МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**НЕЧЕТКАЯ КЛАСТЕРИЗАЦИЯ ДАННЫХ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р. Р. Посевин

(подпись)

Направление подготовки 02.03.03 "Математическое обеспечение и администрирование информационных систем"   курс   3

Направленность (профиль) Технология программирования

Научный руководитель

к. ф.-м. н, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О. Н. Лапина

(подпись, дата)

Нормоконтролер

к. т. н, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е. Е. Полупанова

(подпись, дата)

Краснодар

2021

Содержание

[Введение 3](#_Toc73390155)

[1 Основные понятия кластеризации 4](#_Toc73390156)

[1.1 Кластеризация при заданном числе кластеров 4](#_Toc73390157)

[2 Базовый алгоритм нечетких с-средних 7](#_Toc73390158)

[2.1 Обобщения алгоритма нечетких с-средних 9](#_Toc73390159)

[3 Описание программной реализации алгоритма нечеткой кластеризации c-средних 11](#_Toc73390160)

[3.1 Описание программы 11](#_Toc73390161)

[3.2 Анализ работы программы для различных входных данных 14](#_Toc73390162)

[Заключение 18](#_Toc73390163)

[Список использованных источников 19](#_Toc73390164)

[Приложение А Код программы 20](#_Toc73390165)

# **ВВЕДЕНИЕ**

[Кластеризация](https://wiki.loginom.ru/articles/clustering.html) – объединение в группы схожих объектов – является одной из фундаментальных задач в области анализа данных и [Data Mining](https://wiki.loginom.ru/articles/data-mining.html). Список прикладных областей, где она применяется, широк: сегментация изображений, маркетинг, [борьба с мошенничеством](https://wiki.loginom.ru/articles/antifraud.html), [прогнозирование](https://wiki.loginom.ru/articles/forecasting.html), [анализ текстов](https://wiki.loginom.ru/articles/text-mining.html) и многие другие. На современном этапе кластеризация часто выступает первым шагом при анализе данных. После выделения схожих групп применяются другие методы, для каждой группы строится отдельная модель.

Задачу кластеризации в том или ином виде формулировали в таких научных направлениях, как статистика, распознавание образов, [оптимизация](https://wiki.loginom.ru/articles/optimization.html), [машинное обучение](https://wiki.loginom.ru/articles/machine-learning.html). Отсюда многообразие синонимов понятию кластер [класс](https://wiki.loginom.ru/articles/class.html), таксон, сгущение.

На сегодняшний момент число методов разбиения групп объектов на кластеры довольно велико – несколько десятков алгоритмов и еще больше их модификаций.

Целью данной курсовой работы является исследование и реализация алгоритма нечеткой кластеризации c-средних для двумерных данных на языке программирования Python.

# **Основные понятия кластеризации**

Методы кластерного анализа позволяют разделить изучаемую совокупность объектов на группы “схожих” объектов, называемых кластерами, разнести записи в различные группы, или сегменты. Большинство алгоритмов кластеризации не опираются на традиционные для статистических методов допущения; они могут использоваться в условиях почти полного отсутствия информации о законах распределения данных. Кластеризацию проводят для объектов с количественными (числовыми), качественными или смешанными признаками.

Существует множество методов кластеризации, которые можно классифицировать на четкие и нечеткие. Четкие методы кластеризации разбивают исходное множество объектов X на несколько непересекающихся подмножеств. При этом любой объект из X принадлежит только одному кластеру. Нечеткие методы кластеризации позволяют одному и тому же объекту принадлежать одновременно нескольким (или даже всем) кластерам, но с различной степенью. Нечеткая кластеризация во многих ситуациях более «естественна», чем четкая, например, для объектов, расположенных на границе кластеров.

Методы кластеризации также классифицируются по тому, определено ли количество кластеров заранее или нет. В последнем случае количество кластеров определяется в ходе выполнения алгоритма на основе распределения исходных данных.

## Кластеризация при заданном числе кластеров

Исходной информацией для кластеризации является матрица наблюдений:

Задача кластеризации состоит в разбиении объектов из на несколько подмножеств (кластеров), в которых объекты более схожи между собой, чем с объектами из других кластеров. В метрическом пространстве «схожесть» обычно определяют через расстояние. Расстояние может рассчитываться как между исходными объектами (строчками матрицы ), так и от этих объектов к прототипу кластеров. Обычно координаты прототипов заранее неизвестны - они находятся одновременно с разбиением данных на кластеры.

Нечеткие кластера будут описаны матрицей нечеткого разбиения:

где

содержит степени принадлежности объекта кластерам .

Единственным отличием матрицы степеней принадлежности четкого разбиения о нечеткого является то, что элементы матрицы принимают значения из двухэлементного множества , а не из интервала . Условия для матрицы выглядят следующим образом

Из условия (1.3) следует, что кластеры не могут быть пустыми, (1.4) обеспечивает распределение всех элементов набора данных по всем кластерам, то есть сумма степеней принадлежности по всем кластерам должна быть равна 1 для каждого элемента данных. Алгоритмы нечеткой кластеризации, которые удовлетворяют этим условиям, называются вероятностными алгоритмами кластеризации, так как по сути степени принадлежности данных являются вероятностями, с которыми элементы принадлежат данному кластеру.

Каждый кластер представляется своим прототипом. Поэтому проблема разделения набора данных , по кластерам решается задачей минимизации расстояний от элементов данных до центров кластеров. Это может быть осуществлено минимизацией следующей функции:

с условиями (1.3), (1.4),

где

- центры кластеров;

– экспоненциальный вес, определяющий нечеткость, размазанность кластеров. Обычно выбирают .

Целевая функция обычно минимизируется вычислением и , пока изменение степеней принадлежности не достигнет определенной точности вычисления .

Предложено множество алгоритмов нечеткой кластеризации, основанных на минимизации критерия (1.5). Нахождение матрицы нечеткого разбиения с минимальным значением критерия (1.5) представляет собой задачу нелинейной оптимизации, которая может быть решена разными методами. Наиболее известный и часто применяемый метод решения этой задачи алгоритм нечетких c-средних, в основу которого положен метод неопределенных множителей Лагранжа. Он позволяет найти локальный оптимум, поэтому выполнение алгоритма из различных начальных точек может привести к разным результатам.

# **2 Базовый алгоритм нечетких с-средних**

Самый простой алгоритм нечеткой кластеризации – это нечеткий алгоритм с-средних. Этот алгоритм находит компактные кластеры, например, сферической формы, которые приближены к реальным размерам и формам.

Здесь нечеткое разбиение позволяет просто решить проблему объектов, расположенных на границе двух кластеров - им назначают степени принадлежностей равные 0.5. Недостаток нечеткого разбиения проявляется при работе с объектами, удаленными от центров всех кластеров. Удаленные объекты имеют мало общего с любым из кластеров, поэтому интуитивно хочется назначить для них малые степени принадлежности. Однако, по условию (2.4) сумма их степеней принадлежностей такая же, как и для объектов, близких к центрам кластеров, т. е. равна единице. Для устранения этого недостатка можно использовать возможностное разбиение, которое требует, только чтобы произвольный объект из принадлежал хотя бы одному кластеру. Возможностное разбиение получается следующим ослаблением условия (2.4): .

Алгоритм нечетких c-средних состоит из 7 шагов:

1. Установить параметры алгоритма: - количество кластеров; - экспоненциальный вес; - параметр останова алгоритма.
2. Случайным образом сгенерировать матрицу нечеткого разбиения, удовлетворяющую условиям (1.3), (1.4).
3. Рассчитать центры кластеров: .
4. Рассчитать расстояния между объектами из и центрами кластеров: .
5. Пересчитать элементы матрицы нечеткого разбиения:

если :

если :

1. Проверить условие , где - матрица нечеткого разбиения на предыдущей итерации алгоритма. Если «да», то перейти к шагу 7, иначе – к шагу 3.
2. Конец.

В приведенном алгоритме самым важным параметром является количество кластеров (). Правильно выбрать количество кластеров для реальных задач без какой-либо априорной информации о структурах в данных достаточно сложно. Существует два формальных подхода к выбору числа кластеров.

Первый подход основан на критерии компактности и разделимости полученных кластеров. Логично предположить, что при правильном выборе количества кластеров данные будут разбиты на компактные и хорошие отделимые друг от друга группы. В противном случае, кластеры, вероятно, не будут компактными и хорошо отделимыми. Существует несколько критериев оценки компактности кластеров, однако вопрос о том, как формально и достоверно определить правильность выбора количества кластеров для произвольного набора данных все еще остается открытым.

Второй подход предлагает начинать кластеризацию при достаточно большом числе кластеров, а затем последовательно объединять схожие смежные кластера. При этом используются различные формальные критерии схожести кластеров.

Вторым параметром алгоритма кластеризации является экспоненциальный вес (). Чем больше , тем конечная матрица нечеткого разбиения становится более «размазанной», и при она примет вид , что является очень плохим решением, т. К. все объекты принадлежат ко всем кластерам с одной и той же степенью. Кроме того, экспоненциальный вес позволяет при формировании координат центров кластеров усилить влияние объектов с большими значениями степеней принадлежности и уменьшить влияние объектов с малыми значениями степеней принадлежности. На сегодня не существует теоретически обоснованного правила выбора значения экспоненциального веса. Обычно устанавливают.

## 2.1 Обобщения алгоритма нечетких с-средних

В базовом алгоритме нечетких c-средних расстояние между объектом и центром кластера рассчитывается через стандартную Евклидову норму: . В кластерном анализе применяются и другие нормы, среди которых часто используется диагональная норма и норма Махалонобиса.

В общем виде норму можно задать через симметричную положительно определенную матрицу B размером следующим образом:

где

– операция транспонирования.

Для Евклидовой нормы матрица представляет собой единичную матрицу:

Евклидова норма позволяет выделять кластеры в виде гиперсфер. Для диагональной нормы матрица B задается следующим образом:

Элементы главной диагонали матрицы интерпретируются как веса координат. Диагональная норма позволяет выделять кластеры в виде гиперэллипсоидов, ориентированных вдоль координатных осей.

Норма Махаланобиса позволяет выделять кластеры в виде гиперэллипсоидов, оси которых могут быть ориентированы в произвольных направлениях.

Примеры изолиний различных норм, показаны на рисунке 1. На рисунке 2 приведен пример нечеткой кластеризации методом нечетких c-средних при Евклидовом расстоянии. На левой части рисунка показаны данные для кластеризации. На правой части рисунка приведены результаты нечеткой кластеризации. Центры нечетких кластеров обозначены символами '+'. Восемь изолиний функций принадлежности нечетких кластеров построены для значений 0.67, 0.71, 0.75, 0.79, 0.83, 0.87, 0.91 и 0.95.

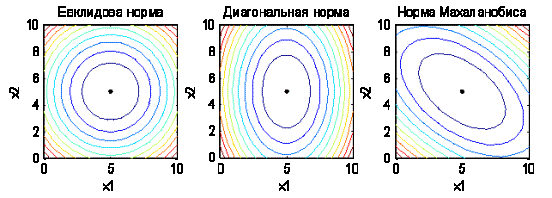


Рисунок 1 – Изолинии различных норм

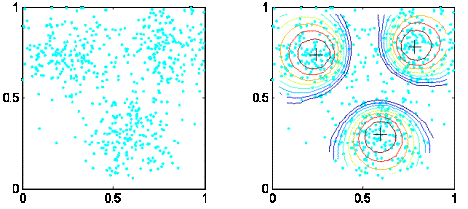


Рисунок 2 – Нечеткая кластеризация при Евклидовой норме

Для некоторых наборов данных «глазная кластеризация» позволяет выделить скопления данных в виде различных геометрических фигур: сферы, эллипсоиды разной ориентации, цепочки и т. П. В результате алгоритмов кластеризации с фиксированной нормой форма всех кластеров получается одинаковой. Алгоритмы кластеризации как бы навязывают данным неприсущую им структуру, что приводит не только к неоптимальным, но иногда и к принципиально неправильным результатам.

# **3 Описание программной реализации алгоритма нечеткой кластеризации c-средних**

Программная реализация алгоритма нечеткой кластеризации c-средних для двумерных данных выполнена на языке программирования Python.

## 3.1 Описание программы

Двумерные данные в программе представлены в виде класса Point, изображенного на рисунке 3.

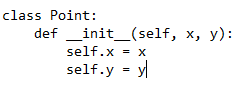


Рисунок 3 – Описание класса Point

Функция, которая начинает работу алгоритма кластеризации называется distribute\_over\_matrix\_u(). Входными параметрами для неё являются список кластеризируемых объектов (объекты класса Point) и коэффициент неопределённости. Функция возвращает матрицу принадлежности и список centers, содержащий центры полученных кластеров. Количество кластеров задается вручную, в константе CLUSTERS\_NUM.

Шаги по нахождению новых центров кластеров и пересчёте матрицы принадлежности ограничены константами MAX\_EXECUTION\_CYCLES и EPSILON, где MAX\_EXECUTION\_CYCLES — ограничивает количество шагов, EPSILON — ограничивает точность нахождения матрицы принадлежности.

Далее представлено пошаговое описание реализованного алгоритма.

1) С помощью функции calculate\_centers рассчитываются центры кластеров.

2) Далее для каждого объекта рассчитывается Евклидово расстояние до центра каждого кластера. Для этой задачи определена функция evklid\_distance().

3) Рассчитывается коэффициент принадлежности u для данного объекта (функция prepare\_u()).

4) Нормализуются коэффициенты u для данного объекта (функция normalize\_u\_matrix\_row()).

5) Рассчитывается значение решающей функции (функция calculate\_decision\_dunction())

6) Далее сравнивается текущее значение решающей функции с предыдущим её значением, и если их разница меньше установленного EPSILON, то алгоритм прекращает работу.

Центры рассчитываются по следующей формуле:

где

— коэффициент принадлежности;

m — коэффициент неопределённости;

p — объект (в самом алгоритме в качестве p выступают составляющие координаты x и y).

Коэффициент принадлежности рассчитывается по формуле:

где

d — расстояние от объекта до центра кластера;

m — коэффициент неопределённости.

Нормализация всех коэффициентов принадлежности объекта — преобразование коэффициентов, чтобы в сумме они давали 1, т. е. фактически каждый коэффициент делится на сумму всех коэффициентов данного объекта.

Решающая функция возвращает сумму всех Евклидовых расстояний каждого объекта к каждому центру кластера, умноженному на коэффициент принадлежности.

Матрица принадлежности выводится в файл с помощью реализованного метода write\_in\_file(). Для анализа корректности работы алгоритма описана функция draw\_graph(). На вход ей подается список, содержащий центры полученных в ходе работы алгоритма кластеров и набор объектов, для которых эти центры были получены. Функция визуализирует полученные данные в виде графика.

## 3.2 Анализ работы программы для различных входных данных

Для проверки корректности работы реализованного алгоритма проведем следующие эксперименты для различных наборов входных данных. Данные для первого эксперимента представлены в таблице 1.

Таблица 1 – Набор точек для первого эксперимента

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| x1 | 1 | 1 | 1 | 3 | 3 | 3 | 5 | 7 | 9 | 11 | 11 | 11 | 13 | 13 | 13 |
| x2 | 1 | 4 | 7 | 2 | 4 | 6 | 4 | 4 | 4 | 2 | 4 | 6 | 1 | 4 | 7 |

Графическое изображение этих данных представляет собой "бабочку" ‑ хорошо известный в теории распознавания образов пример кластеризации. Установим количество кластеров, равное 2. График, полученный в результате работы алгоритма представлен на рисунке 3. Матрица принадлежности, полученная в результате работы программы представлена в таблице 2.

Таблица 2 – разбиение точек на кластеры для первого эксперимента

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x | y | Кластер 1 | Кластер 2 |
| 0 | 1 | 1 | 0.990 | 0.010 |
| 1 | 1 | 4 | 0.999 | 0.001 |
| 2 | 1 | 7 | 0.990 | 0.010 |
| 3 | 3 | 2 | 0.997 | 0.003 |
| 4 | 3 | 4 | 1.000 | 0.000 |
| 5 | 3 | 6 | 0.997 | 0.003 |
| 6 | 5 | 4 | 0.982 | 0.018 |
| 7 | 7 | 4 | 0.499 | 0.501 |
| 8 | 9 | 4 | 0.018 | 0.982 |
| 9 | 11 | 2 | 0.003 | 0.997 |
| 10 | 11 | 4 | 0.000 | 1.000 |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x | y | Кластер 1 | Кластер 2 |
| 11 | 11 | 6 | 0.003 | 0.997 |
| 12 | 13 | 1 | 0.010 | 0.990 |
| 13 | 13 | 4 | 0.001 | 0.999 |
| 14 | 13 | 7 | 0.010 | 0.990 |

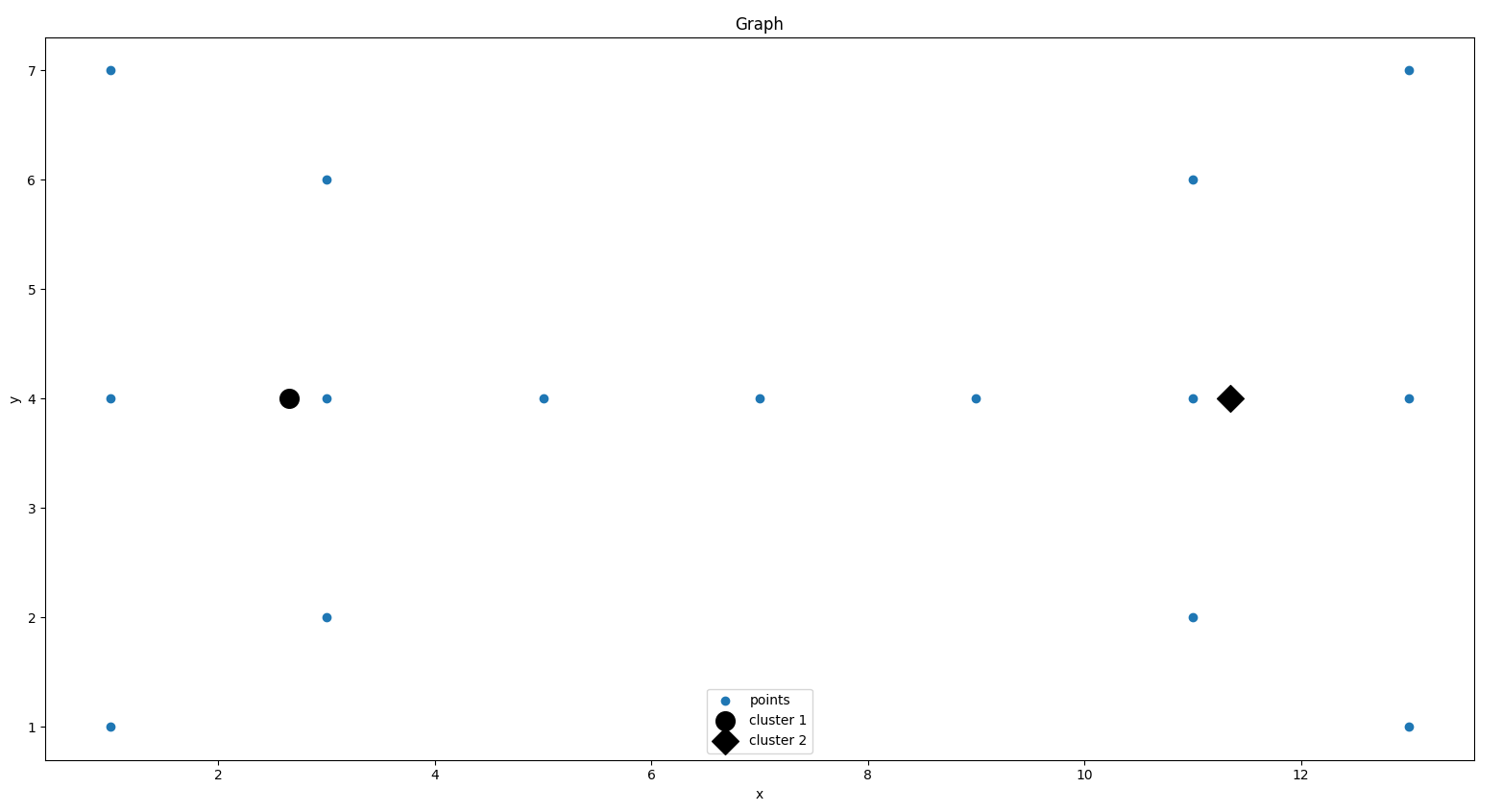


Рисунок 4 – График, полученный в результате первого эксперимента

Попробуем установить количество кластеров, равное 3. Результаты работы программы представлены на рисунке 4 и в таблице 3.

Таблица 3 - разбиение точек на кластеры для второго эксперимента

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | x | y | Кластер 1 | Кластер 2 | Кластер 3 |
| 0 | 1 | 1 | 0.006 | 0.045 | 0.949 |
| 1 | 1 | 4 | 0.000 | 0.001 | 0.999 |
| 2 | 1 | 7 | 0.006 | 0.045 | 0.949 |
| 3 | 3 | 2 | 0.003 | 0.051 | 0.945 |
| 4 | 3 | 4 | 0.000 | 0.003 | 0.997 |
| 5 | 3 | 6 | 0.003 | 0.051 | 0.946 |

Продолжение таблицы 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | x | y | Кластер 1 | Кластер 2 | Кластер 3 |
| 6 | 5 | 4 | 0.007 | 0.781 | 0.212 |
| 7 | 7 | 4 | 0.000 | 1.000 | 0.000 |
| 8 | 9 | 4 | 0.138 | 0.858 | 0.005 |
| 9 | 11 | 2 | 0.934 | 0.063 | 0.003 |
| 10 | 11 | 4 | 0.996 | 0.004 | 0.000 |
| 11 | 11 | 6 | 0.934 | 0.063 | 0.003 |
| 12 | 13 | 1 | 0.944 | 0.050 | 0.006 |
| 13 | 13 | 4 | 0.999 | 0.001 | 0.000 |
| 14 | 13 | 7 | 0.944 | 0.050 | 0.006 |

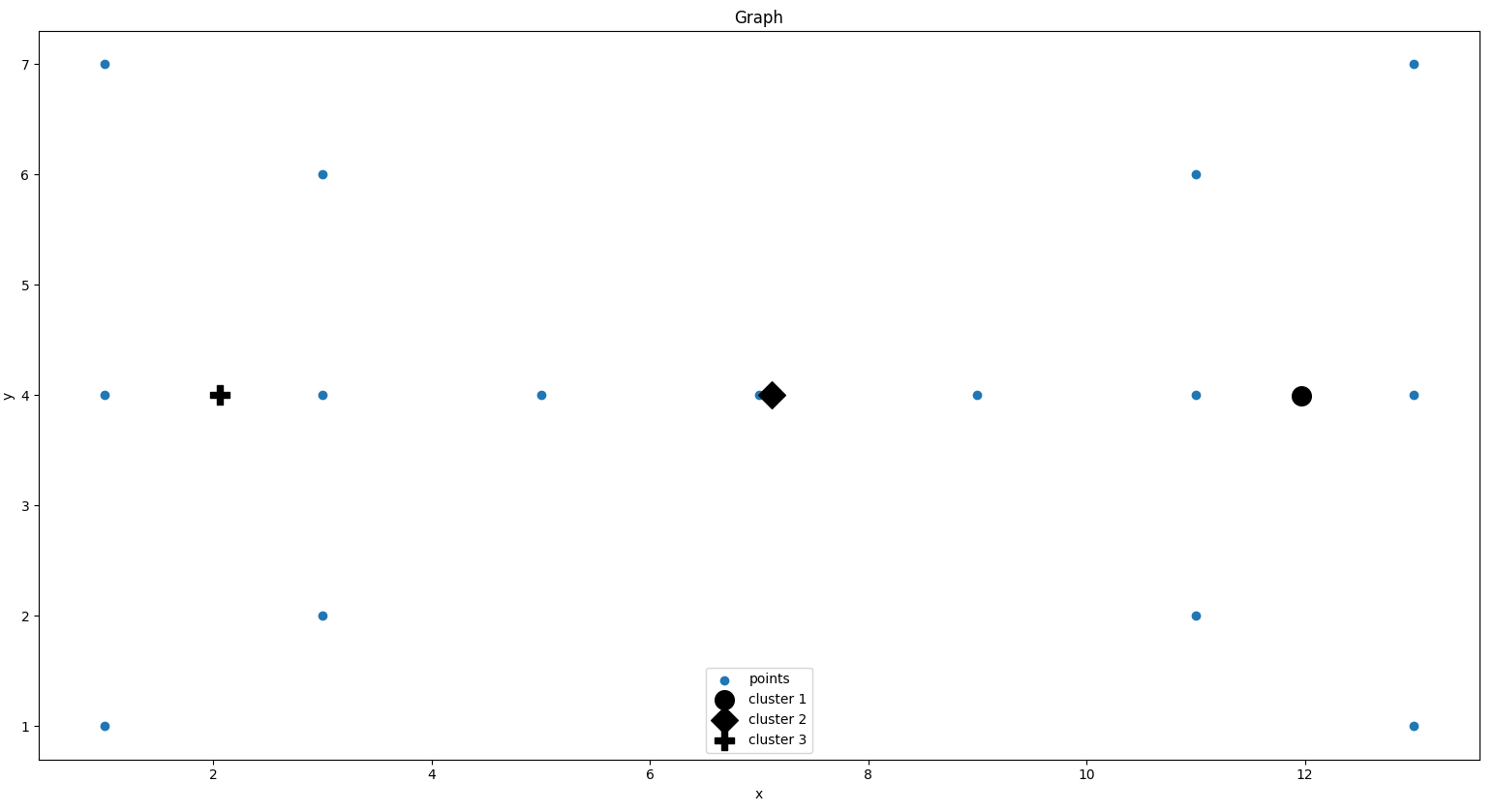


Рисунок 5 – График, полученный в результате второго эксперимента

Как видно из первых двух экспериментов, программа корректно устанавливает центры кластеров и определяет степень принадлежности точек кластерам при малом количестве данных.

Для последующих экспериментов были взяты наборы данных из 5000 точек. Количество кластеров для работы программы равно 15. Результаты работы программы представлены на рисунках 5 и 6.

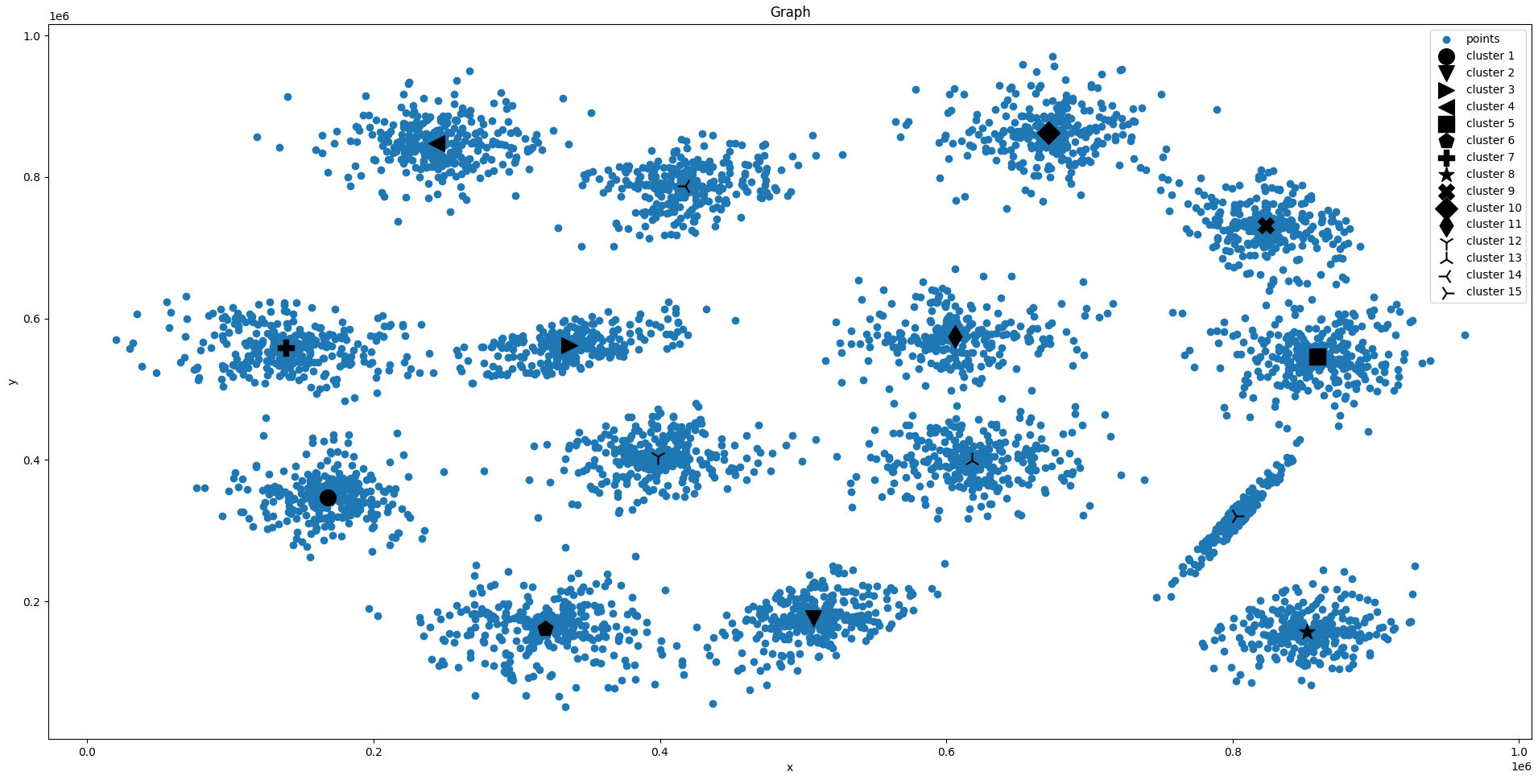


Рисунок 6 – График, полученный в результате третьего эксперимента

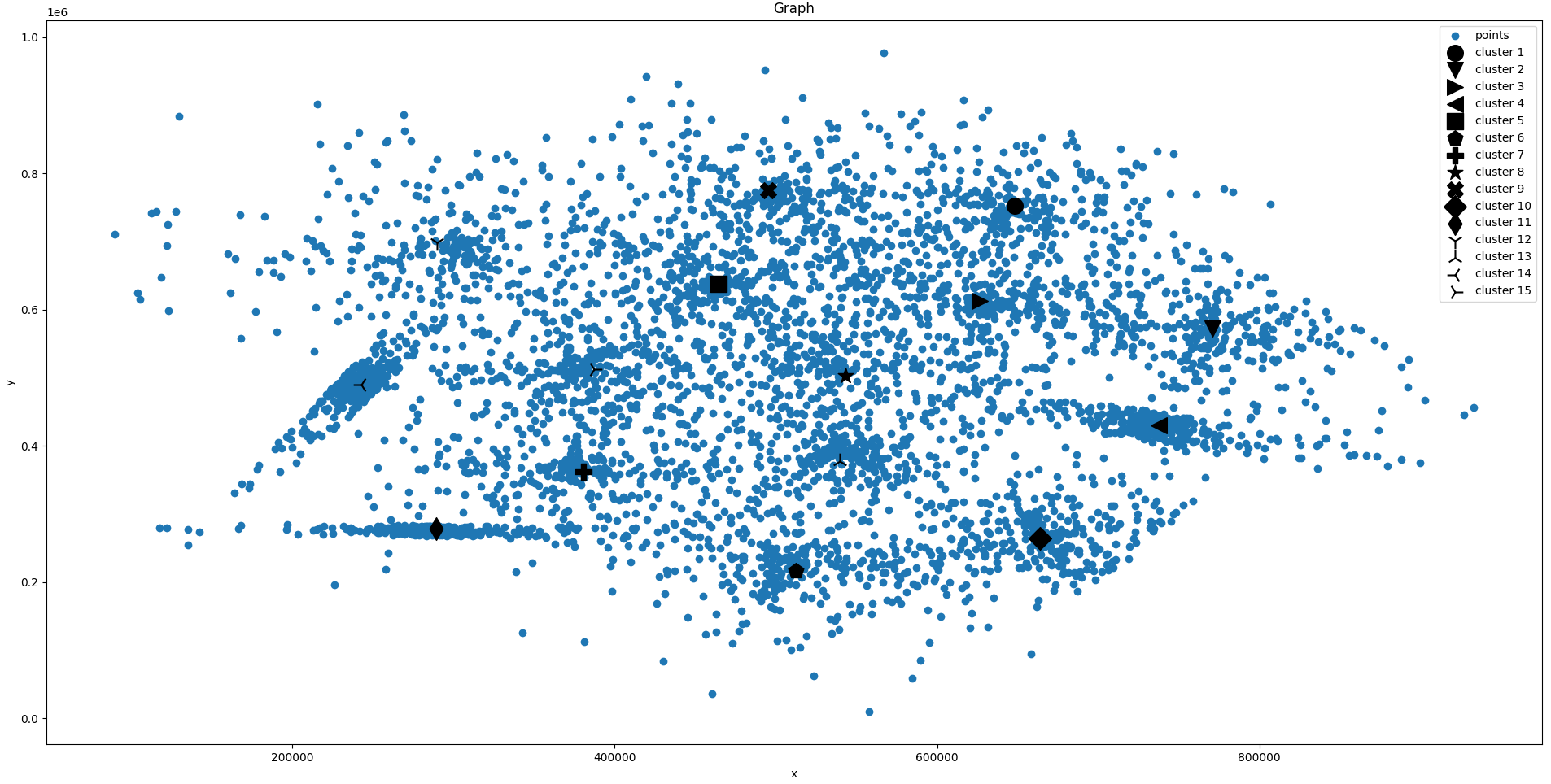


Рисунок 7 – График, полученный в результате четвертого эксперимента

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы был изучен и реализован на языке Python алгоритм нечеткой кластеризации c-средних для двумерных данных.

Проведенные эксперименты показали, что данная программная реализация способна корректно разбивать исходное множество объектов на кластеры, для дальнейшего анализа данных.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Анализ данных и процессов: учеб. пособие / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2009. — 512 с.
2. Кластеризация данных при помощи нечетких отношений в Data Mining [Электронный ресурс]. – URL: <https://ami.nstu.ru/~vms/lecture/data_mining/fuzzy.htm> (дата обращения 15.03.2021).
3. Алгоритмы кластеризации на службе Data Mining [Электронный ресурс]. – URL: <https://loginom.ru/blog/data-mining-clustering> (дата обращения 29.04.2021).
4. Алгоритм нечёткой кластеризации fuzzy c-means [Электронный ресурс]. – URL: <https://habr.com/ru/post/208496> (дата обращения 9.05.2021).
5. Набор данных для кластеризации [Электронный ресурс]. – URL: <http://cs.joensuu.fi/sipu/datasets> (дата обращения 13.05.2021).

# **ПРИЛОЖЕНИЕ А**

**Код программы**

import random as rdm

import pandas as pd

import csv

import matplotlib.pyplot as plt

EPSILON = 0.1

MAX\_EXECUTION\_CYCLES = 150

MIN\_X = 0

MAX\_X = 100

POINTS\_COUNT = 100

CLUSTERS\_NUM = 3

FUZZ = 1.5

MARKERS = (

'o',

'D',

'P',

'X',

'v',

's',

'\*',

'p',

'>',

'<',

'd',

'1',

'2',

'3',

'4',

)

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "({}, {})".format(self.x, self.y)

def \_\_round\_\_(self, n=None):

self.x = round(self.x, n)

self.y = round(self.y, n)

return self

# Алгоритм c-средних

def distribute\_over\_matrix\_u(points, m):

centers = generate\_random\_centers(CLUSTERS\_NUM, points)

matrix\_u = fill\_matrix(len(points),

len(centers))

print(\*points, sep='\n')

previous\_decision\_value = 0

current\_decision\_value = 1

for \_ in range(MAX\_EXECUTION\_CYCLES):

if not abs(previous\_decision\_value - current\_decision\_value) > EPSILON:

break

previous\_decision\_value = current\_decision\_value

centers = calculate\_centers(matrix\_u, m, points)

for key, uRow in enumerate(matrix\_u):

for cluster\_index, u in enumerate(uRow):

distance = evklid\_distance(points[key],

centers[cluster\_index])

matrix\_u[key][cluster\_index] = prepare\_u(distance, m)

matrix\_u[key] = normalize\_u\_matrix\_row(matrix\_u[key])

current\_decision\_value = calculate\_decision\_function(

points,

centers,

matrix\_u)

return matrix\_u, centers

def fill\_matrix(points\_count, clusters\_count):

matrix\_u = []

for i in range(points\_count):

matrix\_u.append([])

for j in range(clusters\_count):

matrix\_u[i].append(rdm.random())

matrix\_u[i] = normalize\_u\_matrix\_row(matrix\_u[i])

return matrix\_u

def calculate\_centers(matrix\_u, m, points):

matrix\_centroids = []

for clusterIndex in range(CLUSTERS\_NUM):

temp\_ax = 0

temp\_bx = 0

temp\_ay = 0

temp\_by = 0

for i in range(len(matrix\_u)):

temp\_ax += matrix\_u[i][clusterIndex] \*\* m

temp\_bx += matrix\_u[i][clusterIndex] \*\* m \* points[i].x

temp\_ay += matrix\_u[i][clusterIndex] \*\* m

temp\_by += matrix\_u[i][clusterIndex] \*\* m \* points[i].y

matrix\_centroids.append(

Point(temp\_bx / temp\_ax,

temp\_by / temp\_ay)

)

return matrix\_centroids

def calculate\_decision\_function(matrix\_point\_x, matrix\_centroids, matrix\_u):

summa = 0

for index, row in enumerate(matrix\_u):

for clusterIndex, u in enumerate(row):

summa += u \* evklid\_distance(

matrix\_centroids[clusterIndex],

matrix\_point\_x[index])

return summa

def evklid\_distance(point\_a, point\_b):

distance1 = (point\_a.x - point\_b.x) \*\* 2

distance2 = (point\_a.y - point\_b.y) \*\* 2

distance = distance2 + distance1

return distance \*\* 0.5

def normalize\_u\_matrix\_row(matrix\_row):

summa = sum(matrix\_row)

out = [num / summa for num in matrix\_row]

return out

def prepare\_u(distance, m):

return pow(1 / distance, 2 / (m - 1))

def generate\_random\_points(count):

points = []

for \_ in range(count):

points.append(

Point(rdm.randint(MIN\_X, MAX\_X),

rdm.randint(MIN\_X, MAX\_X))

)

return points

def generate\_random\_centers(count, points):

centers = []

points\_x = [pt.x for pt in points]

points\_y = [pt.y for pt in points]

min\_x = round(min(points\_x))

max\_x = round(max(points\_x))

min\_y = round(min(points\_y))

max\_y = round(max(points\_y))

for \_ in range(count):

centers.append(Point(

rdm.randint(min\_x, max\_x),

rdm.randint(min\_y, max\_y)

))

return centers

def write\_in\_file(matrix, points, file\_name, x\_label='x', y\_label='y'):

points\_x = [pt.x for pt in points]

points\_y = [pt.y for pt in points]

matrix\_t = list(zip(\*matrix))

data = {

x\_label: points\_x,

y\_label: points\_y,

}

for i in range(CLUSTERS\_NUM):

data['cluster {}'.format(i + 1)] = matrix\_t[i]

df = pd.DataFrame(data)

with open(file\_name, 'w') as file:

file.write(df.to\_string())

def generate\_random\_points\_in(count, mix\_x, max\_x, min\_y, max\_y):

points = []

for \_ in range(count):

points.append(Point(rdm.randint(mix\_x, max\_x),

rdm.randint(min\_y, max\_y)))

return points

def draw\_graph(points, centers, x\_label='x', y\_label='y'):

points\_x = [pt.x for pt in points]

points\_y = [pt.y for pt in points]

plt.title('Graph')

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.scatter(

points\_x,

points\_y,

label='points')

for index, center in enumerate(centers):

plt.scatter(

center.x,

center.y,

s=200,

c='k',

marker=MARKERS[index % len(MARKERS)],

label='cluster {}'.format(index + 1))

plt.legend()

plt.show()

def read\_points\_from\_txt\_file(file\_name):

points = []

with open(file\_name, 'r') as file:

for line in file.readlines():

x\_and\_y = line.split()

points.append(

Point(

int(x\_and\_y[0]),

int(x\_and\_y[1])

)

)

# print(\*points, sep='\n')

return points

def read\_points\_from\_csv\_file(file\_name):

points = []

with open(file\_name, 'r') as file:

reader = csv.DictReader(file)

for row in reader:

if row['Gender'] == 'Male':

points.append(

Point(

float(row['Height']),

float(row['Weight']),

)

)

return points

def c\_means\_points(file\_name=None):

if file\_name:

points = read\_points\_from\_txt\_file(file\_name)

else:

points = generate\_random\_points(POINTS\_COUNT)

matrix\_u, centers = distribute\_over\_matrix\_u(points, FUZZ)

# Округление выходных данных

matrix\_u = [[round(num, 3) for num in row] for row in matrix\_u]

# Выгрузка в файл

write\_in\_file(matrix\_u,

points,

'output.txt')

print(\*centers, sep='\n')

draw\_graph(points, centers)

def c\_means\_w\_h():

points = read\_points\_from\_csv\_file('weight-height.csv')

matrix\_u, centers = distribute\_over\_matrix\_u(points, FUZZ)

# Округление выходных данных

matrix\_u = [[round(num, 3) for num in row] for row in matrix\_u]

points = [Point(point.x\*2.54, point.y\*0.4536) for point in points]

centers = [Point(point.x\*2.54, point.y\*0.4536) for point in centers]

x\_label = 'Рост'

y\_label = 'Вес'

# Выгрузка в файл

write\_in\_file(matrix\_u,

points,

'output.txt',

x\_label=x\_label,

y\_label=y\_label,

)

draw\_graph(points,

centers,

x\_label=x\_label,

y\_label=y\_label,

)

centers = [round(center) for center in centers]

for index, center in enumerate(centers):

print('cluster {} center -> {}'.format(index+1, center))

if \_\_name\_\_ == '\_\_main\_\_':

# c\_means\_w\_h()

c\_means\_points('input\_data\_5.txt')