1. Исследовать зависимость от стратегии начальной инициализации

В данном разделе сравниваются 3 вида инициализации: случайная, min_max_mid, метод "локтя" поиска оптимального количества кластеров. Инициализация min_max_mid состоит в определении самого близкого к началу координат центра кластера, самого дальнего центра от начала координат, среднего по позиции в списке массиве, остальные кластеры случайным образом распределяются. Также существуют другие общеизвестные методы инициализации: силуэтный метод, статистика разрыва и т.д.

Раздел 1.2. одновременно включает и раздел 3.

Для сравнения разных методов рассматривается 10000 объектов, 30 кластеров, точность (centroids_residual) центров кластеров 0.001

1.1. Случайная инициализация

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
k = 30  # Clusters number (centers number)
n_points = 10000 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
```

Default parameters

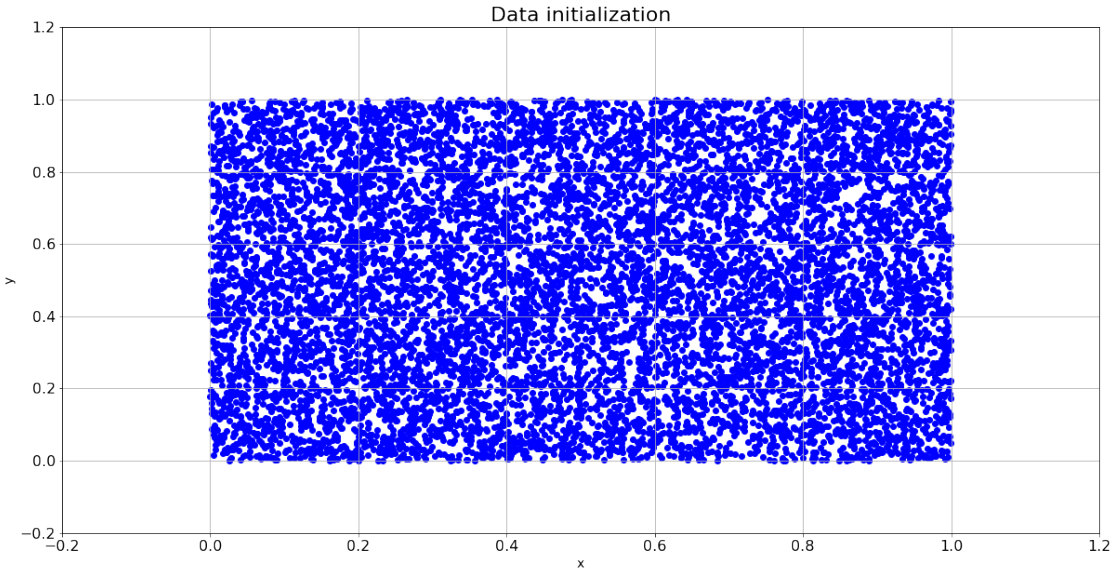
```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1, \ldots, x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
```

```
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
```

```
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

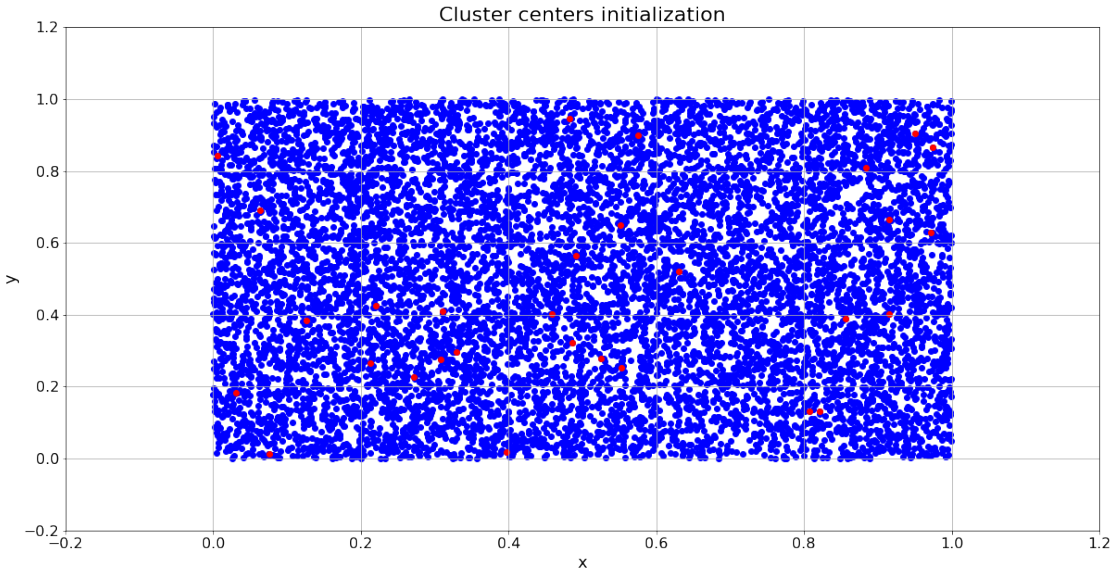
```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
```

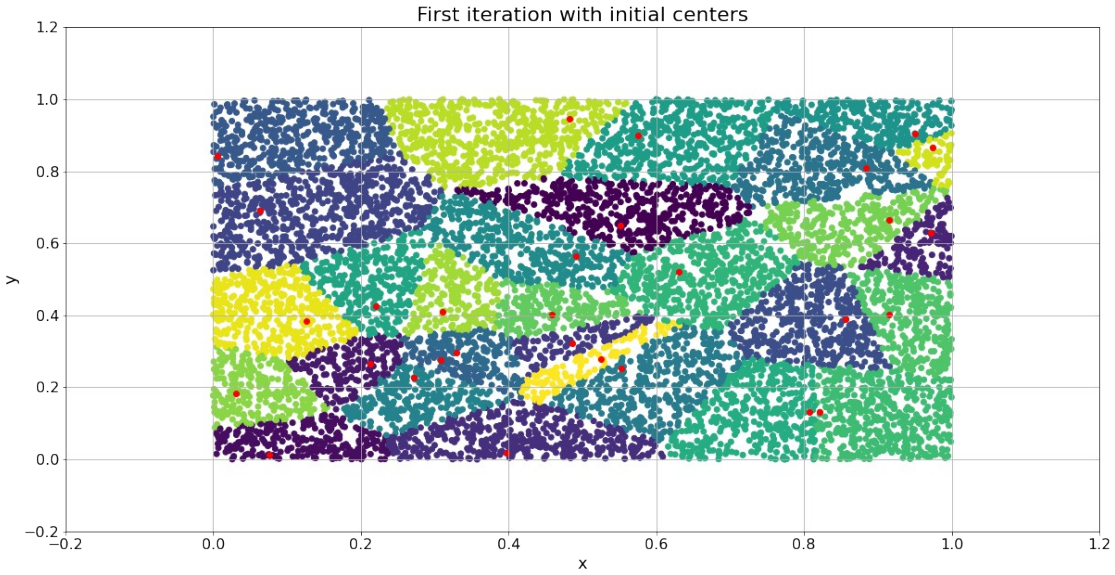
```
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result

#centroids = initialize_centroids(points, k)
centroids = initialize_centroids_random(points, k)

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)
```

```
[ 1 21 28 ... 26 17 23]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

```
[[0.55262471 0.69741458]
 [0.11999808 0.05148961]
 [0.18639314 0.25875081]
 [0.95702218 0.59467513]
 [0.41464496 0.05942241]
 [0.48922307 0.33013836]
 [0.14358451 0.68401275]
 [0.8121685  0.38161012]
 [0.11366646 0.89462634]
 [0.32019146 0.2614977 ]
 [0.35938994 0.29613158]
 [0.82483095 0.80789892]
 [0.59759325 0.21165001]
 [0.27180971 0.16563002]
 [0.4181741  0.59328831]
 [0.8932055  0.94136088]
 [0.62468118 0.88332886]
 [0.2131397  0.49134263]
```

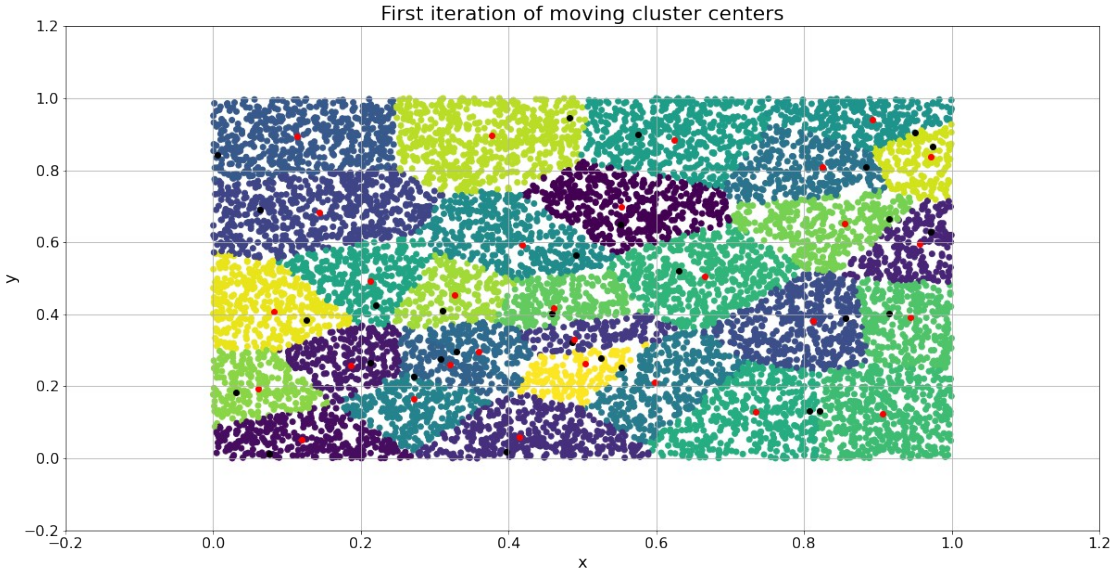
```
 [0.73486285 0.12803295]
 [0.66612998 0.5059908 ]
 [0.90627209 0.12306681]
 [0.94414715 0.39070438]
 [0.46131829 0.41790401]
 [0.85513087 0.65105516]
 [0.06095907 0.19344577]
 [0.32679427 0.45467914]
 [0.37746017 0.89684467]
 [0.97220326 0.83847386]
 [0.0820216  0.40823373]
 [0.50409048 0.26252557]]
[[0.55114113 0.64873295]
 [0.07548229 0.01414086]
 [0.21272366 0.2649603 ]
 [0.9712979  0.62891219]
 [0.39728407 0.01763331]
 [0.4859259  0.32303433]
 [0.06387144 0.69077773]
 [0.85684946 0.38982145]
 [0.00629491 0.84299721]
 [0.30768181 0.27640589]
 [0.32934701 0.29727201]
 [0.88369323 0.80873228]
 [0.55334145 0.25300745]
 [0.27209593 0.22746093]
 [0.49081272 0.56441544]
 [0.95022404 0.90458446]
 [0.57561813 0.89842186]
 [0.22033007 0.42639044]
 [0.80777382 0.13141534]
 [0.63098344 0.52184298]
 [0.82166056 0.13145205]
 [0.9151829  0.40297189]
 [0.45823794 0.4010709 ]
 [0.91484569 0.66526173]
 [0.03064002 0.18275468]
 [0.31077556 0.41099638]
 [0.48249344 0.94493502]
 [0.9744074  0.86626726]
 [0.12666131 0.38310825]
 [0.52502885 0.27807966]]
```

```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[ 1 20 28 ... 26 17 23]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
```

```
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```

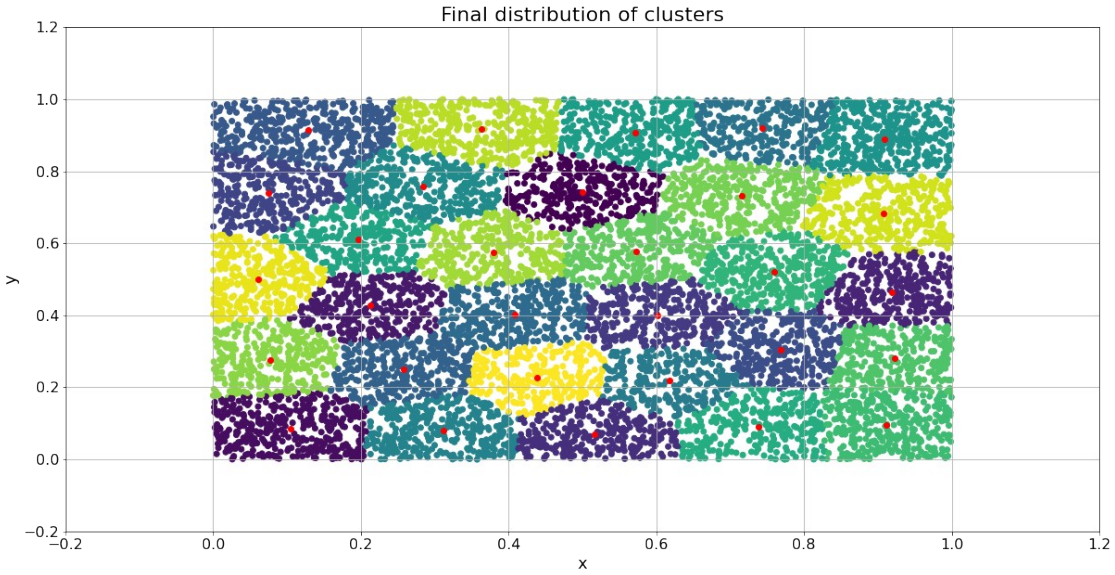



```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)
```

```
Iteration number =  1 False
Iteration number =  2 False
Iteration number =  3 False
Iteration number =  4 False
Iteration number =  5 False
Iteration number =  6 False
Iteration number =  7 False
```

```
Iteration number =  8 False
Iteration number =  9 False
Iteration number =  10 False
Iteration number =  11 False
Iteration number =  12 False
Iteration number =  13 False
Iteration number =  14 False
Iteration number =  15 False
Iteration number =  16 False
Iteration number =  17 False
Iteration number =  18 False
Iteration number =  19 False
Iteration number =  20 False
Iteration number =  21 False
Iteration number =  22 False
Iteration number =  23 False
Iteration number =  24 False
Iteration number =  25 False
Iteration number =  26 False
Iteration number =  27 False
Iteration number =  28 False
Iteration number =  29 False
Iteration number =  30 False
Iteration number =  31 False
Iteration number =  32 False
Iteration number =  33 False
Iteration number =  34 False
Iteration number =  35 False
Iteration number =  36 False
Iteration number =  37 False
Iteration number =  38 False
Iteration number =  39 False
Iteration number =  40 False
Iteration number =  41 False
Iteration number =  42 False
Iteration number =  43 False
Iteration number =  44 False
Iteration number =  45 False
Iteration number =  46 False
Iteration number =  47 False
Iteration number =  48 False
Iteration number =  49 False
Iteration number =  50 False
Iteration number =  51 False
Iteration number =  52 False
Iteration number =  53 False
Iteration number =  54 False
Iteration number =  55 False
Iteration number =  56 True
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Потребовалось 56 итераций для сходимости решения по нахождению центров кластеров

1.2. , 3. min_max_mid инициализация

Данная стратегия также относится к разделу 3, но было принятно решение поместить в раздел 1.

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
k = 30  # Clusters number (centers number)
n_points = 10000 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
```

Default parameters

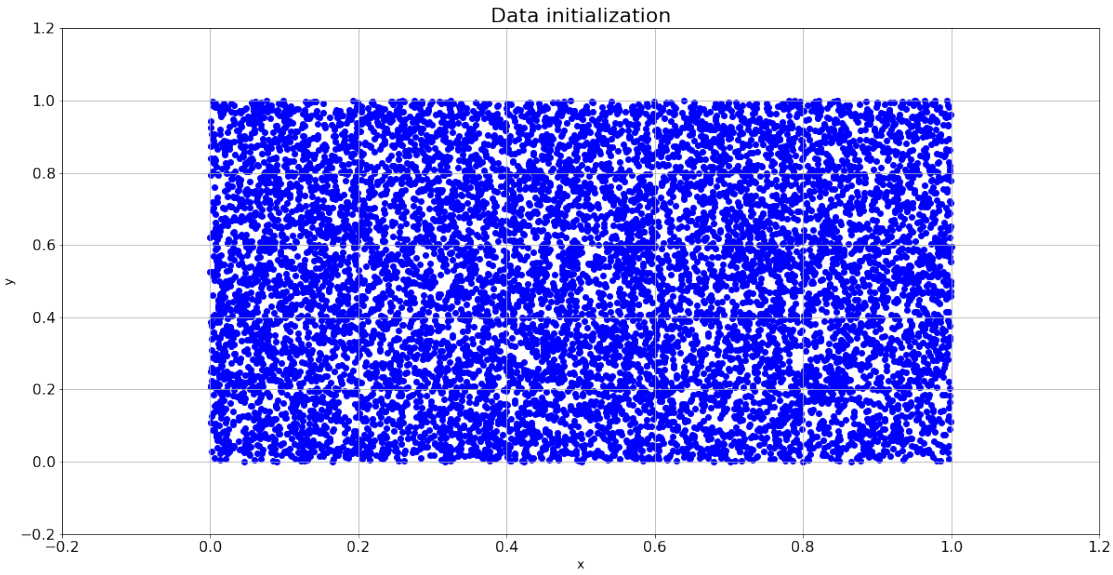
```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

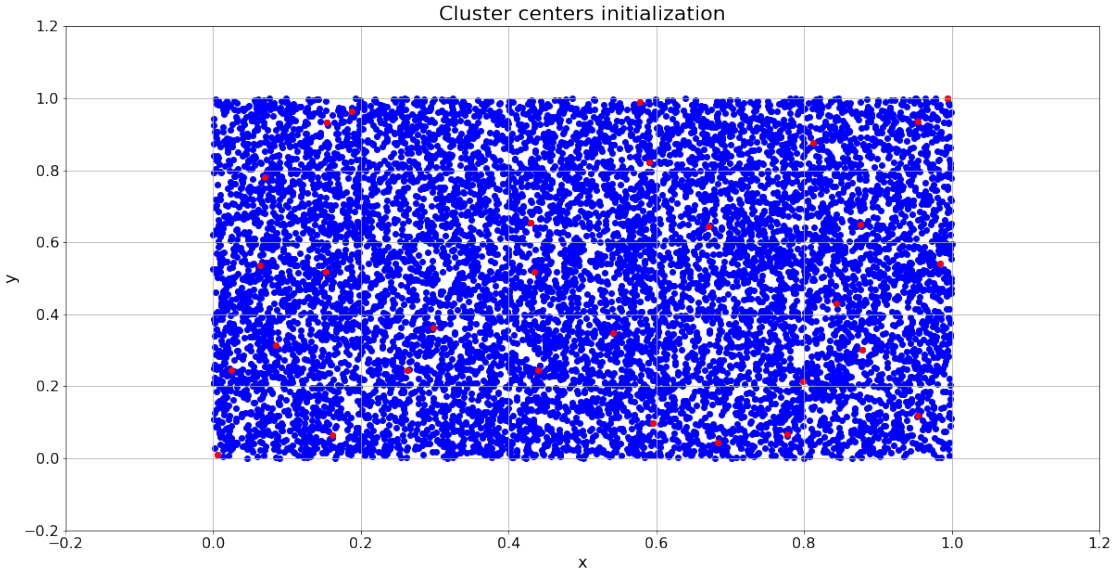
Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Задаётся стратегия начальной инициализации

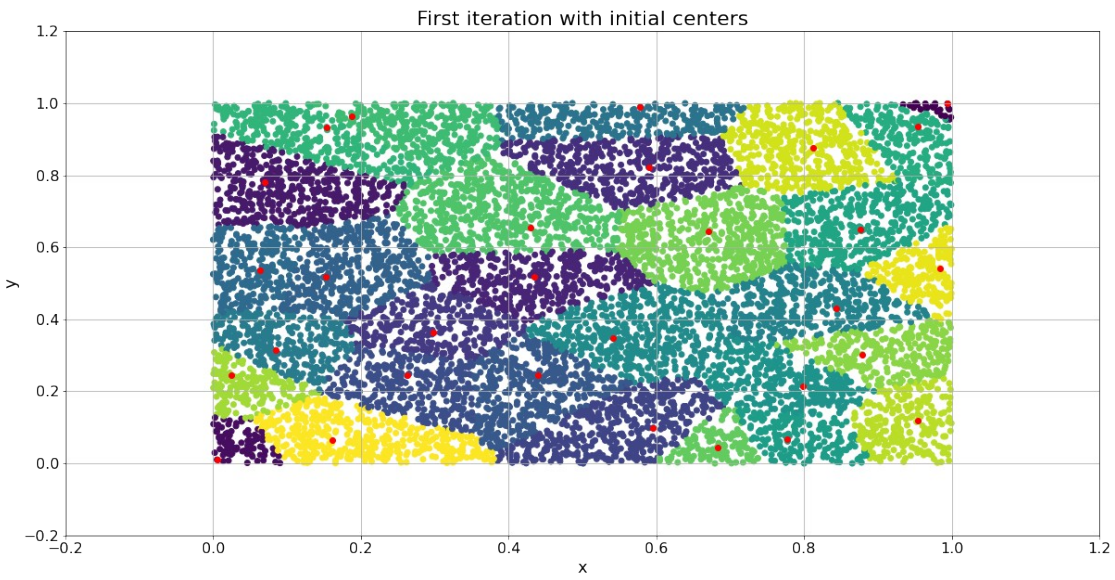
```
centroids = initialize_centroids(points, k)
#centroids = initialize_centroids_random(points, k)

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)
```

```
[ 8  4 13 ...  1 23 11]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

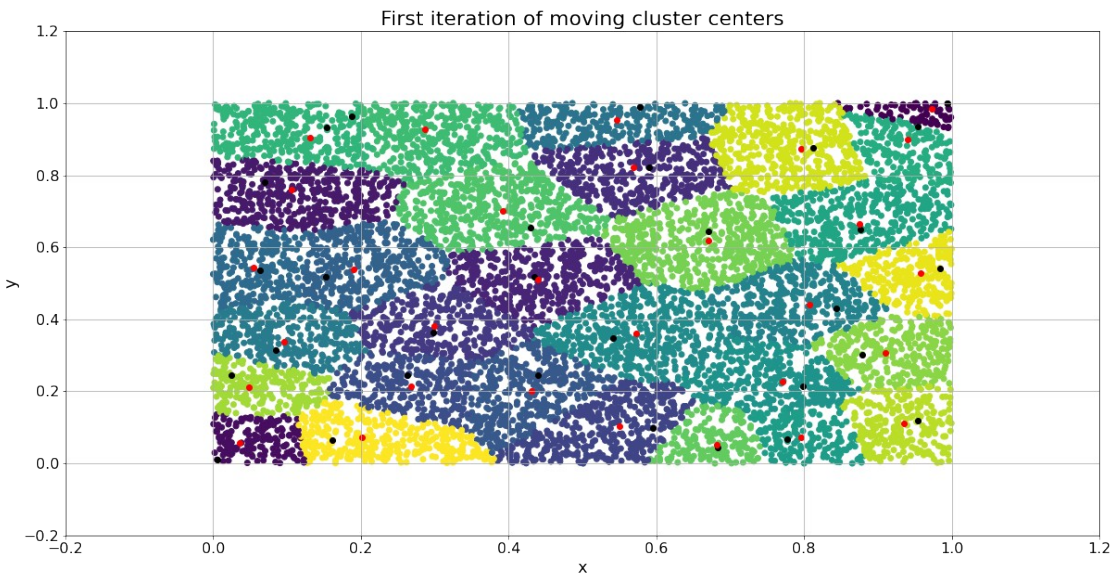
```
[[0.97330578 0.98439922]
 [0.0362337  0.05773838]
 [0.10669679 0.75969625]
 [0.43971631 0.5104455 ]
 [0.56942951 0.82292018]
 [0.29898531 0.38151328]
 [0.549759   0.10358566]
 [0.26807696 0.2128608 ]
 [0.43162976 0.20214515]
 [0.18967227 0.53902682]
 [0.05524942 0.5448467 ]
 [0.54678696 0.95220229]
 [0.09586449 0.33647824]
 [0.80730762 0.44054849]
 [0.57334754 0.36019019]
 [0.7705417  0.22716555]
 [0.79610409 0.07237697]
 [0.87548721 0.66459183]
 [0.93982198 0.89852217]
 [0.13104452 0.9038329 ]
 [0.2862082  0.92676794]
 [0.39272408 0.70125099]
 [0.68200019 0.05143284]
 [0.67124769 0.61832699]
 [0.91075304 0.30639231]
 [0.04814749 0.21261584]
 [0.93486049 0.10993893]
 [0.79663223 0.87288045]
 [0.95805821 0.52947115]
```

```
 [0.2012371  0.07142732]]
[[0.99435841 0.9985053 ]
 [0.00530242 0.01045032]
 [0.06982061 0.78136135]
 [0.43510428 0.51813299]
 [0.58987703 0.82218497]
 [0.29854329 0.36286848]
 [0.59598431 0.09890758]
 [0.26264053 0.24402756]
 [0.44014887 0.24447708]
 [0.15201967 0.51909467]
 [0.06350264 0.53491784]
 [0.57836378 0.98967606]
 [0.08477217 0.31416241]
 [0.84340832 0.42990245]
 [0.54189259 0.34795388]
 [0.7986853  0.21433307]
 [0.77701763 0.06696156]
 [0.87663483 0.64946651]
 [0.95465359 0.9346317 ]
 [0.15383057 0.93152311]
 [0.18806679 0.96283459]
 [0.42929258 0.65569604]
 [0.68266608 0.04539958]
 [0.67092791 0.64345727]
 [0.87886426 0.30264235]
 [0.02452329 0.24584392]
 [0.95455722 0.11847869]
 [0.81248513 0.87528793]
 [0.98398929 0.54226757]
 [0.1613876  0.06409572]]
```

```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[ 8 27 13 ...  1 23 27]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```

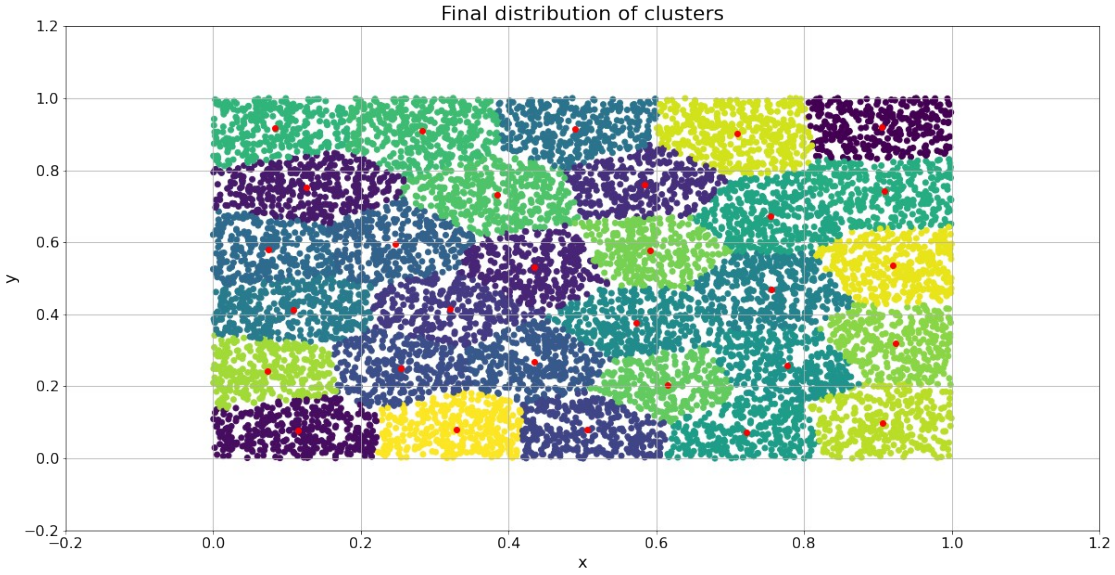



```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)
```

```
Iteration number =  1 False
Iteration number =  2 False
Iteration number =  3 False
Iteration number =  4 False
Iteration number =  5 False
Iteration number =  6 False
Iteration number =  7 False
Iteration number =  8 False
Iteration number =  9 False
Iteration number =  10 False
Iteration number =  11 False
Iteration number =  12 False
Iteration number =  13 False
Iteration number =  14 False
Iteration number =  15 False
Iteration number =  16 False
```

```
Iteration number =  17 False
Iteration number =  18 False
Iteration number =  19 False
Iteration number =  20 False
Iteration number =  21 False
Iteration number =  22 False
Iteration number =  23 False
Iteration number =  24 False
Iteration number =  25 False
Iteration number =  26 False
Iteration number =  27 False
Iteration number =  28 False
Iteration number =  29 False
Iteration number =  30 False
Iteration number =  31 False
Iteration number =  32 False
Iteration number =  33 False
Iteration number =  34 True
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Потребовалось 34 итерации для сходимости решения по нахождению центров кластеров. Данный результат более чем в 1.5 раза быстрее инициализации случаными элементами.

1.3. Инициализация методом "локтя"

В данном методе добавлена точность(elbow_residual) определения функции "локтя"

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
n_points = 10000 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
elbow_residual = 0.001 # Residual for checking centroids
```

Default parameters

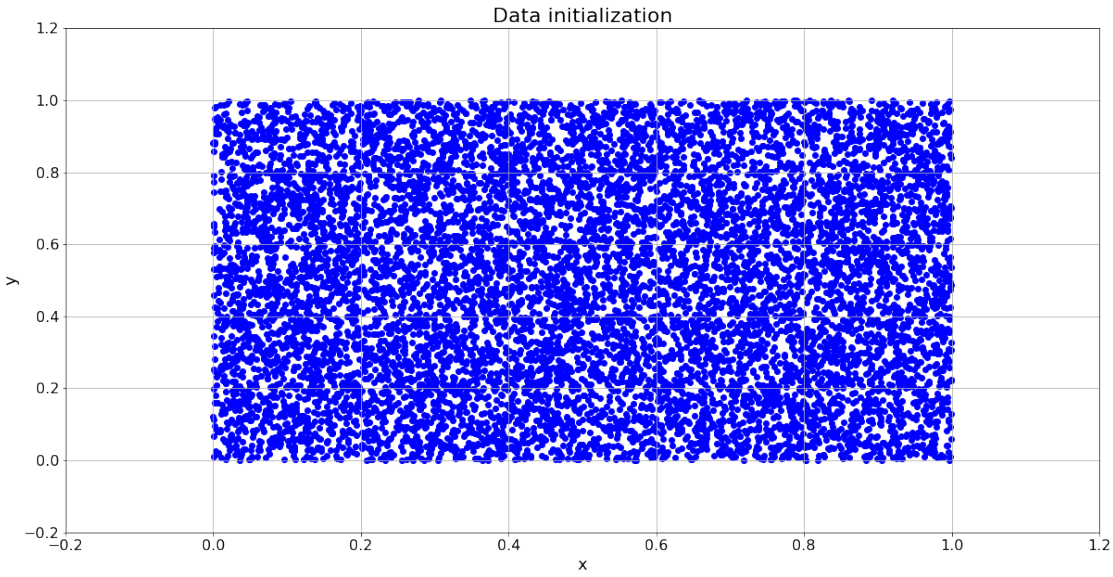
```
k = 1  # Clusters number (centers number)
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_randow(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Метод Elbow - метод локтя. Вычисляется сумма квадратов расстояний от каждой точки до центра кластера, которому он принадлежит. Далее

возвращется средне арифметическое, т.е. общая сумма делится на число кластеров k.

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} \left| p - x_i \right|^2$$

```
def elbow_function(points, centroids, closest_points, k):
    result = 0
    for i in range(k):
        for j in range(len(closest_points)):
            if i == closest_points[j]:
                result += np.sqrt(((points[j][0] - centroids[i][0]) **
2) + ((points[j][1] - centroids[i][1]) ** 2)) ** 2
    return (result / k)
```

Метод Elbow - метод "локтя". С помощью данного методы вычисляется оптимальное количество кластеров. На каждой итерации значение суммы квадратов расстояний до каждого кластера будет уменьшаться, то есть в какой-то момент значение функции будет слабо изменяться, что означает достижении оптимального числа кластеров. В алгоритм ниже добавлена невязка для останова.

```
elbow_dist = np.array([])
k_final = 0
prev = 100
print('Iteration number (clusters number) = ', k)
#centroids = initialize_centroids(points, k)
centroids = initialize_centroids_randow(points, k)
closest_points = closest_centroids(points, centroids, n_points, k)
dist = elbow_function(points, centroids, closest_points, k)
elbow_dist = np.append(elbow_dist, dist)
prev = np.copy(dist)
k += 1
#print(elbow_dist)
#K = range(2, 100, 1)
while True:
    print('Iteration number (clusters number) = ', k)
    #centroids = initialize_centroids(points, k)
    centroids = initialize_centroids_randow(points, k)
    closest_points = closest_centroids(points, centroids, n_points, k)
    dist = elbow_function(points, centroids, closest_points, k)
    #elbow_dist = np.append(elbow_dist, dist)
    if np.absolute(dist - prev) < elbow_residual:
        k_final = k
        elbow_dist = np.append(elbow_dist, dist)
        break
    else:
        prev = np.copy(dist)
        elbow_dist = np.append(elbow_dist, dist)
        k += 1
```

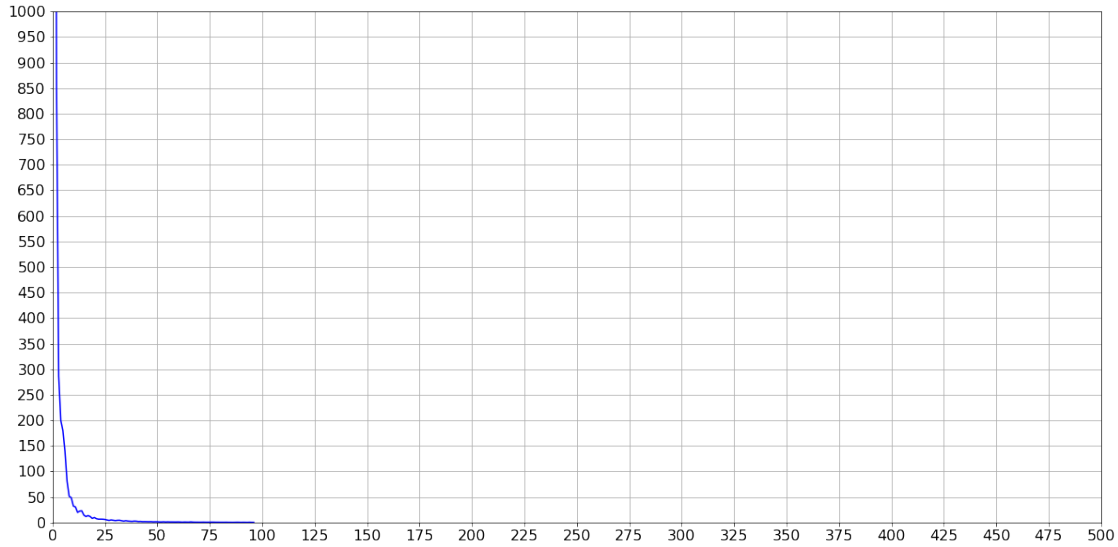
```
print('Elbow method function')
print(elbow_dist)
print('Clusters optimum number = ', k_final)
```

```
Iteration number (clusters number) =  1
Iteration number (clusters number) =  2
Iteration number (clusters number) =  3
Iteration number (clusters number) =  4
Iteration number (clusters number) =  5
Iteration number (clusters number) =  6
Iteration number (clusters number) =  7
Iteration number (clusters number) =  8
Iteration number (clusters number) =  9
Iteration number (clusters number) =  10
Iteration number (clusters number) =  11
Iteration number (clusters number) =  12
Iteration number (clusters number) =  13
Iteration number (clusters number) =  14
Iteration number (clusters number) =  15
Iteration number (clusters number) =  16
Iteration number (clusters number) =  17
Iteration number (clusters number) =  18
Iteration number (clusters number) =  19
Iteration number (clusters number) =  20
Iteration number (clusters number) =  21
Iteration number (clusters number) =  22
Iteration number (clusters number) =  23
Iteration number (clusters number) =  24
Iteration number (clusters number) =  25
Iteration number (clusters number) =  26
Iteration number (clusters number) =  27
Iteration number (clusters number) =  28
Iteration number (clusters number) =  29
Iteration number (clusters number) =  30
Iteration number (clusters number) =  31
Iteration number (clusters number) =  32
Iteration number (clusters number) =  33
Iteration number (clusters number) =  34
Iteration number (clusters number) =  35
Iteration number (clusters number) =  36
Iteration number (clusters number) =  37
Iteration number (clusters number) =  38
Iteration number (clusters number) =  39
Iteration number (clusters number) =  40
Iteration number (clusters number) =  41
Iteration number (clusters number) =  42
Iteration number (clusters number) =  43
Iteration number (clusters number) =  44
Iteration number (clusters number) =  45
```

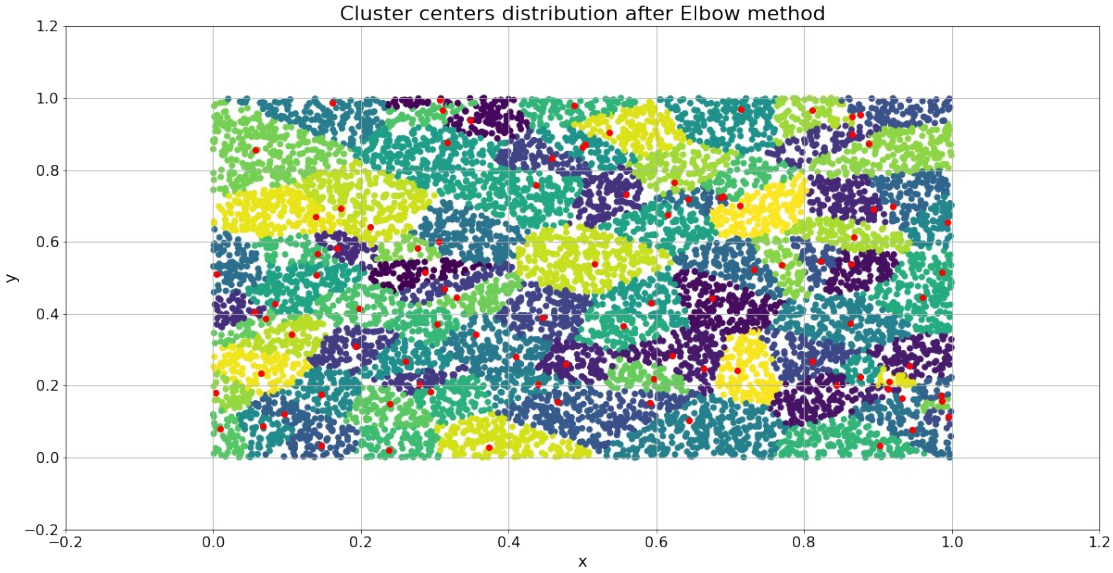
```
Iteration number (clusters number) =  46
Iteration number (clusters number) =  47
Iteration number (clusters number) =  48
Iteration number (clusters number) =  49
Iteration number (clusters number) =  50
Iteration number (clusters number) =  51
Iteration number (clusters number) =  52
Iteration number (clusters number) =  53
Iteration number (clusters number) =  54
Iteration number (clusters number) =  55
Iteration number (clusters number) =  56
Iteration number (clusters number) =  57
Iteration number (clusters number) =  58
Iteration number (clusters number) =  59
Iteration number (clusters number) =  60
Iteration number (clusters number) =  61
Iteration number (clusters number) =  62
Iteration number (clusters number) =  63
Iteration number (clusters number) =  64
Iteration number (clusters number) =  65
Iteration number (clusters number) =  66
Iteration number (clusters number) =  67
Iteration number (clusters number) =  68
Iteration number (clusters number) =  69
Iteration number (clusters number) =  70
Iteration number (clusters number) =  71
Iteration number (clusters number) =  72
Iteration number (clusters number) =  73
Iteration number (clusters number) =  74
Iteration number (clusters number) =  75
Iteration number (clusters number) =  76
Iteration number (clusters number) =  77
Iteration number (clusters number) =  78
Iteration number (clusters number) =  79
Iteration number (clusters number) =  80
Iteration number (clusters number) =  81
Iteration number (clusters number) =  82
Iteration number (clusters number) =  83
Iteration number (clusters number) =  84
Iteration number (clusters number) =  85
Iteration number (clusters number) =  86
Iteration number (clusters number) =  87
Iteration number (clusters number) =  88
Iteration number (clusters number) =  89
Iteration number (clusters number) =  90
Iteration number (clusters number) =  91
Iteration number (clusters number) =  92
Iteration number (clusters number) =  93
Iteration number (clusters number) =  94
Iteration number (clusters number) =  95
```

```
Iteration number (clusters number) =  96
Elbow method function
[2.38990857e+03 8.50467789e+02 2.86482480e+02 1.99745713e+02
 1.79980866e+02 1.39560360e+02 8.08499373e+01 5.12901491e+01
 4.86608481e+01 3.22340999e+01 3.04925136e+01 1.98567927e+01
 2.23852671e+01 2.29387338e+01 1.45824411e+01 1.17437279e+01
 1.36703991e+01 1.17881438e+01 8.10153675e+00 9.85477395e+00
 7.55515452e+00 6.51037372e+00 6.62258332e+00 6.49045597e+00
 6.07403933e+00 4.97426914e+00 4.11544670e+00 5.02377196e+00
 4.43344692e+00 3.74064077e+00 4.31307035e+00 4.44971908e+00
 3.52777183e+00 2.73322541e+00 3.54991289e+00 2.85754349e+00
 2.51467563e+00 2.22749419e+00 2.78361923e+00 2.58617144e+00
 1.97734483e+00 2.13489045e+00 1.73837729e+00 1.81156252e+00
 1.71251467e+00 1.59038819e+00 1.74282252e+00 1.37627607e+00
 1.50203548e+00 1.50637815e+00 1.13519694e+00 1.10663985e+00
 1.39819480e+00 1.03710729e+00 1.29567455e+00 1.09279437e+00
 1.11040875e+00 1.05224538e+00 1.01788017e+00 1.15698551e+00
 9.64451269e-01 7.37156490e-01 9.83113763e-01 8.61218147e-01
 7.98399486e-01 1.21149010e+00 8.34150765e-01 6.85456245e-01
 6.83335450e-01 6.35718810e-01 6.44129455e-01 7.35657180e-01
 6.26451180e-01 6.13593787e-01 6.76099620e-01 7.36236263e-01
 7.00291029e-01 5.77758784e-01 5.86688393e-01 5.28870337e-01
 5.21246671e-01 5.10073080e-01 5.79787182e-01 4.48762325e-01
 4.75451815e-01 4.06158031e-01 4.75612538e-01 5.71937947e-01
 5.36422737e-01 3.87164577e-01 4.98842560e-01 4.44732144e-01
 3.54502172e-01 5.60829484e-01 3.66609690e-01 3.66023073e-01]
Clusters optimum number =  96
```

```
K = range(1, k_final + 1)
K = np.array(K)
#print(K)
#print(K[len(K) - 1])

plt.figure(figsize = (20,10))
plt.title('Elbow method (choosing the value of k)', fontsize = 22)
plt.xlabel('k (number of clusters)', fontsize = 18)
plt.ylabel('Elbow function SSE', fontsize = 18)
plt.axes(xlim=(0, K[len(K) - 1] + 1), ylim=(0, 1000))
plt.grid(True)
plt.plot(K, elbow_dist, c = 'blue')
plt.xticks(np.linspace(0, 500, 21), fontsize = 16)
plt.yticks(np.linspace(0, 1000, 21), fontsize = 16);
#plt.xticks([int(5*i) for i in range(1, 10)])
#plt.yticks([int(50*i) for i in range(1, 20)])
#plt.yticks([0, 10, 20, 30, 40, 50, 60, 70, 80])
```




```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
#plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points,
k_final)
```

```
print(new_centroids)
print(centroids)
```

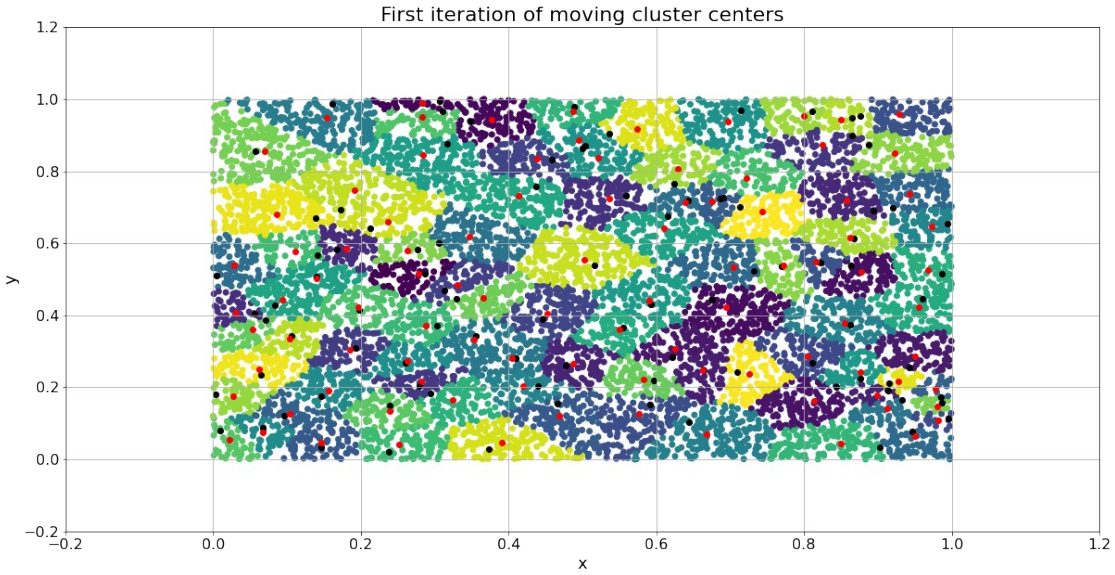
```
[[0.27826156 0.51649027]
 [0.28311441 0.98873853]
 [0.37721905 0.94188894]
 [0.69389319 0.42392815]
 [0.87811799 0.52070571]
 [0.81336021 0.16173031]
 [0.62543388 0.30600434]
 [0.95086219 0.28530611]
 [0.66268735 0.24675709]
 [0.8989373  0.17906411]
 [0.48729516 0.26485483]
 [0.85775021 0.72004728]
 [0.18046533 0.58379103]
 [0.03097829 0.40625949]
 [0.53588907 0.72462411]
 [0.82520048 0.87448452]
 [0.28192258 0.216822  ]
 [0.18562612 0.30499062]
 [0.33038432 0.48511849]
 [0.43843459 0.83354844]
 [0.45209769 0.40575085]
 [0.80535001 0.28506241]
 [0.98169417 0.10799869]
 [0.57711386 0.12764713]
 [0.92877036 0.95950783]
 [0.81579918 0.54845815]
 [0.14640226 0.04677338]
 [0.46812513 0.12125238]
 [0.67567734 0.71620532]
 [0.97754468 0.19508063]
 [0.10372011 0.12727372]
 [0.94349776 0.73779064]
 [0.70408177 0.53466563]
 [0.95005299 0.0641591 ]
 [0.02876486 0.53735982]
 [0.63914966 0.71377228]
 [0.34643113 0.61775806]
 [0.91275387 0.1407553 ]
 [0.40421917 0.28077322]
 [0.3531642  0.33378743]
 [0.15414271 0.94851599]
 [0.66803311 0.07071231]
 [0.85470889 0.38006364]
 [0.41915346 0.20293385]
 [0.97334161 0.64749883]
 [0.15616122 0.19124459]
 [0.06769828 0.07413171]
```

```
 [0.26343438 0.27336337]
 [0.69673803 0.93862693]
 [0.49532319 0.88713703]
 [0.52097673 0.8363526 ]
 [0.87593908 0.24291782]
 [0.59056055 0.44083189]
 [0.28415883 0.84451526]
 [0.14031346 0.50317229]
 [0.61041208 0.642378  ]
 [0.0941829  0.44233721]
 [0.41358161 0.73168667]
 [0.95576067 0.42334907]
 [0.55080987 0.36025292]
 [0.96766022 0.52693749]
 [0.32403585 0.16616559]
 [0.48809232 0.9668782 ]
 [0.85062586 0.04326224]
 [0.28832501 0.37058537]
 [0.25133544 0.04191844]
 [0.11168499 0.57667423]
 [0.7221248  0.77989299]
 [0.19669925 0.42352067]
 [0.28342672 0.95182073]
 [0.02157335 0.05490651]
 [0.23904979 0.13522658]
 [0.05292638 0.36125194]
 [0.58232311 0.22175769]
 [0.36632523 0.44806864]
 [0.06916903 0.85435726]
 [0.97992711 0.14824677]
 [0.26249404 0.57924858]
 [0.77228707 0.53876231]
 [0.92307968 0.85112749]
 [0.02686704 0.17462197]
 [0.79987267 0.95247424]
 [0.62867354 0.80739005]
 [0.86240112 0.6170292 ]
 [0.85013482 0.94229617]
 [0.10316836 0.33591351]
 [0.19145661 0.74596486]
 [0.50201396 0.5542605 ]
 [0.2368724  0.66095183]
 [0.39043088 0.04724683]
 [0.57380325 0.91711273]
 [0.92738122 0.21556935]
 [0.08567674 0.68123936]
 [0.06195216 0.24891132]
 [0.7256938  0.23847039]
 [0.74389153 0.68790282]]
[[0.28717296 0.51538602]
```

```
[0.30722704 0.99475054]
[0.34805597 0.93901032]
[0.67547567 0.44413327]
[0.86388955 0.53739666]
[0.84367995 0.20236839]
[0.62146911 0.28285352]
[0.94277528 0.25450525]
[0.66407123 0.24855157]
[0.91447918 0.19031943]
[0.47803523 0.26085855]
[0.89398349 0.69136105]
[0.16702094 0.58213771]
[0.05536041 0.40577894]
[0.55954792 0.73306019]
[0.8654811  0.89813881]
[0.27984236 0.20401281]
[0.19245438 0.31039387]
[0.31281155 0.4680669 ]
[0.45826017 0.83147579]
[0.44563844 0.3884141 ]
[0.81115905 0.26747042]
[0.99601318 0.11408587]
[0.59131757 0.15153596]
[0.87614807 0.95351175]
[0.82212469 0.5456653 ]
[0.14661318 0.03436899]
[0.46627648 0.15606669]
[0.6864134  0.72401589]
[0.98534709 0.17281242]
[0.09635906 0.1204164 ]
[0.92088233 0.69840636]
[0.73163008 0.52336306]
[0.94689574 0.07677505]
[0.00496279 0.50987107]
[0.6432476  0.71957318]
[0.30533207 0.60095306]
[0.93259617 0.16429941]
[0.41003551 0.28104518]
[0.35533893 0.34163723]
[0.16183784 0.98700228]
[0.64481015 0.10278013]
[0.86271194 0.37474188]
[0.44043528 0.20413351]
[0.99437236 0.65537063]
[0.14627996 0.17493191]
[0.06780685 0.08871883]
[0.26102484 0.26791671]
[0.71430999 0.96939339]
[0.50321457 0.87099639]
[0.50042108 0.86396483]
```

```
 [0.876524   0.22545558]
 [0.59278389 0.42945765]
 [0.31691761 0.8754458 ]
 [0.13945849 0.50782951]
 [0.61597405 0.67456737]
 [0.08296666 0.42886287]
 [0.43741826 0.75701749]
 [0.96065389 0.446224  ]
 [0.55562107 0.36553368]
 [0.98628966 0.51538118]
 [0.29442053 0.18200832]
 [0.48851244 0.97878248]
 [0.90258675 0.03369864]
 [0.30270832 0.37159754]
 [0.23777254 0.0209386 ]
 [0.14158551 0.568078  ]
 [0.69063789 0.72785484]
 [0.19716913 0.4141157 ]
 [0.31107295 0.96730835]
 [0.00923979 0.08045169]
 [0.23858062 0.15076318]
 [0.0712491  0.3857177 ]
 [0.59622189 0.21965303]
 [0.32977091 0.44531894]
 [0.05711546 0.85602352]
 [0.98679379 0.15743517]
 [0.27704064 0.5820586 ]
 [0.76978734 0.53566382]
 [0.88755728 0.87450004]
 [0.00382236 0.18005328]
 [0.8117271  0.96548175]
 [0.62373192 0.76570489]
 [0.86812732 0.61328705]
 [0.86500441 0.94735263]
 [0.10557913 0.34382547]
 [0.17303702 0.69234532]
 [0.51662327 0.53823543]
 [0.21225928 0.64071494]
 [0.37293127 0.02748643]
 [0.53615034 0.9051594 ]
 [0.91491005 0.21090288]
 [0.13933385 0.66931318]
 [0.06423642 0.2342847 ]
 [0.71009006 0.24373724]
 [0.71273674 0.70030781]]

closest_points = closest_centroids(points, new_centroids, n_points,
k_final)
print(closest_points)

[89 63 34 ... 93 75 79]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k_final)
    closest_points = closest_centroids(points, new_centroids,
n_points, k_final)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)
print('Final centroids')
print(new_centroids)
print('Closest_points')
print(closest_points)
```

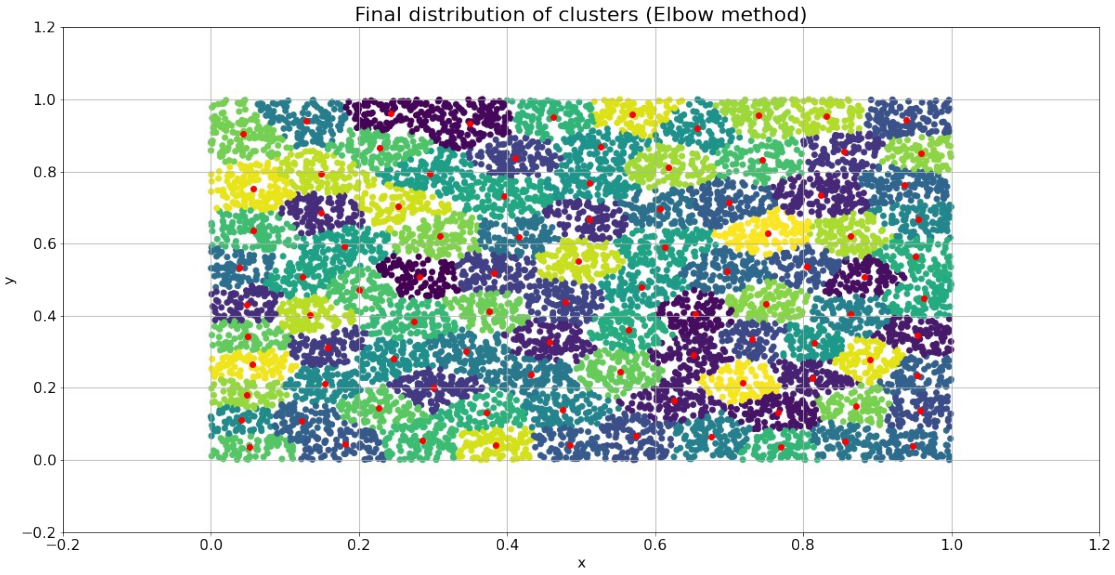
```
Iteration number =  1 False
Iteration number =  2 False
Iteration number =  3 False
Iteration number =  4 False
Iteration number =  5 False
Iteration number =  6 False
Iteration number =  7 False
Iteration number =  8 False
Iteration number =  9 False
Iteration number =  10 False
Iteration number =  11 False
Iteration number =  12 False
Iteration number =  13 False
Iteration number =  14 False
Iteration number =  15 False
Iteration number =  16 False
Iteration number =  17 False
Iteration number =  18 False
Iteration number =  19 False
Iteration number =  20 False
Iteration number =  21 False
Iteration number =  22 False
Iteration number =  23 False
Iteration number =  24 False
Iteration number =  25 False
Iteration number =  26 False
Iteration number =  27 False
Iteration number =  28 False
Iteration number =  29 False
Iteration number =  30 False
Iteration number =  31 False
Iteration number =  32 False
Iteration number =  33 False
Iteration number =  34 False
Iteration number =  35 False
Iteration number =  36 False
Iteration number =  37 False
Iteration number =  38 False
Iteration number =  39 False
Iteration number =  40 False
Iteration number =  41 False
Iteration number =  42 False
Iteration number =  43 False
Iteration number =  44 False
Iteration number =  45 False
Iteration number =  46 False
Iteration number =  47 False
Iteration number =  48 False
Iteration number =  49 False
Iteration number =  50 False
```

```
Iteration number =  51 False
Iteration number =  52 False
Iteration number =  53 False
Iteration number =  54 False
Iteration number =  55 False
Iteration number =  56 False
Iteration number =  57 True
Final centroids
[[0.28176217 0.50716003]
 [0.24248618 0.96041308]
 [0.34921718 0.93305827]
 [0.65426267 0.40464451]
 [0.88254827 0.50696953]
 [0.76626435 0.13173551]
 [0.65151238 0.29032097]
 [0.95463279 0.34442938]
 [0.62581659 0.16409256]
 [0.81288451 0.22819755]
 [0.45751022 0.32844514]
 [0.82381835 0.7337473 ]
 [0.14848181 0.68618447]
 [0.04871852 0.43024468]
 [0.51094376 0.66803439]
 [0.85567666 0.85544101]
 [0.30102903 0.20057465]
 [0.15781273 0.31108207]
 [0.38274133 0.51725552]
 [0.41156779 0.83814441]
 [0.479034   0.43893456]
 [0.73069394 0.33643273]
 [0.9580117  0.13744296]
 [0.57462093 0.06617177]
 [0.93927525 0.94185134]
 [0.80541202 0.53662574]
 [0.18174565 0.04403217]
 [0.48467785 0.04233259]
 [0.69924481 0.71336745]
 [0.95396057 0.23495977]
 [0.12217754 0.10766067]
 [0.93656916 0.76258747]
 [0.69683353 0.5231897 ]
 [0.94736585 0.03933818]
 [0.03827338 0.53359324]
 [0.60709821 0.69530651]
 [0.41574746 0.61741212]
 [0.85654407 0.0523382 ]
 [0.4325318  0.23618653]
 [0.34472452 0.30049153]
 [0.12909609 0.94042823]
 [0.67595527 0.06409688]
```

```
[0.8644229  0.40379381]
[0.47545708 0.13906129]
[0.95603013 0.66712452]
[0.15400231 0.2119817 ]
[0.04073411 0.11029963]
[0.24723919 0.28095467]
[0.65643738 0.9201807 ]
[0.52620422 0.8687924 ]
[0.51194064 0.7674623 ]
[0.81464068 0.32387118]
[0.58139097 0.47854551]
[0.29591816 0.793719  ]
[0.17991659 0.59170057]
[0.61254793 0.59012759]
[0.12423213 0.5083159 ]
[0.39622401 0.73055752]
[0.96340526 0.44853848]
[0.56370178 0.36112262]
[0.9519918  0.56526038]
[0.37196615 0.13154526]
[0.46184635 0.94954387]
[0.77009026 0.03594385]
[0.27395483 0.38402239]
[0.28602912 0.05399202]
[0.05769895 0.63543913]
[0.74466255 0.83182967]
[0.20008759 0.47138566]
[0.22758071 0.86498361]
[0.05251447 0.03552638]
[0.22600558 0.14530908]
[0.0501615  0.34251813]
[0.55335387 0.24393181]
[0.37547804 0.41367183]
[0.04323843 0.90383026]
[0.8712033  0.14985024]
[0.30975944 0.6202246 ]
[0.74992439 0.43336405]
[0.95871931 0.84955444]
[0.04865446 0.1819013 ]
[0.74005903 0.95557151]
[0.61740702 0.81103331]
[0.8635877  0.62126724]
[0.83089283 0.95368995]
[0.13420402 0.40202465]
[0.14820054 0.79237433]
[0.49612402 0.55246079]
[0.2534408  0.70444282]
[0.38492991 0.04237082]
[0.5694081  0.95772491]
[0.88964371 0.27832239]
```

```
 [0.05676475 0.75249952]
 [0.05589119 0.26475783]
 [0.71820352 0.21309236]
 [0.75171151 0.62848604]]
Closest_points
[89 37 34 ... 80 86 79]
```

```
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Final distribution of clusters (Elbow method)', fontsize =
22)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Данный метод по решению задачи занимает больше всего времени. Отметим также, что оптимальное количество кластеров в 3 раза больше, чем задавалась пользователем для других методов. Потребовалось 57 итераций для сходимости решения по нахождению центров кластеров.

2. Исследовать, для каких задач подходит / не подходит

В данном разделе рассматриваются случаи, когда метод кластеризации k-means даёт неоднозначные результаты. Подходящие задачи были рассмотрены в разделе 1.

2.1. Малое количество объектов и кластеров

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
#Малое число объёктов и кластеров
k = 3  # Clusters number (centers number)
n_points = 10 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
#centroids = initialize_centroids(points, k)  # min_max_mid initial
#centroids = initialize_centroids_random(points, k)  # random initial
```

Default parameters

```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

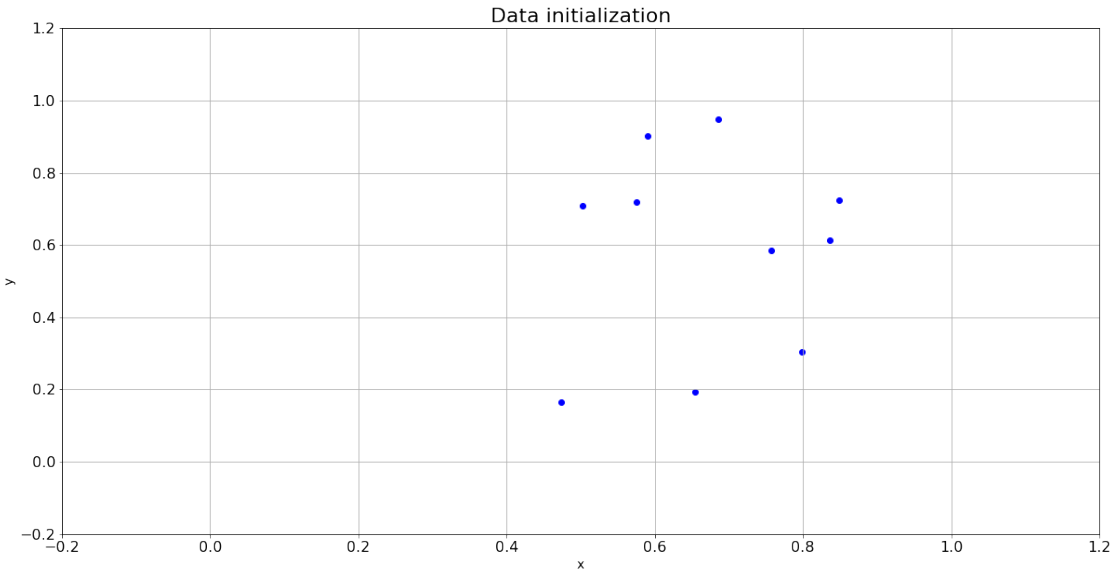
Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
```

```
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
```

```
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
```

```
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

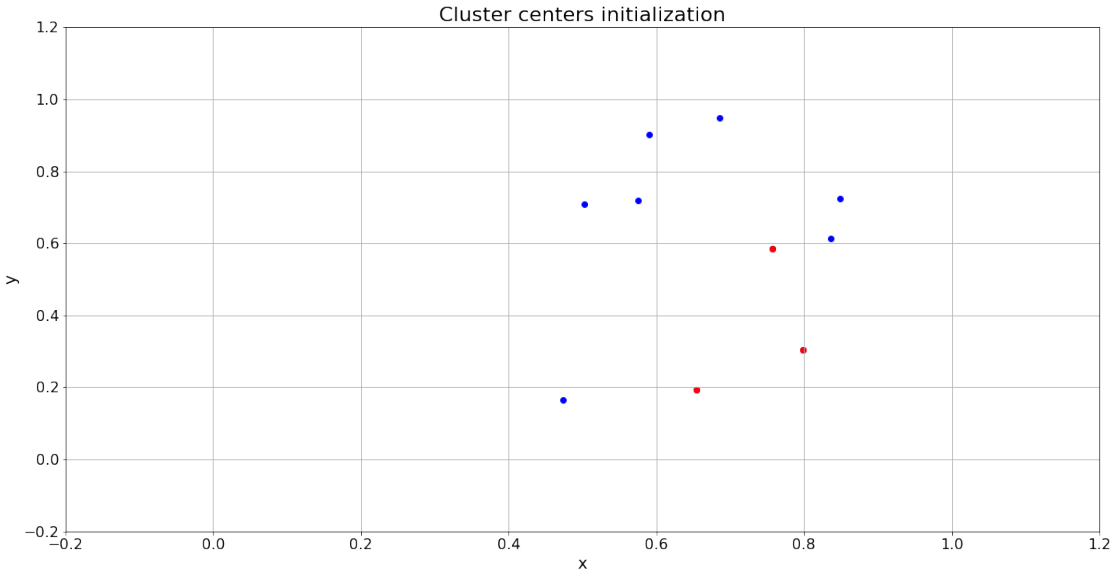
$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
```

```
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Entered by the user

```
#centroids = initialize_centroids(points, k)  # min_max_mid initial
centroids = initialize_centroids_random(points, k)  # random initial

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```

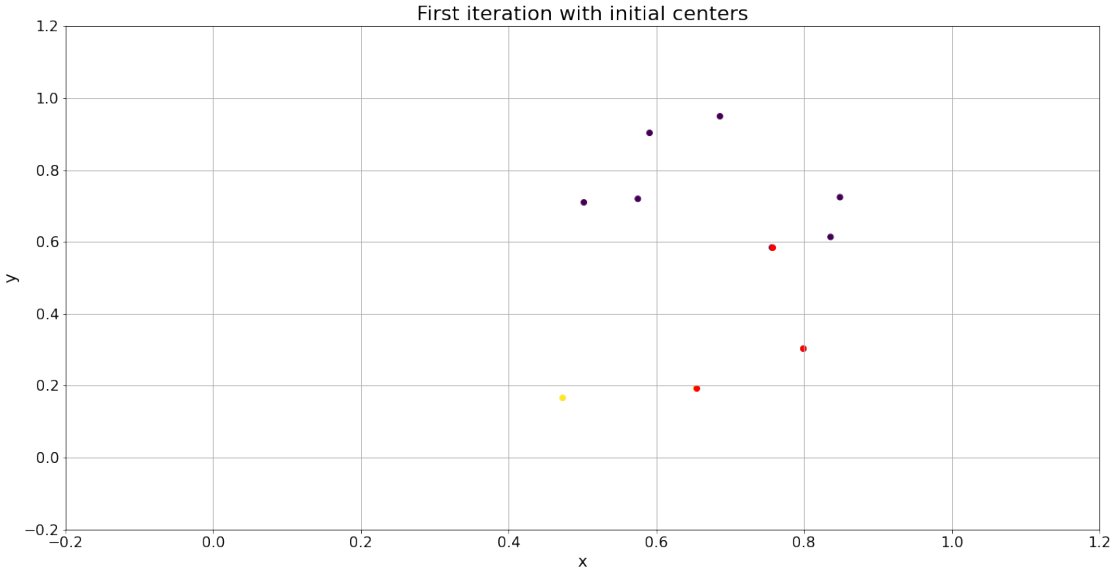



```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)
```

```
[1 2 0 2 0 0 0 0 0 0]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
```

```
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



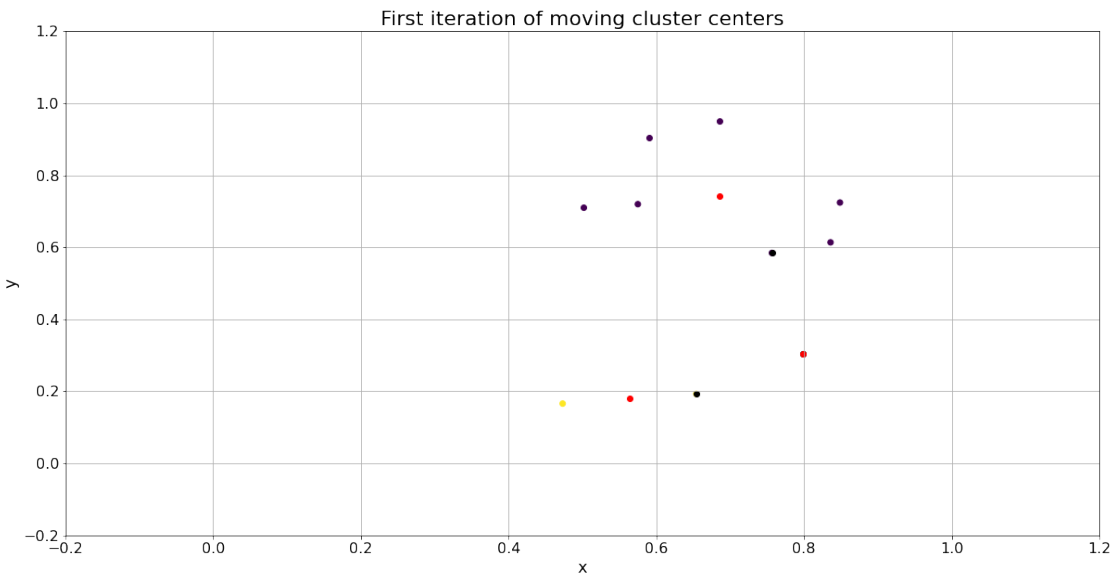

```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

```
[[0.68510542 0.74337261]
 [0.79922241 0.30330335]
 [0.56376357 0.17956131]]
[[0.75664287 0.58424333]
 [0.79922241 0.30330335]
 [0.65425786 0.19300162]]
```

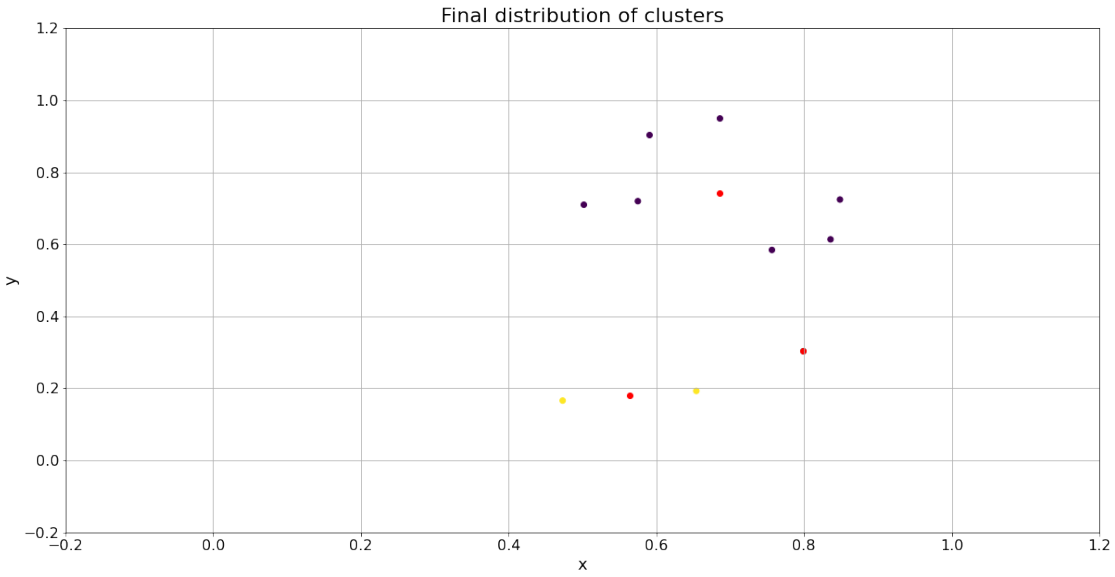
```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[1 2 0 2 0 0 0 0 0 0]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)

Iteration number =  1 False
Iteration number =  2 True

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

При малом числе объектов и кластеров результат противоречит кластерной структуре множества данных. Центры кластеров лежат вне сгруппированных объектов на значительном отдалении.

2.2. Большое количество объектов и малое количество кластеров

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
#Малое число объёктов и кластеров
k = 3  # Clusters number (centers number)
n_points = 1000 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
```

Default parameters

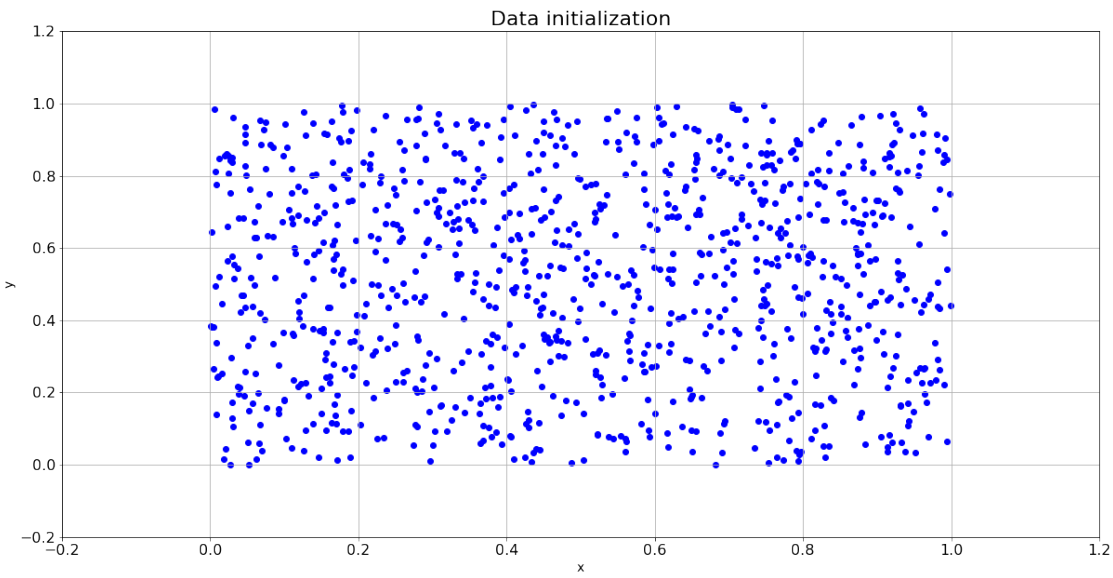
```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

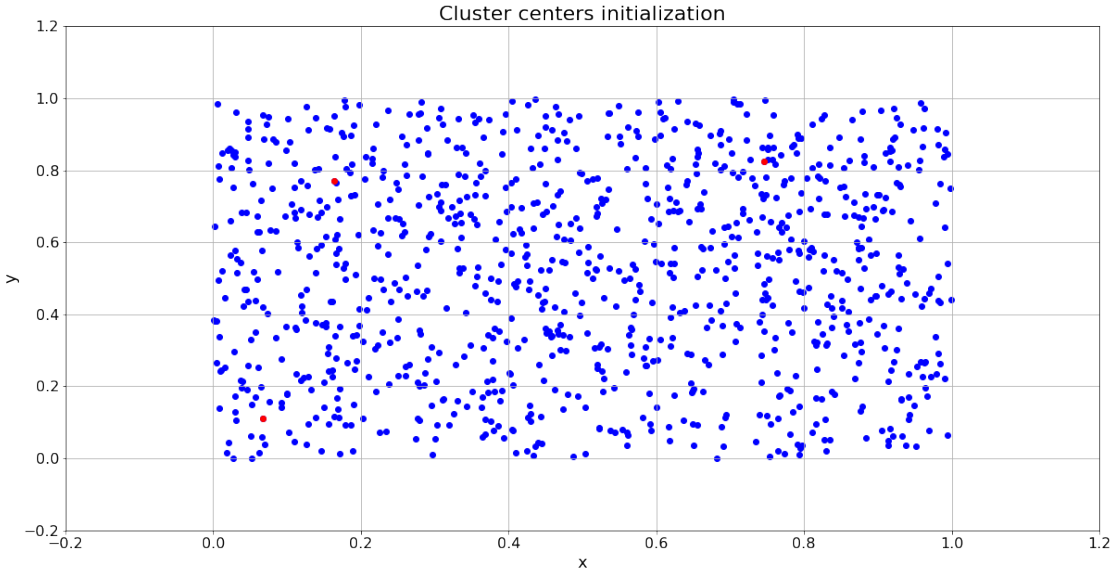
Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Entered by the user

```
#centroids = initialize_centroids(points, k)  # min_max_mid initial
centroids = initialize_centroids_random(points, k)  # random initial

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```

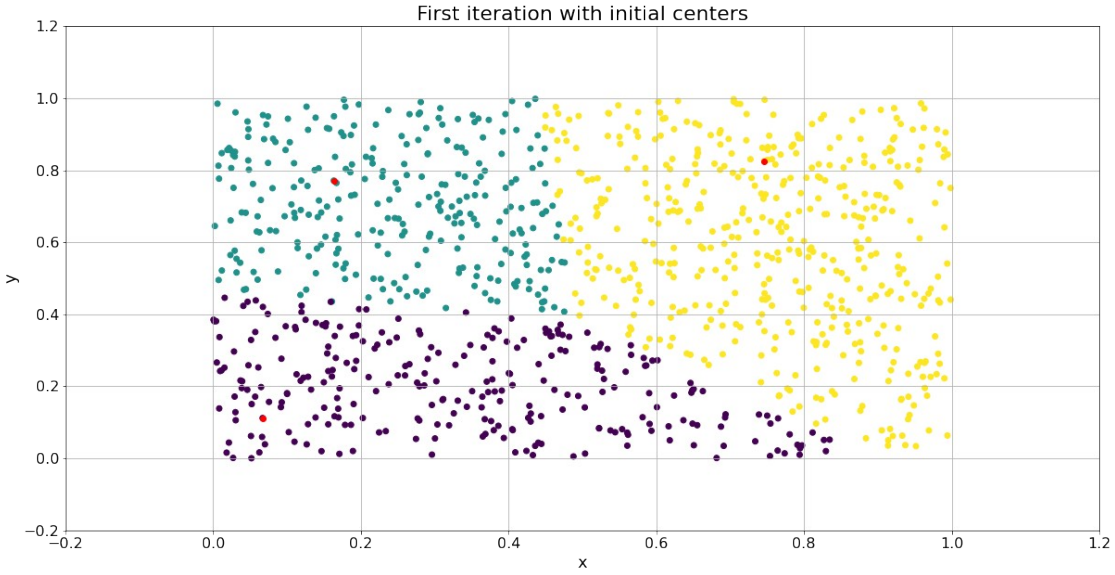



```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)
```

```
[1 1 1 1 0 0 2 1 2 2 2 2 1 0 2 2 2 1 1 0 0 2 1 0 2 1 0 2 0 2 2 2 2 1 2
0 0
 0 2 2 0 0 2 1 2 1 1 1 2 1 2 1 1 2 0 0 0 0 0 1 0 0 0 1 0 1 2 2 2 1 1 2
2 2
 0 1 2 2 2 0 2 2 0 2 1 1 2 2 1 2 1 0 1 2 1 0 0 1 2 2 0 2 2 2 2 0 2 2 2
2 2
 1 0 2 2 0 1 0 1 1 2 0 0 1 1 2 0 2 2 0 0 2 2 2 0 2 0 2 2 2 2 2 2 1 0 2
1 2
 1 2 2 2 2 2 2 0 0 1 2 1 2 2 2 2 1 0 1 2 1 2 1 2 2 1 2 2 2 1 2 1 0 2 0
1 1
 0 1 2 2 0 0 2 2 2 2 2 0 0 1 2 2 0 2 2 0 1 2 1 2 1 0 0 0 2 2 2 2 2 1 1
0 0
 2 2 2 2 0 1 1 0 2 1 2 0 2 2 1 1 2 2 2 2 1 0 0 0 1 2 2 1 1 2 1 0 1 1 2
2 2
 2 0 0 0 1 0 2 0 2 2 0 0 0 2 1 1 0 2 1 2 2 1 1 0 1 1 2 1 0 0 0 0 2 2 0
```

```
0 2
 0 1 0 1 0 2 2 1 2 2 2 1 1 2 0 0 2 2 2 1 0 1 2 0 2 0 1 0 2 1 0 0 0 2 2
1 2
 1 0 2 0 2 1 2 2 2 0 0 0 2 0 0 1 2 0 2 1 2 0 1 2 0 1 2 2 2 0 2 2 0 2 2
2 2
 1 2 2 0 2 0 0 0 0 0 2 2 0 2 2 2 2 2 2 2 1 1 1 1 0 0 0 2 2 1 1 1 0 1 2
1 2
 2 1 2 1 1 0 2 1 0 1 1 0 2 1 1 1 2 0 2 2 2 2 0 0 0 0 1 1 0 0 1 1 1 0 2
2 1
 2 2 2 2 0 1 2 1 0 1 2 1 0 2 1 2 1 0 0 2 2 1 2 1 2 0 0 1 1 1 2 1 1 2 2
1 2
 0 0 1 2 1 0 1 2 2 2 0 2 2 0 1 2 2 1 0 0 2 2 1 0 2 2 1 1 2 1 1 1 0 2 2
0 1
 0 1 2 0 1 2 0 2 1 2 2 2 1 0 2 0 1 1 2 1 2 2 0 1 2 0 2 0 2 2 2 2 2 0 1
2 0
 0 1 2 0 2 0 2 2 1 0 1 1 2 1 1 1 0 2 2 1 1 0 0 1 2 2 1 2 1 0 0 2 2 1 2
2 2
 2 0 1 2 0 0 2 2 2 2 0 0 2 2 2 2 2 1 0 2 2 2 2 2 1 2 0 2 1 1 2 2 1 1 0
2 2
 0 2 1 0 0 1 2 2 2 2 2 1 2 0 2 2 0 2 2 0 2 1 2 2 2 2 1 1 0 0 2 0 0 1 2
1 1
 1 2 0 1 0 2 1 2 0 0 0 2 1 2 1 1 2 2 1 0 2 2 1 2 1 2 2 2 2 2 2 1 0 1 2
2 2
 1 0 2 2 1 1 1 1 2 2 0 1 2 2 0 2 2 2 0 0 2 1 2 1 2 0 1 0 2 1 2 2 1 2 2
2 1
 1 0 1 2 2 1 1 2 0 1 1 0 0 2 2 2 2 1 2 0 2 2 0 1 1 2 2 2 2 2 1 1 2 2 2
2 1
 2 0 0 0 0 0 0 1 1 2 2 1 0 2 1 1 1 1 2 2 0 2 2 0 2 2 2 0 0 2 2 0 2 2 1
2 2
 2 0 2 0 1 2 0 2 0 1 0 2 1 2 2 2 0 1 0 1 0 0 1 0 2 2 2 2 1 2 2 1 0 0 1
0 0
 1 0 2 1 2 2 2 1 0 2 0 0 2 2 0 0 0 0 2 1 1 2 2 2 0 2 1 0 0 2 1 2 1 2 1
2 0
 2 0 0 1 1 2 1 0 2 1 1 2 2 1 0 2 0 0 1 1 2 1 1 0 1 0 2 1 2 2 2 2 2 2 0
0 2
 2 0 1 0 1 2 1 2 2 0 1 1 0 0 0 2 0 1 2 0 2 0 2 2 0 0 2 2 2 2 1 2 2 0 0
2 0
 1 2 2 2 0 1 2 2 1 2 2 2 2 0 2 2 2 2 0 2 1 1 2 0 0 2 1 0 1 2 1 0 1 1 0
2 2
 1]

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
```

```
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

```
[[0.33702618 0.19916604]
 [0.24423211 0.7041486 ]
 [0.75725122 0.59552386]]
[[0.06697835 0.11199254]
 [0.16340801 0.77111744]
 [0.74633219 0.82522638]]
```

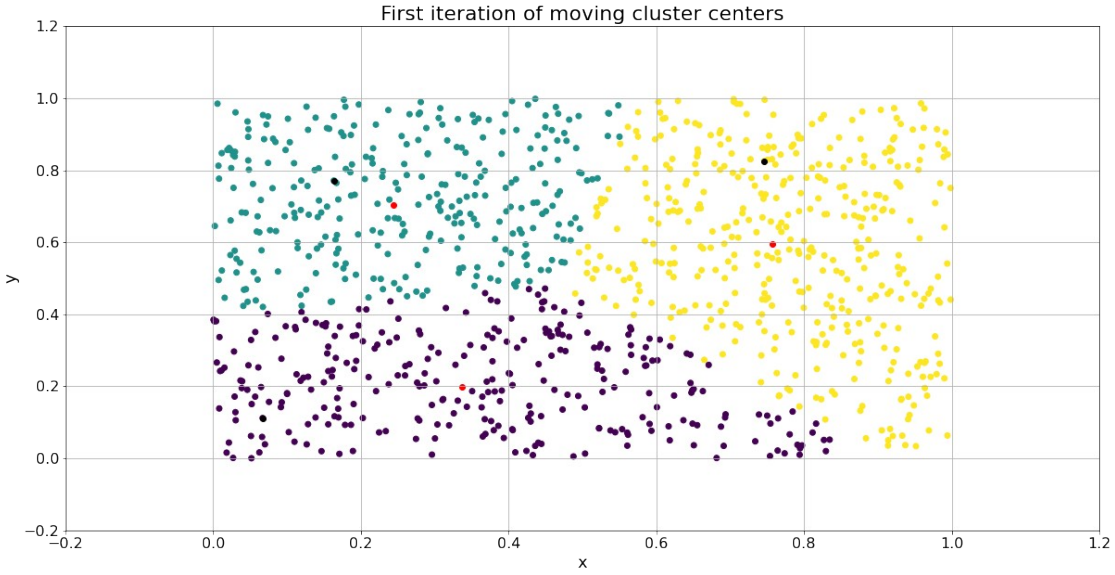
```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[0 1 1 1 0 0 2 1 2 2 2 2 1 0 2 0 0 1 1 0 0 2 1 0 1 1 0 2 0 2 2 2 2 1 2
0 0
 0 2 2 0 0 2 1 2 1 1 1 2 1 2 1 1 2 1 0 0 0 0 1 0 0 0 1 0 1 2 2 2 1 1 2
2 2
 0 1 2 2 2 0 2 0 0 2 1 1 2 2 1 2 1 0 1 2 1 0 0 1 2 2 0 2 2 2 2 0 2 2 1
1 1
 1 0 2 2 0 0 0 1 1 2 0 0 1 1 2 0 2 0 0 0 2 2 2 0 2 0 1 2 2 2 2 2 1 0 2
1 2
 1 2 2 2 2 2 2 0 0 1 2 1 2 2 2 2 1 0 1 2 1 2 1 2 2 1 2 2 2 1 1 1 0 2 0
0 1
 0 1 2 0 0 0 2 2 2 2 2 0 0 1 2 2 0 1 2 0 1 2 1 2 1 0 0 0 2 2 2 2 2 1 1
0 0
 2 2 2 2 0 1 1 0 2 1 2 0 2 2 1 1 2 2 2 2 1 0 0 0 1 2 2 1 1 2 1 0 1 1 2
2 2
 2 0 0 0 1 0 2 0 2 2 0 0 0 2 1 1 0 2 1 2 2 1 1 0 1 1 2 1 1 0 0 0 2 2 0
0 2
```

```
 0 1 0 1 0 2 2 1 2 2 2 1 1 2 0 0 2 2 2 1 0 1 2 0 2 0 1 0 2 1 0 0 0 2 2
1 2
 1 0 2 0 2 1 2 2 2 0 0 0 2 0 0 1 2 0 2 1 2 0 1 2 0 1 2 2 1 0 2 1 1 2 2
2 2
 1 2 1 0 2 0 0 0 0 0 2 2 0 2 1 0 2 2 2 1 1 1 1 1 0 0 0 2 2 1 1 1 0 1 1
1 2
 2 1 2 1 1 0 2 1 0 1 1 0 2 1 1 1 2 0 2 2 2 2 0 0 0 0 1 1 0 0 1 1 1 0 2
2 1
 2 2 2 2 0 0 1 0 0 1 2 1 0 2 1 2 1 0 1 2 2 1 2 1 2 0 0 1 1 1 2 1 1 2 2
1 2
 0 0 1 2 1 0 1 2 2 2 0 2 2 0 0 0 2 1 0 0 2 2 0 0 2 2 1 1 2 1 1 1 0 2 2
0 1
 0 1 2 0 1 2 0 2 1 2 2 2 1 0 1 0 1 1 2 1 2 2 0 0 2 0 1 0 2 2 2 2 2 0 1
2 0
 0 1 2 0 2 0 2 2 1 0 1 1 2 1 1 1 0 2 2 1 1 0 0 1 2 2 1 2 1 0 0 2 2 1 2
2 2
 2 0 1 2 0 0 2 2 2 2 0 0 2 2 2 2 2 1 0 2 2 2 2 2 1 2 0 2 1 1 2 2 1 1 0
2 2
 0 2 1 0 0 0 2 2 1 2 2 1 2 0 0 2 0 2 2 0 2 1 2 2 2 2 1 1 0 0 2 0 0 1 2
1 1
 1 2 0 1 0 2 1 2 0 0 0 2 1 2 1 1 2 2 1 0 2 0 1 2 1 2 2 2 2 2 2 1 0 1 2
1 2
 1 0 2 2 1 1 1 1 2 2 0 1 2 2 0 2 1 2 0 0 2 1 2 1 0 0 1 0 1 1 2 2 1 1 2
2 1
 1 0 1 2 2 1 1 2 0 1 1 0 0 2 2 2 2 0 2 0 2 2 0 1 1 2 2 2 2 2 1 1 2 2 2
2 0
 2 0 0 0 0 0 0 1 1 2 1 1 0 2 1 1 1 1 2 2 0 2 2 0 2 2 2 0 0 2 2 0 0 1 1
2 2
 2 0 2 0 1 2 0 2 0 1 0 2 1 2 2 2 0 1 0 1 0 0 1 0 2 2 2 2 1 2 2 1 1 0 1
0 0
 1 0 2 1 2 2 2 1 0 2 0 0 1 2 0 0 0 0 0 1 1 2 2 2 0 2 1 0 0 2 1 2 1 2 1
2 0
 2 0 0 1 1 2 1 0 2 1 1 2 2 1 0 2 0 1 1 1 1 1 1 0 1 0 2 1 1 2 2 2 2 0 0
0 2
 2 0 0 0 1 2 1 2 2 0 1 1 0 0 0 2 0 1 2 0 2 0 1 2 0 0 2 2 2 2 1 2 2 0 0
1 0
 1 2 2 2 0 1 2 2 1 2 0 2 2 0 1 2 2 2 1 2 1 0 2 0 0 2 1 0 1 2 1 0 1 1 0
2 2
 1]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
```

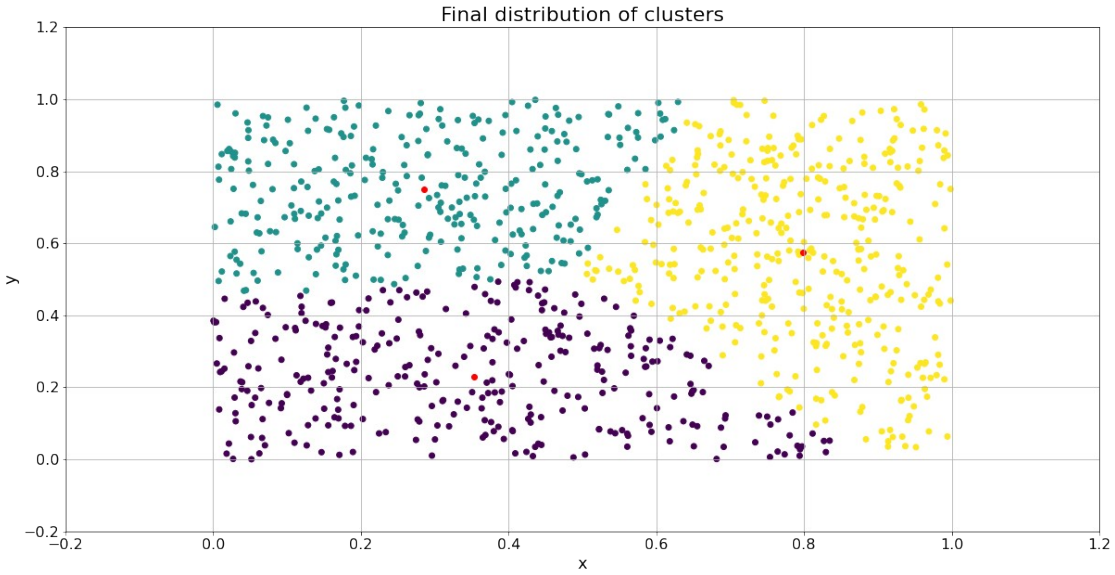
```
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)
```

```
Iteration number =  1 False
Iteration number =  2 False
Iteration number =  3 False
Iteration number =  4 False
Iteration number =  5 False
Iteration number =  6 False
Iteration number =  7 False
Iteration number =  8 False
Iteration number =  9 False
Iteration number =  10 False
Iteration number =  11 False
Iteration number =  12 False
Iteration number =  13 True
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

При большом числе объектов и кластеров результат сложно интерпретировать. Центры кластеров лежат вне сгруппированных объектов на значительном отдалении.

2.3. Малое количество объектов

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
n_points = 100 # Data quantity
centroids_residual = 0.001 # Residual for checking centroids
elbow_residual = 0.001 # Residual for checking centroids
```

Default parameters

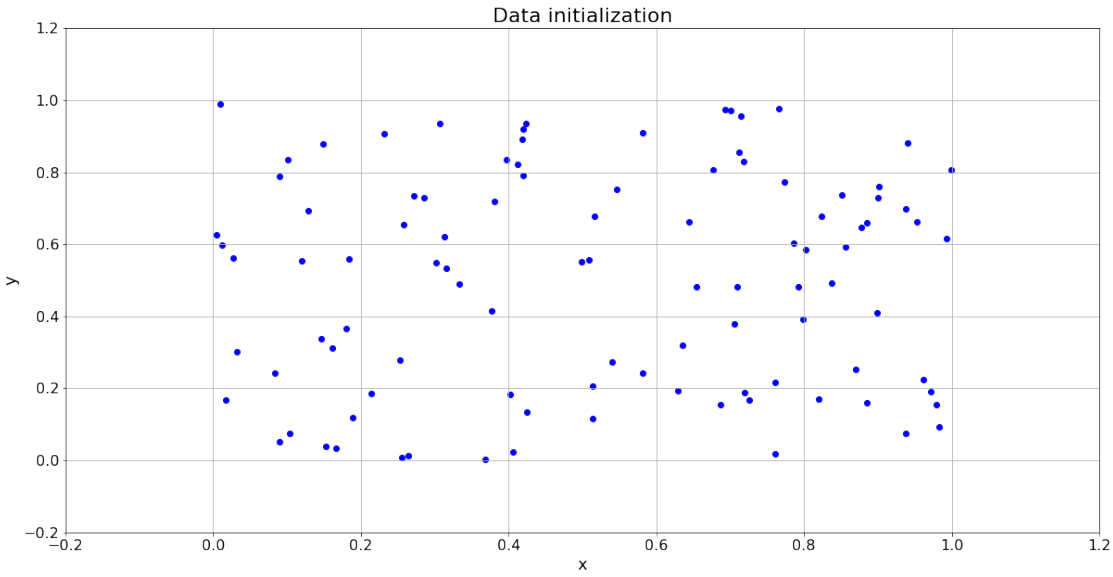
```
k = 1  # Clusters number (centers number)
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_randow(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Метод Elbow - метод локтя. Вычисляется сумма квадратов расстояний от каждой точки до центра кластера, которому он принадлежит. Далее

возвращется средне арифметическое, т.е. общая сумма делится на число кластеров k.

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} \left| p - x_i \right|^2$$

```
def elbow_function(points, centroids, closest_points, k):
    result = 0
    for i in range(k):
        for j in range(len(closest_points)):
            if i == closest_points[j]:
                result += np.sqrt(((points[j][0] - centroids[i][0]) **
2) + ((points[j][1] - centroids[i][1]) ** 2)) ** 2
    return (result / k)
```

Метод Elbow - метод "локтя". С помощью данного методы вычисляется оптимальное количество кластеров. На каждой итерации значение суммы квадратов расстояний до каждого кластера будет уменьшаться, то есть в какой-то момент значение функции будет слабо изменяться, что означает достижении оптимального числа кластеров. В алгоритм ниже добавлена невязка для останова.

```
elbow_dist = np.array([])
k_final = 0
prev = 100
print('Iteration number (clusters number) = ', k)
#centroids = initialize_centroids(points, k)
centroids = initialize_centroids_randow(points, k)
closest_points = closest_centroids(points, centroids, n_points, k)
dist = elbow_function(points, centroids, closest_points, k)
elbow_dist = np.append(elbow_dist, dist)
prev = np.copy(dist)
k += 1
#print(elbow_dist)
#K = range(2, 100, 1)
while True:
    print('Iteration number (clusters number) = ', k)
    #centroids = initialize_centroids(points, k)
    centroids = initialize_centroids_randow(points, k)
    closest_points = closest_centroids(points, centroids, n_points, k)
    dist = elbow_function(points, centroids, closest_points, k)
    #elbow_dist = np.append(elbow_dist, dist)
    if np.absolute(dist - prev) < elbow_residual:
        k_final = k
        elbow_dist = np.append(elbow_dist, dist)
        break
    else:
        prev = np.copy(dist)
        elbow_dist = np.append(elbow_dist, dist)
        k += 1
```

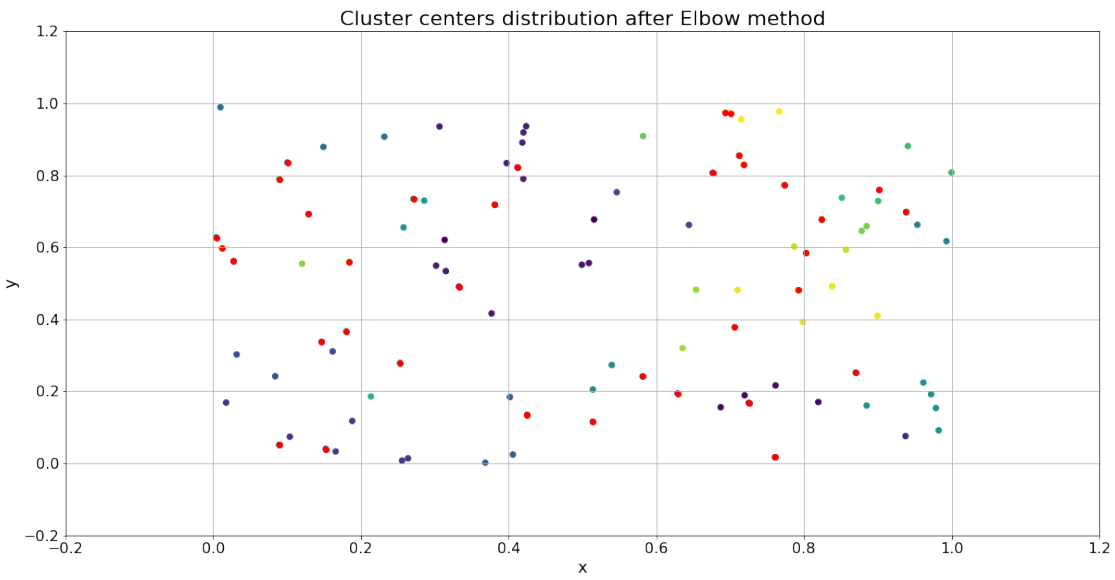
```
print('Elbow method function')
print(elbow_dist)
print('Clusters optimum number = ', k_final)
```

```
Iteration number (clusters number) =  1
Iteration number (clusters number) =  2
Iteration number (clusters number) =  3
Iteration number (clusters number) =  4
Iteration number (clusters number) =  5
Iteration number (clusters number) =  6
Iteration number (clusters number) =  7
Iteration number (clusters number) =  8
Iteration number (clusters number) =  9
Iteration number (clusters number) =  10
Iteration number (clusters number) =  11
Iteration number (clusters number) =  12
Iteration number (clusters number) =  13
Iteration number (clusters number) =  14
Iteration number (clusters number) =  15
Iteration number (clusters number) =  16
Iteration number (clusters number) =  17
Iteration number (clusters number) =  18
Iteration number (clusters number) =  19
Iteration number (clusters number) =  20
Iteration number (clusters number) =  21
Iteration number (clusters number) =  22
Iteration number (clusters number) =  23
Iteration number (clusters number) =  24
Iteration number (clusters number) =  25
Iteration number (clusters number) =  26
Iteration number (clusters number) =  27
Iteration number (clusters number) =  28
Iteration number (clusters number) =  29
Iteration number (clusters number) =  30
Iteration number (clusters number) =  31
Iteration number (clusters number) =  32
Iteration number (clusters number) =  33
Iteration number (clusters number) =  34
Iteration number (clusters number) =  35
Elbow method function
[2.68620366e+01 6.44952793e+00 4.95402416e+00 1.70032783e+00
 1.90483752e+00 1.04255331e+00 4.81767039e-01 4.21621901e-01
 5.74264677e-01 2.41146714e-01 2.97476031e-01 6.15128404e-01
 3.18837581e-01 1.22635897e-01 1.52404017e-01 1.00566981e-01
 1.29549119e-01 8.38997017e-02 1.06530784e-01 8.24526388e-02
 5.72667703e-02 5.52854129e-02 4.15564979e-02 5.29565829e-02
 5.06839968e-02 4.86750472e-02 4.20173706e-02 3.24037174e-02
 2.80984207e-02 2.47504913e-02 3.01407235e-02 2.29658563e-02
```

```
 2.44908129e-02 1.95135379e-02 1.96315823e-02]
Clusters optimum number =  35
```

```
K = range(1, k_final + 1)
K = np.array(K)
#print(K)
#print(K[len(K) - 1])

plt.figure(figsize = (20,10))
plt.title('Elbow method (choosing the value of k)', fontsize = 22)
plt.xlabel('k (number of clusters)', fontsize = 18)
plt.ylabel('Elbow function SSE', fontsize = 18)
plt.axes(xlim=(0, K[len(K) - 1] + 1), ylim=(0, 1000))
plt.grid(True)
plt.plot(K, elbow_dist, c = 'blue')
plt.xticks(np.linspace(0, 500, 21), fontsize = 16)
plt.yticks(np.linspace(0, 1000, 21), fontsize = 16);
#plt.xticks([int(5*i) for i in range(1, 10)])
#plt.yticks([int(50*i) for i in range(1, 20)])
#plt.yticks([0, 10, 20, 30, 40, 50, 60, 70, 80])
```


```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
#plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points,
k_final)
print(new_centroids)
print(centroids)
```

```
[[0.40357267 0.67190549]
 [0.74266209 0.18000798]
 [0.38917484 0.5162395 ]
 [0.39982572 0.87507686]
 [0.84915665 0.04646993]
 [0.07034275 0.09819655]
 [0.20527933 0.04265077]
 [0.62240503 0.7404448 ]
 [0.10615284 0.29821125]
 [0.6288205  0.19432837]
 [0.40035027 0.0865651 ]
 [0.02772675 0.56182506]
 [0.12287543 0.90230516]
 [0.0039688  0.62735935]
 [0.12905529 0.69208823]
 [0.96162358 0.65915393]
 [0.5451554  0.24005708]
 [0.94165117 0.17916643]
 [0.27160868 0.70627558]
 [0.51397008 0.11613331]
 [0.18012119 0.36669969]
 [0.71202432 0.85463352]
 [0.23325362 0.23253638]
 [0.91884571 0.78273275]
 [0.08949936 0.78884309]
 [0.77331269 0.77311334]
 [0.63781437 0.94088687]
 [0.86226793 0.6608472 ]
```

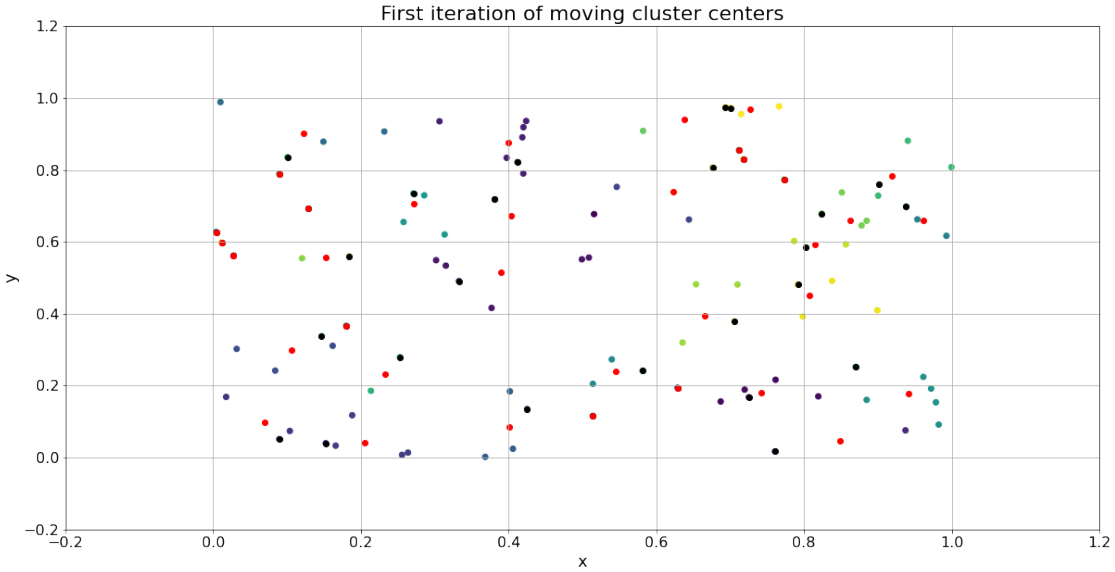
```
 [0.15234104 0.55694663]
 [0.66518033 0.39352313]
 [0.71820971 0.82946412]
 [0.81550643 0.59304909]
 [0.01152484 0.59781368]
 [0.80756636 0.45125008]
 [0.7272295  0.9681704 ]]
[[0.38142977 0.71806381]
 [0.72538091 0.16794366]
 [0.33286564 0.49059517]
 [0.41213612 0.82154638]
 [0.76079163 0.01710851]
 [0.08956883 0.05150041]
 [0.15243849 0.04018891]
 [0.67636782 0.80656433]
 [0.14691362 0.33755098]
 [0.6288205  0.19432837]
 [0.42514704 0.13509027]
 [0.02772675 0.56182506]
 [0.10053601 0.83524022]
 [0.0039688  0.62735935]
 [0.12905529 0.69208823]
 [0.93838069 0.69816977]
 [0.5815109  0.24176512]
 [0.86999557 0.25219324]
 [0.27117487 0.73430525]
 [0.51397008 0.11613331]
 [0.18012119 0.36669969]
 [0.71202432 0.85463352]
 [0.25286814 0.27928558]
 [0.90197477 0.75920601]
 [0.08949936 0.78884309]
 [0.77331269 0.77311334]
 [0.69342949 0.9732361 ]
 [0.82382272 0.67846849]
 [0.18425616 0.55956573]
 [0.70596832 0.37843545]
 [0.71820971 0.82946412]
 [0.80299683 0.58385562]
 [0.01152484 0.59781368]
 [0.79181948 0.48115254]
 [0.70034903 0.9723696 ]]
```

```
closest_points = closest_centroids(points, new_centroids, n_points,
k_final)
print(closest_points)
```

```
[12 27  3  6 28 31 27 18 30 12  3 23 25  2 11 20 34  8 21 10 29  2 29
31
 15  5  0 32 17  6 29 34  6  1  3 22 13 16  4 16 33 23 16  4 19  0  9
29
```

```
 17  3  2  1  5 17 22 23 12 18  8 27 34  1  1 28  8  3  6 18 33  6 17
5
  2  3  3 24 10  2 15 20  2 10  1  7 27 31 15 33 33 26  7 34 24 23 14
30
 10 18 17 17]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



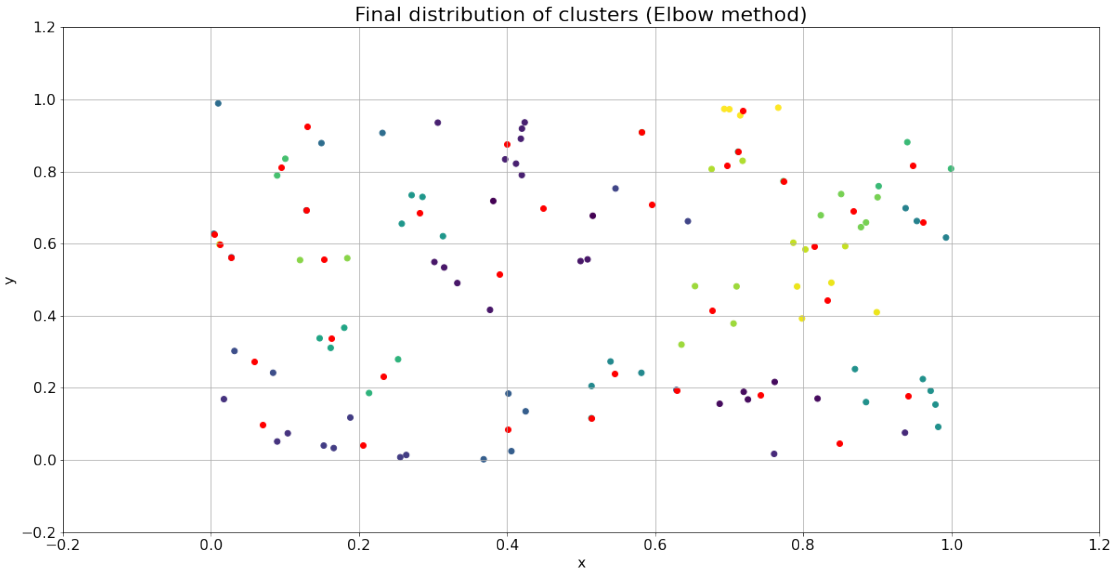

```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k_final)
    closest_points = closest_centroids(points, new_centroids,
n_points, k_final)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
```

```
    print(exit_condition)
print('Final centroids')
print(new_centroids)
print('Closest_points')
print(closest_points)
```

```
Iteration number =  1 False
Iteration number =  2 False
Iteration number =  3 False
Iteration number =  4 True
Final centroids
[[0.44862816 0.69759043]
 [0.74266209 0.18000798]
 [0.38917484 0.5162395 ]
 [0.39982572 0.87507686]
 [0.84915665 0.04646993]
 [0.07034275 0.09819655]
 [0.20527933 0.04265077]
 [0.59542363 0.70738504]
 [0.05793953 0.27216029]
 [0.6288205  0.19432837]
 [0.40035027 0.0865651 ]
 [0.02772675 0.56182506]
 [0.1303219  0.92466014]
 [0.0039688  0.62735935]
 [0.12905529 0.69208823]
 [0.96162358 0.65915393]
 [0.5451554  0.24005708]
 [0.94165117 0.17916643]
 [0.28207194 0.68484059]
 [0.51397008 0.11613331]
 [0.16295117 0.33840803]
 [0.71202432 0.85463352]
 [0.23325362 0.23253638]
 [0.94745916 0.81599531]
 [0.09501769 0.81204166]
 [0.77331269 0.77311334]
 [0.58219925 0.90853765]
 [0.86773097 0.68964389]
 [0.15234104 0.55694663]
 [0.67639868 0.41548904]
 [0.69728877 0.81801423]
 [0.81550643 0.59304909]
 [0.01152484 0.59781368]
 [0.83194451 0.4437159 ]
 [0.7187795  0.96943682]]
Closest_points
[12 27  3  6 28 31 27 18 30 12  3 23 25  2 11 20 34 20 21 10 29  2 29
31
 15  5  0 32 17  6 29 34  6  1  3 22 13 16  4 16 33 23 16  4 19  0  9
```

```
29
 17  3  2  1  5 17 22 27 12 18  8 27 34  1  1 28  8  3  6 18 33  6 17
5
  2  3  3 24 10  2 15 20  2 10  1  7 27 31 15 33 33 26  7 34 24 23 14
30
 10 18 17 17]
```

```
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Final distribution of clusters (Elbow method)', fontsize =
22)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Количество оптимальных кластеров сравнимо с заданным количество объектов и не всегда может привести к решению, если задана высокая точность определения функции "локтя". В данном примере при 10 объектах и заданной точности не получилось получить количество кластеров меньше числа объектов. Для решения задачи количество объектов было увеличено до 100 и получено 35 кластеров.

4. Исследовать зависимость (скорости настройки) от объёма данных / сложности задачи

На скорость сходимости решения влияет оптимальный код, количество кластеров, количество объектов, точность нахождения центров кластеров.

4.1. Случайная инициализация

Задаётся 10000 объектов, 30 кластеров (как в разделе 1) и устаналивается значение точности нахождения центров кластеров 0.1

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
#Малое число объёктов и кластеров
k = 30  # Clusters number (centers number)
n_points = 10000 # Data quantity
centroids_residual = 0.1 # Residual for checking centroids
#centroids = initialize_centroids(points, k)  # min_max_mid initial
#centroids = initialize_centroids_random(points, k)  # random initial
```

Default parameters

```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

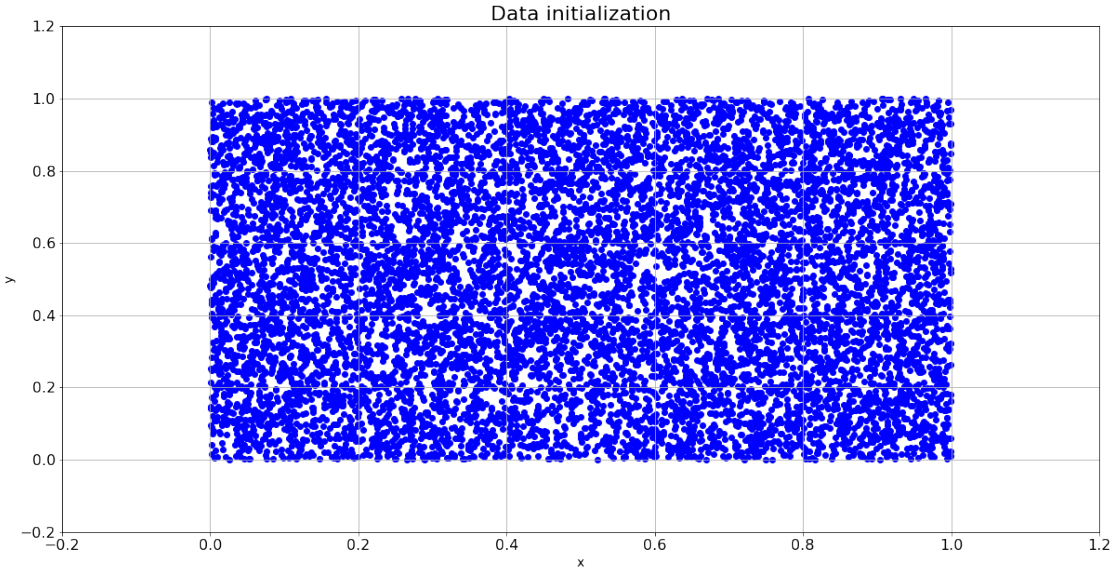
Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]
```

```
#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
```

```
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
```

```
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

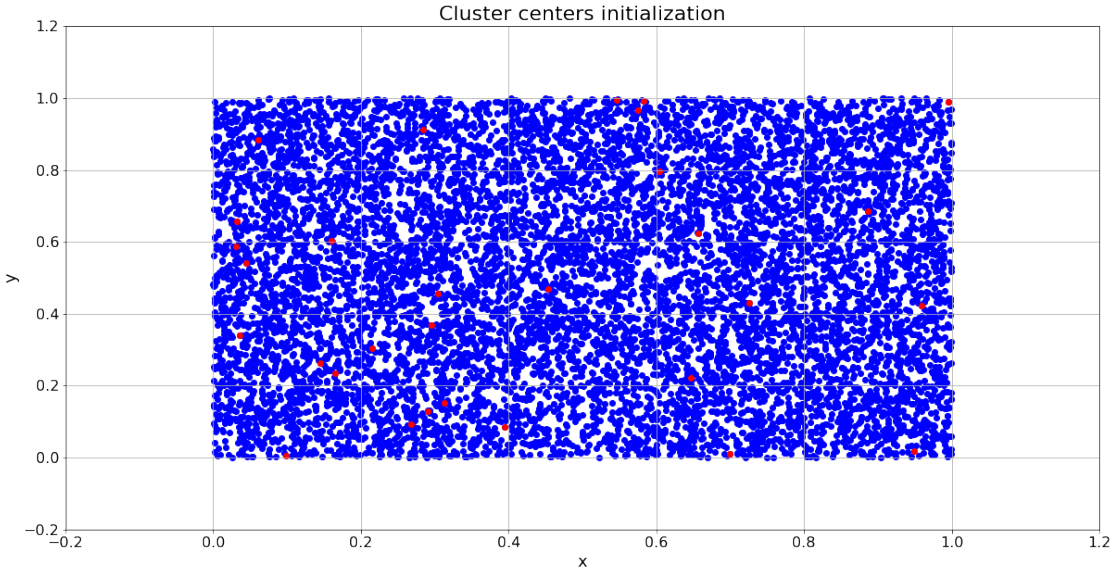
$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
```

```
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Entered by the user

```
#centroids = initialize_centroids(points, k)  # min_max_mid initial
centroids = initialize_centroids_random(points, k)  # random initial

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```

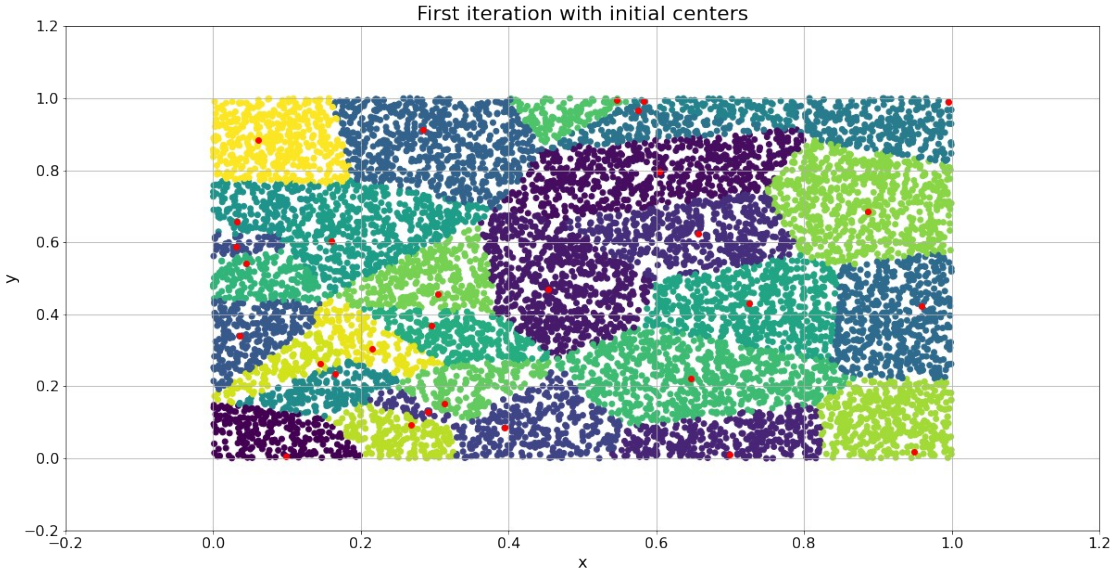



```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)
```

```
[16  4 27 ... 10 26  2]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
```

```
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

```
[[0.08449748 0.0690809 ]
 [0.59877703 0.79516122]
 [0.46646003 0.49077159]
 [0.69888176 0.05922932]
 [0.65068886 0.62384123]
 [0.26740153 0.14140985]
 [0.44201363 0.08991767]
 [0.04644438 0.59274506]
 [0.05563497 0.34516554]
 [0.3012604  0.87207978]
 [0.92103296 0.38883928]
 [0.59335423 0.92010377]
 [0.90497135 0.92165955]
 [0.70261213 0.97081401]
 [0.15330245 0.18966822]
 [0.06317433 0.70365297]
 [0.21656875 0.63614348]
 [0.72960159 0.42194136]
 [0.33203785 0.34001659]
 [0.07188448 0.49460606]
```

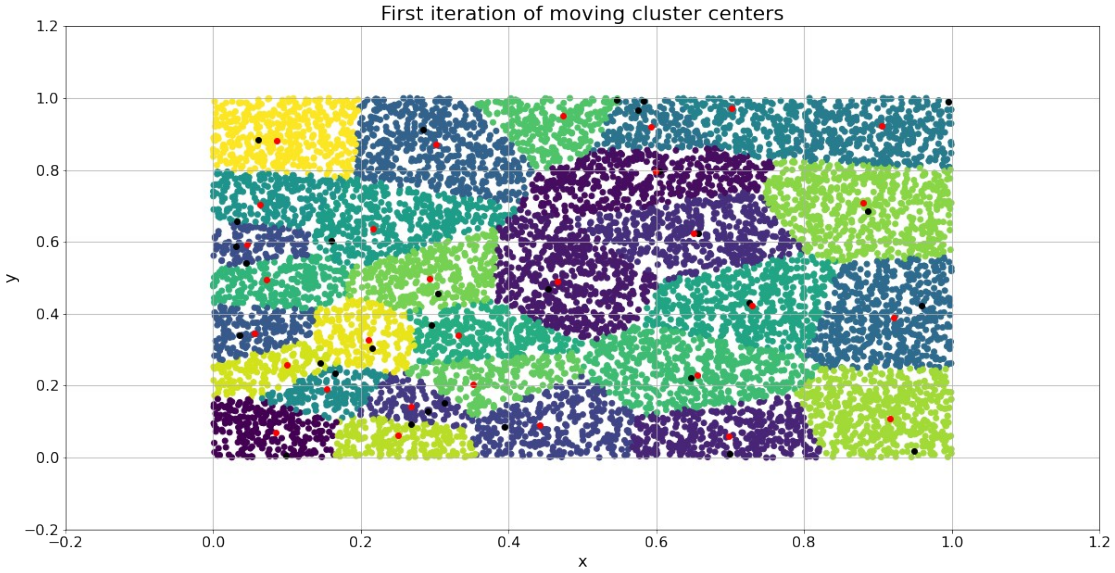
```
 [0.65501363 0.22873103]
 [0.4741053  0.95112594]
 [0.3526281  0.20480288]
 [0.29348197 0.49749057]
 [0.88055706 0.70842922]
 [0.9160814  0.10915754]
 [0.2507099  0.0625357 ]
 [0.09926705 0.25693117]
 [0.21028349 0.32831211]
 [0.08585875 0.88107292]]
[[0.09922328 0.00623681]
 [0.60433529 0.79717906]
 [0.4532025  0.46811322]
 [0.69897387 0.01152609]
 [0.65724948 0.62253848]
 [0.29046862 0.13027778]
 [0.39461957 0.08505484]
 [0.03148113 0.58647384]
 [0.03528252 0.34016373]
 [0.28467476 0.91300849]
 [0.95957593 0.42244683]
 [0.57538839 0.96720426]
 [0.9956949  0.98957332]
 [0.58267884 0.99100794]
 [0.16544791 0.23447257]
 [0.03242756 0.65805957]
 [0.15982827 0.60422079]
 [0.72648545 0.43118778]
 [0.29555354 0.36853143]
 [0.04453953 0.54030292]
 [0.64718394 0.22287589]
 [0.54664493 0.99480592]
 [0.31331297 0.15137105]
 [0.30492667 0.45543125]
 [0.88686581 0.68571983]
 [0.94941536 0.01801777]
 [0.26855182 0.09249419]
 [0.14457794 0.26298339]
 [0.21496215 0.30552403]
 [0.06097786 0.88488977]]
```

```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[16  4 28 ... 10 26  2]
```

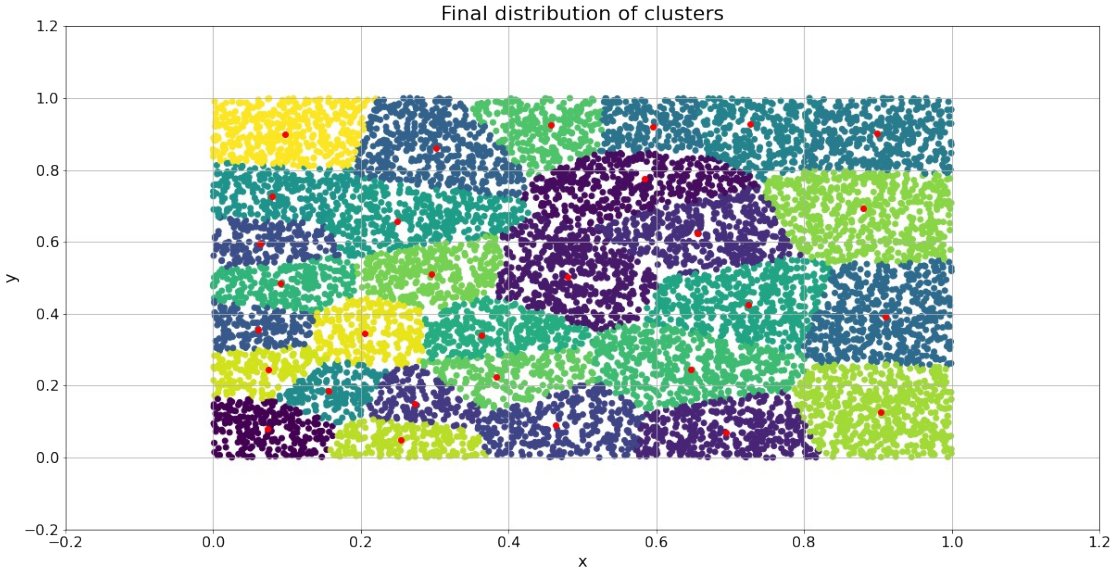
```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
```

```
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)

Iteration number =  1 False
Iteration number =  2 True

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
```

```
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Потребовалось 2 итерации для нахождения центров кластеров при уменьшении точности решения в 100 раз. Это в 28 раз быстрее, чем в разделе 1 получилось при точности 0.001.

4.2. min_max_mid инициализация

Задаётся 10000 объектов, 30 кластеров (как в разделе 1) и устаналивается значение точности нахождения центров кластеров 0.1

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
k = 30  # Clusters number (centers number)
n_points = 10000 # Data quantity
centroids_residual = 0.1 # Residual for checking centroids
```

Default parameters

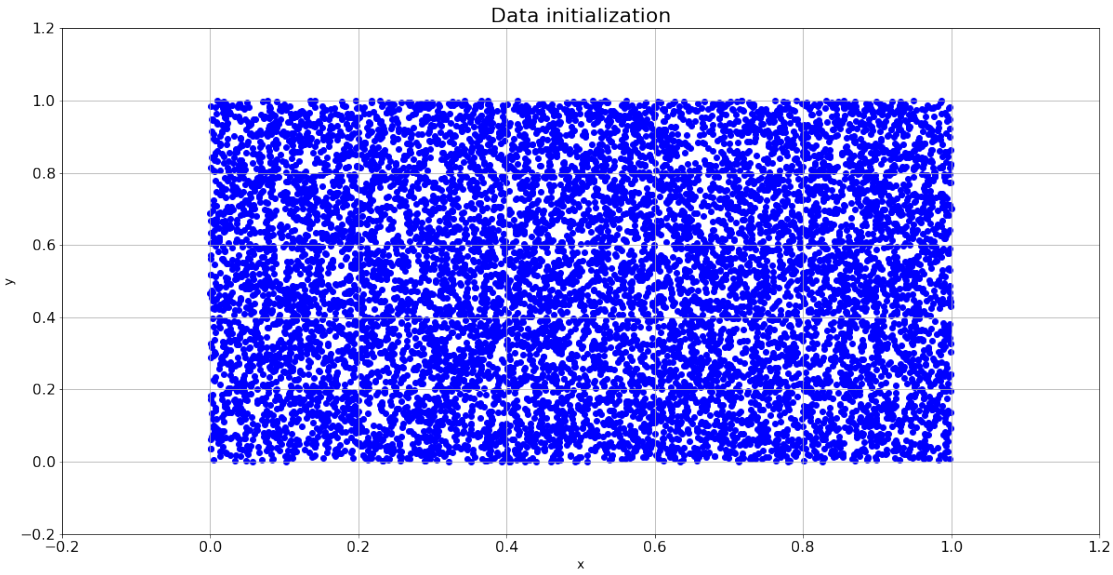
```
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 14)
plt.ylabel('y', fontsize = 14)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_random(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

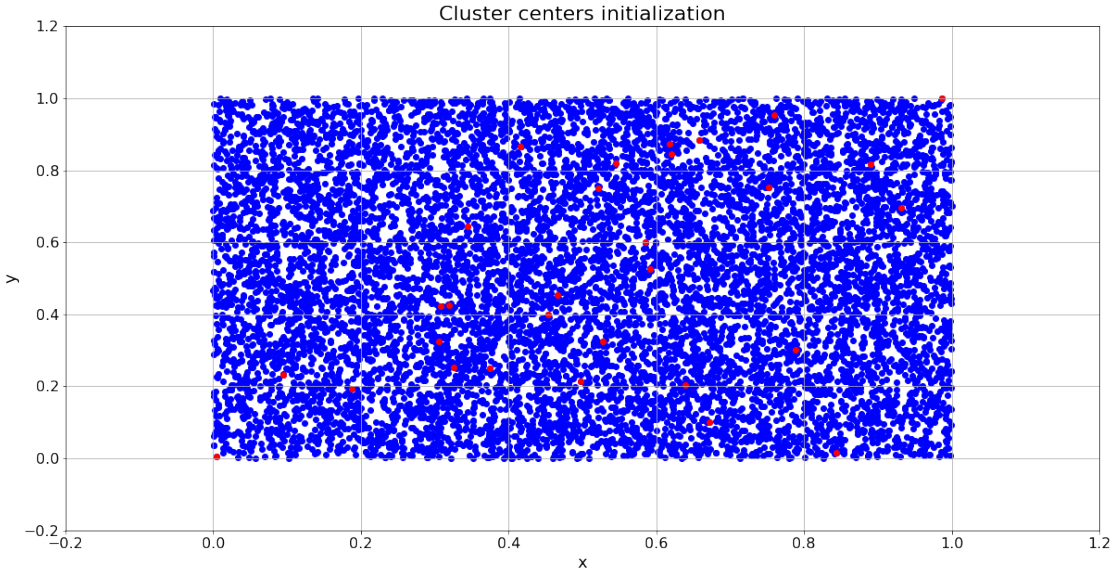
Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

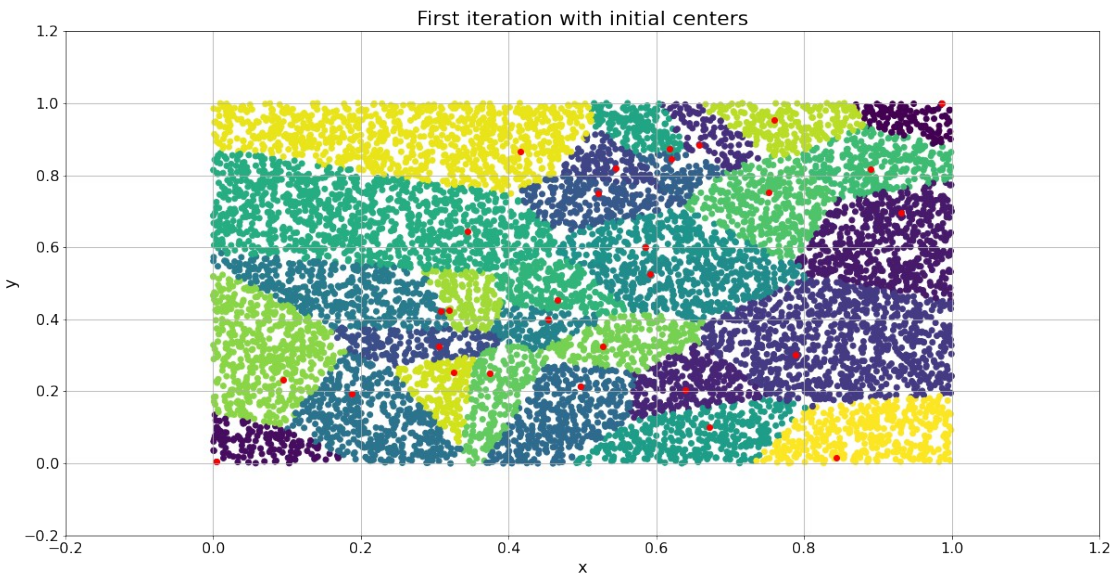
```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Задаётся стратегия начальной инициализации

```
centroids = initialize_centroids(points, k)
#centroids = initialize_centroids_random(points, k)

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
closest_points = closest_centroids(points, centroids, n_points, k)
print(closest_points)

[25 13  2 ... 15  2  7]

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration with initial centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points, k)
print(new_centroids)
print(centroids)
```

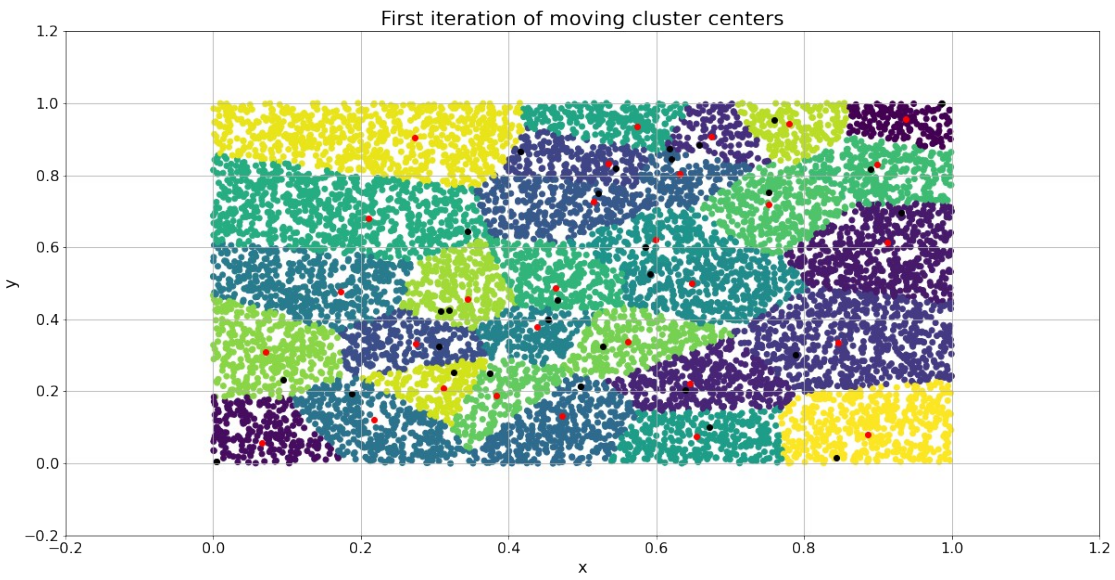
```
[[0.93838833 0.9549191 ]
 [0.06598658 0.05716968]
 [0.91234519 0.6124375 ]
 [0.64594693 0.22168013]
 [0.67466463 0.90754677]
 [0.8458925  0.33505769]
 [0.53492195 0.83104382]
 [0.27373266 0.33289798]
 [0.5151955  0.72745721]
 [0.63128705 0.80443492]
 [0.47306819 0.13136476]
 [0.21748775 0.12195974]
 [0.17322961 0.47724933]
 [0.43907683 0.37815889]
 [0.64865224 0.49984772]
 [0.59894716 0.62056821]
 [0.65374617 0.07619422]
 [0.57443714 0.93496493]
 [0.20995127 0.68142699]
 [0.46311338 0.4868144 ]
 [0.89934373 0.8290301 ]
 [0.75277496 0.71814491]
 [0.38391268 0.18768697]
 [0.56113545 0.33654222]
 [0.07071081 0.30954211]
 [0.34509233 0.45731052]
 [0.7797549  0.9431282 ]
 [0.31220309 0.20792026]
 [0.2730259  0.90390679]
```

```
 [0.8862523  0.08136575]]
[[0.98665891 0.99880395]
 [0.00466005 0.00429538]
 [0.9320292  0.69684984]
 [0.63926241 0.20439039]
 [0.65818657 0.88485884]
 [0.78802813 0.30161308]
 [0.54515333 0.81948526]
 [0.30513599 0.32589537]
 [0.52198332 0.75035804]
 [0.61997539 0.84610986]
 [0.49779102 0.21285325]
 [0.18817314 0.19270069]
 [0.30863604 0.4223796 ]
 [0.45358508 0.40079543]
 [0.59128742 0.52549035]
 [0.58511358 0.60099841]
 [0.67170742 0.09989417]
 [0.61843781 0.87443889]
 [0.34435383 0.64296126]
 [0.46658727 0.45273738]
 [0.88997309 0.81571168]
 [0.75282541 0.7534498 ]
 [0.37505887 0.24949192]
 [0.52784466 0.32362367]
 [0.09436473 0.23124727]
 [0.31935793 0.42536204]
 [0.75958694 0.95325437]
 [0.32540287 0.252493  ]
 [0.41580901 0.86568237]
 [0.84423938 0.01565287]]
```

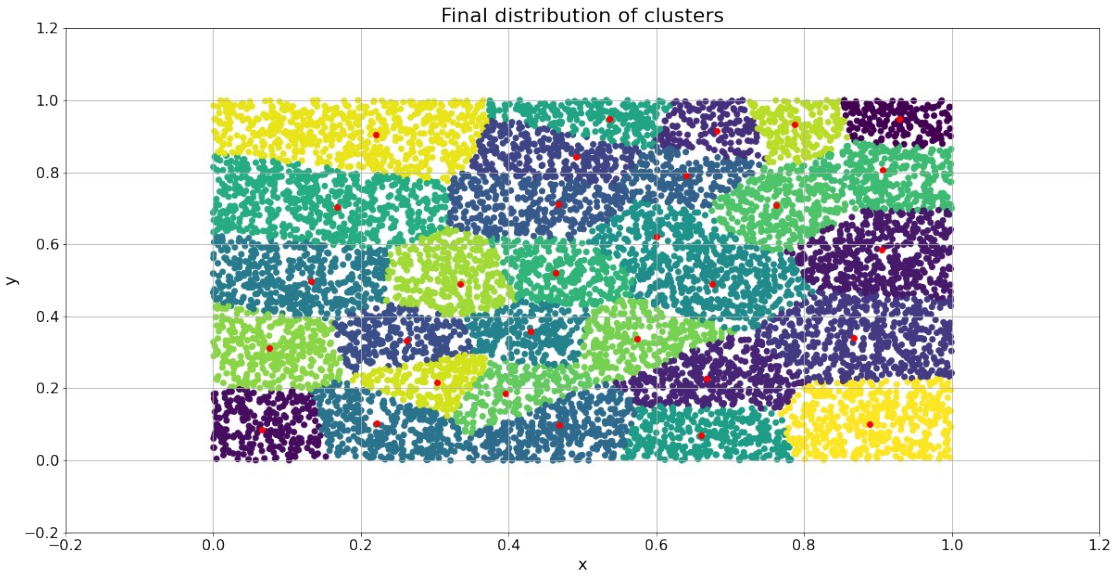
```
closest_points = closest_centroids(points, new_centroids, n_points, k)
print(closest_points)
```

```
[25 13  2 ... 15  2  7]
```

```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k)
    closest_points = closest_centroids(points, new_centroids,
n_points, k)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)

Iteration number =  1 False
Iteration number =  2 True

plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Final distribution of clusters', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Потребовалось 2 итерации для нахождения центров кластеров при уменьшении точности решения в 100 раз. Это в 17 раз быстрее, чем в разделе 1 получилось при точности 0.001.

Также отмечаем совпадение количества итераций с расчётом при случайной инициализации.

4.3. Инициализация методом "локтя"

Задаётся 10000 объектов, 30 кластеров (как в разделе 1) и устаналивается значение точности нахождения центров кластеров 0.1

```
import numpy as np
import matplotlib.pyplot as plt
```

Entered by the user

```
n_points = 10000 # Data quantity
centroids_residual = 0.1 # Residual for checking centroids
elbow_residual = 0.1 # Residual for checking centroids
```

Default parameters

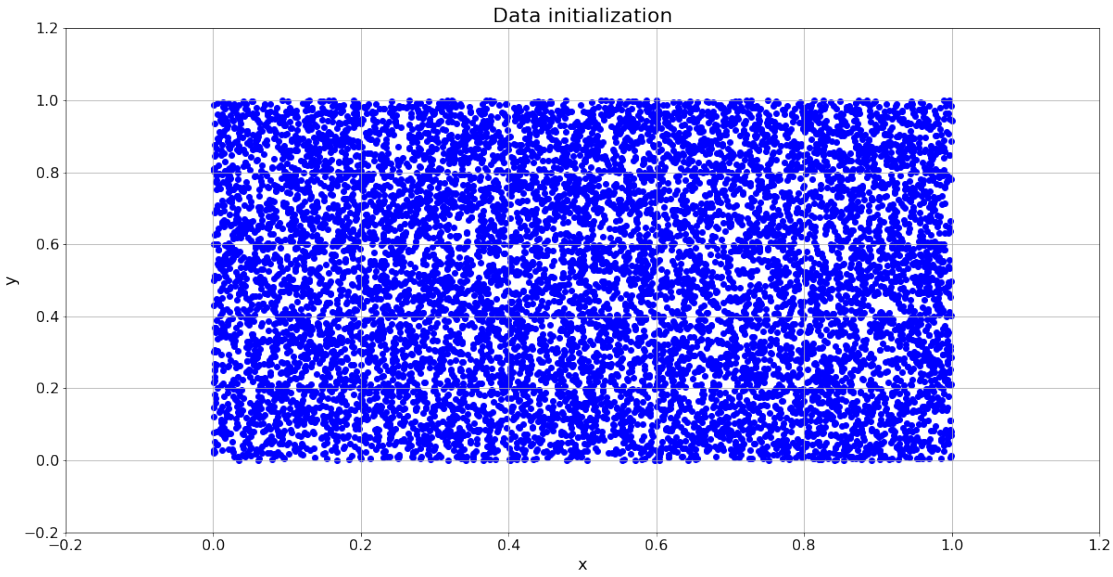
```
k = 1  # Clusters number (centers number)
exit_condition = False

#for plots of clusters distribution
xlim_min = -0.2
xlim_max = 1.2
ylim_min = -0.2
ylim_max = 1.2
```

Data initialization

Вход:

$\{x_1,\ldots,x_m\} \subseteq R^n$

```
points = np.random.rand(n_points,2)
#points = np.array([[1,2], [2,3], [3,3], [4,4], [7,5], [8,6], [9,7],
[8,8], [9,9], [10,10]])
#points = [[int(i) for j in range(2)] for i in range(10)]

#print(points)

#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Data initialization', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = 'blue')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Инициализация k центров кластеров.

$\{\mu_1, \ldots, \mu_k\} \subseteq R^n$

```
def initialize_centroids(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    dist = np.sqrt(np.sum(data**2, axis = 1))
    if k >= 3:
        centroids[0] = data[np.argmax(dist)]
        random_n.append(np.argmax(dist))
        k -= 1
        if np.argmin(dist) not in random_n:
            centroids[1] = data[np.argmin(dist)]
            random_n.append(np.argmin(dist))
            k -= 1
            if (len(data) // 2) not in random_n:
                centroids[2] = data[len(data) // 2]
                random_n.append(len(data) // 2)
                k -= 1
                i = 3
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
```

```
            else:
                while k != 0:
                    n = np.random.randint(0, (len(data) - 1))
                    if n not in random_n:
                        random_n.append(n)
                        centroids[i] = data[n]
                        k -= 1
                        i += 1
                    else:
                        continue
        else:
            while k != 0:
                n = np.random.randint(0, (len(data) - 1))
                if n not in random_n:
                    random_n.append(n)
                    centroids[i] = data[n]
                    k -= 1
                    i += 1
                else:
                    continue

    elif k == 1:
        centroids[0] = data[np.argmax(dist)]
    elif k == 2:
        centroids[0] = data[np.argmax(dist)]
        centroids[1] = data[np.argmin(dist)]
    return centroids

def initialize_centroids_randow(data, k):
    dist = []
    centroids = np.zeros((k, 2))
    random_n = []
    n = 0
    i = 0
    while k != 0:
        n = np.random.randint(0, (len(data) - 1))
        if n not in random_n:
            random_n.append(n)
            centroids[i] = data[n]
            k -= 1
            i += 1
        else:
            continue
    return centroids
```

Нахождение ближайшего центроида к каждой точке (assignment). Каждый объект приписать к тому кластеру, к центру которого он ближе

$$C_t = \{ i \mid k = argmin \| x_i - \mu_t \|^2 \}$$

```
def closest_centroids(points, centroids, n_points, k):
    dist = np.zeros((n_points, k))
    #print(dist)
    for i in range(n_points):
        for j in range(k):
            dist[i][j] = np.sqrt(((points[i][0] - centroids[j][0]) **
2) + ((points[i][1] - centroids[j][1]) ** 2))
    #print(dist)
    return np.argmin(dist, axis = 1)
```

Пересчет центров кластеров (update). Подсчитывается среднеарифметическое значение для каждого центра кластера. При неопределённости вида "деление на ноль" оставляем центр неизменным.

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

```
def move_centroids(points, centroids, closest_points, k):
    new_centroids = np.zeros((k, 2))
    new_centroids = np.array([points[closest_points==i].mean(axis=0)
if i in closest_points else centroids[i] for i in
range(centroids.shape[0])])
    return new_centroids
```

Расчёт невязки. Подсчитывается разность между текущими новыми и предыдущими центроидами. Значение невязки задаётся в программе

$$\mu_{cur} - \mu_{prev} < \mu_{residual}$$

```
def residual_check(new_centroids, centroids, residual):
    result = True
    residual_arr = np.absolute(new_centroids - centroids)
    checking = np.zeros((residual_arr.shape[0],
residual_arr.shape[1]))
    for i in range(residual_arr.shape[0]):
        for j in range(residual_arr.shape[1]):
            if residual_arr[i][j] <= residual:
                checking[i][j] = True
            else:
                checking[i][j] = False
    #print(checking)
    for i in range(checking.shape[0]):
        for j in range(checking.shape[1]):
            if checking[i][j] == False:
                result = False
            else:
                continue
    return result
```

Метод Elbow - метод локтя. Вычисляется сумма квадратов расстояний от каждой точки до центра кластера, которому он принадлежит. Далее

возвращется средне арифметическое, т.е. общая сумма делится на число кластеров k.

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} \left| p - x_i \right|^2$$

```
def elbow_function(points, centroids, closest_points, k):
    result = 0
    for i in range(k):
        for j in range(len(closest_points)):
            if i == closest_points[j]:
                result += np.sqrt(((points[j][0] - centroids[i][0]) **
2) + ((points[j][1] - centroids[i][1]) ** 2)) ** 2
    return (result / k)
```

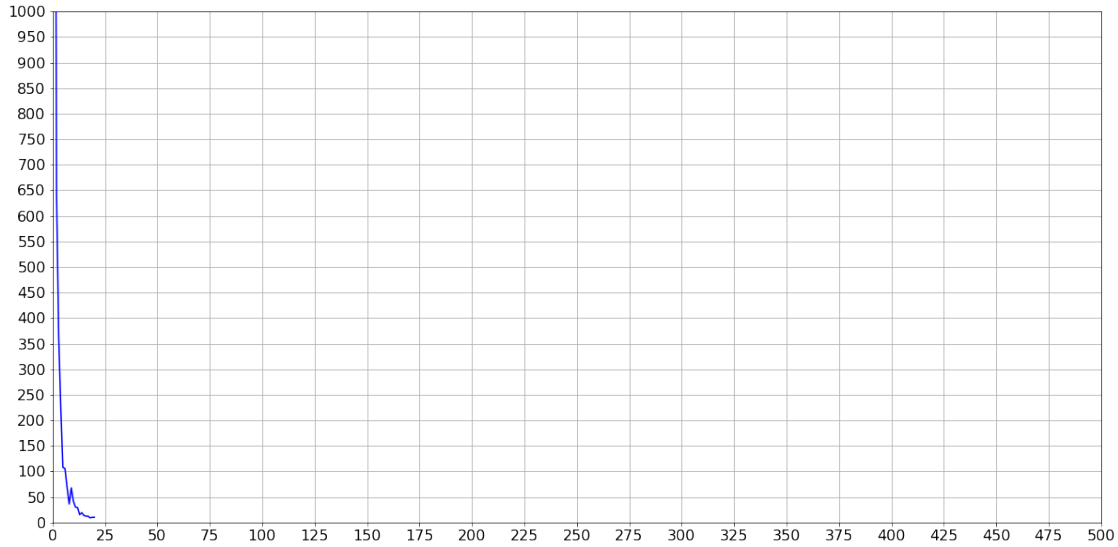
Метод Elbow - метод "локтя". С помощью данного методы вычисляется оптимальное количество кластеров. На каждой итерации значение суммы квадратов расстояний до каждого кластера будет уменьшаться, то есть в какой-то момент значение функции будет слабо изменяться, что означает достижении оптимального числа кластеров. В алгоритм ниже добавлена невязка для останова.

```
elbow_dist = np.array([])
k_final = 0
prev = 100
print('Iteration number (clusters number) = ', k)
#centroids = initialize_centroids(points, k)
centroids = initialize_centroids_randow(points, k)
closest_points = closest_centroids(points, centroids, n_points, k)
dist = elbow_function(points, centroids, closest_points, k)
elbow_dist = np.append(elbow_dist, dist)
prev = np.copy(dist)
k += 1
#print(elbow_dist)
#K = range(2, 100, 1)
while True:
    print('Iteration number (clusters number) = ', k)
    #centroids = initialize_centroids(points, k)
    centroids = initialize_centroids_randow(points, k)
    closest_points = closest_centroids(points, centroids, n_points, k)
    dist = elbow_function(points, centroids, closest_points, k)
    #elbow_dist = np.append(elbow_dist, dist)
    if np.absolute(dist - prev) < elbow_residual:
        k_final = k
        elbow_dist = np.append(elbow_dist, dist)
        break
    else:
        prev = np.copy(dist)
        elbow_dist = np.append(elbow_dist, dist)
        k += 1
```

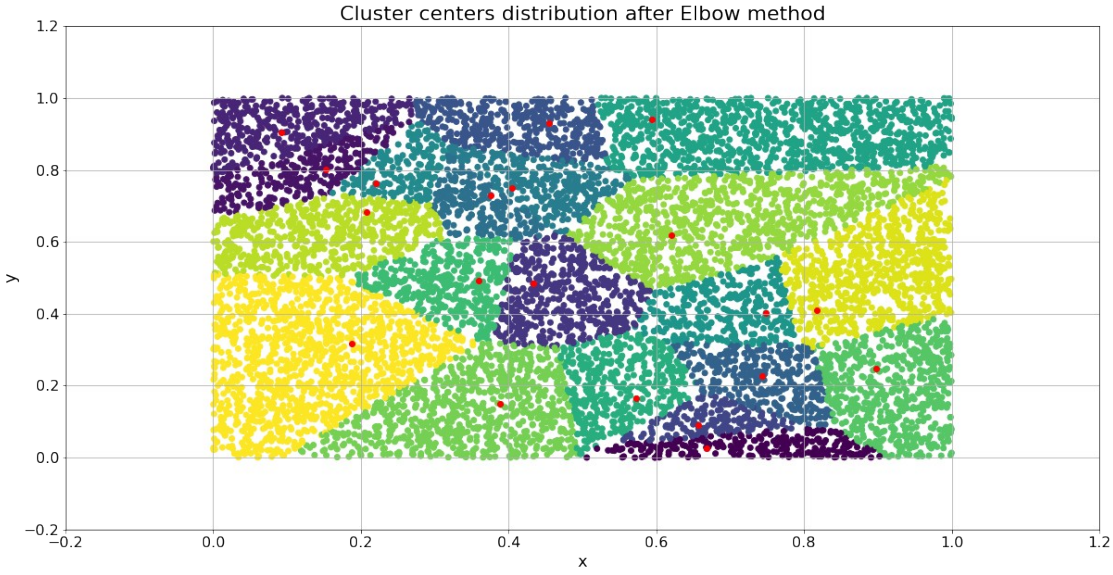
```
print('Elbow method function')
print(elbow_dist)
print('Clusters optimum number = ', k_final)
```

```
Iteration number (clusters number) =  1
Iteration number (clusters number) =  2
Iteration number (clusters number) =  3
Iteration number (clusters number) =  4
Iteration number (clusters number) =  5
Iteration number (clusters number) =  6
Iteration number (clusters number) =  7
Iteration number (clusters number) =  8
Iteration number (clusters number) =  9
Iteration number (clusters number) =  10
Iteration number (clusters number) =  11
Iteration number (clusters number) =  12
Iteration number (clusters number) =  13
Iteration number (clusters number) =  14
Iteration number (clusters number) =  15
Iteration number (clusters number) =  16
Iteration number (clusters number) =  17
Iteration number (clusters number) =  18
Iteration number (clusters number) =  19
Iteration number (clusters number) =  20
Elbow method function
[2050.31231422  640.83286486  364.32305239  222.97891962  107.8956129
  105.66103021   69.05382396   36.21251515   67.83998885   41.96735729
   30.19174437   29.27444804   15.48498863   19.3811133    13.91026478
   12.50606198   12.30713163    9.12006696   10.26240691
10.23088769]
Clusters optimum number =  20
```

```
K = range(1, k_final + 1)
K = np.array(K)
#print(K)
#print(K[len(K) - 1])

plt.figure(figsize = (20,10))
plt.title('Elbow method (choosing the value of k)', fontsize = 22)
plt.xlabel('k (number of clusters)', fontsize = 18)
plt.ylabel('Elbow function SSE', fontsize = 18)
plt.axes(xlim=(0, K[len(K) - 1] + 1), ylim=(0, 1000))
plt.grid(True)
plt.plot(K, elbow_dist, c = 'blue')
plt.xticks(np.linspace(0, 500, 21), fontsize = 16)
plt.yticks(np.linspace(0, 1000, 21), fontsize = 16);
#plt.xticks([int(5*i) for i in range(1, 10)])
#plt.yticks([int(50*i) for i in range(1, 20)])
#plt.yticks([0, 10, 20, 30, 40, 50, 60, 70, 80])
```




```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
#plt.title('Cluster centers distribution after Elbow method', fontsize
= 22)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
new_centroids = move_centroids(points, centroids, closest_points,
k_final)
```

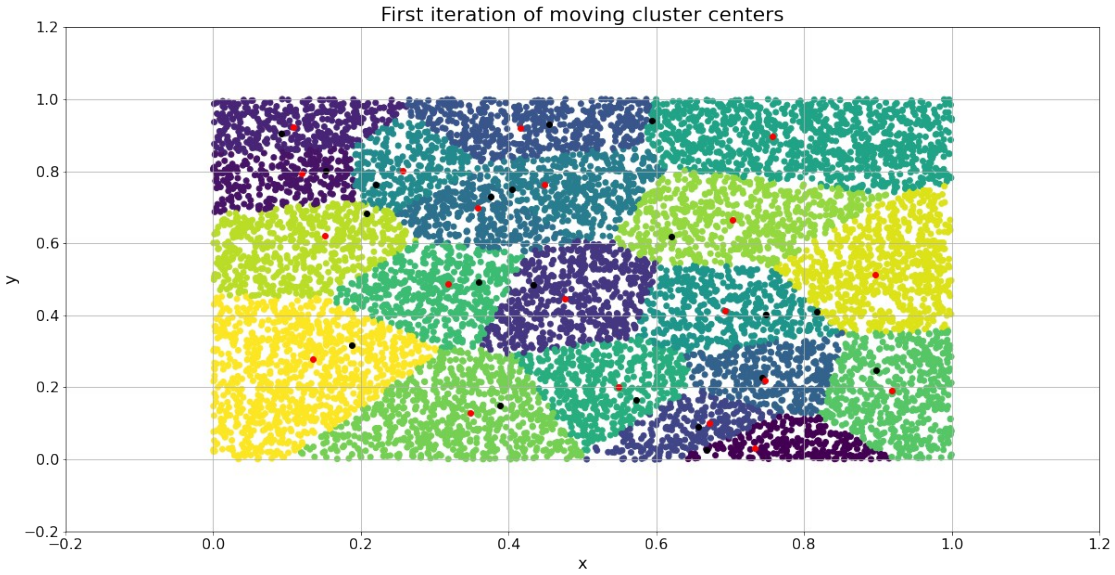
```
print(new_centroids)
print(centroids)
```

```
[[0.73281325 0.03165371]
 [0.12021018 0.79256557]
 [0.10823442 0.92314278]
 [0.47615857 0.44587808]
 [0.67197259 0.10065984]
 [0.41600556 0.92026698]
 [0.7466383  0.22011228]
 [0.35878902 0.69945416]
 [0.44841239 0.76155886]
 [0.25683834 0.80237907]
 [0.6938645  0.41131285]
 [0.75758839 0.89656132]
 [0.54951611 0.20010395]
 [0.31809336 0.48597354]
 [0.91909649 0.19190342]
 [0.3480918  0.13007431]
 [0.70324561 0.66385677]
 [0.15138161 0.62026496]
 [0.8970426  0.51289551]
 [0.13523781 0.2777634 ]]
[[0.66856527 0.02519469]
 [0.15308036 0.80231544]
 [0.09275386 0.90426955]
 [0.43347546 0.48335464]
 [0.65666698 0.09125634]
 [0.45541949 0.92958642]
 [0.74376911 0.22692684]
 [0.37576353 0.7299776 ]
 [0.4045289  0.74897465]
 [0.22004763 0.76293653]
 [0.7485253  0.40208215]
 [0.59386546 0.93979508]
 [0.57293119 0.16405645]
 [0.36014693 0.49214053]
 [0.89731822 0.24879362]
 [0.38895918 0.14868248]
 [0.62081452 0.61973929]
 [0.20724845 0.68242633]
 [0.81779309 0.4106401 ]
 [0.18824092 0.31643378]]
```

```
closest_points = closest_centroids(points, new_centroids, n_points,
k_final)
print(closest_points)
```

```
[11  2 11 ... 11  5 14]
```

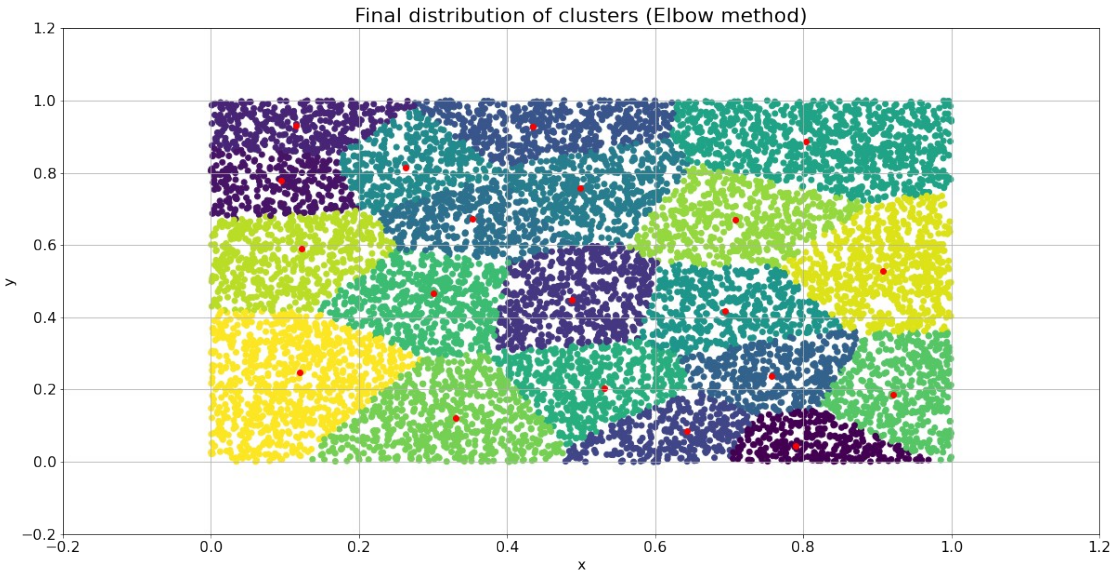
```
plt.figure(figsize = (20,10))
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.title('First iteration of moving cluster centers', fontsize = 22)
plt.xlabel('x', fontsize = 18)
plt.ylabel('y', fontsize = 18)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(centroids[:,0], centroids[:,1], c = 'black')
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```




```
iteration = 1
exit_condition = residual_check(new_centroids, centroids,
centroids_residual)
print('Iteration number = ', iteration, end = ' ')
print(exit_condition)
while exit_condition == False:
    iteration += 1
    prev_centroids = np.copy(new_centroids)
    new_centroids = move_centroids(points, prev_centroids,
closest_points, k_final)
    closest_points = closest_centroids(points, new_centroids,
n_points, k_final)
    exit_condition = residual_check(new_centroids, prev_centroids,
centroids_residual)
    print('Iteration number = ', iteration, end = ' ')
    print(exit_condition)
print('Final centroids')
print(new_centroids)
print('Closest_points')
print(closest_points)
```

```
Iteration number =  1 False
Iteration number =  2 True
Final centroids
[[0.78940566 0.04401518]
 [0.09489559 0.77764885]
 [0.11497743 0.93092924]
 [0.48689972 0.44784317]
 [0.64245621 0.08526144]
 [0.43488166 0.92726564]
 [0.75669472 0.23743255]
 [0.35309121 0.67322319]
 [0.49848989 0.75671804]
 [0.26272935 0.81531026]
 [0.6951084  0.41789445]
 [0.80384011 0.8855155 ]
 [0.53128264 0.20466463]
 [0.30122059 0.46637939]
 [0.92202915 0.18696448]
 [0.33010024 0.12199426]
 [0.70792505 0.66909229]
 [0.12210546 0.58937805]
 [0.90728056 0.52846725]
 [0.11989376 0.24642729]]
Closest_points
[11  2 11 ... 11  5 14]
```

```
#plt.axes(xlim=(xlim_min, xlim_max), ylim=(ylim_min, ylim_max))
plt.figure(figsize = (20,10))
plt.title('Final distribution of clusters (Elbow method)', fontsize =
22)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
plt.grid(True)
plt.scatter(points[:,0], points[:,1], c = closest_points)
plt.scatter(new_centroids[:,0], new_centroids[:,1], c = 'red')
plt.xticks(np.linspace(xlim_min, xlim_max, 8), fontsize = 16)
plt.yticks(np.linspace(ylim_min, ylim_max, 8), fontsize = 16);
```



Вывод.

Потребовалось 2 итерации для нахождения центров кластеров при уменьшении точности решения в 100 раз. Это в 28.5 раз быстрее, чем в разделе 1 получилось при точности 0.001.

Количество оптимальных кластеров получилось 20. Это в 4.8 раз меньше, чем в разделе 1.