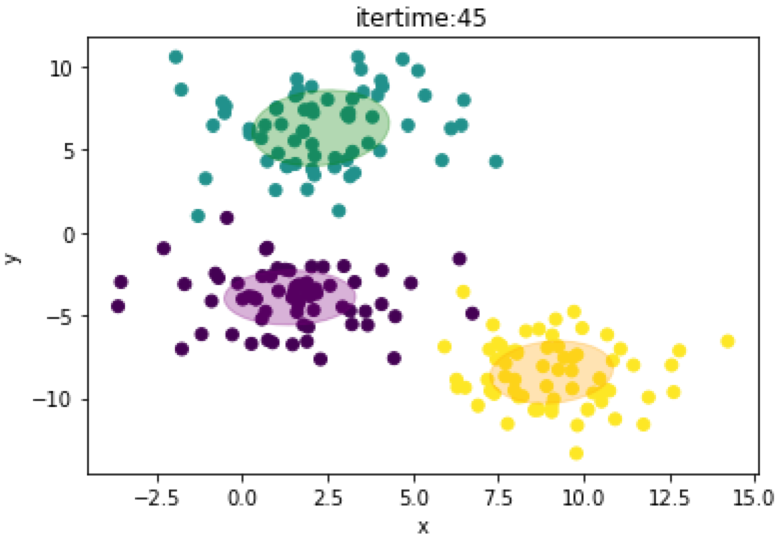

y
x
5
0
−5
−10
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0

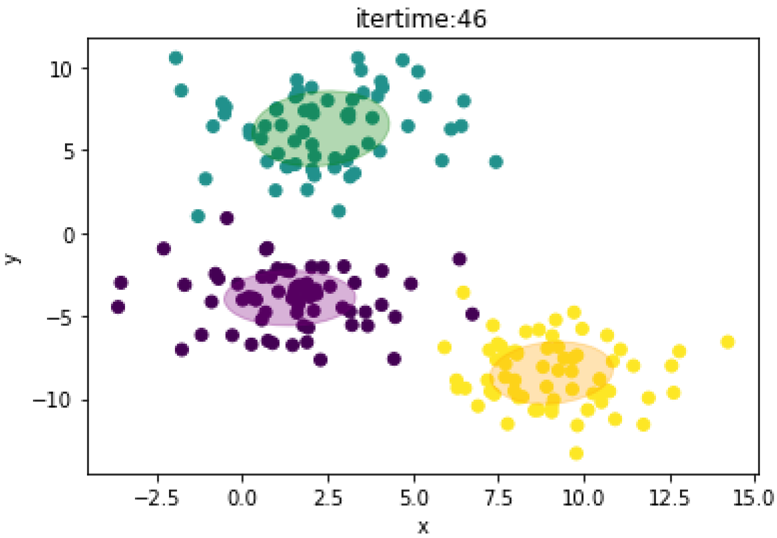

itertime:41
y
x

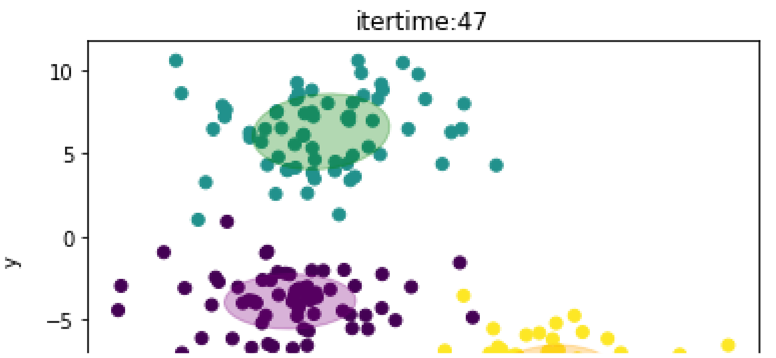

itertime:42
y
x


itertime:43
y


itertime:44
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:45
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:46
10
5
0
−5
−10
y
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x


itertime:47
10
5
0
−5
y