In [363…

```python
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

In [364…

```python
def euclidean_metric(x1, y1, x2, y2):
        return np.power((np.power(abs(x1 - x2), 2) + np.power(abs(y1 - y2), 2)), 0.5)
```

In [365…

```python
def color_rand():
    return f'#{"".join([hex(np.random.choice((range(15))))[2:] for i in range(6)])}'
```

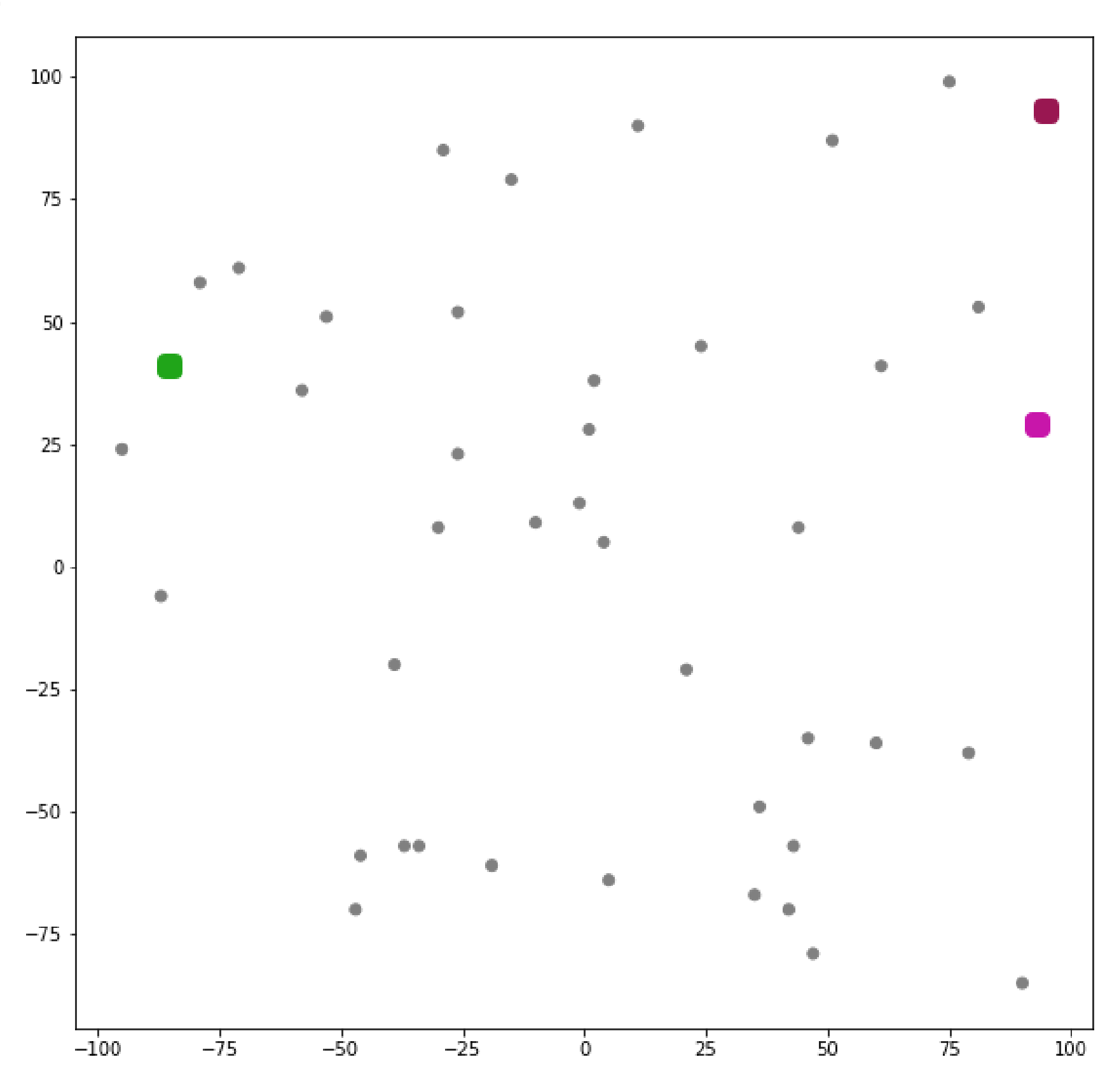
Реализация k-means с случайным расположением центроид

In [366…

```python
points_amount = 40
clusters_amount = 3
x_coord_max = 100
y_coord_max = 100


point_coord_x = np.array([np.random.randint(low=-x_coord_max, high=x_coord_max) for i in range(points_amount)])
point_coord_y = np.array([np.random.randint(low=-y_coord_max, high=y_coord_max) for i in range(points_amount)])
point_colors = np.array(['grey'] * points_amount, dtype='object')

point_coords = np.array([point_coord_x, point_coord_y]).transpose()


centr_coord_x = np.array([np.random.randint(low=-x_coord_max, high=x_coord_max) for i in range(clusters_amount)
centr_coord_y = np.array([np.random.randint(low=-y_coord_max, high=y_coord_max) for i in range(clusters_amount)
#centr_colors = np.array(['black', 'red', 'green', 'cyan'], dtype='object')
centr_colors = np.array([color_rand() for i in range(clusters_amount)], dtype='object')

centr_coords = np.array([centr_coord_x, centr_coord_y]).transpose()


figure, ax = plt.subplots(figsize=(10,10))
ax.scatter(x=point_coords[:, 0], y=point_coords[:, 1], c=point_colors)
ax.scatter(x=centr_coords[:, 0], y=centr_coords[:, 1], c=centr_colors, \
           marker='s', linewidths=8, alpha=1)
```

Out[366…

```
<matplotlib.collections.PathCollection at 0x7fa105473220>
```



Красим вершины в первый раз

In [367…

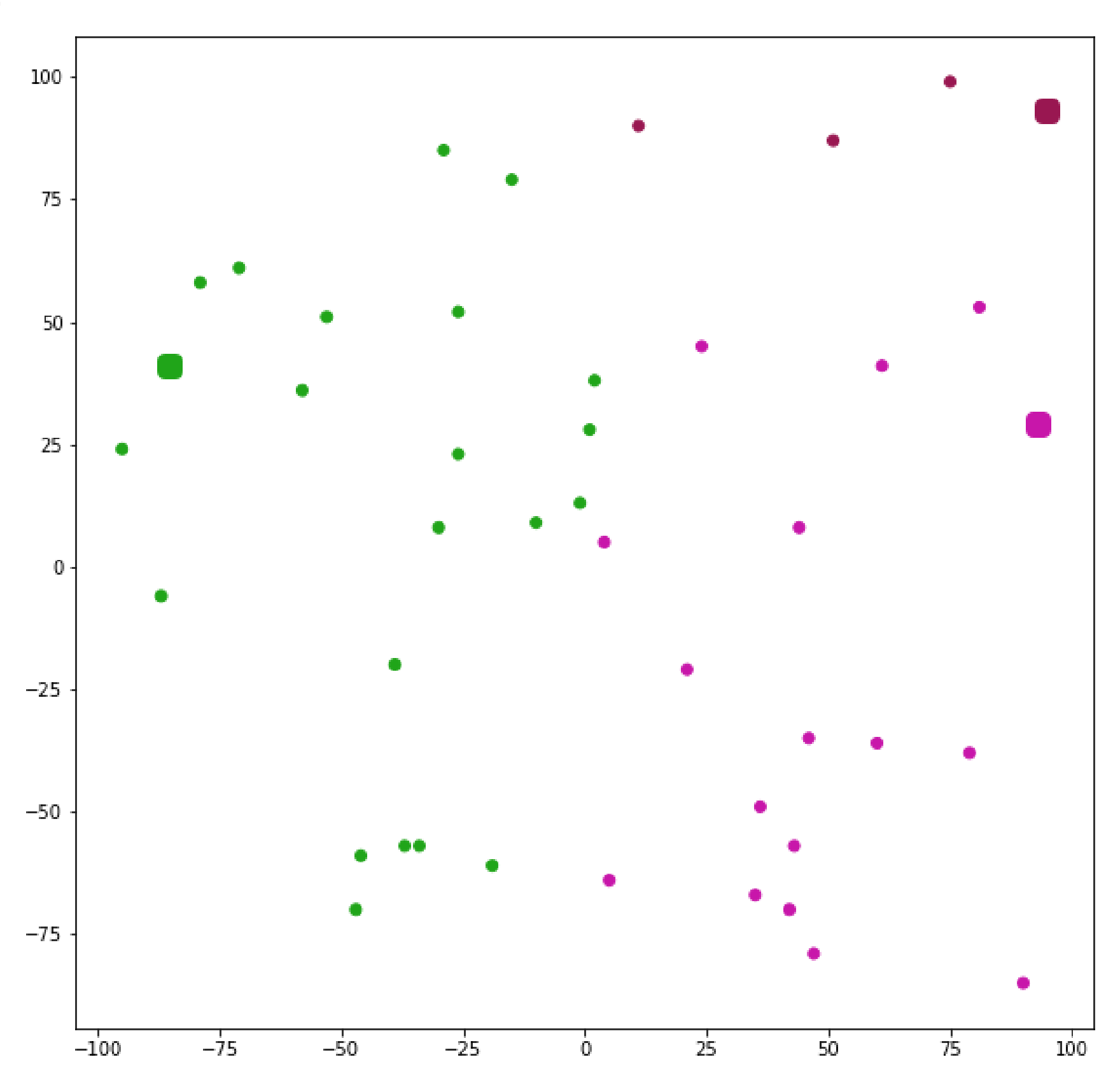
```python
for i in range(points_amount):
    min_distance = euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                                    centr_coords[0, 0], centr_coords[0, 1])
    min_cluster_index = 0

    for j in range(0, clusters_amount):
        if (euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                             centr_coords[j, 0], centr_coords[j, 1]) < min_distance):
            min_distance = euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                                            centr_coords[j, 0], centr_coords[j, 1])
            min_cluster_index = j

    point_colors[i] = centr_colors[min_cluster_index]
```

In [368…

```python
figure, ax = plt.subplots(figsize=(10,10))

ax.scatter(x=point_coords[:, 0], y=point_coords[:, 1], c=point_colors)
ax.scatter(x=centr_coords[:, 0], y=centr_coords[:, 1], c=centr_colors, marker='s', linewidths=8)
```

Out[368…

```
<matplotlib.collections.PathCollection at 0x7fa10282ee80>
```



Оставшиеся итерации алгоритма

Будем изменять положение центра до тех пор, пока оно изменяется больше, чем на `difference`, в сравнении с предыдущей итерацией. После того, как мы изменяем положение центров, вновь красим точки.

In [369…

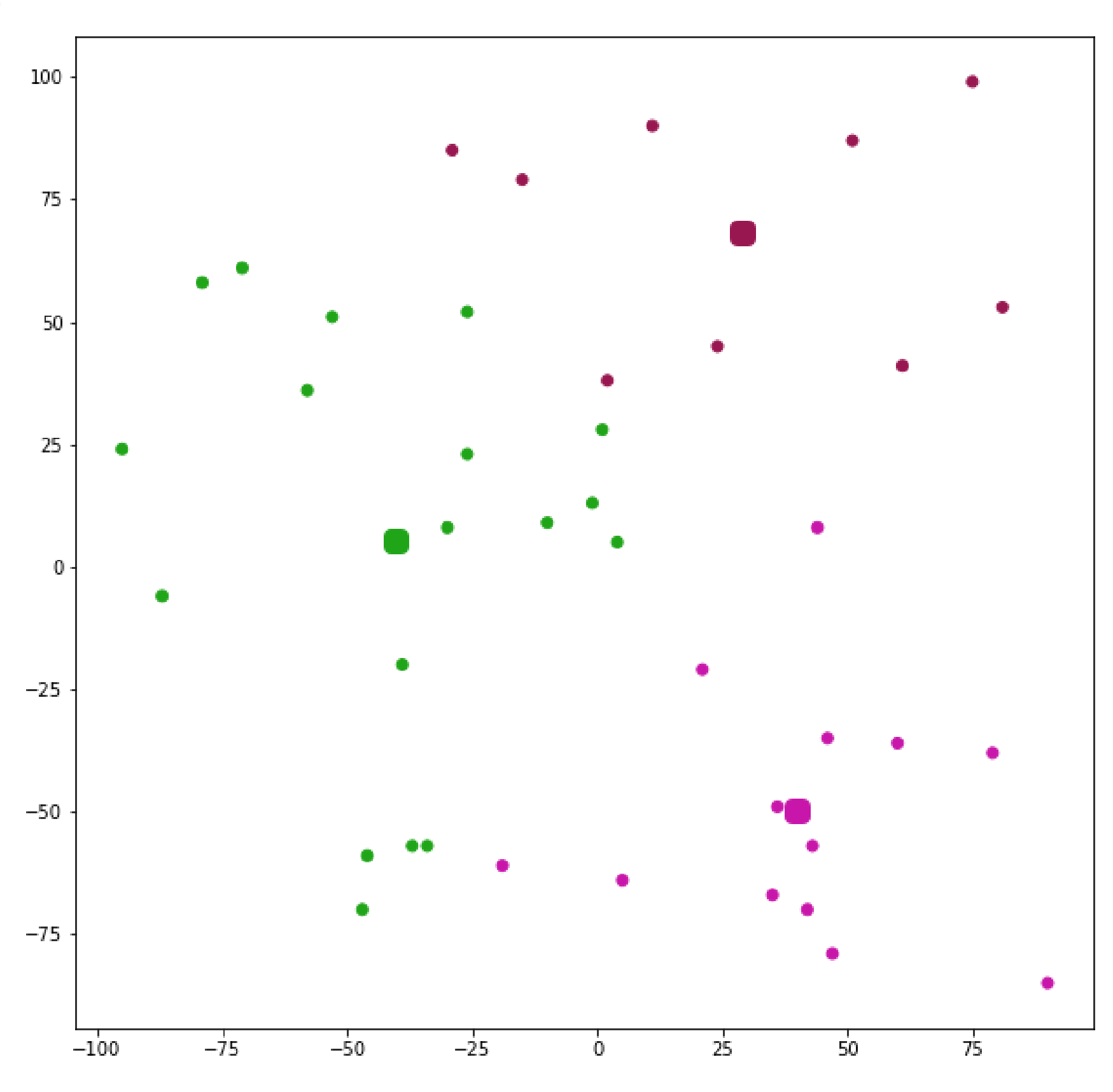
```python
difference = 1
new_centr_coords = np.copy(centr_coords)
last_centr_coords = np.copy(centr_coords) + 1.5

while not (abs(new_centr_coords - last_centr_coords) < difference).all():

    for h in range(clusters_amount):
        last_centr_coords = np.copy(new_centr_coords)
        array_specified_color = point_coords[point_colors == centr_colors[h]]
        try:
            new_x, new_y = sum(array_specified_color) / len(array_specified_color)
        except ZeroDivisionError:
            pass
        new_centr_coords[h] = np.array([new_x, new_y])

    for i in range(points_amount):
        min_distance = euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                                        new_centr_coords[0, 0], new_centr_coords[0, 1])
        min_cluster_index = 0

        for j in range(0, clusters_amount):
            if (euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                                 new_centr_coords[j, 0], new_centr_coords[j, 1]) < min_distance):
                min_distance = euclidean_metric(point_coords[i, 0], point_coords[i, 1], \
                                                new_centr_coords[j, 0], new_centr_coords[j, 1])
                min_cluster_index = j

        point_colors[i] = centr_colors[min_cluster_index]
```

In [370…

```python
figure, ax = plt.subplots(figsize=(10,10))

ax.scatter(x=point_coords[:, 0], y=point_coords[:, 1], c=point_colors)
ax.scatter(x=new_centr_coords[:, 0], y=new_centr_coords[:, 1], c=centr_colors, marker='s', linewidths=8)
```

Out[370…

```
<matplotlib.collections.PathCollection at 0x7fa105639400>
```



Реализация k-means++

Первый центроид выбирается случайно из всех точек, остальные будут выбраны так, чтобы расстояние между центроидами было максимальным.

In [371…

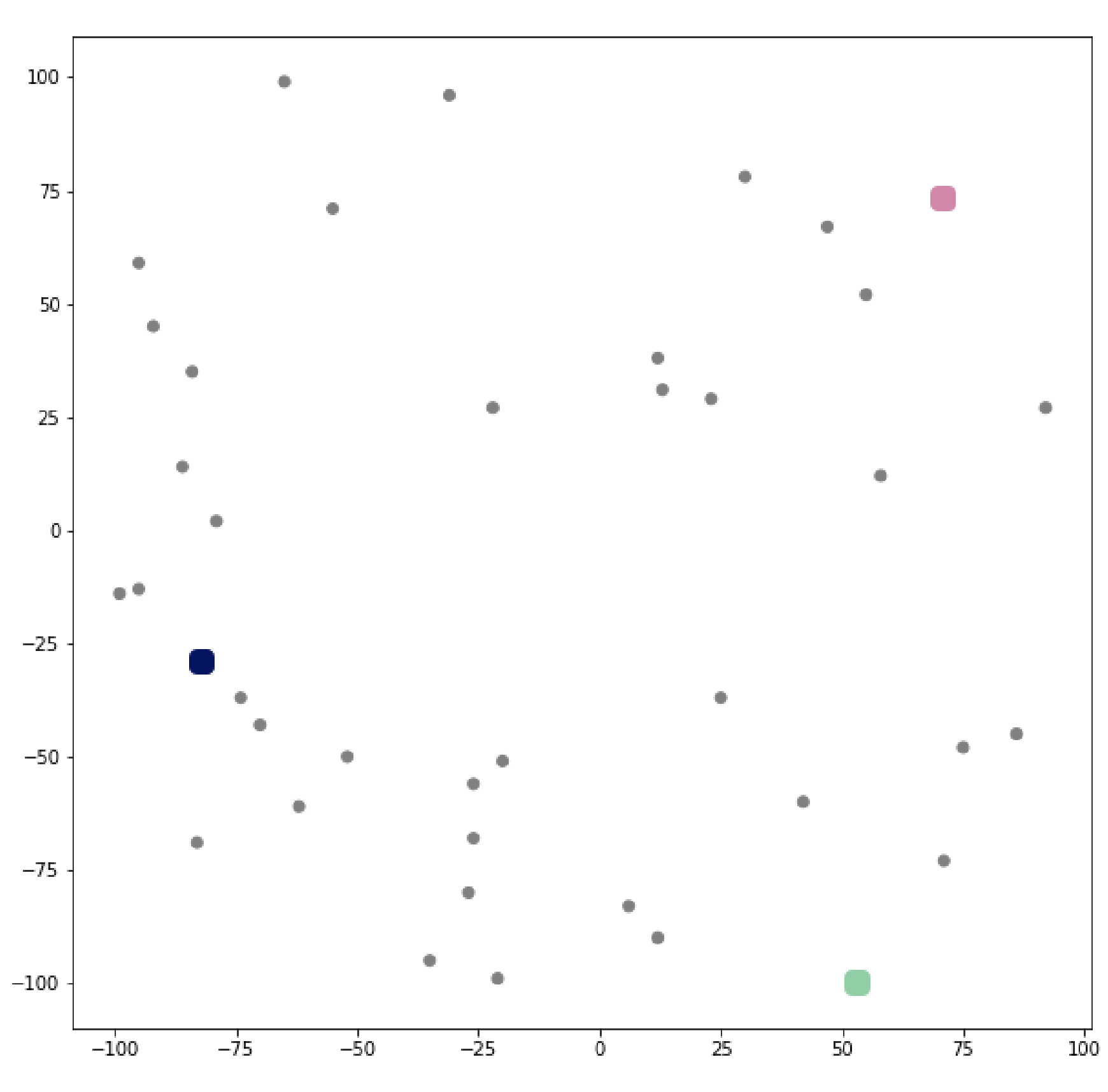
```python
point_coord_x = np.array([np.random.randint(low=-x_coord_max, high=x_coord_max) for i in range(points_amount)])
point_coord_y = np.array([np.random.randint(low=-y_coord_max, high=y_coord_max) for i in range(points_amount)])
point_coord_x_old = np.copy(point_coord_x) # сохраняется для использования в матрице расстояний
point_coord_y_old = np.copy(point_coord_y)
is_centr = np.array([0] * points_amount)
point_colors = np.array(['grey'] * points_amount, dtype='object')
point_coords_old = np.array([point_coord_x_old, point_coord_y_old]).T


point_data = np.array([point_coord_x, point_coord_y, is_centr]).transpose()
chosen = np.random.randint(low=0, high=points_amount-1)
point_data[chosen][2] = 1
centr_colors = np.array([color_rand() for i in range(clusters_amount)], dtype='object')
index_for_color_pick = 0
point_colors[chosen] = centr_colors[index_for_color_pick]


for i in range(1, clusters_amount):
    index_for_color_pick += 1
    dist_array = np.array([0] * points_amount)
    dist_is_centr_array = np.array([dist_array, point_data[:, 2]]).transpose()

    for centroid in point_data[point_data[:,2] == 1,:]:

        for j in range(points_amount):
            if dist_is_centr_array[j, 1] == 0:
                dist_is_centr_array[j, 0] += euclidean_metric(centroid[0], centroid[1], \
                                                              point_data[j, 0], point_data[j, 1])

    index_max = np.argmax(dist_is_centr_array[:,0])
    point_data[index_max, 2] = 1
    point_colors[index_max] = centr_colors[index_for_color_pick]
```

In [372…

```python
figure, ax = plt.subplots(figsize=(10, 10))

ax.scatter(x=point_data[point_data[:,2] == 0, 0], y=point_data[point_data[:,2] == 0, 1], \
           c=point_colors[point_data[:,2] == 0])
ax.scatter(x=point_data[point_data[:,2] == 1, 0], y=point_data[point_data[:,2] == 1, 1], \
           c=point_colors[point_data[:,2] == 1], marker='s', linewidths=8)
```

Out[372…

```
<matplotlib.collections.PathCollection at 0x7fa105422d00>
```



Красим вершины в первый раз

Сначала отделяю центроиды от остальных точек для того, чтобы использовать код, который был написан выше.

In [373…

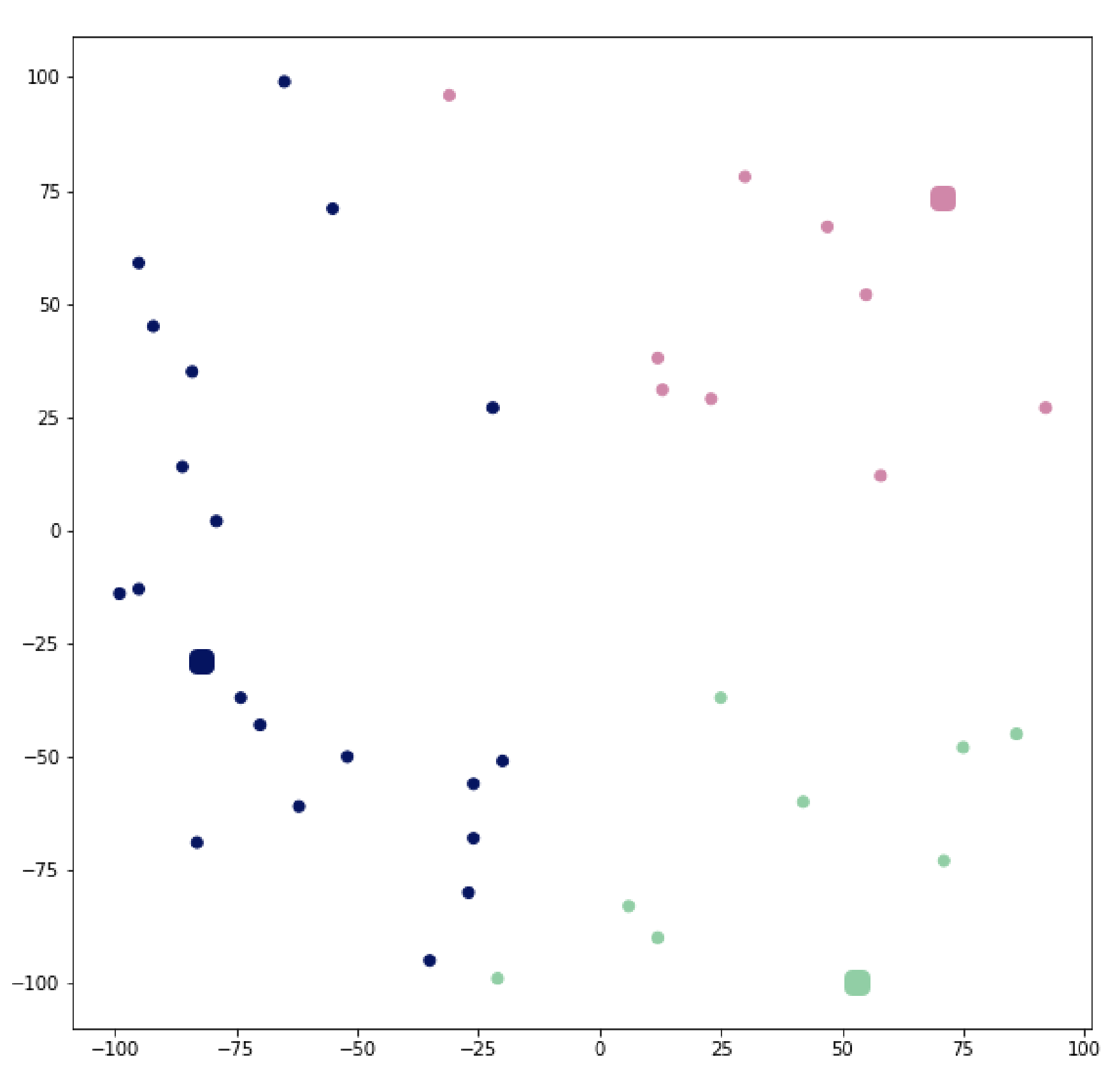
```python
mask = (point_data[:,2] == 1)
simple_point_coords = np.array([point_data[np.logical_not(mask), 0], \
                                point_data[np.logical_not(mask), 1]]).transpose()
simple_point_colors = point_colors[np.logical_not(mask)]

centr_coords = np.array([point_data[(mask), 0], point_data[(mask), 1]]).transpose()
centr_colors = point_colors[mask]

for i in range(points_amount - clusters_amount):
    min_distance = euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                                    centr_coords[0, 0], centr_coords[0, 1])
    min_cluster_index = 0

    for j in range(0, clusters_amount):
        if euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                            centr_coords[j, 0], centr_coords[j, 1]) < min_distance:

            min_distance = euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                                            centr_coords[j, 0], centr_coords[j, 1])
            min_cluster_index = j

    simple_point_colors[i] = centr_colors[min_cluster_index]
```

In [374…

```python
figure, ax = plt.subplots(figsize=(10,10))

ax.scatter(x=simple_point_coords[:, 0], y=simple_point_coords[:, 1], c=simple_point_colors)
ax.scatter(x=centr_coords[:, 0], y=centr_coords[:, 1], c=centr_colors, marker='s', linewidths=8)
```

Out[374…

```
<matplotlib.collections.PathCollection at 0x7fa1055cfa90>
```



Оставшиеся итерации алгоритма

Полностью повторяется код из реализации обычного k-means, только выделен массив для точек, не являющихся центроидами.

In [375…

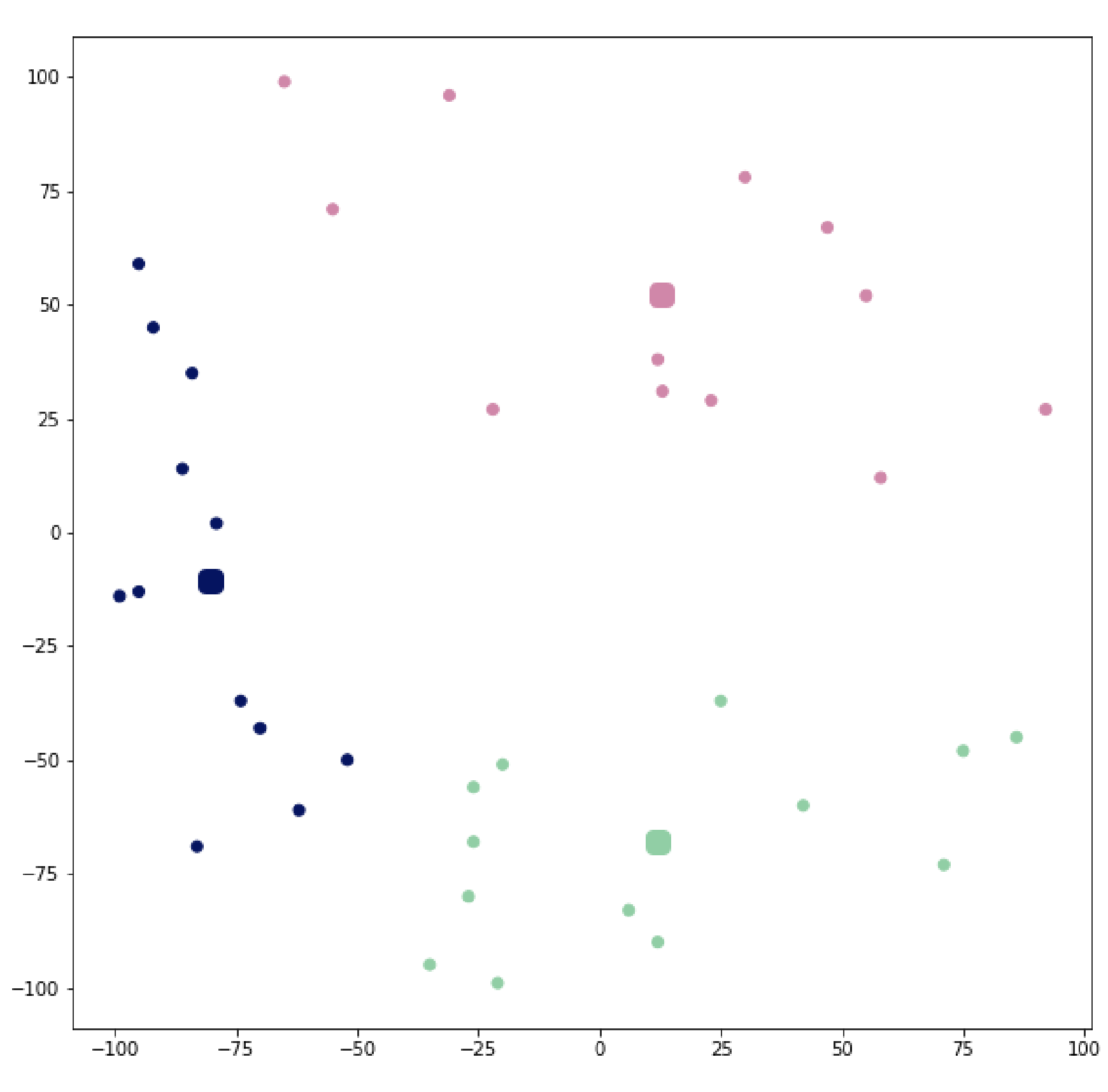
```python
difference = 1
new_centr_coords = np.copy(centr_coords)
last_centr_coords = np.copy(centr_coords) + 1.5

while not (abs(new_centr_coords - last_centr_coords) < difference).all():

    for h in range(clusters_amount):
        last_centr_coords = np.copy(new_centr_coords)
        array_specified_color = simple_point_coords[simple_point_colors == centr_colors[h]]
        try:
            new_x, new_y = sum(array_specified_color) / len(array_specified_color)
        except ZeroDivisionError:
            pass
        new_centr_coords[h] = np.array([new_x, new_y])

    for i in range(points_amount - clusters_amount):
        min_distance = euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                                        new_centr_coords[0, 0], new_centr_coords[0, 1])
        min_cluster_index = 0

        for j in range(0, clusters_amount):
            if (euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                                 new_centr_coords[j, 0], new_centr_coords[j, 1]) < min_distance):
                min_distance = euclidean_metric(simple_point_coords[i, 0], simple_point_coords[i, 1], \
                                                new_centr_coords[j, 0], new_centr_coords[j, 1])
                min_cluster_index = j

        simple_point_colors[i] = centr_colors[min_cluster_index]
```

In [376…

```python
figure, ax = plt.subplots(figsize=(10,10))

ax.scatter(x=simple_point_coords[:, 0], y=simple_point_coords[:, 1], c=simple_point_colors)
ax.scatter(x=new_centr_coords[:, 0], y=new_centr_coords[:, 1], c=centr_colors, marker='s', linewidths=8)
```

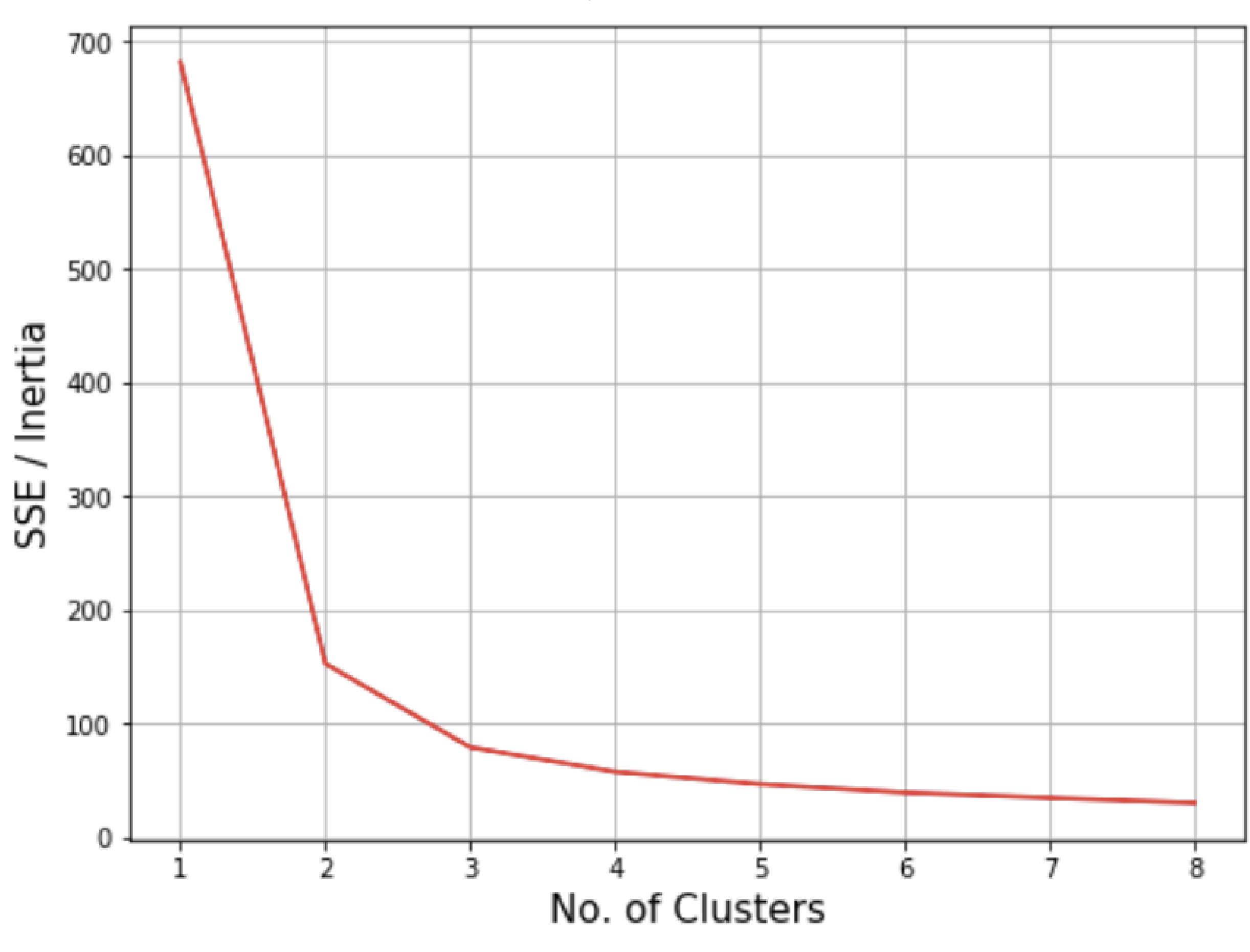
Out[376…

```
<matplotlib.collections.PathCollection at 0x7fa104d9e7f0>
```



Стратегия выбора количества кластеров

Существует способ выбора числа центроидов - **Elbow method**.\ Одна из вариаций метода основывается на подсчете **инерции** - суммы расстояний от точек до ближайших к ним центроидов.\ Далее строится график зависимости **инерции** от количества центроидов. Пример показан на картинке снизу.



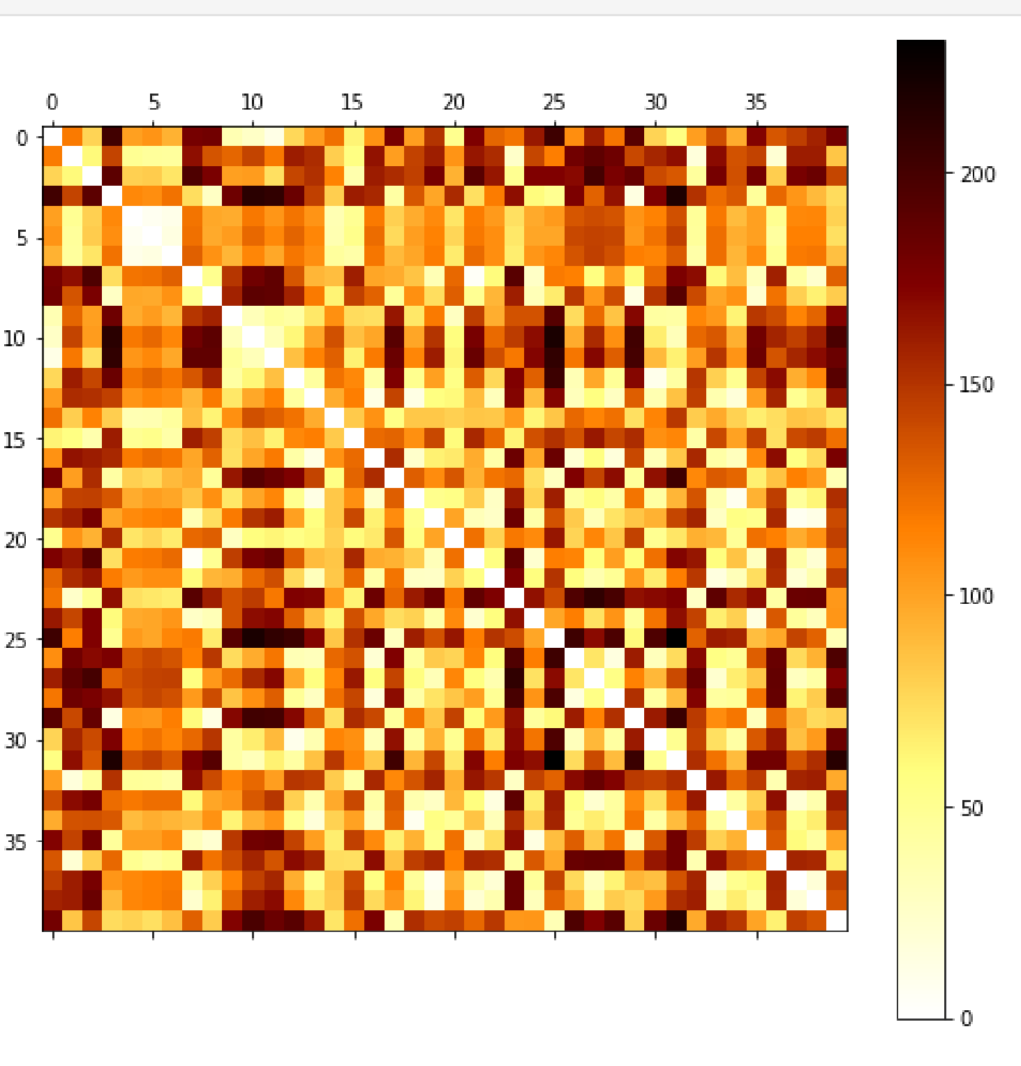
Оптимальным количеством центров кластеров будет то, в котором происходит изгиб графика (локоть (**elbow**), если представить, что график изображает руку). \ В примере сверху это количество будет равно 3, так как после этого значения **инерция** начинает уменьшаться примерно линейно.

Матрица попарных расстояний до и после кластеризации

До кластеризации

In [377…

```python
dm_old = np.linalg.norm(point_coords_old[:, None, :] - point_coords_old[None, :, :], axis=-1)
fig, ax = plt.subplots(1, 1, figsize=(9, 9))
cax = ax.matshow(dm_old, cmap='afmhot_r')
cbar = fig.colorbar(cax)
```



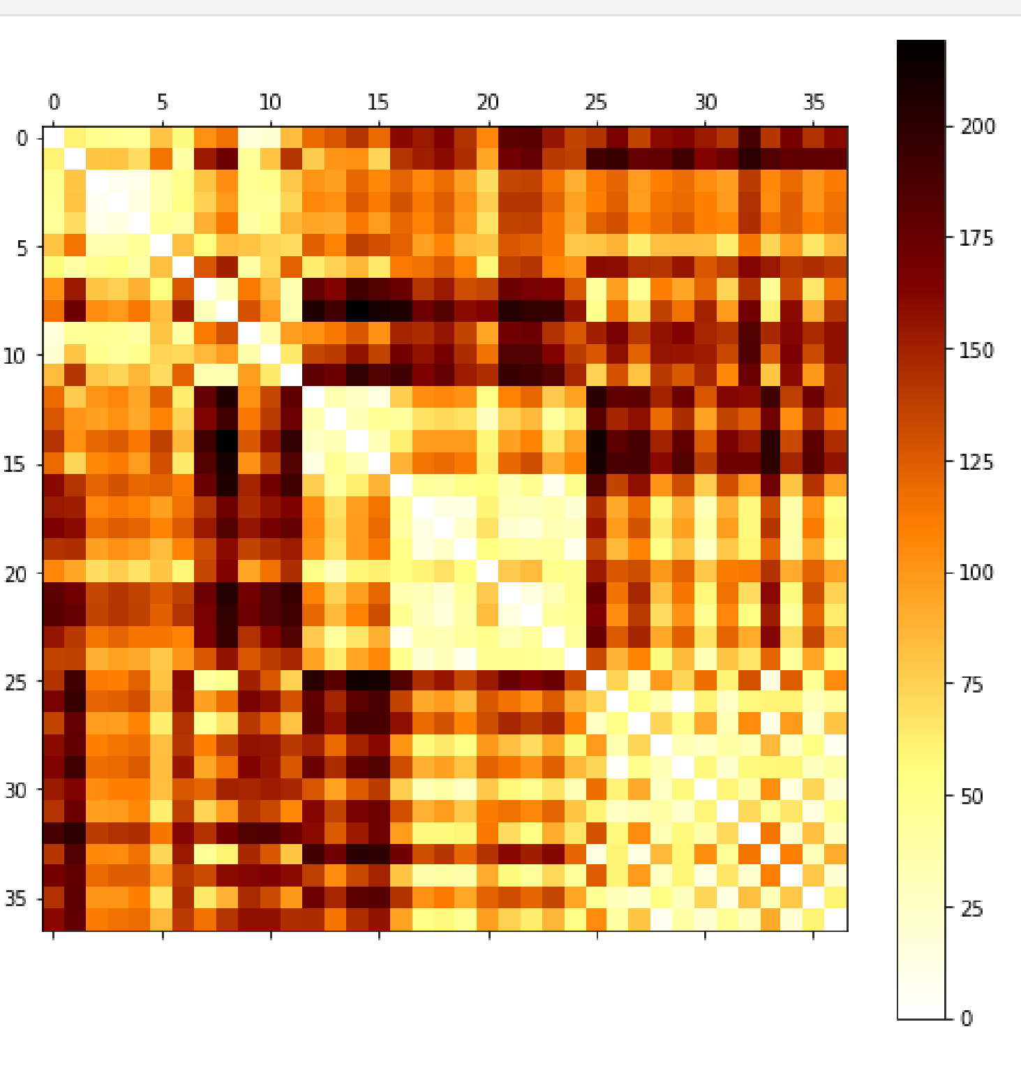
In [378…

```python
point_coords_new = np.array(simple_point_coords[simple_point_colors == centr_colors[0]])

for color in centr_colors[1:]:
    point_coords_new = np.append(point_coords_new, simple_point_coords[simple_point_colors == color],
                                 axis=0)
```

После кластеризации

In [379…

```python
dm_new = np.linalg.norm(point_coords_new[:, None, :] - point_coords_new[None, :, :], axis=-1)
fig, ax = plt.subplots(1, 1, figsize=(9, 9))
cax = ax.matshow(dm_new, cmap='afmhot_r')
cbar = fig.colorbar(cax)
```



Картина сильно улучшилась после кластеризации, но все равно недостаточно. Проблема в том, что точки распределяются случайно по всей плоскости. Если бы они располагались кластерами, было бы в разы лучше.