Кластерный анализ. Метод k-средних

Особенности метода k-means

- Он быстрый. Но это не означает, что он небрежный.
- Он требует меньше памяти по сравнению с методами иерархической кластеризации.

Ограничение 1. Число кластеров k определяется заранее.

Это непросто. Далее мы обсудим процедуру для определения числа кластеров.

Алгоритм k-means

Выбирается k точек — центры начальных кластеров.

Далее в цикле применяем следующие правила.

Правило 1

Каждый объект приписывается к тому кластеру, чей центр ближайший.

Правило 2

После того, как объекты приписаны к кластерам, вычисляем новые центры кластеров. Центр кластера — центр тяжести объектов кластера.

$$\bar{x}_j = \frac{1}{N_j} \sum_i^{N_j} x_i, \quad \bar{y}_j = \frac{1}{N_j} \sum_i^{N_j} y_i.$$

Ограничение 2. Используется только евклидово расстояние.

Недостаток исправляется в других вариантах метода k-средних. Например, в методе k-медоидов.

Алгоритм кластеризации k-medoids похож на алгоритм k-средних, но в отличие от него на каждой итерации ищет центры кластеров не как среднее точек, а как медоиды точек. То есть, центр кластера должен обязательно являться одной из его точек.

Реализован в пакете **flexclust**.

Шаг 1. Создадим набор данных, который предстоит кластеризовать

```
# Зададим зерно для датчика случайных чисел
set.seed(1234)
# Количество точек в каждом кластере
n.obj <- 900
# Стандартные отклонения (корень из дисперсии)
sd.1 <- 0.06

# Генерируем кластеры
#   точки из 1-го кластера
x1 <- rnorm(n.obj, mean = 0.2,  sd = sd.1)
y1 <- rnorm(n.obj, mean = 0.38, sd = sd.1)
#   точки из 2-го кластера
x2 <- rnorm(n.obj, mean = 0.49, sd = sd.1)
y2 <- rnorm(n.obj, mean = 0.25, sd = sd.1)
...
# Объединяем данные в матрицу
x.0 <- c(x1, x2, x3, x4, x5); y.0 <- c(y1, y2, y3, y4, y5)
data.0 <- cbind(x.0, y.0)
```

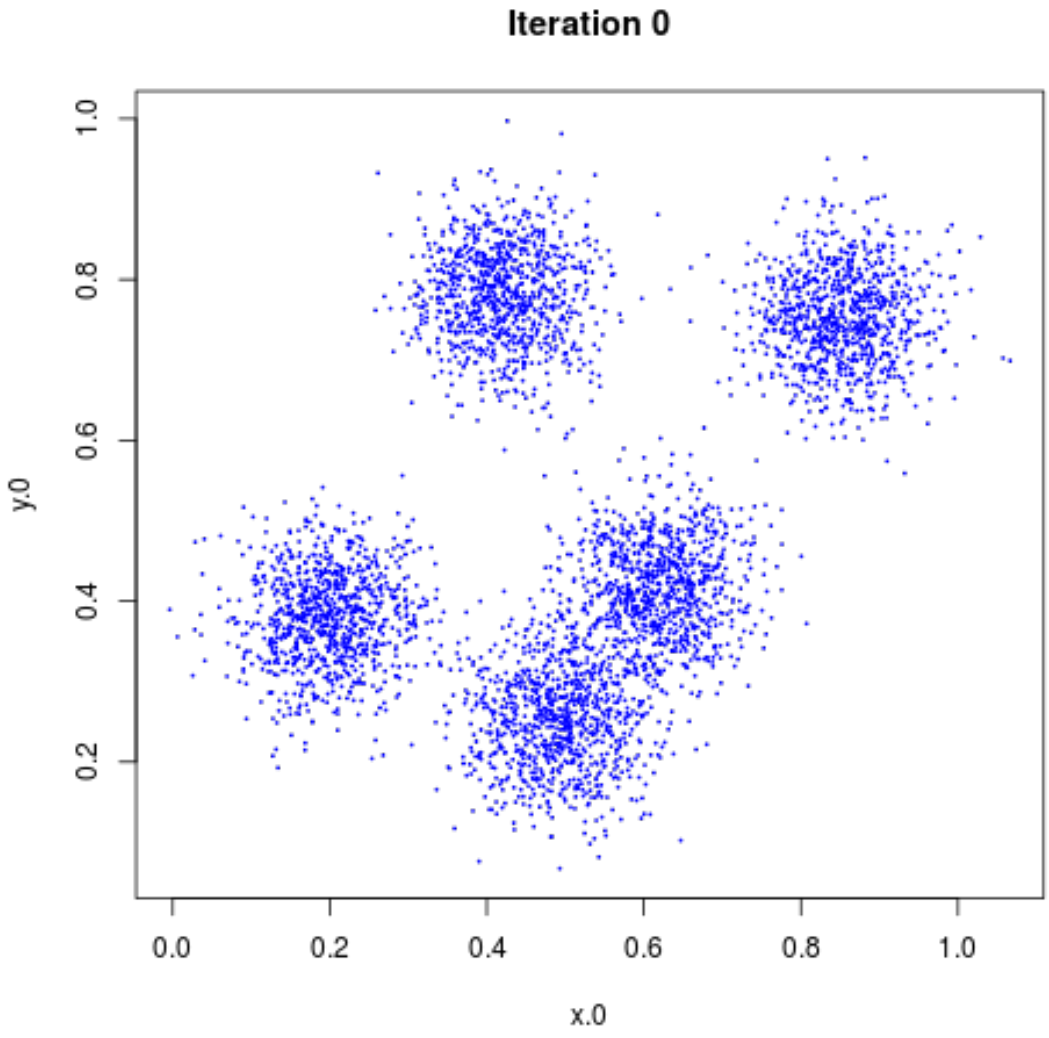


Figure 1:

Шаг 2. Задаем начальные центры кластеров

```
# На практике они - случайные.
# Сейчас берем такие, чтобы процесс кластеризации
# выглядел выразительно.

# Абсциссы точек
x.start <- c(0.50, 0.41, 0.43, 0.62, 0.38)
# Ординаты точек
y.start <- c(0.20, 0.22, 0.32, 0.36, 0.71)
# Объединяем данные в матрицу и помещаем ее в список
# для удобства работы с результатами кластеризации
clus.00 <- list()
clus.00$centers <- cbind(x.start, y.start)
```

Шаг 3. Проводим кластеризацию по шагам

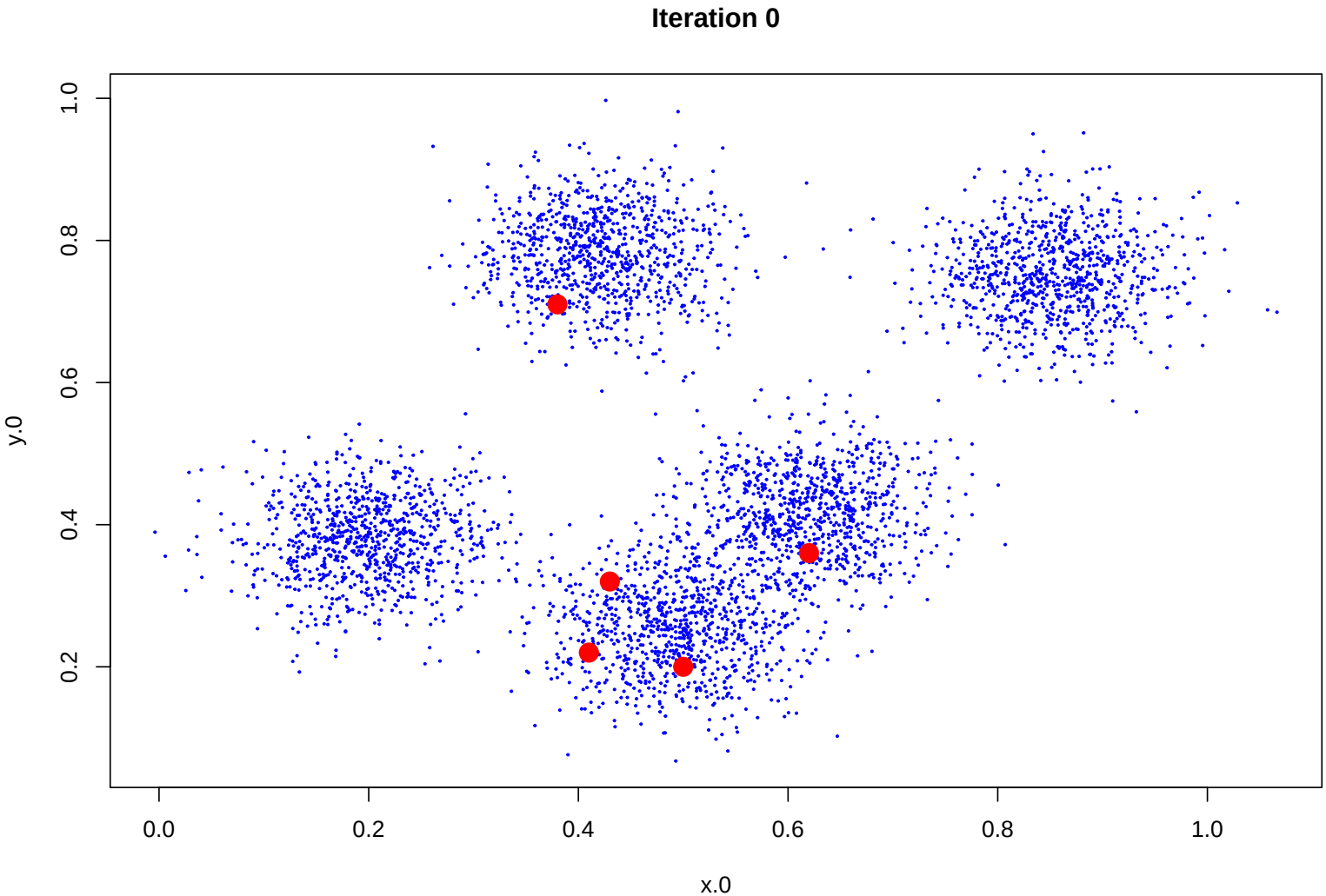
```
#  Вместо выполнения 15 итераций разом,
#  15 раз выполняем по одной итерации.
#  Чтобы посмотреть, как работает процедура

clus.01 <- kmeans(data.0, centers=clus.00$centers,
                  iter.max=1, algorithm = "Lloyd")
clus.02 <- kmeans(data.0, centers=clus.01$centers,
                  iter.max=1, algorithm = "Lloyd")
...
clus.15 <- kmeans(data.0, centers=clus.14$centers,
                  iter.max=1, algorithm = "Lloyd")
```

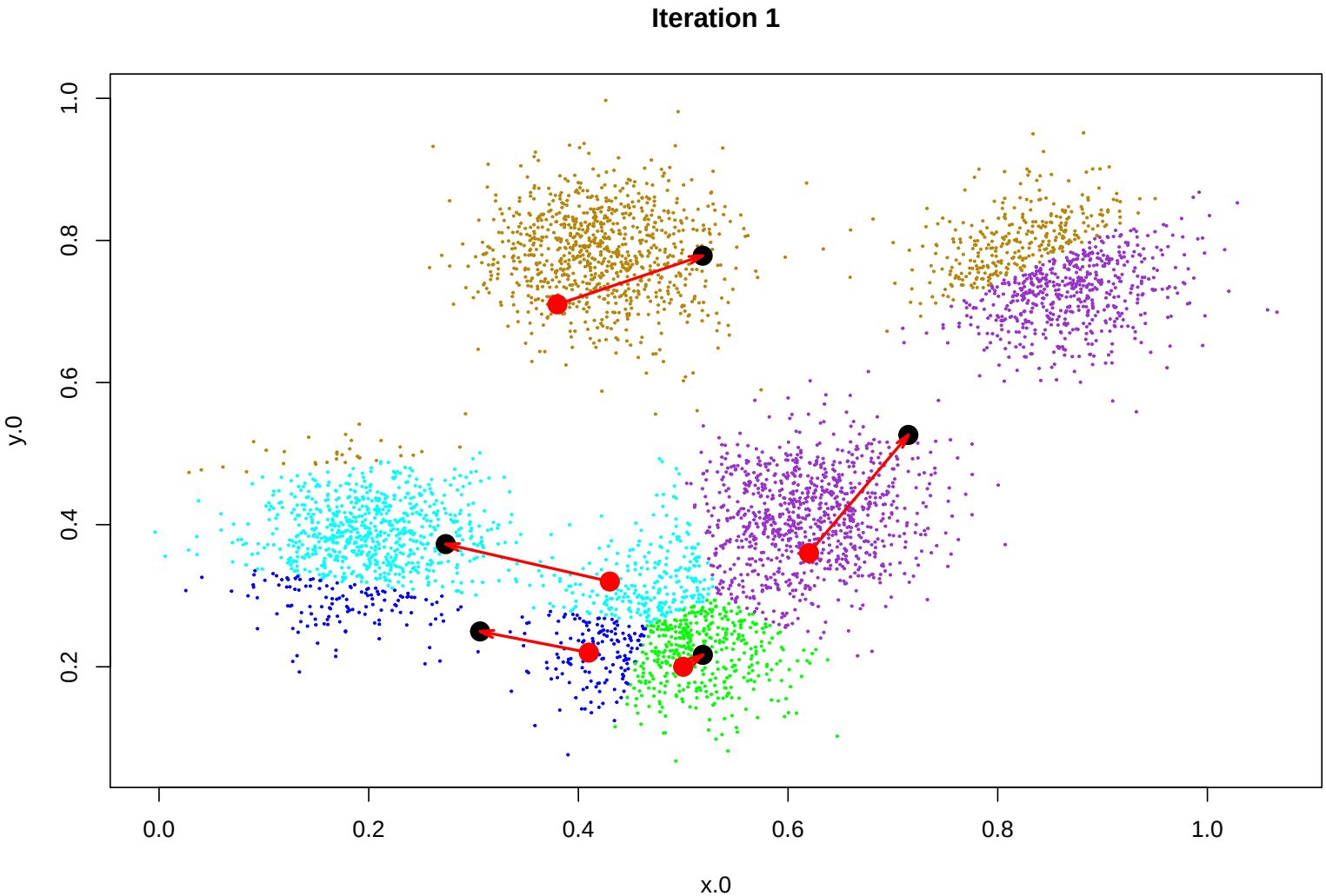
Будет появляться предупреждение:

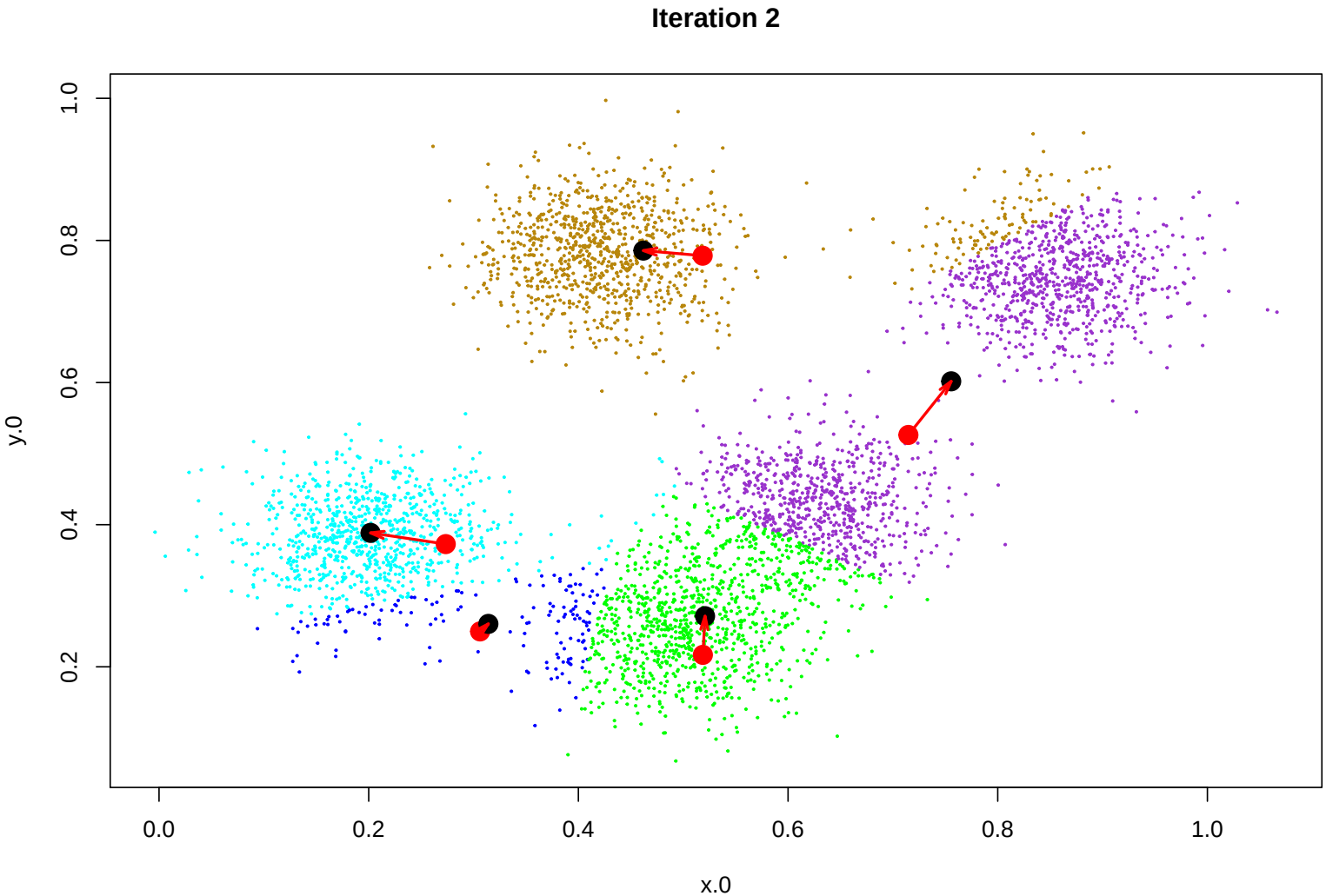
```
## Warning: did not converge in 1 iteration
```

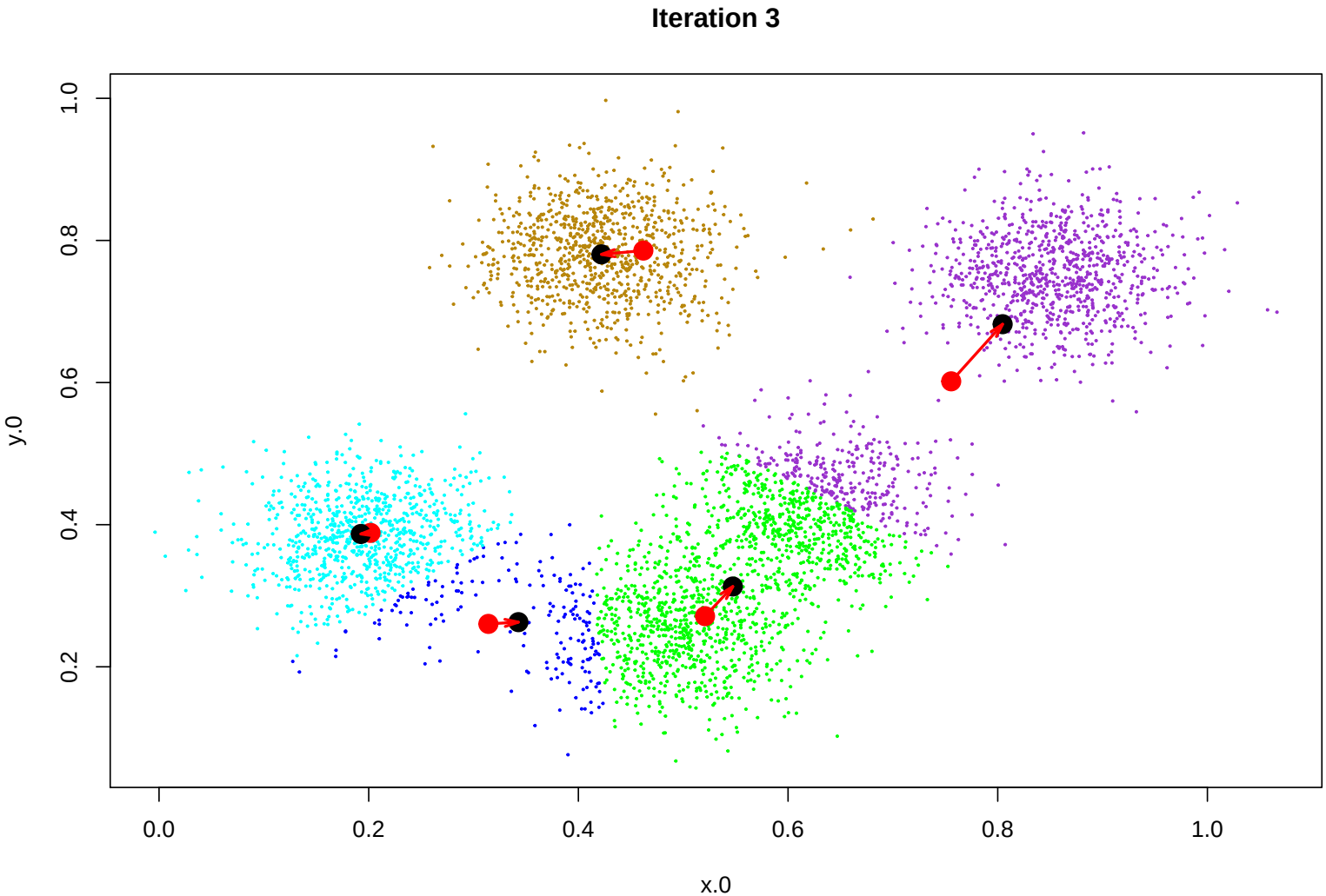
Шаг 4. Строим графики процесса кластеризации

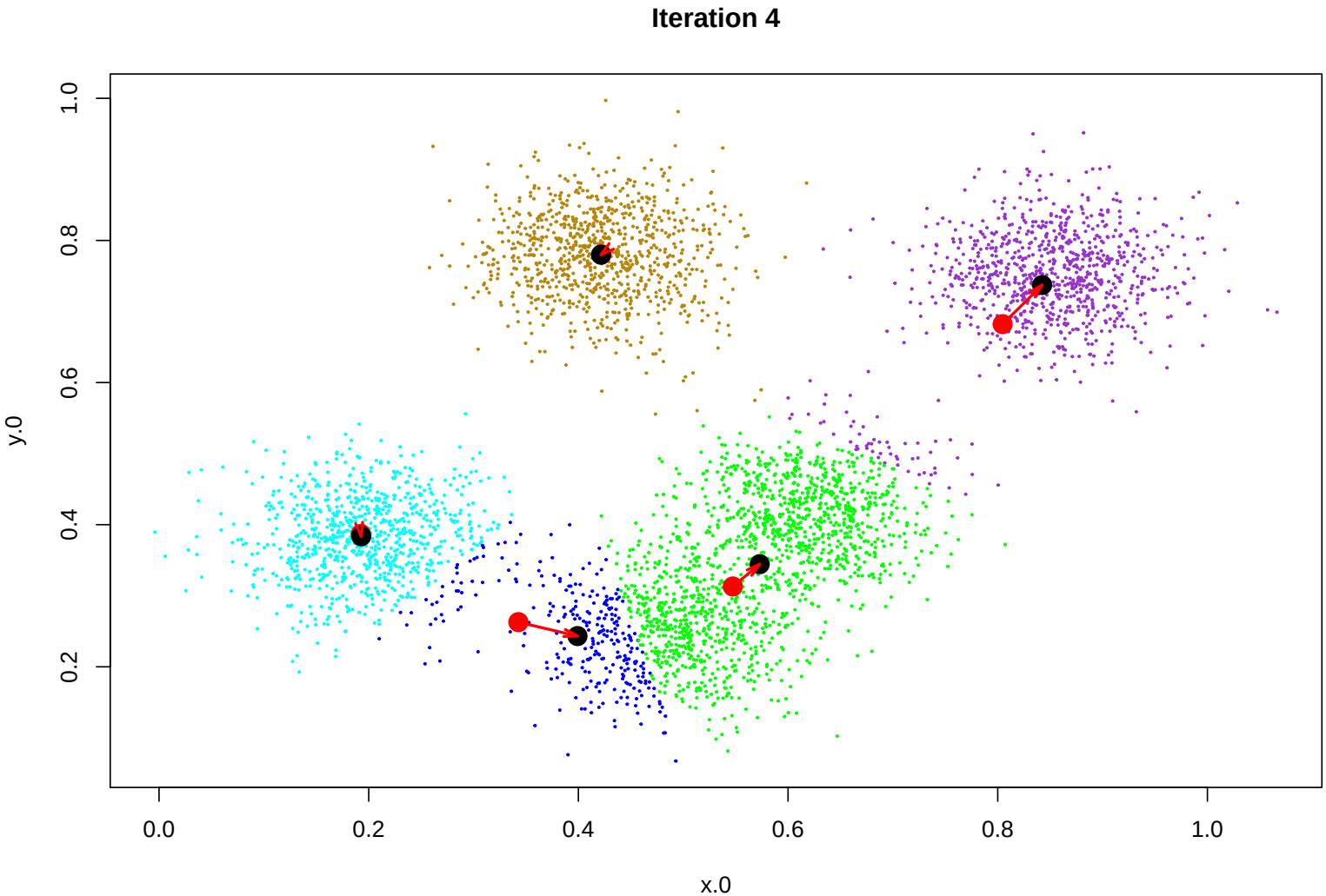
```
# Задаем размер точек на графиках
cex.1 <- 0.2
# Задаем цвета кластеров
col.1 <- c("green", "blue", "cyan", "darkorchid",
           "darkgoldenrod")
# -------------------- Итерация 1 --------------------------------
# Исходные данные
plot(data.0, col="blue", pch=19, main="Iteration 0", cex=cex.1)
# Центры кластеров в начале итерации
points(clus.00$centers, col="red", pch=19, cex=9*cex.1)
# Распределяем объекты по кластерам
plot(data.0, col=col.1[clus.01$cluster], pch=19, main="Iteration 1",
     cex=cex.1)
# Исходные центры кластеров
points(clus.00$centers, col="red", pch=19, cex=9*cex.1)
# Новые центры кластеров
points(clus.01$centers, col="black", pch=19, cex=9*cex.1)
# Стрелки - перемещение центров кластеров
arrows(clus.00$centers[,1], clus.00$centers[,2],
       x1=clus.01$centers[,1], y1=clus.01$centers[,2],
       col="red", lwd=2, angle=15, length=0.1)
```

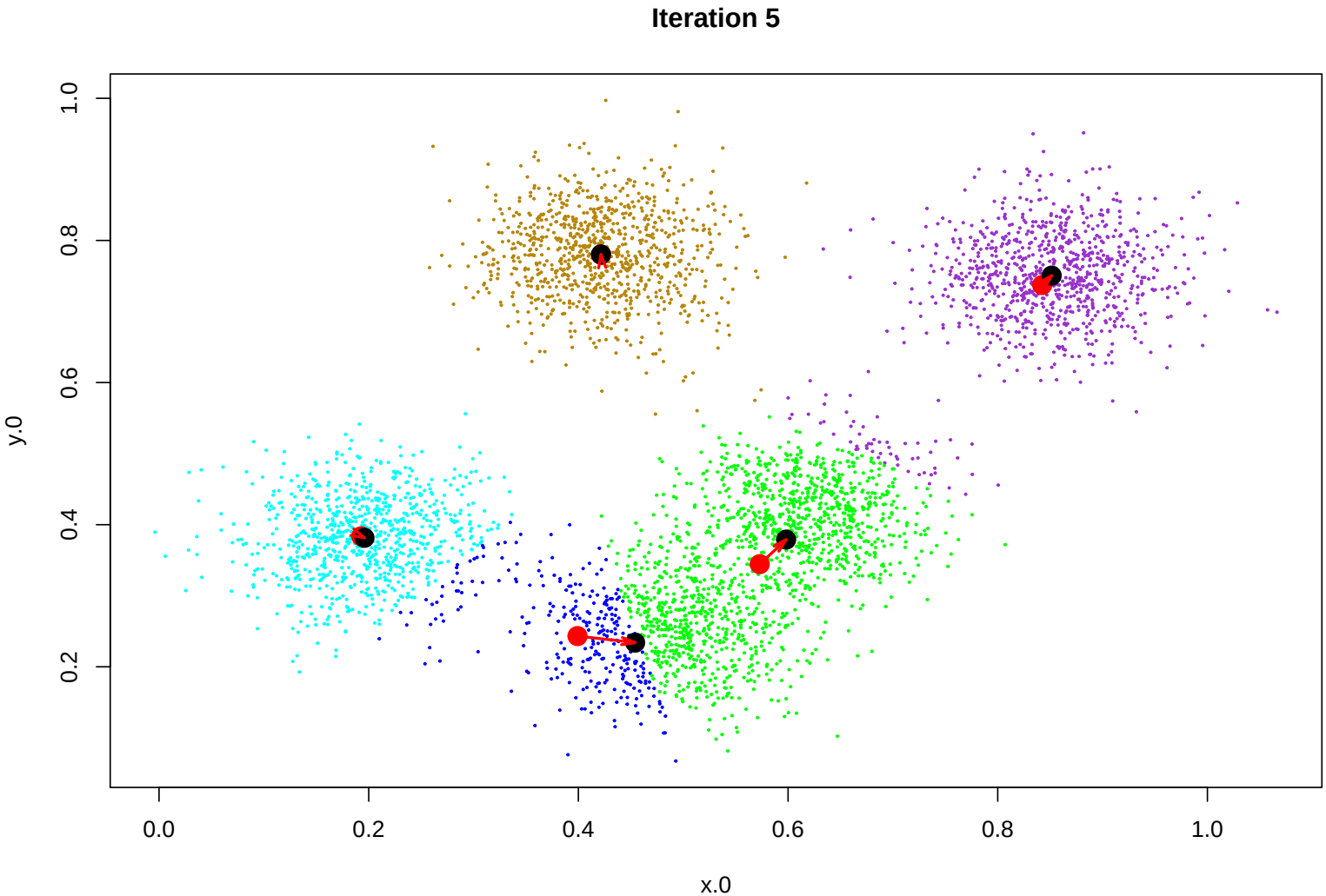

Iteration 0
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

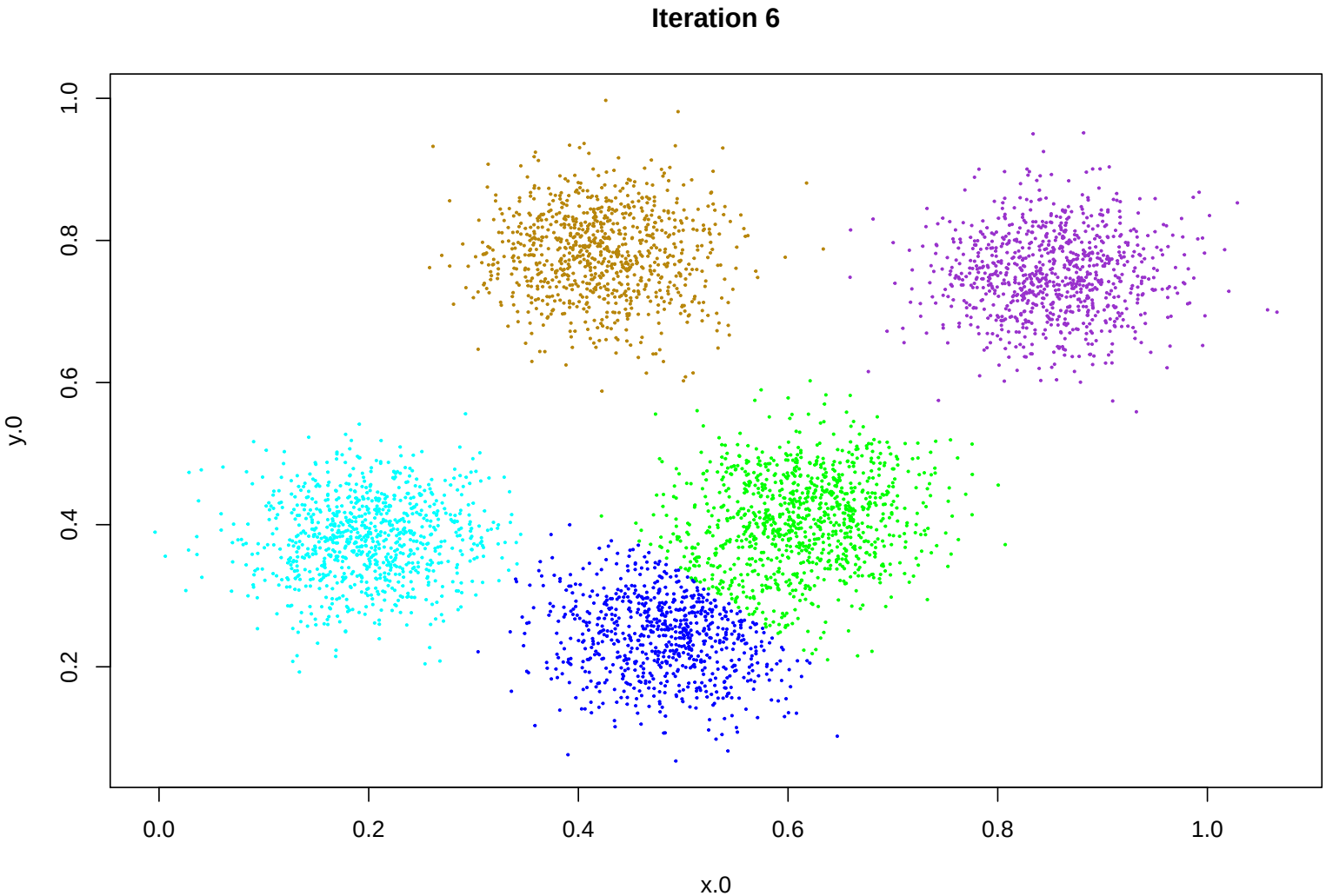

Iteration 1
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

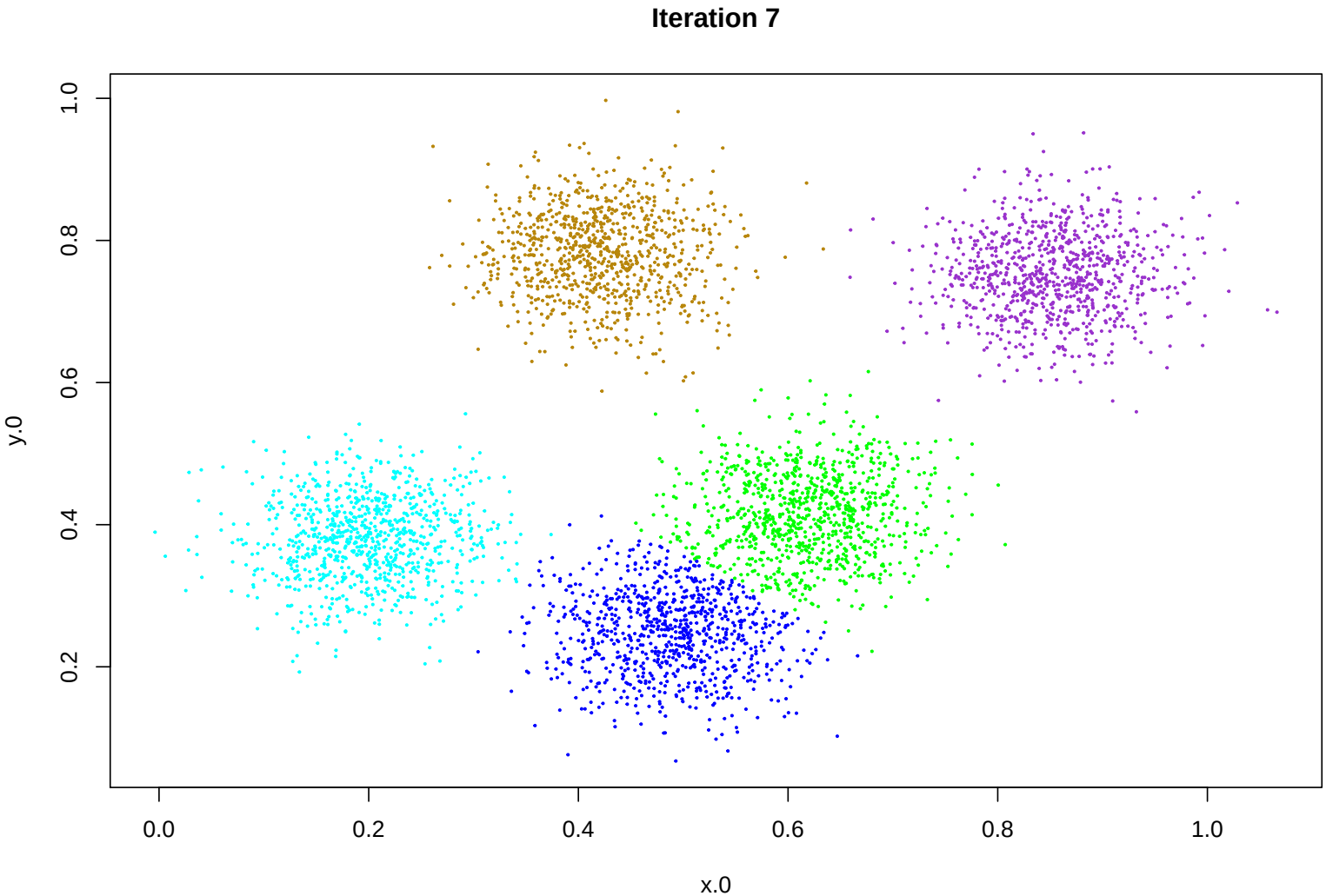

Iteration 2
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

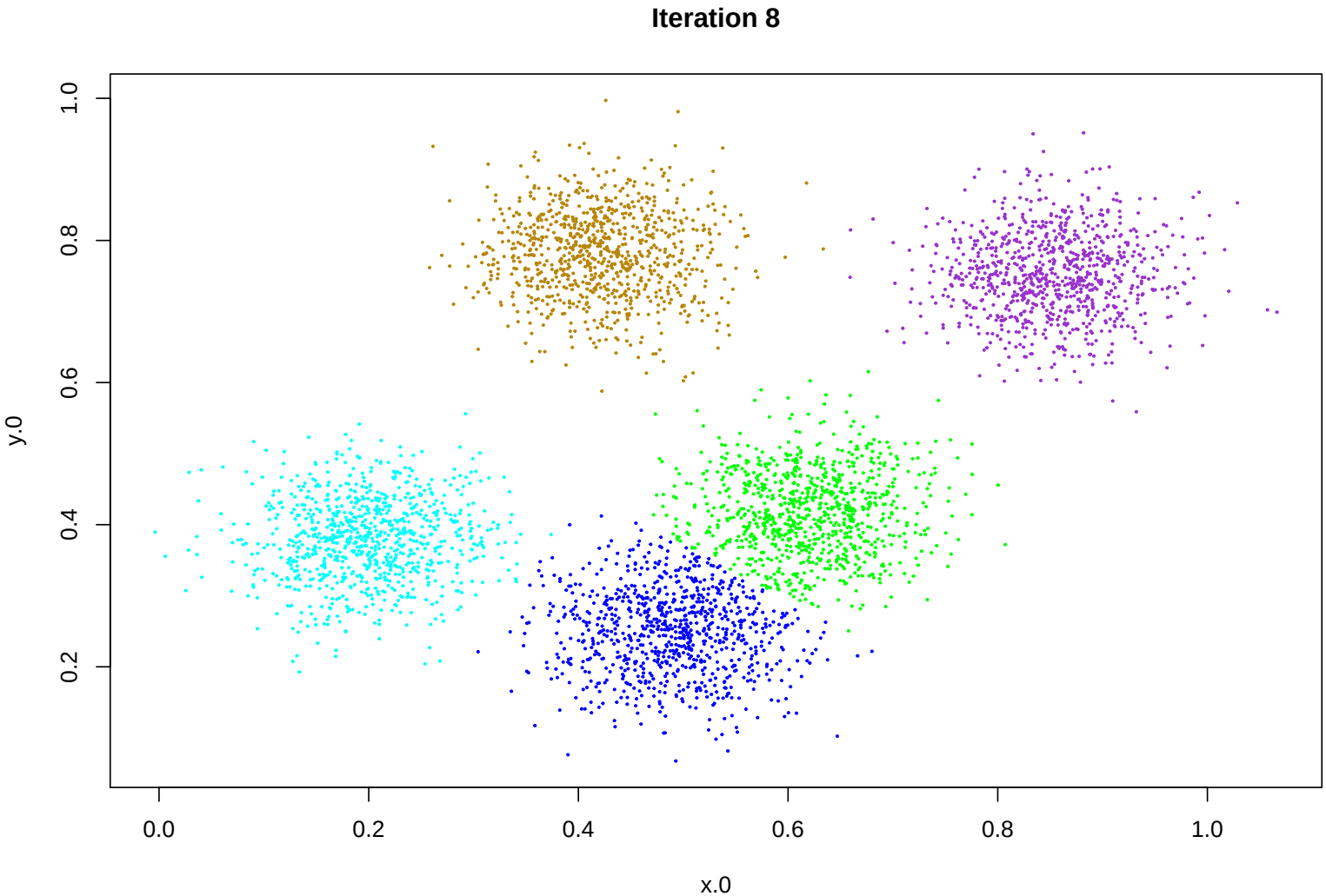

Iteration 3
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

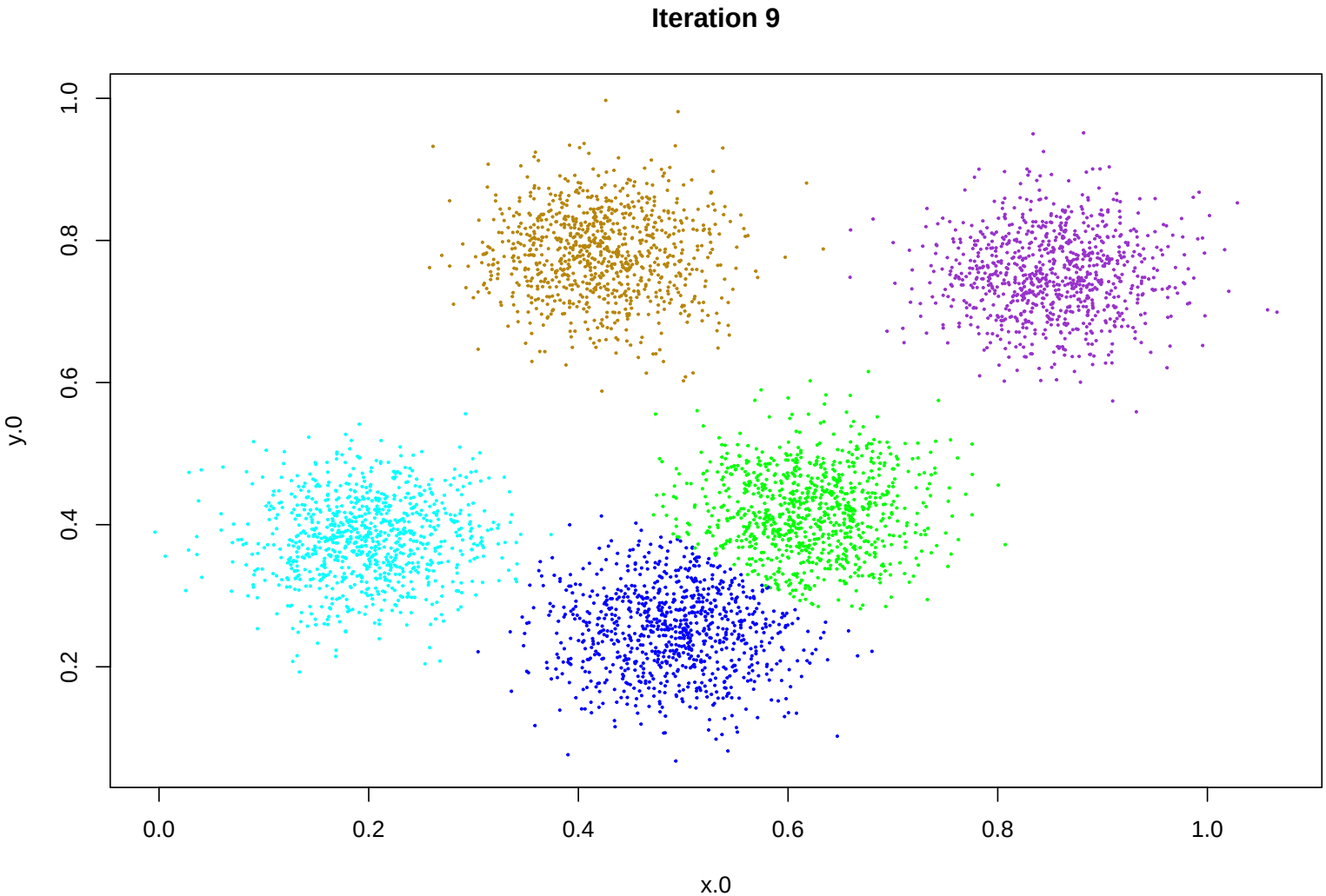

Iteration 4
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

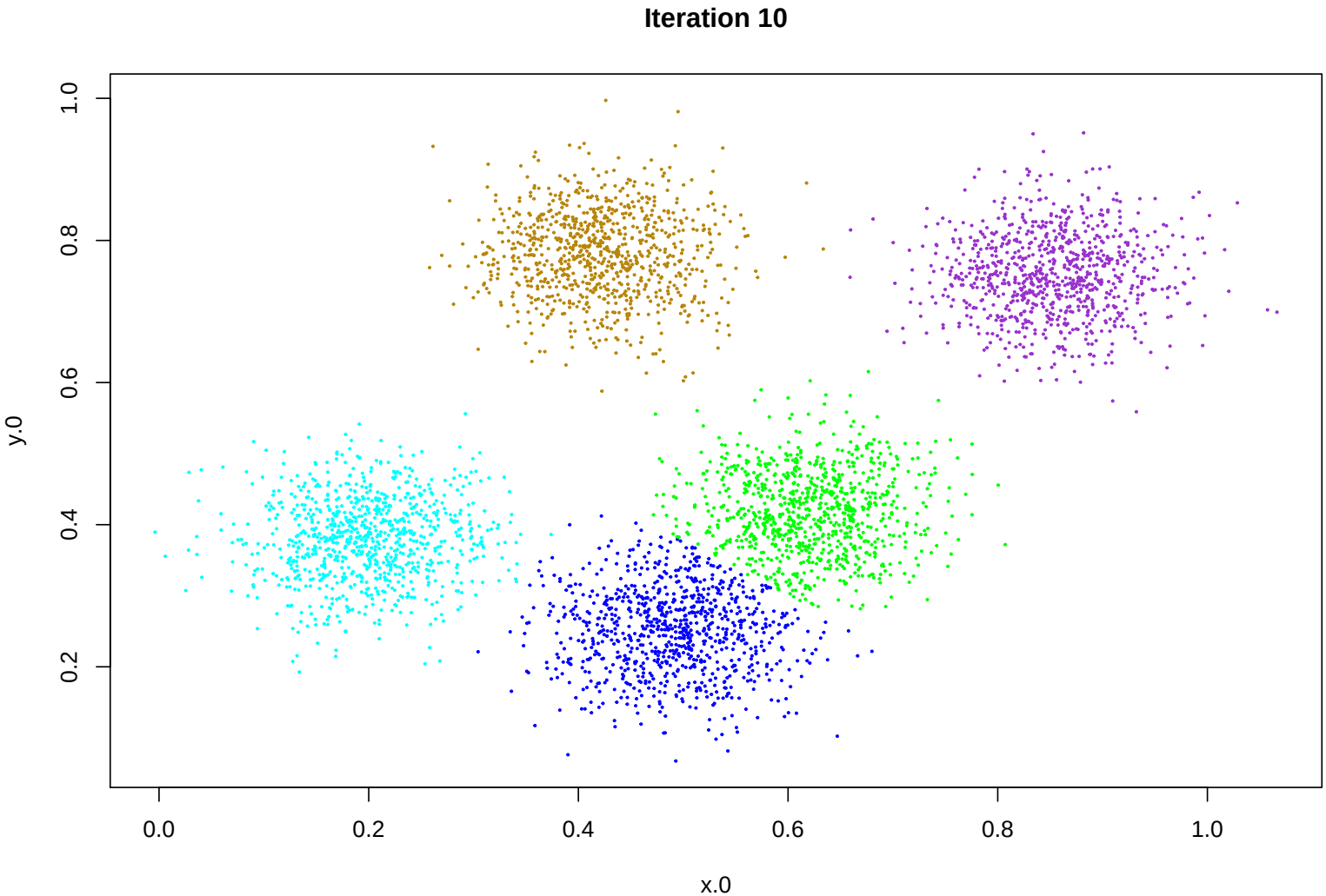

Iteration 5
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0

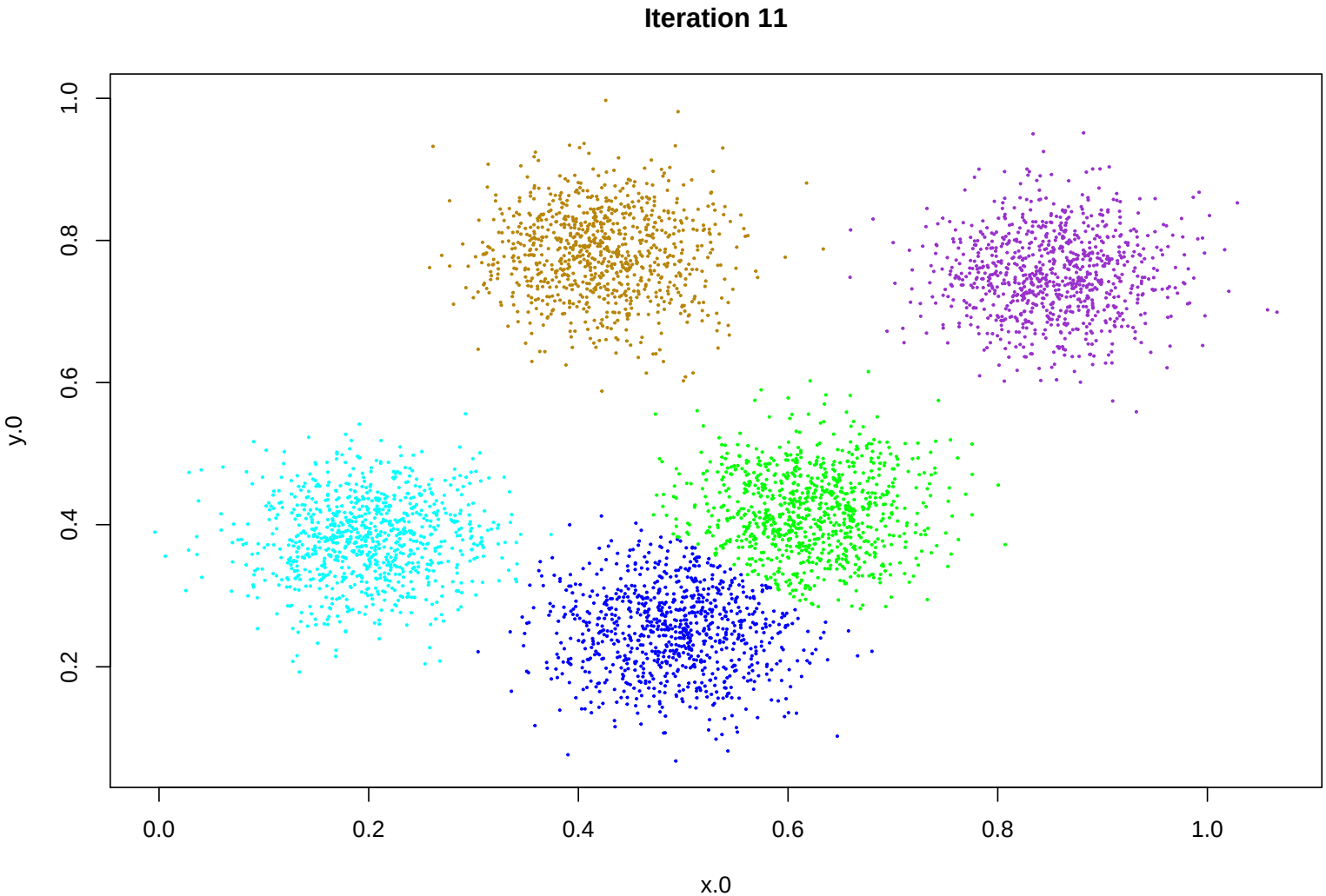

Iteration 6
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0


Iteration 7
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 8
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0


Iteration 9
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0


Iteration 10
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0


Iteration 11
y.0
x.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0

Сохранение графика в файл

```
png(file = "iter0.png", bg = "transparent")
plot(data.0, col="blue", pch=19, main="Iteration 0")
dev.off()
```

Выбор начального расположения центров кластеров

Ограничение 3. Результат зависит от начальных центров кластеров.

Наиболее популярны два метода.

1. Forgy (фамилия)

Случайным образом выбираются k наблюдений. Они и будут начальными центрами кластеров.

2. Случайное разбиение (Random Partition)

Каждое наблюдение случайным образом приписывается к одному из кластеров. Находятся центры тяжести кластеров. Они и будут начальными центрами.

k-means++ — улучшенная версия алгоритма кластеризации k-means. Суть улучшения заключается в нахождении более «хороших» начальных значений центроидов кластеров.

Определение числа кластеров

Мы не знаем точное число кластеров, но представляем себе в каком диапазоне оно находится.

Можно прогнать процедуру кластеризации много раз, задавая разное число кластеров, и выбрать наилучшую кластеризацию.

Вопрос: какая кластеризация наилучшая?

Математическая модель

Действие алгоритма k-средних таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

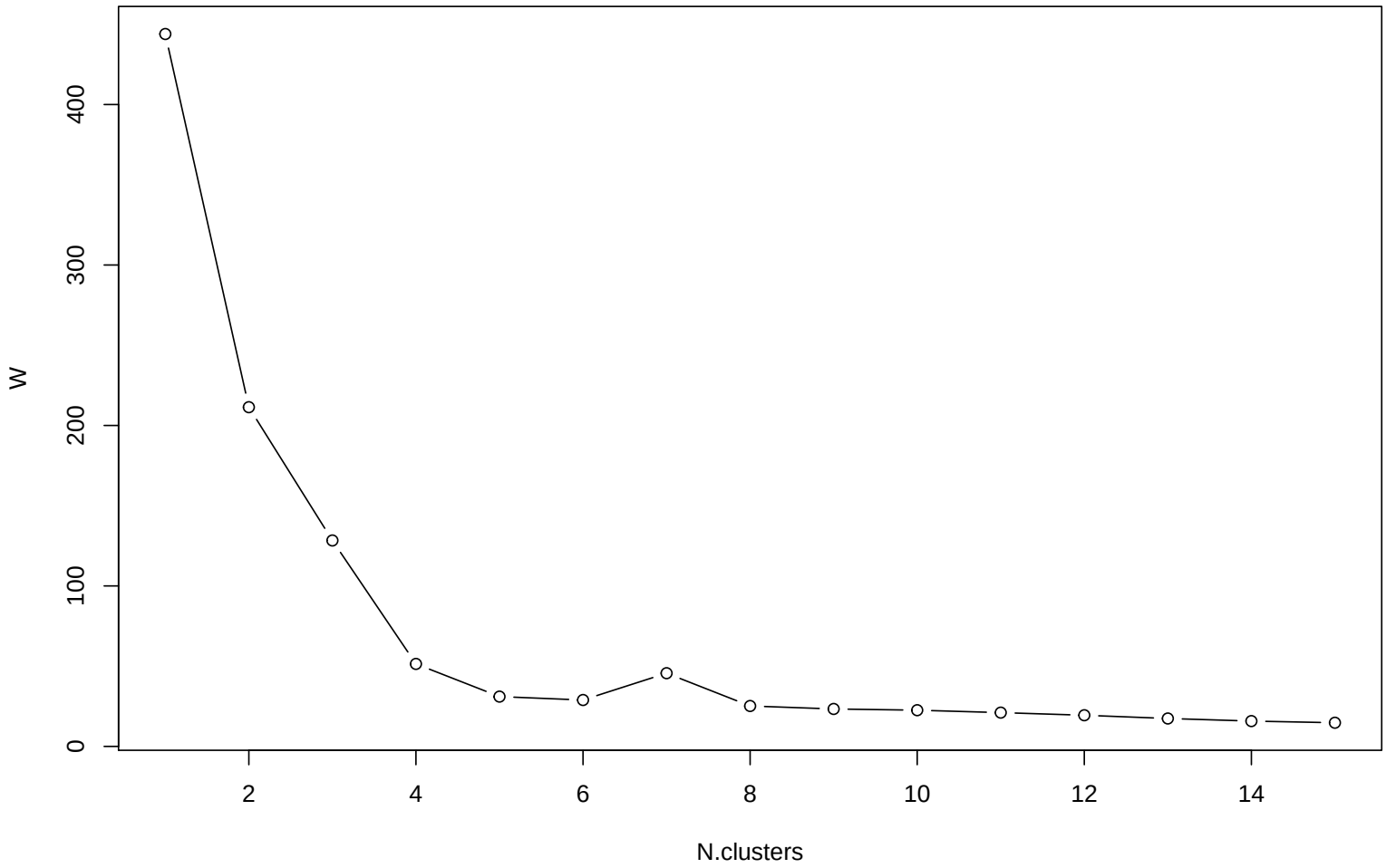
$$W = \underset{S}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2,$$

k — число кластеров; S_i — i-й кластер; $i = 1, 2, \ldots, k$; μ_i — центр тяжести i-го кластера.

W — от Within-cluster sum of squares (WCSS), tot.withinss в функции kmeans().

График “каменистая осыпь”

```
# Таблица для хранения результатов
cost_df <- data.frame()
# Запускаем kmeans для k от 1 до 15
for(k in 1:15){
  kmeans<- kmeans(x=data.0, centers=k, iter.max=100,
                  algorithm = "Lloyd")
  # Соберем число кластеров и оценку внутрикластерного расс
  # и запишем в таблицу
  cost_df<- rbind(cost_df, cbind(k, kmeans$tot.withinss))
}
# Даем имена столбцам таблицы
names(cost_df) <- c("N.clusters", "W")
# Строим график
plot(cost_df, type="b")
```


W
0
100
200
300
400
2
4
6
8
10
12
14
N.clusters

Если кластеризовать нужно много...

Понадобится способ автоматического определения излома (числа кластеров)

```
# Установим новый пакет. Кавычки обязательны!
install.packages("NbClust")

library(NbClust)

Best <- NbClust(data.0, distance = "euclidean",
                min.nc = 2, max.nc = 15,
                method = "ward.D", index = "alllong")
```

Приготовьтесь подождать несколько минут...

Недостатки метода k-средних

- Число кластеров k нужно знать заранее. *Но есть способ его определить.*
- Только евклидово расстояние. *Переходим к k-medoids.*
- Результат зависит от начальных центров кластеров. *Лечится числом попыток выбора начальных центров.*
- Слишком много вычислений расстояний. На поздних итерациях мало точек переходят из кластера в кластер, вычисления для “определившихся” точек можно исключить. Только как?