**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

Курсовая РАБОТА

**по дисциплине «Статистические методы обработки**

**экспериментальных данных»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Фиалковский М.С. |
| Преподаватель |  | Середа В.И. |

Санкт-Петербург

2020

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Фиалковский М.С. | | |
| Группа 6381 | | |
|  | | |
| Исходные данные:  Генеральная совокупность из 400 пар значений, размер выборки для последующего анализа | | |
| Содержание пояснительной записки:  Содержание, Введение, Основные теоретические положения, Ход работы, Заключение, Список использованных источников | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 07.04.2020 | | |
| Дата сдачи работы: 14.04.2020 | | |
| Дата защиты работы: | | |
| Студент гр. 6381 |  | Фиалковский М.С. |
| Преподаватель |  | Середа В.И. |

**Аннотация**

В данной работе рассматриваются и применяются на практике различные методы статистической обработки экспериментальных данных: первичная обработка выборки, вычисление точечных и интервальных оценок параметров распределения, проверка статистической гипотезы согласия с нормальным законом распределения с помощью критерия Пирсона, корреляционный, регрессионный анализ и кластерный анализы.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
|  | Основные теоретические положения | 6 |
| 1. | Выравнивание статистических рядов |  |
| 1.1. | Первичная обработка выборки по первому признаку (1) | 18 |
| 1.2 | Нахождение выборочных оценок параметров распределения | 20 |
| 1.3 | Определение доверительных интервалов основных оценок | 22 |
| 1.4 | Проверка гипотезы о нормальном распределении по критерию Пирсона | 23 |
| 2. | Корреляционный и регрессионный анализ |  |
| 2.1. | Первичная обработка выборки по второму признаку | 25 |
| 2.2. | Нахождение выборочных оценок параметров распределения | 27 |
| 2.3. | Исследование корреляции признаков | 28 |
| 2.4 | Построение выборочных прямых среднеквадратической регрессии | 30 |
| 2.5 | Нахождение оценок корреляционного отношения | 32 |
| 2.6 | Построение выборочных кривых параболической среднеквадратической регрессии | 33 |
| 3. | Кластерный анализ |  |
| 3.1. | Метод k-средних | 35 |
| 3.2. | Метод поиска сгущений | 37 |
|  | Заключение | 41 |
|  | Список использованных источников | 43 |
|  | Приложение А. Исходный код | 44 |

**введение**

Цель работы заключается в применении и закреплении на практике навыков первичной статистической обработки экспериментальных данных, корреляционного и кластерного анализа. В работе объединяются темы, затронутые в течение всего семестрового курса.

**Основные теоретические положения**

Совокупность взаимно независимых реализаций случайной величины образует *выборку* объема :

где – числовая реализация случайной величины в -ом эксперименте (.

Случайная величина , реализации которой наблюдаются, называют *генеральной совокупностью*.

*Репрезентативной* называется выборка, отражающая значимые для исследования характеристики генеральной совокупности.

*Ранжированный ряд* – это распределение отдельных единиц выборки в порядке возрастания или убывания исследуемого признака.

*Вариационный ряд* – это статистический ряд, показывающий распределение изучаемого явления по величине какого-либо количественного признака.

*Варианта* – отдельные значения признака, по которому производится группировка.

*Частота* – число, показывающее, как часто встречается та или иная варианта. Сумма всех частот равно общему числу наблюдений.

*Интервальный ряд* – это таблица, состоящая из двух столбцов (строк) – интервалов значений признака и числа единиц выборки, попадающих в данный интервал.

Число групп в интервальном ряду определяют по формуле Стерджесса:

Тогда шаг, с которым строится интервальный ряд вычисляется по формуле:

*Гистограмма* – это наглядное представление функции вероятности некоторой случайной величины, построенное по выборке. Гистограмма строится с помощью интервального ряда.

*Полигон частот* – это один из способов графического представления плотности вероятности распределения выборки.

*Эмпирической функцией* распределения, построенной по выборке объема , называется случайная функция , при каждом равная

*Начальным моментом* -го порядка случайной величины называется математическое ожидание -ой степени случайной величины , т.е. величина .

*Центральным моментом* -го порядка случайной величины называется величина , определяемая формулой .

Начальный момент первого порядка равен математическому ожиданию, а центральный момент второго порядка равен дисперсии случайной величины.

Существуют формулы, позволяющие выразить центральные моменты случайной величины через ее начальные моменты:

Центральный момент , будучи соответствующим образом нормализован, является коэффициентом асимметрии распределения:

Если , то имеет место левая асимметрия графика функции плотности распределения, а если , то имеет место правая асимметрия. При этом соответствует симметричному графику плотности нормального распределения.

Коэффициент эксцесса получается из четвертого центрального момента

Если , то график функции плотности распределения имеет более острый пик, в случае – наоборот. А случай соответствует нормальному распределению.

Оценку параметра называют *точечной*, если она выражается одним числом. Любая точечная оценка, вычисленная на основании опытных данных, является их функцией и поэтому сама должна представлять собой случайную величину с распределением, зависящим от распределения исходной случайной величины, от самого оцениваемого параметра и от числа опытов .

К точечным оценкам предъявляется ряд требований, определяющих их пригодность для описания самих параметров.

* Оценка называется *состоятельной*, если при увеличении числа наблюдений она приближается (сходится по вероятности) к значению оцениваемого параметра.
* Оценка называется *несмещенной*, если ее математическое ожидание равно оцениваемому параметру.
* Оценка называется *эффективной*, если ее дисперсия меньше дисперсии любой другой оценки данного параметра.

Оценки моментов можно получить с использованием следующих формул для условных вариант

Выборочные оценки начальных моментов:

Выборочные оценки центральных моментов:

Отсюда получаем формулы для точечных оценок параметров распределения:

* Оценка математического ожидания
* Смещенная оценка дисперсии
* Оценка среднеквадратического отклонения
* Исправленная оценка дисперсии
* Исправленная оценка СКО
* Оценка асимметрии
* Оценка эксцесса

Оценка неизвестного параметра называется *интервальной*, если она определяется двумя числами – концами интервала.

Интервал , накрывающий с вероятностью истинное значение параметра , называется *доверительным интервалом*, а вероятность – *надежностью* оценки или *доверительной вероятностью*.

Число характеризует точность оценки: чем меньше разность , тем точнее оценка.

*Доверительные интервалы для параметров нормального распределения.*

Доверительный интервал заданной надежности для математического ожидания имеет следующий вид:

где – оценка математического ожидания, – исправленная оценка среднеквадратического отклонения, – объем выборки, – квантиль распределения Стьюдента уровня .

Доверительный интервал заданной надежности для среднеквадратического отклонения имеет следующий вид:

где – определяется по таблице по заданным и .

*Критерий согласия Пирсона* (критерий согласия ) позволяет проверить гипотезу о том, что случайная величина подчиняется определенному закону распределения. Данный критерий основан на сравнении теоретической плотности распределения и гистограммы.

Для того, чтобы проверить гипотезу о том, что выборка взяла из генеральной совокупности , имеющей распределение, заданное функцией распределения необходимо множество значений разбить на интервалы:

Для каждого интервала необходимо вычислить наблюдаемые частоты , а также теоретические (ожидаемые) частоты При этом интервалы должны быть сформированы таким образом, чтобы ожидаемые частоты удовлетворяли условию .

Наблюдаемые частоты равны количеству элементов выборки, попавших в соответствующий -ый интервал. Теоретические частоты вычисляются по формуле , где – объем выборки, а - вероятность попадания в -ый интервал, вычисляемая следующим образом:

где

а , где - граница интервала, – оценка математического ожидания, – исправленная оценка среднеквадратического отклонения.

Статистика критерия имеет вид:

Статистика критерия имеет распределение c степенями свободы, где – число интервалов, а – число неизвестных параметров распределения. В случае для нормального распределения .

Для проверки гипотезы необходимо сравнить значение статистики критерия, полученное по выборке с критическим значением для степенями свободы и на уровне значимости .

В случае, если , то гипотеза отвергается в пользу гипотезы , в противном случае у нас не достаточно оснований, чтобы отвергнуть нулевую гипотезу.

*Коэффициент корреляции Пирсона* характеризует существование линейной зависимости между двумя величинами.

Пусть даны два интервальных ряда и тогда точечная оценка коэффициента корреляции Пирсона рассчитывается по формуле:

Где , – количество интервалов в интервальных рядах для выборок значений случайных величин и соответственно; , – условные варианты, значения которых соответствуют выборочным (интервальным) значениям случайных величин и соответственно; – общий объем выборки.

Можно перейти к условным вариантам, тогда формула приобретает следующий вид:

*Построение доверительного интервала для коэффициента корреляции Пирсона.*

Распределение выборочного коэффициента корреляции Пирсона сложное, поэтому используют преобразование Фишера:

Для полученного среднеквадратическое отклонение равно , где – объем выборки.

Доверительный интервал имеет следующий вид:

Выполнив обратное преобразование, получим доверительный интервал для коэффициента корреляции Пирсона:

где – левая граница доверительного интервала для , а – правая граница доверительного интервала для .

*Проверка гипотезы о равенстве коэффициента корреляции Пирсона нулю.*

Основная и альтернативная гипотезы имеет следующий вид:

Статистика критерия по выборке вычисляется по следующей формуле:

Статистика критерия имеет распределение Стьюдента с степенями свободы. Для заданного уровня значимости по таблице находят критическое значение .

Тогда если , то имеется достаточно оснований, чтобы отклонить гипотезу на заданном уровне значимости. В противном случае, оснований недостаточно.

*Уравнения выборочных прямых среднеквадратической регрессии.*

Пусть имеется двумерная выборка , где и – зависимы.

Уравнения выборочных прямых среднеквадратической регрессии имеют следующий вид:

Где , – выборочные оценки математических ожиданий; , – выборочные оценки среднеквадратических отклонений; – выборочная оценка коэффициента корреляции.

, обозначают статистические оценки условных математических ожиданий как функций значений и соответственно.

*Корреляционное отношение*

Пусть выборочные данные для случайных величин и представлены корреляционной таблицей:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Тогда можно вычислить оценки дисперсий и .

И соответственно оценки среднеквадратических отклонений:

Тогда выборочное корреляционное отношение к вычисляется следующим образом:

Аналогично может быть вычислено выборочное корреляционное отношение к :

*Уравнения выборочных кривых для параболической среднеквадратической регрессии*

Выборочное уравнение регрессии на имеет вид:

По методу наименьших квадратов значения коэффициентов и определяются из системы линейных уравнений следующего вида:

Здесь – сумма частот в -ой строке корреляционной таблицы; - условные выборочные средние (групповые средние) значения при ,

Аналогично может быть получено уравнение параболической регреcсии на .

*Задача кластерного анализа* заключается в том, чтобы на основании данных, характеризующих исследуемые объекты, содержащиеся во множестве , разбить исходные множество объектов на кластеров так, чтобы каждый объект принадлежал одному и только одному подмножеству разбиения. Объекты, принадлежащие каждому из кластеров, должны удовлетворять при этом некоторой заданной мере сходства.

В качестве расстояния между объектами в данной работе используется евклидово расстояние:

*Описание алгоритма k-средних.*

Пусть имеется наблюдений, каждое из которых характеризуется признаками . Эти наблюдения необходимо разбить на кластеров.

*Инициализация.* Из наблюдений случайным образом выбираются или задаются исследователем точек. Выбранные точки принимаются за начальные приближения центров кластеров. Каждому центру кластера присваивается порядковый номер, который одновременно является и номером кластера.

На каждом шаге алгоритма для каждого из объектов, проверяется, к какому центру он ближе всего. В соответствии с этим ему сопоставляется номер кластера. После чего значения центров кластеров обновляются.

Критерием остановки может служить заданное количество итераций или то, что составы кластеров перестали изменяться.

Возможны две модификации алгоритма *k-*средних:

* В первой модификации центр кластера пересчитываются после каждого изменения его состава.
* Во второй модификации центры кластеров пересчитываются после просмотра всех объектов из выборки.

Для оценки качества кластеризации используются функционалы качества:

1. Сумма квадратов расстояний до центров кластеров (

Наилучшим разбиением является такое, при котором значение было бы минимальным.

1. Сумма внутрикластерных расстояний между объектами ()

Наилучшим разбиением следует считать такое, при котором достигается минимальное значение , т.е. получены кластеры большой «плотности».

1. Сумма внутрикластерных дисперсий.

Наилучшим разбиением является такое, при котором сумма внутрикластерных дисперсий минимальна.

Одним из итеративных методов кластеризации является *метод поиска сгущений.* Суть алгоритма, реализующего данный метод, заключается в применении гиперсферы (в данной работе – окружности) заданного радиуса, которая перемещается в пространстве признаков с целью поиска локальных сгущений объектов.

На первом шаге центром сферы служит объект, в ближайший окрестности которого расположено наибольшее число соседей. На основании заданного радиуса сферы определяется совокупность точек внутри этой сферы, и для них вычисляются координаты центра. Положение сферы пересчитывается, либо после каждого включения в нее нового элемента, либо после того, как для текущего положения центра сферы определены все, входящие в нее элементы.

Когда очередной пересчет координат при водит к такому же результату, как и на предыдущем шаге, перемещение сферы прекращается, а точки, попавшие в нее, образуют кластер, и из дальнейшего процесса кластеризации исключаются.

Рассмотренная процедура повторяется для оставшихся точек.

Работа алгоритма завершается за конечное число шагов, когда все точки оказываются распределенными по кластерам. Число образовавшихся кластеров заранее неизвестно и зависит от радиуса сферы.

**1. Выравнивание статистических рядов**

**1.1. Первичная обработка данных по первому признаку**

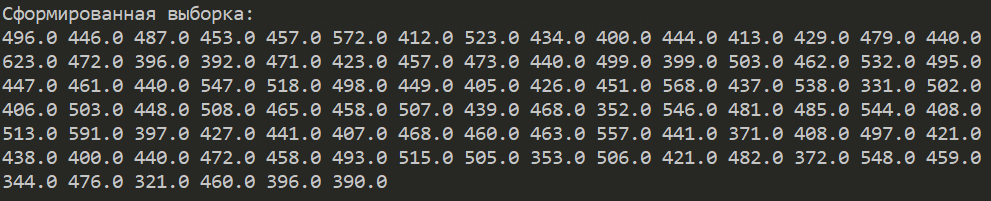
Произведём выборку из представленной генеральной совокупности экспериментальных данных размером в 96 элементов для первого признака. Результат на рис. 1. Для обеспечения получения одинаковых результатов при множественных запусках программы передаём генератору случайных чисел заранее определенное зерно. Код программы представлены в приложении А.

Рис. 1. Начальная выборка

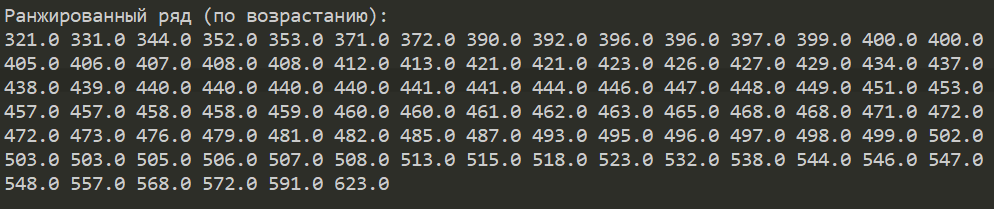
Ранжированный ряд представлен на рис.2.

Рис. 2. Ранжированный ряд

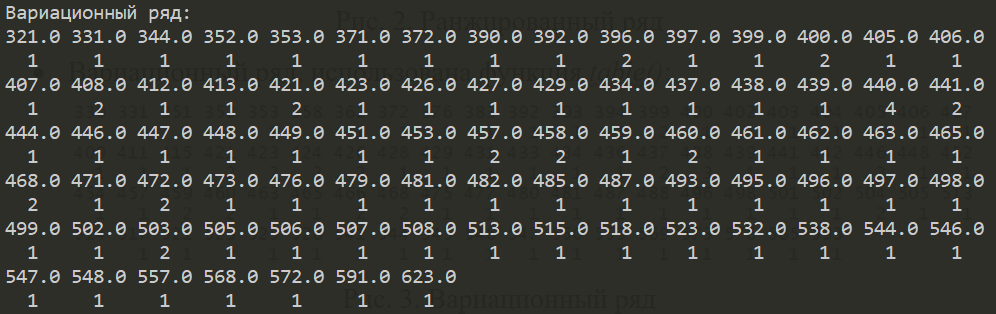
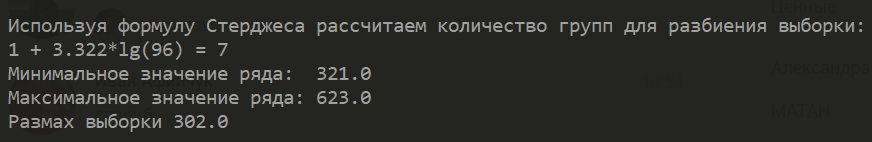
Вариационный ряд представлен на рис.3.

Рис. 3. Вариационный ряд

Для построения интервальный ряда требуется рассчитать количество различных групп. Требуемые расчёты представлены на рис. 4. Построенный интервальный ряд представлен на рис. 5.

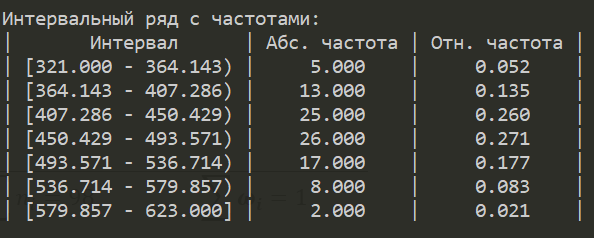
Рис. 4. Расчёты для интервального ряда

Рис. 5. Интервальный ряд с частотами (сумма отн. частот равна 1)

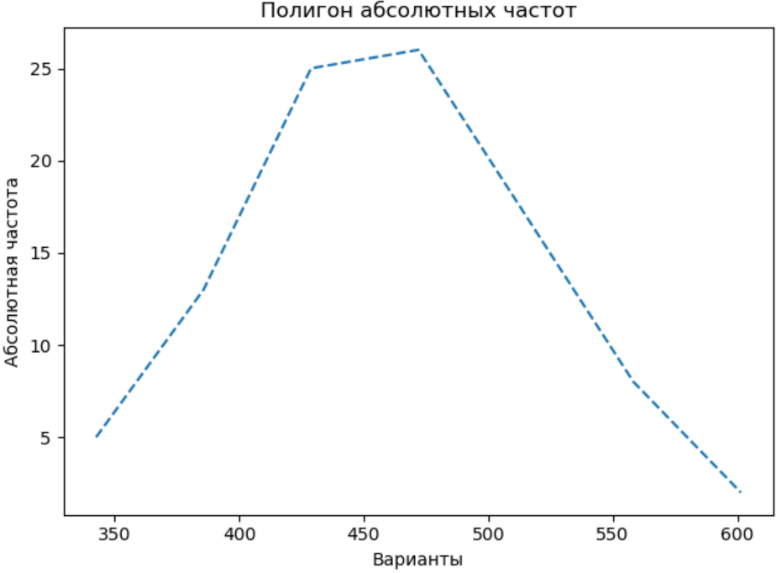
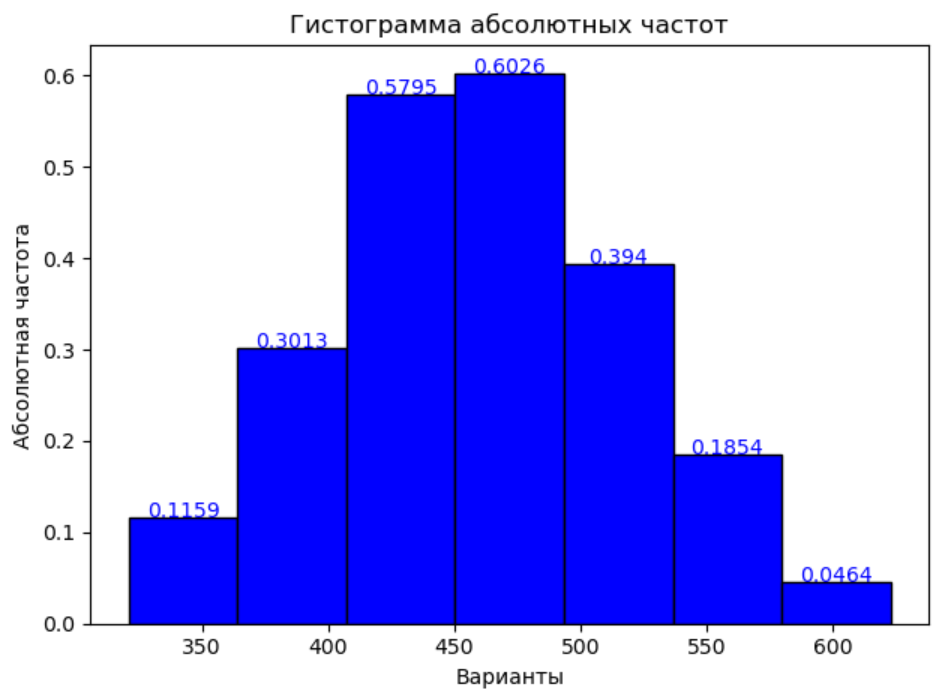
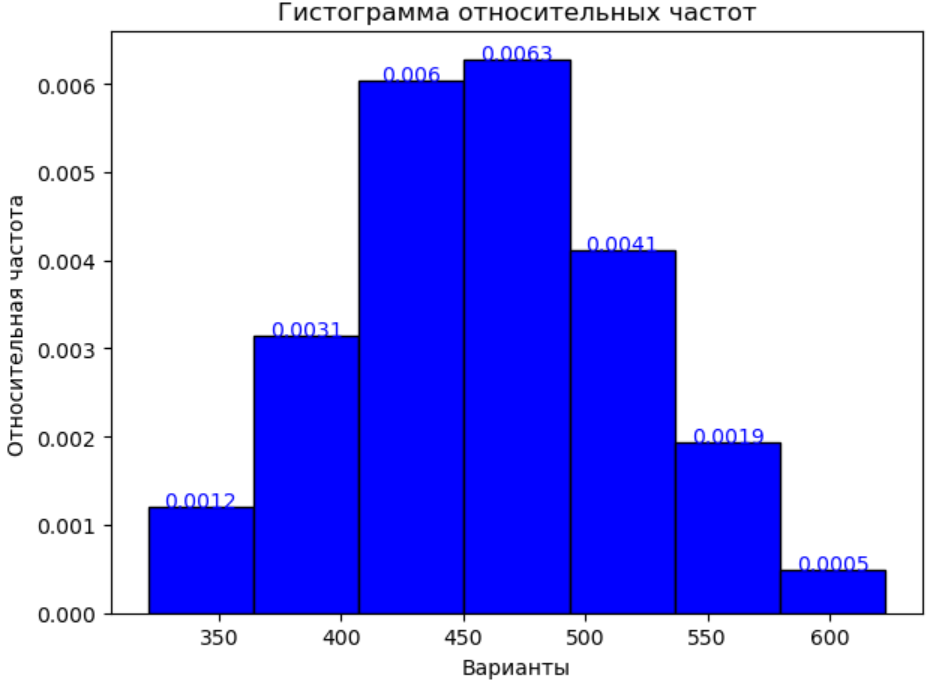
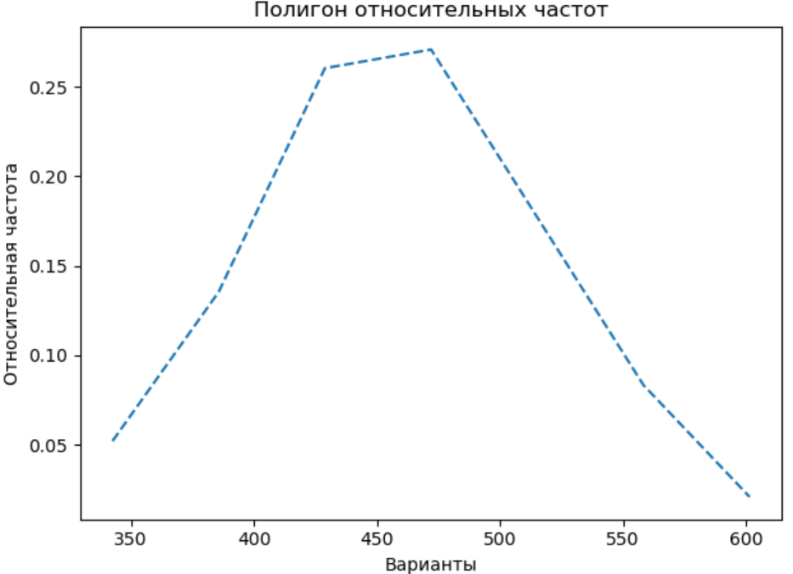
Визуализируем полученный интервальный ряд, построив гистограмму и полигон абсолютных (рис. 6) и относительных частот (рис. 7).

Рис. 6. Гистограмма и полигон абсолютных частот

Рис. 7. Гистограмма и полигон относительных частот

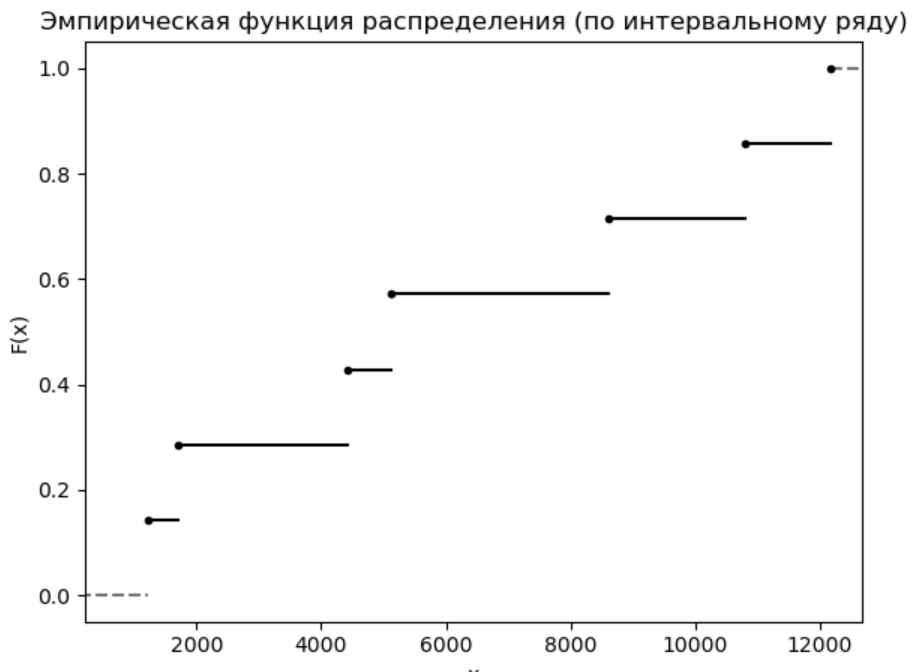
Далее построим эмпирическую функцию распределения по интервальному ряду. Результат на рис. 8.

Рис. 8. Эмпирическая функция распределения (по интервальному ряду)

**1.2. Нахождение выборочных оценок параметров распределения**

Для построения точечных оценок параметров составим таблицу для вычисления моментов.

Ведём следующее обозначение: - условная варианта.

Шаг интервального ряда . В качестве значения параметра используется середина 4 интервала в интервальном ряду, т.е. .

Правильность составления таблицы можно проверить, используя следующую формулу:

Таблица для вычисления моментов первого признака

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | 272 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Проверим программно правильность построения. Результат на рис. 9.

Рис. 9. Исходные данные

Можем сделать вывод, что таблица рассчитана правильно.

Начальные выборочные моменты с первого по четвертый:

Далее будем вычислять центральные выборочные моменты со второго по четвертый:

Теперь вычислим оценки параметров исследуемого распределения первого признака.

* Оценка математического ожидания
* Смещенная оценка дисперсии
* Оценка среднеквадратического отклонения
* Исправленная оценка дисперсии
* Исправленная оценка СКО
* Оценка асимметрии
* Оценка эксцесса

**1.3. Определение доверительных интервалов основных оценок**

Найдем границы доверительного интервала для математического ожидания случайной величины первого признака, предположительно имеющей нормальное распределение, в случае неизвестной дисперсии.

Воспользуемся формулой подставив туда уже найденный для первого признака значения:

Используя таблицу и параметры и получим   
 Получим:

Далее будем находить границы доверительного интервала для СКО. Для этого по таблице найдем для и . Получим , проинтерполировав значение для 90 и 100. Будем использовать формулу

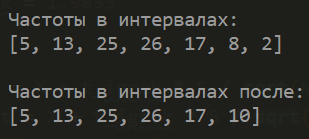
и получим*:*

**1.4. Проверка гипотезы о нормальном распределении по критерию Пирсона**

Теперь проверим гипотезу о согласии с нормальным распределением с параметрами (459.91, 59.42). Для этого воспользуемся критерием для проверки сложной гипотезы. При этом проверяемые гипотезы будут формулироваться следующим образом:

выборка взята из нормального распределения с параметрами и .

:выборка взята не из нормального распределения с параметрами и .

Для использования критерия для проверки гипотезы сформируем интервалы таким образом, чтобы ожидаемые частоты в них были не менее Объединим последние 2 интервала. Получим:

На основании данных, полученных при расчёте значений критерия, построим таблицу для 6 интервалов (ниже).

Таким образом:

статистика критерия:

Таблица ля расчёта статистики критерия первого признака

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | 1.29 |  |  |  |  |
| 536.71 | 623.00 | 1.29 |  | 10 | 0.10 | 9.41 | 0.0364 |

Критическое значение статистики критерия на уровне значимости  
 и при степенях свободы равно .

Получаем, что . Поэтому принимаем гипотезу для первого признака.

**2. Корреляционный и регрессионный анализ**

**2.1. Первичная обработка данных по второму признаку**

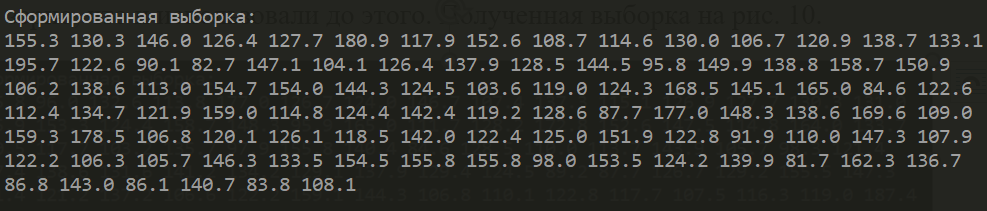
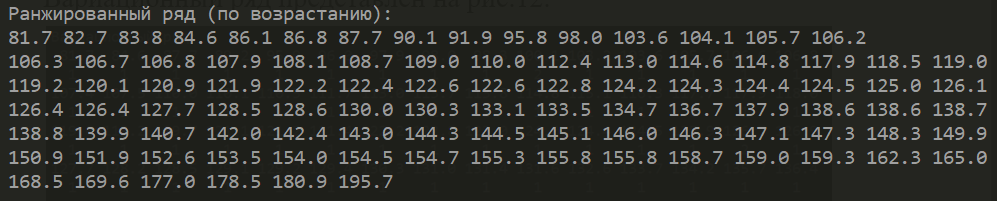
Теперь произведём выборку из представленной генеральной совокупности экспериментальных данных размером в 96 элементов для 2 признака и проведём аналогичные первой части исследования. Для обеспечения получения согласованных с выборкой по 1 признаку результатов при множественных запусках программы передаём генератору случайных то же зерно, что и использовали до этого. Полученная выборка на рис. 10.

Рис. 10. Начальная выборка 2 признака

Ранжированный ряд, полученный из выборки, представлен на рис. 11.

Рис. 11 – Ранжированный ряд второго признака

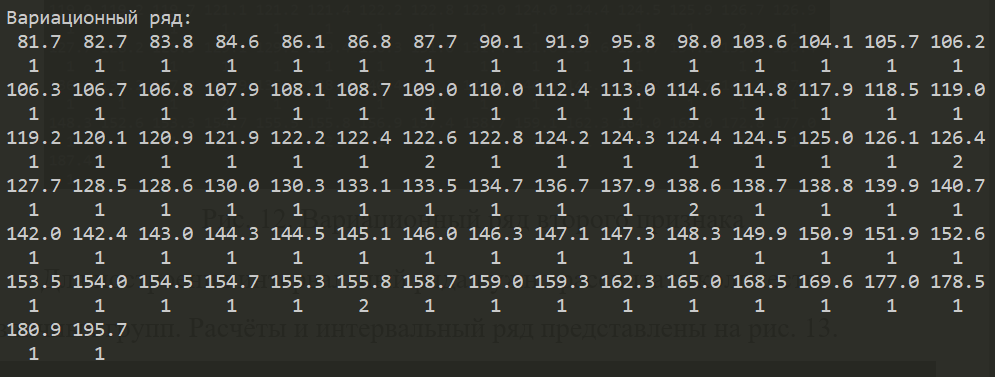
Вариационный ряд представлен на рис.12.

Рис. 12. Вариационный ряд второго признака

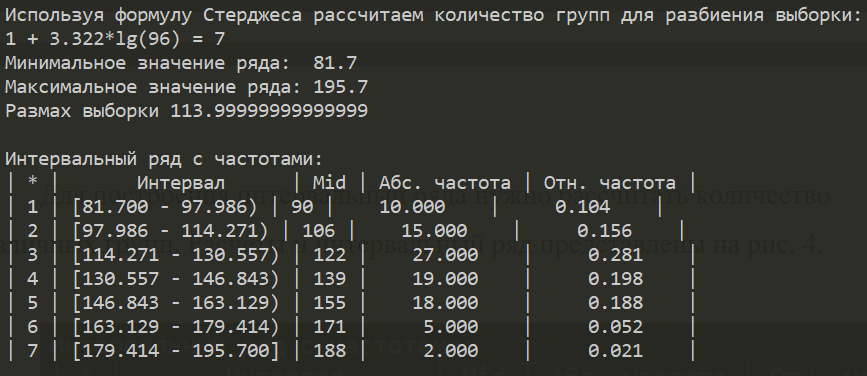
Для построения интервальный ряда нужно рассчитать количество различных групп. Расчёты и интервальный ряд представлены на рис. 13.

Рис. 13. Расчёты для интервального ряда второго признака

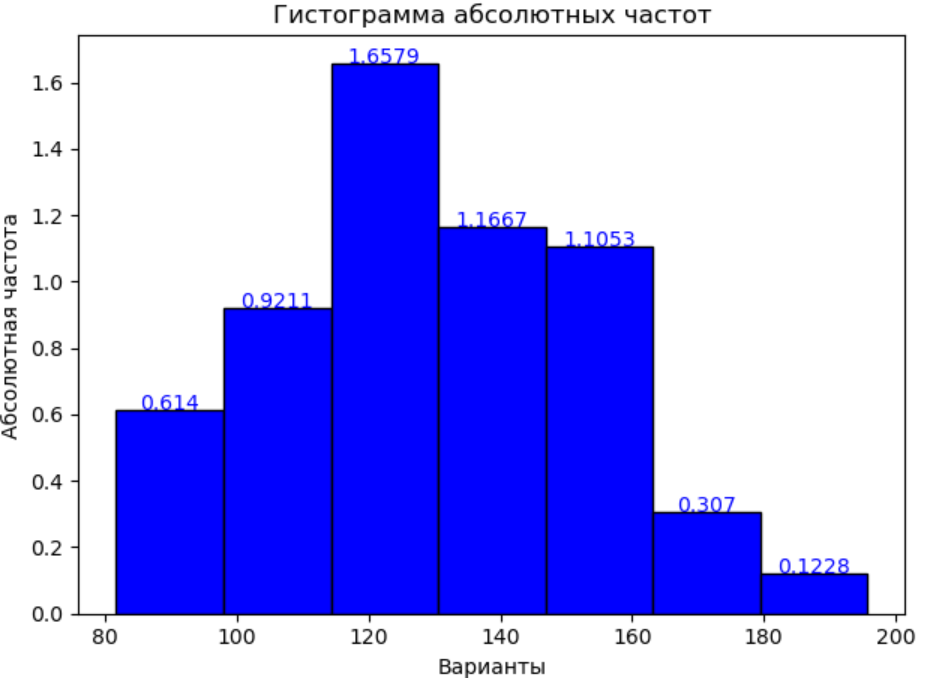
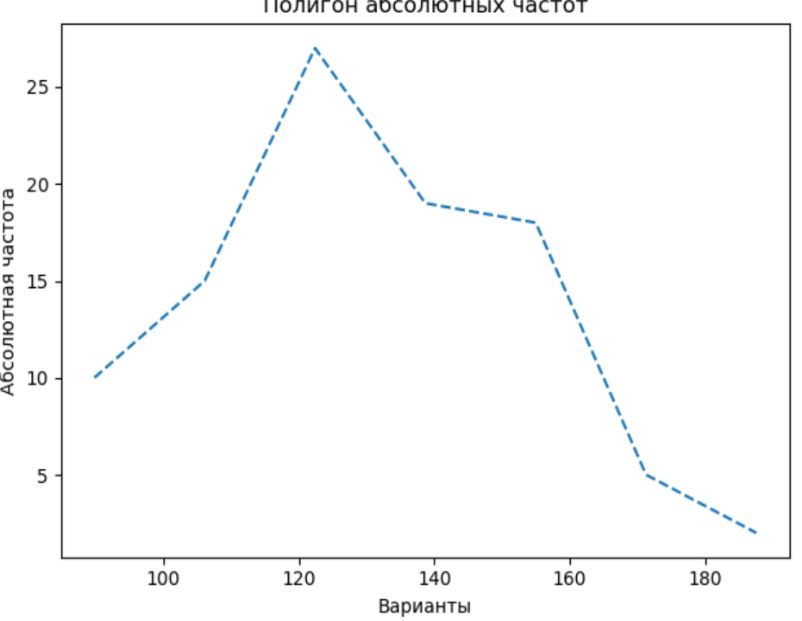
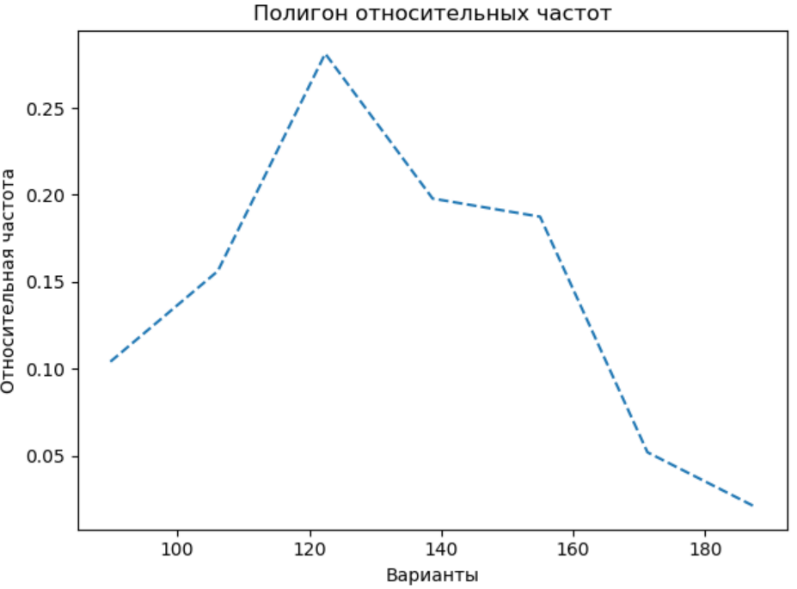
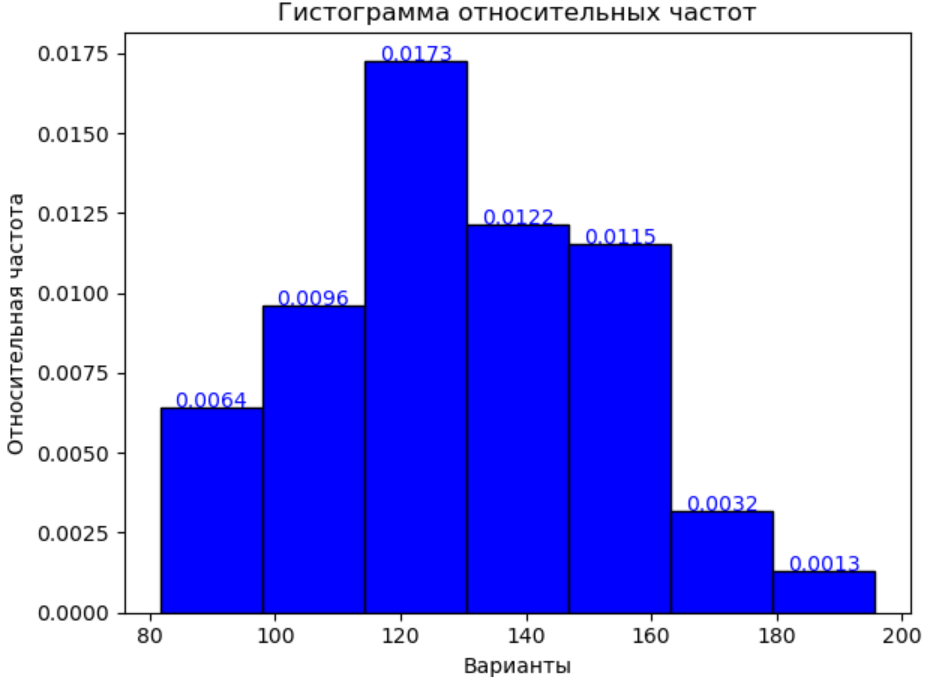
Визуализируем полученный интервальный ряд, построив гистограмму и полигон абсолютных (рис. 14) и относительных (рис. 15) частот.

Рис. 14. Гистограмма и полигон абсолютных частот

Рис. 15. Гистограмма и полигон относительных частот

**2.2. Нахождение выборочных оценок параметров распределения**

Далее для оценки параметров распределения построим таблицу (ниже), используя построенный только что интервальный ряд. Условные варианты вычислены по формуле , где , .

Таблица для вычисления моментов первого признака:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Правильность составления таблицы проверим по этой формуле:

Равенство выполняется, поэтому таблица построена верно.

По таблице найдем точечные оценки параметров распределения второго признака, опустив выкладки вычислений выборочных и центральных выборочных моментов.

* Оценка математического ожидания
* Смещенная оценка дисперсии
* Оценка среднеквадратического отклонения
* Исправленная оценка дисперсии
* Исправленная оценка СКО
* Оценка асимметрии
* Оценка эксцесса

**2.3. Исследование корреляции признаков**

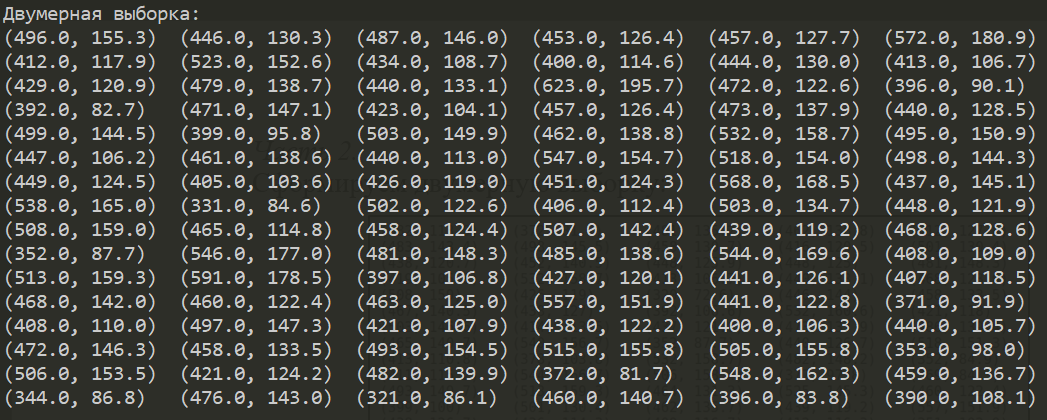
Для исследования корреляции формируем двумерную выборку (рис. 16).

Рис. 16. Полученная двумерная выборка.

И строим корреляционную таблицу:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Зафиксируем результаты промежуточных вычислений. Оценки математического ожидания условных вариант:

Оценки стандартных отклонений уловных вариант:

Оценка коэффициента корреляции:

Теперь построим доверительный интервал для коэффициента корреляции. Для этого построим интервал для :

После расчётов получим:

Воспользуемся обратным преобразованием для перехода к доверительному интервалу для коэффициента корреляции:

Итого после расчётов получим доверительный интервал, покрывающий истинное значение параметра с 95% доверительной вероятностью:

Теперь сформулируем нулевую и альтернативную гипотезы о корреляции исследуемых значений (уровень значимости ):

: Коэффициент корреляции равен нулю.

: Коэффициент корреляции не равен нулю.

Найдем значение статистики критерия:

*=>*

Найдем в таблице критическое значение , используя тот факт, что статистика критерия имеет распределение Стьюдента с степенями свободы.

Получаем, что отвергаем гипотезу в пользу .

**2.4. Построение выборочных прямых среднеквадратической регрессии**

Используя уже найденные оценки математических ожиданий, среднеквадратических отклонений и коэффициента корреляции Пирсона для обоих признаков, построим уравнения выборочных прямых среднеквадратической регрессии с помощью следующих формул:

Получим:

Вычислим оценку остаточной дисперсии X относительно Y:

Вычислим оценку остаточной дисперсии Y относительно X:

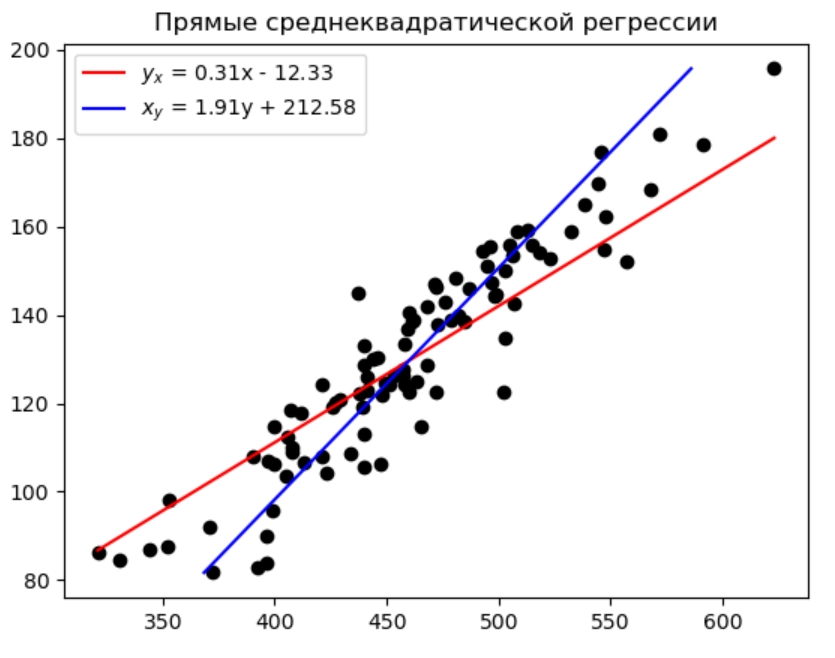
Полученные уравнения прямых изображены на рис.17.

Рис. 17 – Прямые среднеквадратичной регрессии

**2.5. Нахождение оценок корреляционного отношения**

Рассмотрим корреляционную таблицу:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |

Найдем выборочное корреляционное отношение к :

;

;

Теперь найдем корреляционное отношение к :

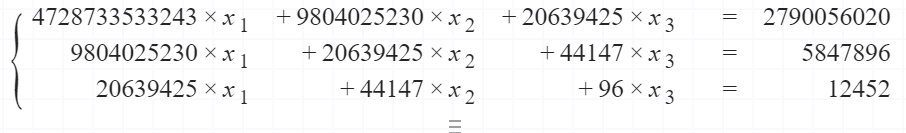
;

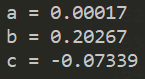
;

Получили, что и равны между собой и больше . На основании этого факта можно сделать вывод, что и в выборочных данные связаны корреляционной зависимостью, но характер этой зависимости определить пока не удаётся.

**2.6. Построение выборочных кривых параболической среднеквадратической регрессии**

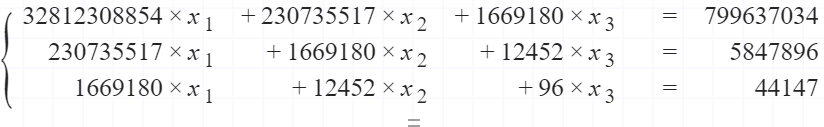
Теперь, используя корреляционную таблицу из предыдущего пункта, составим систему линейных уравнений третьего порядка для нахождения коэффициентов и в уравнениях и

Вычислим коэффициенты для СЛУ и найдём её решение:

Коэффициенты уравнения параболической регрессии на :

Тогда уравнение параболической регрессии на имеет вид:

Аналогично найдем и решим уравнение параболической регрессии на :

Решим систему уравнений:

Тогда уравнение параболической регрессии к имеет вид:

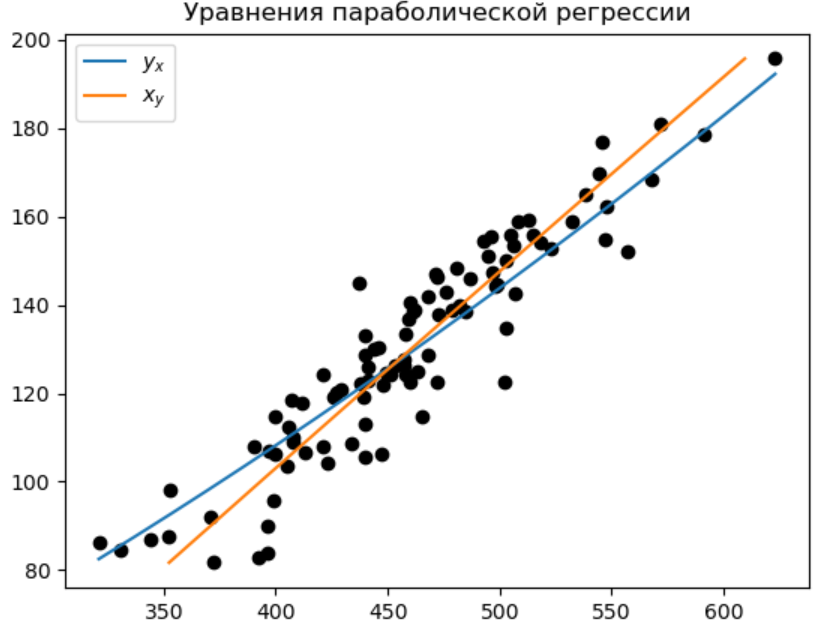
 Построим на рис. 18 графическое представление получившихся уравнений:

Рис. 18 – Уравнения параболической регрессии

**3. Нахождение оценок корреляционного отношения**

**3.1. Метод k-средних**

Для выполнения алгоритма кластеризации k-средних нужно найти первые приближения возможных центров кластеров. Эта задача была выполнена с помощью нахождения средних значений в равных по размаху диапазонах для каждой переменной.

Далее при сравнении размахов выборок по каждому признаку можно заметить, что они заметно отличаются. Из-за этого при подсчёте евклидового расстояния вклад по первому признаку будет заметно выше, чем по второму. Поэтому при расчёте расстояния между точками значения второго признака будет умножаться на поправочный коэффициент, равный отношению размахов первой выборки ко второй. В абсолютной величине он будет равняться ≈ 2.65.

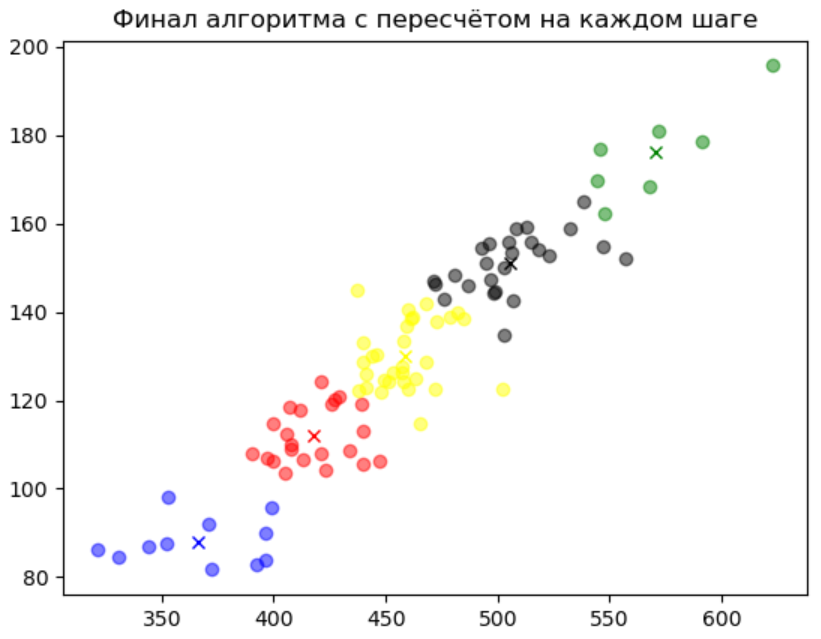
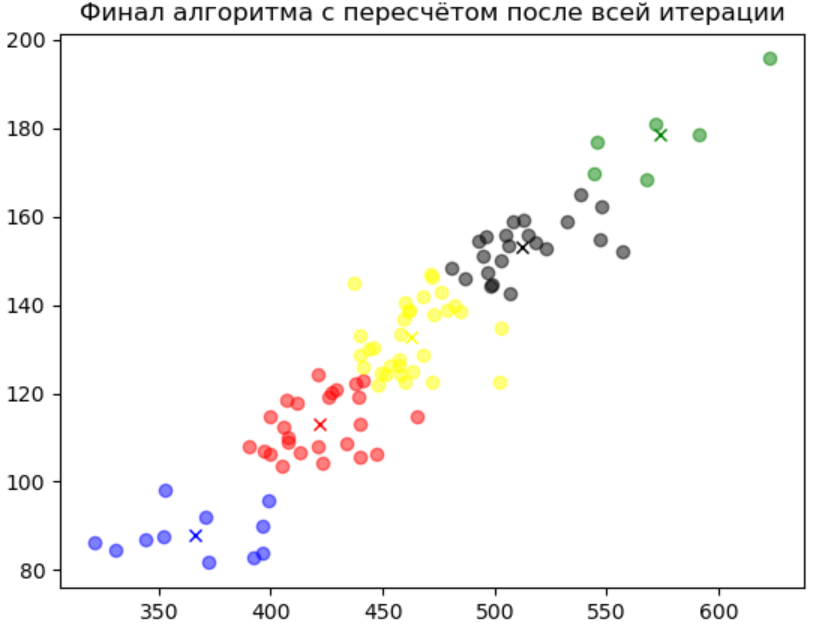
После выполнения подготовительных действий, используя реализованный метод *k*-средних, было выполнено разбиение на 5 кластеров, причём двумя способами: с пересчётом центров кластеров сразу после изменения их наполнения (1) и с пересчётом только после полной итерации (2). Графические изображения результатов представлены на рис. 19 и на рис. 20 соответственно.

Рис. 19 – Результат k-средних (1)

Рис. 20 – Результат k-средних (2)

На этих рисунках точки, относящиеся к одному кластеру закрашены одним цветом. Центры этих кластеров показаны крестиками.

Составим таблицы с промежуточными данными для каждой итерации алгоритмов разбиения. Таблица для варианта (1) алгоритма с пересчётом на каждом шаге:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| 0 | (353.00; 98.00) | (412.00; 117.90) | (473.00; 137.90) | (532.00; 158.70) | (591.00; 178.50) | - | - | - |
| 1 | (366.09; 88.11) | (416.48; 112.22) | (462.12; 131.13) | (509.64; 152.72) | (576.43; 178.39) | 2334.36 | 36199 | 3951.53 |
| 2 | (366.09; 88.11) | (417.86; 111.95) | (458.61; 130.16) | (505.60; 150.99) | (570.29; 176.07) | 2324.37 | 35339 | 4195.10 |

Здесь алгоритм закончил работу на третьем шаге, т.к. полученные на этом шаге кластеры совпали с кластерами на предыдущем. Функциональные качества разбиения F1 иF2 стали лучше, а в то время как F3 стало хуже.

Таблица для варианта (2) алгоритма с пересчётом после всей итерации:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| 0 | (353.00; 98.00) | (412.00; 117.90) | (473.00; 137.90) | (532.00; 158.70) | (591.00; 178.50) | - | - | - |
| 1 | (366.09; 88.11) | (422.58; 113.96) | (470.94; 136.31) | (522.24; 157.09) | (581.83; 179.85) | 2422.33 | 41117 | 3957.54 |
| 2 | (366.09; 88.11) | (423.19; 113.73) | (467.94; 134.97) | (518.78; 156.12) | (580.00; 180.12) | 2461.18 | 41208 | 4204.10 |
| 3 | (366.09; 88.11) | (422.5; 113.25) | (464.09; 133.90) | (516.29; 154.35) | (580.00; 180.12) | 2413.35 | 38800 | 4173.75 |
| 4 | (366.09; 88.11) | (422.5; 113.25) | (462.81; 133.05) | (512.09; 153.00) | (574.00; 178.37) | 2375.06 | 36620 | 4152.49 |
| 5 | (366.09; 88.11) | (421.48; 122.91) | (462.34; 132.70) | (512.09; 153.00) | (574.00; 178.37) | 2369.39 | 36764 | 4141.25 |

Здесь алгоритм закончил работу на шестом шаге, т.к. полученные на этом шаге кластеры совпали с кластерами на предыдущем. Видно, что все функциональные качества разбиения деградировали на втором шаге, но затем стали постепенно улучшаться. Но F3 так и не вернулся на прежние позиции.

**3.2. Метод поиска сгущений**

Далее сравним алгоритм k-средних c другим подобным алгоритмом – алгоритмом поиска сгущений. Для начала работы алгоритма сгущения следует выбрать способ выбора точек для центров окружностей. Введём параметр дельта, который будет равен радиусу окрестности вокруг точки, в который считается количество соседей. После этого каждый раз из ещё необработанных точек выбирается такая, у которой в этой самой окрестности самое большое количество других точек из текущего множества. Если ни у одной точки в соседях никого не осталось, то дельта увеличивается на 1. Если дельта достигла размера радиуса, то автоматически выбирается первая из последовательности точка.

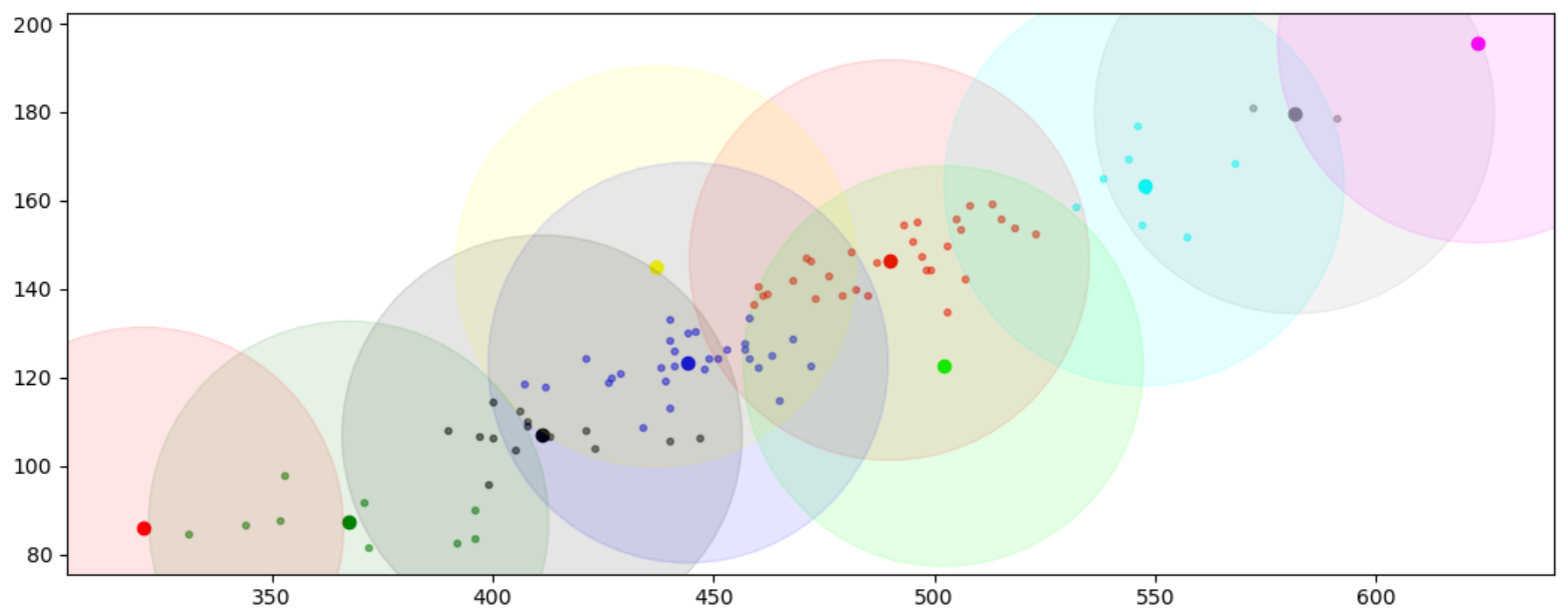
После нахождения центра окружности в работу вступает основная часть алгоритма сгущения. В результате его работы получаем кластеры с центрами. После этого нужно найти устойчивое разбиение. При постепенном увеличении радиуса на 0.7% получим разбиение из 10 кластеров с помощью окружностей радиусом 42.29. Картинка этого разбиения представлена на рисунке 21.

Рис. 21 – Устойчивое разбиение алгоритма сгущения при шаге в 0.7%

На каждом шаге с увеличением радиуса будет считать текущие функционалы и разместим результаты в таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R** | **Число кластеров** | **𝐹1** | **𝐹2** | **𝐹3** |
| 3.23 | 89 | 14.39 | 14.39 | 8.12 |
| 6.02 | 75 | 77.28 | 114.78 | 76.63 |
| 8.81 | 59 | 212.74 | 385.61 | 319.76 |
| 11.60 | 48 | 363.55 | 1228.83 | 509.78 |
| 14.39 | 38 | 548.94 | 2810.27 | 800.27 |
| 17.18 | 32 | 681.15 | 3714.34 | 1132.63 |
| … | | | | |
| 33.92 | 12 | 1629.96 | 20350.12 | 2712.77 |
| 36.71 | 12 | 1677.84 | 22405.97 | 2777.10 |
| 39.50 | 10 | 1820.48 | 26324.19 | 2859.14 |
| 42.29 | 10 | 1924.79 | 30209.38 | 2844.59 |

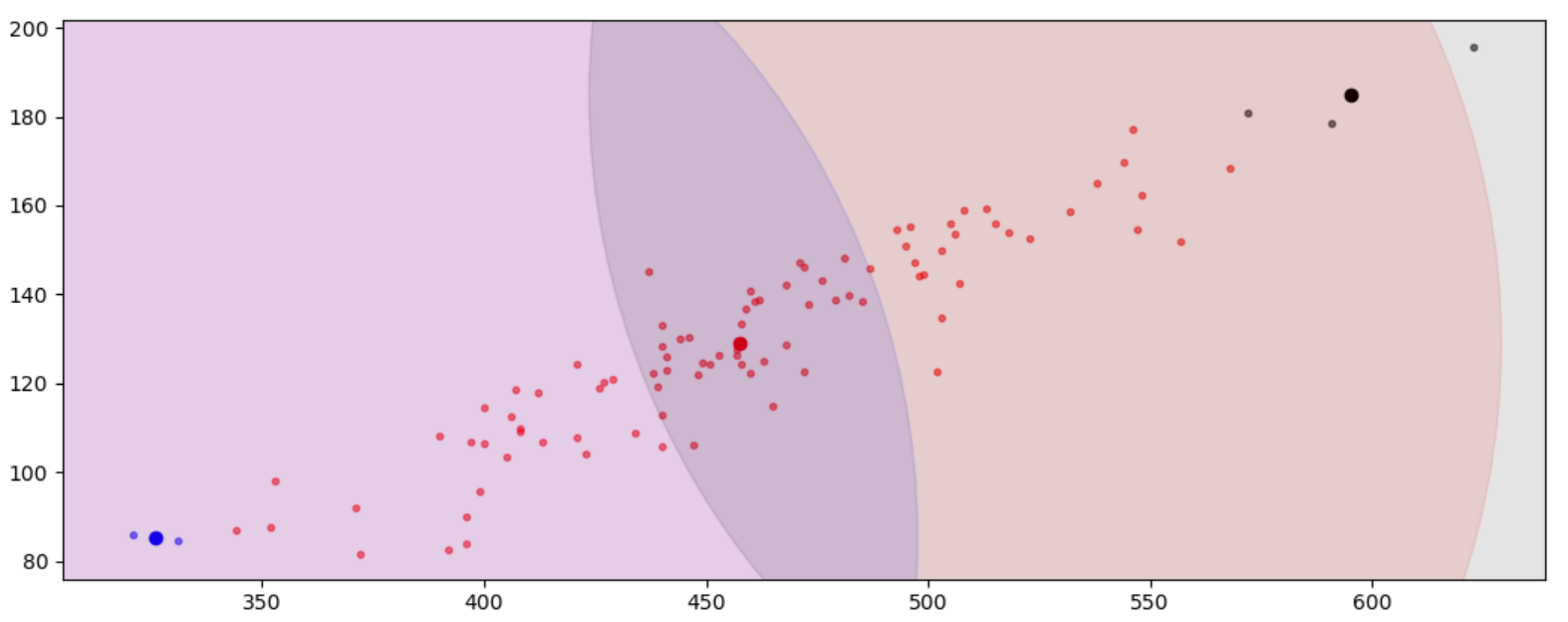
Если увеличить шаг интервала до 2%, получим устойчивое разбиение при радиусе окружности в 130.48. Картинка такого разбиения показана на рис. 22.

Рис. 22 – Устойчивое разбиение алгоритма сгущения при шаге в 2%

Функционалы ведут себя ожидаемо также – непреклонно увеличиваются с каждой итерацией. Зафиксируем их на конечном шаге: *R1* = 4868, *R2* = 253660, *R3* = 5950.

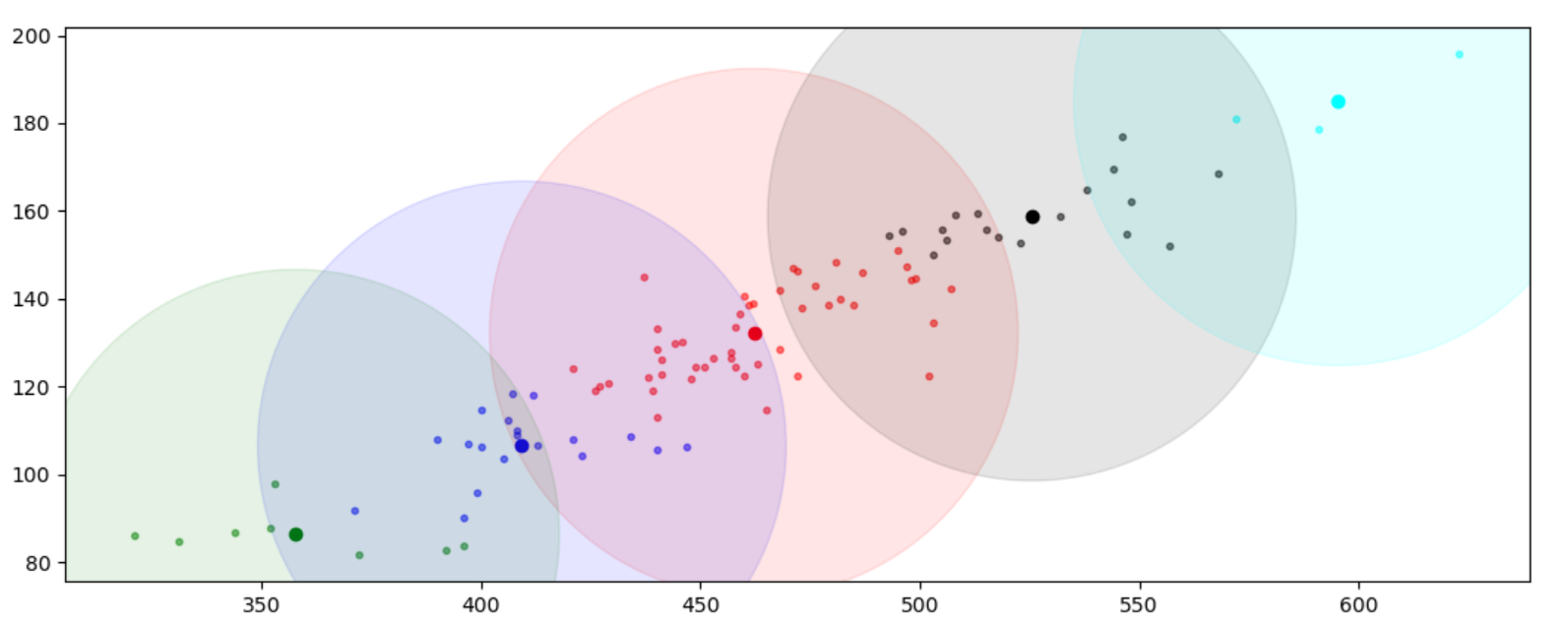
Теперь подберём такой минимальный радиус (хотя бы для неустойчивый), чтобы количество кластеров было равно количеству кластеров из прошлой работы, то есть пяти. После недолгого подбора коэффициентов получаем радиус размером приблизительно в 59,5. Изобразим разбиение на рис. 23.

Рис. 23 – Случай 6 кластеров

Получили интересную картинку. Для сравнения с предыдущим алгоритмом зафиксируем центры окружностей и центры функционалов:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| k-средних | (357.63; 86.43) | (409.32; 106.54) | (462.19; 132.21) | (525.55; 158.74) | (595.33; 185.03) | 2717 | 61631 | 4364 |
| Сгущений | (366.09; 88.11) | (421.48; 122.91) | (462.34; 132.70) | (512.09; 153.00) | (574.00; 178.37) | 2369 | 36764 | 4141 |

Исходя из данных в таблице можно сделать вывод, что на данной выборке метод k-средних в обоих своих вариантах сработал лучше.

**заключение**

В ходе выполнения первой части работы была сформирована выборка по первому признаку и проведена начальная статистическая обработка, итогом которой были построенные гистограммы, эмпирическая функции распределения и интервальный ряд, благодаря которому затем были вычислены выборочные точечные и интервальные оценки параметров распределения. Значения полученных описательных статистик позволили предположить, что выборка была сделана из генеральной совокупности, имеющей нормальный закон распределения. Затем это гипотеза была проверена с помощью критерия Пирсона. Результата проверки не дал оснований отвергнуть эту гипотезу.

Во второй части работы выборка по первому признаку была дополнена до двумерной, с сопутствующим первичным статистическим анализом дополняющей выборки по второму признаку. В отношении полученной двумерной выборки были применены методы корреляционного и регрессионного анализа. Так, был вычисленный коэффициент корреляции Пирсона, значение которого позволяет сделать вывод о наличии корреляционной зависимости между двумя величинами. Также была проверена гипотеза о равенстве коэффициента корреляции нулю. Результат проверки не был ошеломительными - эта гипотеза была отвергнута, из чего можно сделать вывод, что скорее всего зависимость между величинами не была получена случайно, а действительно существует.

После этого стало возможным вычисление и интерпретация корреляционного отношения одной переменной к другой. Вычисленное значение позволило сделать вывод, что выборочные данные согласованы с предположением о том, что исследуемые величины связаны корреляционной зависимостью. Но характер этой зависимости определить не удалось.

Далее были вычислены коэффициенты для уравнений выборочных прямых среднеквадратической регрессии и уравнений выборочных кривых для параболической среднеквадратической регрессии. Построение полученных уравнений на графике позволяют сказать, что полученные уравнения скорее всего были найдена правильно. Также в параболическом уравнении значения коэффициентов при близки к нулю. Этот факт косвенно подтверждает догадку об отсутствии квадратичной зависимости между исследуемыми параметрами. что может говорить о том, что скорее всего квадратичной зависимости нет.

В третьей части работы по отношению к выборке был проведён первичный кластерный анализ, итогом которого стали различные разбиения исходного множества на кластеры. Эти разбиения были получены применением двух алгоритмов: k-средних (две версии) и поиска сгущений. Оба метода выдали примерно одинаковые результаты, но, опираясь на полученные значения функционалов финальных разбиений, можно сделать вывод, что алгоритм k-средних выполнил свою задачу лучше, так как найденные им кластеры оказались более «кучными».

В заключении можно с достаточной долей уверенности сделать предположительный вывод, что между двумя признаками в генеральной совокупности всё-таки существует линейна зависимость (отчасти на это также указывают центры кластеров), и что сам наборы значений по каждому отдельному признаку согласованы с нормальным распределением.

**список использованных источников**

1. Севастьянов Б. А. Курс теории вероятностей и математической статистики. — Москва-Ижевск: Институт компьютерных исследований, 2004.
2. Гмурман В.Е Теория вероятностей и математическая статистика. – Москва : Высшее образование, 2008.

Приложение А

исходный код разработанной прораммы

from collections import namedtuple

from collections import Counter

import random

import math

from operator import attrgetter

Measurement = namedtuple("Measurement", ['density', 'elastic'])

selection\_size = 96 # объём выборочной совокупности

data\_file\_name = "Tabl.txt"

sample\_seed\_1 = 120 # None to random # Зерно для выборки, константа, чтобы всегда одинаково

block\_output\_size = 15 # (c учётом, что на одну запись уходит 6 символов(с пробелом))

def read\_data(filename):

# Извлекаем из файла данные и формируем генеральную совокупность.

general\_population = []

with open(filename) as file:

for line in file:

line = line.replace(',', '.').replace(' ', '').strip().split("\t")

general\_population += [Measurement(float(line[i]), float(line[i+1])) for i in range(0, len(line), 2) if line[i]]

return general\_population

def get\_sample(gen\_pop, size\_of\_gen\_pop):

# Формируем выборку из генеральной совокупности

prev\_state = random.getstate()

random.seed(sample\_seed\_1)

sample = random.sample(gen\_pop, size\_of\_gen\_pop)

random.setstate(prev\_state)

return sample

def get\_sample\_first(gen\_pop, size\_of\_gen\_pop):

prev\_state = random.getstate()

random.seed(sample\_seed\_1)

sample = random.sample(gen\_pop, size\_of\_gen\_pop)

random.setstate(prev\_state)

sample = [elem[0] for elem in sample]

return sample

def get\_sample\_second(gen\_pop, size\_of\_gen\_pop):

prev\_state = random.getstate()

random.seed(sample\_seed\_1)

sample = random.sample(gen\_pop, size\_of\_gen\_pop)

random.setstate(prev\_state)

sample = [elem[1] for elem in sample]

return sample

def print\_beautiful\_sample(sample: list):

for i, value in enumerate(sample):

if i % block\_output\_size == 0 and i != 0:

print(end="\n")

print(value, end=" ")

print(end="\n")

def print\_beautiful\_variation(variation: Counter):

sorted\_density\_values = list(sorted(variation\_series\_density))

for i in range(int(len(variation)/block\_output\_size) + 1):

for index in range(i\*block\_output\_size, (i+1)\*block\_output\_size):

if index >= len(variation): break

key = sorted\_density\_values[index]

print("{0:^6}".format(key), end="")

print()

for index in range(i\*block\_output\_size, (i+1)\*block\_output\_size):

if index >= len(variation): break

key = sorted\_density\_values[index]

print("{0:^6}".format(variation[key]), end="")

print()

def get\_interval\_sample(sample):

buckets\_number = int(1 + 3.322 \* math.log10(selection\_size))

min\_density, max\_density = min(sample), max(sample)

range\_density = max\_density - min\_density

isInBucket = lambda x: min(int((abs(x) - min\_density) / range\_density \* buckets\_number), buckets\_number-1)

borders = [(min\_density + range\_density/buckets\_number\*i, min\_density + range\_density/buckets\_number\*(i+1)) for i in range(buckets\_number)]

buckets = [[] for i in range(buckets\_number)]

for value in sample:

buckets[isInBucket(value)].append(value)

return borders, buckets

def print\_beautiful\_interval\_freq(buckets, borders):

print("\nИнтервальный ряд с частотами:")

print("| \* | Интервал | Mid | Абс. частота | Отн. частота |")

for i, border in enumerate(borders):

if i != len(borders)-1:

print("| {2} | [{0:.3f} - {1:.3f}) | {3} |".format(border[0], border[1], i+1, round((border[0]+border[1])/2)), end=" ")

else:

print("| {2} | [{0:.3f} - {1:.3f}] | {3} |".format(border[0], border[1], i+1, round((border[0]+border[1])/2)), end=" ")

print("{0:^13.3f}| {1:^13.3f}|".format(len(buckets[i]), len(buckets[i])/selection\_size))

def print\_beautiful\_interval\_values(buckets, borders):

print("\nИнтервальный ряд со значениями:")

for i, border in enumerate(borders):

if i != len(borders)-1:

print("[{0:.3f} - {1:.3f}):".format(border[0], border[1]), end=" ")

else:

print("[{0:.3f} - {1:.3f}]:".format(border[0], border[1]), end=" ")

for elem in buckets[i]:

print(elem, end=" ")

print(end="\n")

if \_\_name\_\_ == "\_\_main\_\_":

general\_population = read\_data(filename=data\_file\_name)

sample = get\_sample(general\_population, selection\_size)

sample\_density = [pair.density for pair in sample]

sample\_elastic = [pair.elastic for pair in sample]

print("\nСформированная выборка:")

print\_beautiful\_sample(sample\_elastic)

# ranked\_row = sorted(sample, reverse=False, key=attrgetter('density'))

ranked\_row = sorted(sample\_elastic)

print("\nРанжированный ряд (по возрастанию): ")

print\_beautiful\_sample(ranked\_row)

# Вариационный ряд значений плотности

variation\_series\_density = Counter(sample\_elastic)

print("\nВариационный ряд:")

print\_beautiful\_variation(variation\_series\_density)

# Для интервального ряда нужно оценить длину частичного интервала

# Для этого воспользуемся формулой Стерджеса: k = 1 + 3.322\*log10(n)

buckets\_number = int(1 + 3.322 \* math.log10(selection\_size))

min\_density, max\_density = min(sample\_elastic), max(sample\_elastic)

range\_density = max\_density - min\_density

print("\nИспользуя формулу Стерджеса рассчитаем количество групп для разбиения выборки:")

print("1 + 3.322\*lg({0}) = {1}".format(selection\_size, buckets\_number))

print("Минимальное значение ряда: ", min\_density)

print("Максимальное значение ряда:", max\_density)

print("Размах выборки", range\_density)

isInBucket = lambda x: min(int((abs(x) - min\_density) / range\_density \* buckets\_number), buckets\_number-1)

borders = [(min\_density + range\_density/buckets\_number\*i, min\_density + range\_density/buckets\_number\*(i+1)) for i in range(buckets\_number)]

buckets = [[] for i in range(buckets\_number)]

for value in sample\_elastic:

buckets[isInBucket(value)].append(value)

print\_beautiful\_interval\_freq(buckets, borders)

print\_beautiful\_interval\_values(buckets, borders)

# --------------------------------- Рисуночки! ---------------------------------

import matplotlib.pyplot as plt

import numpy as np

from statistics import mean

# Гистограммы

fig, ax = plt.subplots()

# ax.hist(sample\_elastic, bins=buckets\_number, density=False, edgecolor='black', facecolor='blue')

# ax.hist(sample\_elastic, bins=buckets\_number, density=False, edgecolor='black', facecolor='blue')

y = [len(bucket) for bucket in buckets]

center\_of\_borders = [(border[0] + border[1])/2 for border in borders]

ax.plot(center\_of\_borders, y, '--')

ax.set\_xlabel('Варианты')

ax.set\_ylabel('Абсолютная частота')

ax.set\_title('Полигон абсолютных частот')

fig.tight\_layout()

# plt.show()

fig, ax = plt.subplots()

h = borders[0][1]-borders[0][0]

y\_h = [val/h for val in y]

ax.bar(center\_of\_borders, y\_h, h, edgecolor='black', color='b')

for i in range(len(center\_of\_borders)):

ax.text(center\_of\_borders[i], y\_h[i], str(round(y\_h[i], 4)), ha='center', color='b')

ax.set\_xlabel('Варианты')

ax.set\_ylabel('Абсолютная частота')

ax.set\_title('Гистограмма абсолютных частот')

fig.tight\_layout()

# plt.show()

fig, ax = plt.subplots()

# ax.hist(sample\_elastic, buckets\_number, weights=np.ones(len(sample\_elastic)) / len(sample\_elastic), density=False, edgecolor='black', facecolor='blue')

center\_of\_borders = [(border[0] + border[1])/2 for border in borders]

y = [len(bucket)/selection\_size for bucket in buckets]

ax.plot(center\_of\_borders, y, '--')

ax.set\_xlabel('Варианты')

ax.set\_ylabel('Относительная частота')

ax.set\_title('Полигон относительных частот')

fig.tight\_layout()

fig, ax = plt.subplots()

# ax.hist(sample\_elastic, buckets\_number, weights=np.ones(len(sample\_elastic)) / len(sample\_elastic), density=False, edgecolor='black', facecolor='blue')

center\_of\_borders = [(border[0] + border[1])/2 for border in borders]

y = [len(bucket)/selection\_size for bucket in buckets]

# ax.plot(center\_of\_borders, y, '--')

h = borders[0][1]-borders[0][0]

y\_h = [val/h for val in y]

ax.bar(center\_of\_borders, y\_h, h, edgecolor='black', color='b')

for i in range(len(center\_of\_borders)):

ax.text(center\_of\_borders[i], y\_h[i], str(round(y\_h[i], 4)), ha='center', color='b')

ax.set\_xlabel('Варианты')

ax.set\_ylabel('Относительная частота')

ax.set\_title('Гистограмма относительных частот')

fig.tight\_layout()

plt.show()

exit(12)

# Эмпирическая функция распределения

from statsmodels.distributions.empirical\_distribution import ECDF

fig, ax = plt.subplots()

ecdf = ECDF(sample\_elastic)

ax.set\_xlabel('x')

ax.set\_ylabel('F(x)')

ax.set\_title('Эмпирическая функция распределения (по выборке)')

ax.axis(xmin=min\_density-50, xmax=max(ecdf.x)+50)

ax.axis(ymin=-0.05, ymax=1.05)

for i in range(len(ecdf.x)-1):

xs = [min\_density, ecdf.x[i]]

ys = [ecdf.y[i]] \* 2

# ax.plot(xs, ys, 'r:', alpha=0.2)

ax.plot(ecdf.x[i], ecdf.y[i], "k.") # точки

xs = [ecdf.x[i], ecdf.x[i+1]]

ys = [ecdf.y[i]] \* 2

ax.plot(xs, ys, 'k-')

ax.plot([min\_density-500, ecdf.x[1]], [0, 0], 'k--', alpha=0.1)

ax.plot([max(ecdf.x), max(ecdf.x)+50], [1, 1], 'k--', alpha=0.1)

ax.plot(ecdf.x[-1], ecdf.y[-1], "k.") # точки

fig.tight\_layout()

plt.show()

fig, ax = plt.subplots()

ecdf = ECDF([sum(bucket) for bucket in buckets])

ax.set\_xlabel('x')

ax.set\_ylabel('F(x)')

ax.set\_title('Эмпирическая функция распределения (по интервальному ряду)')

ax.axis(xmin=min\_density-100, xmax=max(ecdf.x)+500)

ax.axis(ymin=-0.05, ymax=1.05)

for i in range(len(ecdf.x)-1):

xs = [min\_density, ecdf.x[i]]

ys = [ecdf.y[i]] \* 2

# ax.plot(xs, ys, 'r:', alpha=0.2)

ax.plot(ecdf.x[i], ecdf.y[i], "k.") # точки

xs = [ecdf.x[i], ecdf.x[i+1]]

ys = [ecdf.y[i]] \* 2

ax.plot(xs, ys, 'k-')

ax.plot([min\_density-1000, ecdf.x[1]], [0, 0], 'k--', alpha=0.1)

ax.plot([max(ecdf.x), max(ecdf.x)+1000], [1, 1], 'k--', alpha=0.1)

ax.plot(ecdf.x[-1], ecdf.y[-1], "k.") # точки

fig.tight\_layout()

plt.show()

### lab\_2

import lab1

import math

from collections import Counter

def build\_table(sample\_density):

borders, buckets = lab1.get\_interval\_sample(sample\_density)

mid\_borders = [round((border[0] + border[1])/2) for border in borders]

build\_table.C = mid\_borders[max([(len(bucket), i) for i, bucket in enumerate(buckets)])[1]]

build\_table.h = int((mid\_borders[-1] - mid\_borders[0]) / (len(mid\_borders)-1))

table = [[0 for i in range(8)] for i in range(len(mid\_borders)+1)]

for i, bucket in enumerate(buckets):

xi = mid\_borders[i]

n = len(buckets[i])

ui = (xi - build\_table.C) / build\_table.h

table [i][0] = xi

table[-1][0] += xi

table [i][1] = n

table[-1][1] += n

table [i][2] = ui

table[-1][2] += ui

table [i][3] = ui \* n

table[-1][3] += ui \* n

table [i][4] = ui \*\* 2 \* n

table[-1][4] += ui \*\* 2 \* n

table [i][5] = ui \*\* 3 \* n

table[-1][5] += ui \*\* 3 \* n

table [i][6] = ui \*\* 4 \* n

table[-1][6] += ui \*\* 4 \* n

table [i][7] = (ui+1) \*\* 4 \* n

table[-1][7] += (ui+1) \*\* 4 \* n

return table

build\_table.C = 0

build\_table.h = 0

def check\_result(table):

lhs = table[-1][6] + 4\*table[-1][5] + 6\*table[-1][4] + 4\*table[-1][3] + table[-1][1]

rhs = table[-1][7]

print("\nПроверка:", lhs, "=", rhs, sep=" ")

assert abs(lhs - rhs) < 0.0001, "Должны совпадать!"

def get\_main\_values\_from\_table(table):

n = table[-1][1]

M1 = table[-1][3] / n

M2 = table[-1][4] / n

M3 = table[-1][5] / n

M4 = table[-1][6] / n

h = build\_table.h # sugar

m2 = (M2 - pow(M1,2)) \* pow(h,2)

m3 = (M3 - 3\*M2\*M1 + 2\*pow(M1,3)) \* pow(h,3)

m4 = (M4 - 4\*M3\*M1 + 6\*M2\*pow(M1,2) - 3\*pow(M1, 4)) \* pow(h,4)

X\_cherta = M1 \* build\_table.h + build\_table.C

Dv = m2

sigma = math.sqrt(Dv)

s2 = (n / (n-1)) \* Dv

S = math.sqrt(s2)

As = m3 / pow(sigma, 3)

E = (m4 / pow(sigma, 4)) - 3

return X\_cherta, S

if \_\_name\_\_ == "\_\_main\_\_":

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

# sample = lab1.get\_sample(general\_population, lab1.selection\_size)

sample\_density = lab1.get\_sample\_first(general\_population, lab1.selection\_size)

sample\_elastic = lab1.get\_sample\_second(general\_population, lab1.selection\_size)

curr\_sample = sample\_elastic

borders, buckets = lab1.get\_interval\_sample(curr\_sample)

lab1.print\_beautiful\_interval\_freq(buckets, borders)

mid\_borders = [round((border[0] + border[1])/2) for border in borders]

C = mid\_borders[max([(len(bucket), i) for i, bucket in enumerate(buckets)])[1]]

# h = mid\_borders[1] - mid\_borders[0]

h = int((mid\_borders[-1] - mid\_borders[0]) / (len(mid\_borders)-1)) # вроде работает, на за счёт округление может быть шляпа

table = build\_table(curr\_sample)

for row in table:

for number in row:

print("{0:.2f}".format(number), end=" ")

print(end="\n")

check\_result(table)

C = build\_table.C

h = build\_table.h

print("C =", C)

print("h =", h)

n = table[-1][1]

M1 = table[-1][3] / n

M2 = table[-1][4] / n

M3 = table[-1][5] / n

M4 = table[-1][6] / n

print("\nНачальные выборочные моменты с 1 по 4:")

print("M1: {0:.4f}".format(M1))

print("M2: {0:.4f}".format(M2))

print("M3: {0:.4f}".format(M3))

print("M4: {0:.4f}".format(M4))

print()

m2 = (M2 - pow(M1,2)) \* pow(h,2)

m3 = (M3 - 3\*M2\*M1 + 2\*pow(M1,3)) \* pow(h,3)

m4 = (M4 - 4\*M3\*M1 + 6\*M2\*pow(M1,2) - 3\*pow(M1, 4)) \* pow(h,4)

print("Центральные выборочные моменты:")

print("m2: {0:.4f}".format(m2))

print("m3: {0:.4f}".format(m3))

print("m4: {0:.4f}".format(m4))

print()

X\_cherta = M1\*h + C

Dv = m2

sigma = math.sqrt(Dv)

s2 = (n / (n-1)) \* Dv

S = math.sqrt(s2)

As = m3 / pow(sigma, 3)

E = (m4 / pow(sigma, 4)) - 3

print("Оценка мат ожидания:", X\_cherta)

print("Смещённая оценка дисперсии:", Dv)

print("Оценка среднеквадратичного отклонения:", sigma)

print("Исправленная оценка дисперсии", s2)

print("Исправленная оценка СКО", S)

print("Оценка асимметрии", As)

print("Оценка эксцесса", E)

### лаб\_3

import lab1

import lab2

from scipy.stats import t, chi2

from scipy.special import erf

Ф = lambda x: erf(x/2\*\*0.5)/2 # функция лапласа

from math import sqrt

def print\_2digits(list, string):

print(string)

for elem in list:

print("{0:.2f}".format(elem), end=" ")

print(end="\n")

N = lab1.selection\_size # 96

n = N-1 # 95

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

sample = lab1.get\_sample(general\_population, lab1.selection\_size)

sample\_density = [pair.density for pair in sample]

table = lab2.build\_table(sample\_density)

X, S = lab2.get\_main\_values\_from\_table(table)

print("С прошлой работы получим значения:\nX = {0:.2f}\nS = {1:.2f}".format(X, S))

igrek = 0.95

t\_igrek = 1.9855

a\_left = X - t\_igrek \* S / sqrt(N)

a\_right = X + t\_igrek \* S / sqrt(N)

print("Доверительный интервал для математического ожидания: ", end="")

print("({0:.4f}; {1:.4f})\n".format(a\_left, a\_right))

gamma = 0.95

q = 0.147

sd\_left = S\*(1-q)

sd\_right = S\*(1+q)

print("Доверительный интервал для среднеквадратического отклонения: ", end="")

print("({0:.4f}; {1:.4f})\n".format(sd\_left, sd\_right))

borders, buckets = lab1.get\_interval\_sample(sample\_density)

# print(borders, buckets)

vals = [len(bucket) for bucket in buckets]

# print(borders)

# print(buckets)

print("Частоты в интервалах:")

print(vals)

print()

# Объединяем последние интервалы

borders[-2] = (borders[-2][0], borders[-1][1])

del borders[-1]

buckets[-2] += buckets[-1]

del buckets[-1]

vals = [len(bucket) for bucket in buckets]

# print(borders)

# print(buckets)

print("Частоты в интервалах после:")

print(vals)

print()

# границы интервалов в интервальном ряду

bins = [elem[0] for elem in borders] + [borders[-1][1]]

print\_2digits(bins, "Границы интервалов:")

# # стандартизуем границы интервалов

z = [(e - X)/S for e in bins]

z[0] = -float('inf')

z[-1] = float('inf')

print\_2digits(z, "Значения Z:")

p = []

for i in range(1, len(z)):

p.append(Ф(z[i])-Ф(z[i-1]))

print\_2digits(p, "Вероятности попадания в интервалы:")

print(end="\n")

# freq\_theor = list(map(lambda x: x\*n, p))

freq\_theor = [x\*N for x in p]

print\_2digits(freq\_theor, "Теоретические частоты по интервалам:")

print(end="\n")

freq\_expect = vals

print\_2digits(freq\_expect, "Ожидаемые частоты по интервалам:")

print(end="\n")

# посчитаем статистику критерия Хи-квадрат

stats\_crit = []

for i in range(len(freq\_expect)):

stats\_crit.append((freq\_expect[i] - freq\_theor[i])\*\*2/freq\_theor[i])

chi2\_calculated = sum(stats\_crit)

print("Cтатистики критерия Хи-квадрат:")

for elem in stats\_crit:

print("{0:.4f}".format(elem), end=" ")

print(end="\n")

print('Общее значение: {}'.format(chi2\_calculated))

print(end="\n")

k = 5 # число интервалов

l = 2 # число параметров

alpha = 0.05 # уровень значимости

# критическое значение статистики критерия

chi2\_criterion = chi2.ppf(1-alpha, df=k-l-1)

print('Критическое значение: {}'.format(chi2\_criterion))

# принимаем решение

if chi2\_calculated > chi2\_criterion:

print('Отвергаем гипотезу H0')

else:

print('Не отвергем гипоетезу H0')

# лаб\_4

from math import sqrt, log, exp

import lab1

def print\_beauty(sample: list, size):

for i in range(1, len(sample)+1):

print(sample[i-1], end='\t')

if i % size == 0:

print(end="\n")

def check(value\_1, value\_2, border\_1, border\_2):

is\_1\_in\_interval = value\_1 >= border\_1[0] and value\_1 <= border\_1[1]

is\_2\_in\_interval = value\_2 >= border\_2[0] and value\_2 <= border\_2[1]

return is\_1\_in\_interval and is\_2\_in\_interval

def build\_corr\_table(sample\_2D, borders\_1, borders\_2):

table = []

for i in range(len(borders\_1)):

table.append([])

for j in range(len(borders\_2)):

tmp = map(lambda x: check(x[0], x[1], borders\_1[i], borders\_2[j]), sample\_2D)

table[i].append(sum(tmp))

return table

def countSum(table, v, u):

sum\_from\_table = 0

for i in range(7):

for j in range(7):

sum\_from\_table += table[i][j] \* v[i] \* u[j]

return sum\_from\_table

def get\_correl\_coef(sample\_density, sample\_elastic):

import lab1

n = lab1.selection\_size

sample\_2D = list(zip(sample\_density, sample\_elastic))

borders\_1, buckets\_1 = lab1.get\_interval\_sample(sample\_density)

borders\_2, buckets\_2 = lab1.get\_interval\_sample(sample\_elastic)

freqs\_1 = [len(array) for array in buckets\_1]

freqs\_2 = [len(array) for array in buckets\_2]

mid\_borders\_1 = [(border[0] + border[1])/2 for border in borders\_1]

mid\_borders\_2 = [(border[0] + border[1])/2 for border in borders\_2]

step\_size\_1 = borders\_1[0][1] - borders\_1[0][0]

step\_size\_2 = borders\_2[0][1] - borders\_2[0][0]

C\_1 = 472 # из прошлых работ

C\_2 = 122

# СКО для условных вариант

v = [(elem - C\_1) / step\_size\_1 for elem in mid\_borders\_1]

u = [(elem - C\_2) / step\_size\_2 for elem in mid\_borders\_2]

mean\_v = sum([x[0] \* x[1] for x in zip(v, freqs\_1)]) / n

mean\_u = sum([x[0] \* x[1] for x in zip(u, freqs\_2)]) / n

S\_v = sqrt(sum([x[0]\*\*2 \* x[1] for x in zip(v, freqs\_1)]) / n - mean\_v\*\*2)

S\_u = sqrt(sum([x[0]\*\*2 \* x[1] for x in zip(u, freqs\_2)]) / n - mean\_u\*\*2)

table = build\_corr\_table(sample\_2D, borders\_1, borders\_2)

sum\_from\_table = countSum(table, v, u)

r = (sum\_from\_table + n \* mean\_v \* mean\_u) / (n \* S\_v \* S\_u)

return r

if \_\_name\_\_ == "\_\_main\_\_":

n = lab1.selection\_size

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

sample\_density = lab1.get\_sample\_first(general\_population, n)

sample\_elastic = lab1.get\_sample\_second(general\_population, n)

borders\_1, buckets\_1 = lab1.get\_interval\_sample(sample\_density)

borders\_2, buckets\_2 = lab1.get\_interval\_sample(sample\_elastic)

freqs\_1 = [len(array) for array in buckets\_1]

freqs\_2 = [len(array) for array in buckets\_2]

# Расчёты прогоняем, меняя источник данных в лабе 2

# Часть 2

sample\_2D = list(zip(sample\_density, sample\_elastic))

print("Двумерная выборка:")

print\_beauty(sample\_2D, size=6)

print(end="\n")

table = build\_corr\_table(sample\_2D, borders\_1, borders\_2)

print("Данные для таблицы:")

for row in table:

print(row)

print(end="\n")

mid\_borders\_1 = [(border[0] + border[1])/2 for border in borders\_1]

mid\_borders\_2 = [(border[0] + border[1])/2 for border in borders\_2]

step\_size\_1 = borders\_1[0][1] - borders\_1[0][0]

step\_size\_2 = borders\_2[0][1] - borders\_2[0][0]

C\_1 = 472 # из прошлых работ

C\_2 = 122

# Условные варианты

v = [(elem - C\_1) / step\_size\_1 for elem in mid\_borders\_1]

u = [(elem - C\_2) / step\_size\_2 for elem in mid\_borders\_2]

# средние для условных вариант

mean\_v = sum([x[0] \* x[1] for x in zip(v, freqs\_1)]) / n

mean\_u = sum([x[0] \* x[1] for x in zip(u, freqs\_2)]) / n

# СКО для условных вариант

S\_v = sqrt(sum([x[0]\*\*2 \* x[1] for x in zip(v, freqs\_1)]) / n - mean\_v\*\*2)

S\_u = sqrt(sum([x[0]\*\*2 \* x[1] for x in zip(u, freqs\_2)]) / n - mean\_u\*\*2)

print('u: Среднее: {0:.3f}, СКО: {1:.3f}'.format(mean\_u, S\_u))

print('v: Среднее: {0:.3f}, СКО: {1:.3f}'.format(mean\_v, S\_v))

print(end="\n")

sum\_from\_table = countSum(table, v, u)

print(sum\_from\_table)

# коэффициент корреляции Пирсона

r = (sum\_from\_table + n \* mean\_v \* mean\_u) / (n \* S\_v \* S\_u)

print('Коэффициент корреляции П ирсона {0:.4f}'.format(r))

# нахождение доверительного интервала

z = 0.5 \* log((1 + r)/(1 - r))

# доверительный интервал для z

left = z - 1.96 / sqrt(n - 3)

right = z + 1.96 / sqrt(n - 3)

print('Доверительный интервал для Z: ({0:.3f}, {1:.3f})'.format(left, right))

# обратное преобразование

left = (exp(2 \* left) - 1) / (exp(2 \* left) + 1)

right = (exp(2 \* right) - 1) / (exp(2 \* right) + 1)

print('Доверительный интервал: ({0:.3f}, {1:.3f})'.format(left, right))

# проверка гипотезы

t\_sample = r \* sqrt(n-2) / sqrt(1 - r\*\*2)

t\_cr = 1.98

print(t\_cr)

print('t выборочное = {0:.3f}'.format(t\_sample))

print('t критическое = {0:.3f}'.format(t\_cr))

if t\_sample <= t\_cr:

print('Не достаточно отснований, чтобы отклонить H0')

else:

print('Отвергаем гипотезу H0')

### lab\_5

from math import sqrt

import lab1

import lab2

import lab4

n = lab1.selection\_size

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

sample\_density = lab1.get\_sample\_first(general\_population, n)

sample\_elastic = lab1.get\_sample\_second(general\_population, n)

table\_x = lab2.build\_table(sample\_density)

mean\_x, S\_x = lab2.get\_main\_values\_from\_table(table\_x)

table\_y = lab2.build\_table(sample\_elastic)

mean\_y, S\_y = lab2.get\_main\_values\_from\_table(table\_y)

r = lab4.get\_correl\_coef(sample\_density, sample\_elastic)

print("Для X:\n\tmean = {0:.2f}\n\tS = {1:.2f}".format(mean\_x, S\_x))

print("Для Y:\n\tmean = {0:.2f}\n\tS = {1:.2f}".format(mean\_y, S\_y))

print("Коэффициент корреляции: {0:.3f}".format(r))

coef\_y\_x = r \* S\_y / S\_x

free\_number\_y\_x = mean\_y + coef\_y\_x\*mean\_x\*(-1)

print(free\_number\_y\_x)

coef\_x\_y = r \* S\_x / S\_y

free\_number\_x\_y = mean\_x + coef\_x\_y\*mean\_y\*(-1)

print(free\_number\_x\_y)

resudial\_disp\_x = pow(S\_x,2) \* pow(1-r, 2)

resudial\_disp\_y = pow(S\_y,2) \* pow(1-r, 2)

# print(resudial\_disp\_x, resudial\_disp\_y)

# import matplotlib.pyplot as plt

# fig, ax = plt.subplots()

# ax.plot(sample\_density, sample\_elastic, 'ok')

# min\_val, max\_val = min(sample\_density), max(sample\_density)

# y\_minval = mean\_y + coef\_y\_x\*(min\_val - mean\_x)

# y\_maxval = mean\_y + coef\_y\_x\*(max\_val - mean\_x)

# ax.plot([min\_val, max\_val], [y\_minval, y\_maxval], "-r", label="$y\_x$ = 0.31x - 12.33") # yx

# min\_val, max\_val = min(sample\_elastic), max(sample\_elastic)

# print(min\_val, max\_val)

# x\_minval = mean\_x + coef\_x\_y\*(min\_val - mean\_y)

# x\_maxval = mean\_x + coef\_x\_y\*(max\_val - mean\_y)

# ax.plot([x\_minval, x\_maxval], [min\_val, max\_val], "-b", label="$x\_y$ = 1.91y + 212.58") # xy

# ax.legend()

# # ax.set\_xlabel('Варианты')

# # ax.set\_ylabel('Абсолютная частота')

# ax.set\_title('Прямые среднеквадратической регрессии')

# # fig.tight\_layout()

# plt.show()

borders\_x, buckets\_x = lab1.get\_interval\_sample(sample\_density)

borders\_y, buckets\_y = lab1.get\_interval\_sample(sample\_elastic)

freqs\_x = [len(array) for array in buckets\_x]

freqs\_y = [len(array) for array in buckets\_y]

mid\_borders\_x = [(border[0] + border[1])/2 for border in borders\_x]

mid\_borders\_y = [(border[0] + border[1])/2 for border in borders\_y]

for elem in mid\_borders\_x:

print("{0:.2f}".format(elem), end=" ")

print()

# print(freqs\_x)

for elem in mid\_borders\_y:

print("{0:.2f}".format(elem), end=" ")

print()

# print(freqs\_y)

sample\_2D = list(zip(sample\_density, sample\_elastic))

table = lab4.build\_corr\_table(sample\_2D, borders\_x, borders\_y)

for row in table:

print(row)

print(end="\n")

n\_y = []

x\_y\_cherta = []

for i in range(len(table)):

summ = 0

num = 0

for j in range(len(table)):

num += table[j][i]

summ += table[j][i] \* mid\_borders\_x[j]

n\_y.append(num)

x\_y\_cherta.append(summ / num)

# n\_y[-1] +=1

n\_x = []

y\_x\_cherta = []

for i in range(len(table)):

summ = 0

num = 0

for j in range(len(table)):

num += table[i][j]

summ += table[i][j] \* mid\_borders\_y[j]

n\_x.append(num)

y\_x\_cherta.append(summ / num)

# n\_x[-1] += 1

D\_o\_xy = sum([n\_x[i] \* pow((mid\_borders\_x[i]-mean\_x),2) for i in range(len(table))]) / n

sigma\_x\_xy = sqrt(D\_o\_xy)

D\_m\_xy = sum([n\_y[i] \* pow((x\_y\_cherta[i]-mean\_x),2) for i in range(len(table))]) / n

sigma\_xy = sqrt(D\_m\_xy)

nu\_xy = sigma\_xy / sigma\_x\_xy

print("D(общ): {0:.2f}".format(D\_o\_xy))

print("σ(x): {0:.2f}".format(sigma\_x\_xy))

print("D(меж): {0:.2f}".format(D\_m\_xy))

print("σ(xy): {0:.2f}".format(sigma\_xy))

print("η(xy): {0:.3f}".format(nu\_xy))

D\_o\_xy = sum([n\_y[i] \* pow((mid\_borders\_y[i]-mean\_y),2) for i in range(len(table))]) / n

sigma\_x\_xy = sqrt(D\_o\_xy)

D\_m\_xy = sum([n\_x[i] \* pow((y\_x\_cherta[i]-mean\_y),2) for i in range(len(table))]) / n

sigma\_xy = sqrt(D\_m\_xy)

nu\_xy = sigma\_xy / sigma\_x\_xy

print()

print("D(общ): {0:.2f}".format(D\_o\_xy))

print("σ(x): {0:.2f}".format(sigma\_x\_xy))

print("D(меж): {0:.2f}".format(D\_m\_xy))

print("σ(yx): {0:.2f}".format(sigma\_xy))

print("η(yx): {0:.3f}".format(nu\_xy))

# 3 часть

import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot(sample\_density, sample\_elastic, 'ok')

sum\_1 = sum([n\_x[i] \* mid\_borders\_x[i]\*\*1 for i in range(len(table))])

sum\_2 = sum([n\_x[i] \* mid\_borders\_x[i]\*\*2 for i in range(len(table))])

sum\_3 = sum([n\_x[i] \* mid\_borders\_x[i]\*\*3 for i in range(len(table))])

sum\_4 = sum([n\_x[i] \* mid\_borders\_x[i]\*\*4 for i in range(len(table))])

sum\_y\_1 = sum([n\_x[i] \* y\_x\_cherta[i] \* mid\_borders\_x[i]\*\*1 for i in range(len(table))])

sum\_y\_2 = sum([n\_x[i] \* y\_x\_cherta[i] \* mid\_borders\_x[i]\*\*2 for i in range(len(table))])

sum\_y\_3 = sum([n\_x[i] \* y\_x\_cherta[i] \* mid\_borders\_x[i]\*\*0 for i in range(len(table))])

print(round(sum\_1))

print(round(sum\_2))

print(round(sum\_3))

print(round(sum\_4))

print(round(sum\_y\_1))

print(round(sum\_y\_2))

print(round(sum\_y\_3))

a = 1330980770795 / 7822224861288736

b = 1585312202138487 / 7822224861288736

c = (-574079274021227) / 7822224861288736

print()

print("a = {:.5f}".format(a))

print("b = {:.5f}".format(b))

print("c = {:.5f}".format(c))

print()

min\_val, max\_val = min(sample\_density), max(sample\_density)

xs = [min\_val+((max\_val-min\_val)/10000\*i) for i in range(10000)]

ys = [a\*xs[i]\*\*2 + b\*xs[i] + c for i in range(10000)]

ax.plot(xs, ys, label="$y\_x$") # yx

sum\_1 = sum([n\_y[i] \* mid\_borders\_y[i]\*\*1 for i in range(len(table))])

sum\_2 = sum([n\_y[i] \* mid\_borders\_y[i]\*\*2 for i in range(len(table))])

sum\_3 = sum([n\_y[i] \* mid\_borders\_y[i]\*\*3 for i in range(len(table))])

sum\_4 = sum([n\_y[i] \* mid\_borders\_y[i]\*\*4 for i in range(len(table))])

sum\_y\_1 = sum([n\_y[i] \* x\_y\_cherta[i] \* mid\_borders\_y[i]\*\*2 for i in range(len(table))])

sum\_y\_2 = sum([n\_y[i] \* x\_y\_cherta[i] \* mid\_borders\_y[i]\*\*1 for i in range(len(table))])

sum\_y\_3 = sum([n\_y[i] \* x\_y\_cherta[i] \* mid\_borders\_y[i]\*\*0 for i in range(len(table))])

print(round(sum\_1))

print(round(sum\_2))

print(round(sum\_3))

print(round(sum\_4))

print(round(sum\_y\_1))

print(round(sum\_y\_2))

print(round(sum\_y\_3))

a = 3400468111 / 11432286831000

b = 24837124860883 / 11432286831000

c = 7906387274955121 / 45729147324000

print()

print("a = {:.5f}".format(a))

print("b = {:.5f}".format(b))

print("c = {:.5f}".format(c))

print()

min\_val, max\_val = min(sample\_elastic), max(sample\_elastic)

ys = [min\_val+((max\_val-min\_val)/10000\*i) for i in range(10000)]

xs = [a\*ys[i]\*\*2 + b\*ys[i] + c for i in range(10000)]

ax.plot(xs, ys, label="$x\_y$")

ax.legend()

ax.set\_title('Уравнения параболической регрессии')

plt.show()

#lab\_\_\_\_\_\_6

from math import sqrt

import matplotlib.pyplot as plt

import copy

import lab1

K\_num = 6

def calcFirstCenters(sameple):

# Прикидываем центры для начала работы

min\_elem, max\_elem = min(sameple), max(sameple)

step = (max\_elem - min\_elem) / (K\_num)

borders = [min\_elem]

for i in range(K\_num):

borders.append(borders[i] + step)

centers = []

for i in range(K\_num):

centers.append((borders[i] + borders[i+1]) / 2)

return centers

def findRealClosestPointIndex(im\_point, sample):

# Ищем наиболее близкие точки в реальной выборке наивным поиском

indexOfMin = 0

currMinDistance = getDistance(im\_point, sample[0])

for i in range(1, len(sample)):

dist = getDistance(im\_point, sample[i])

if dist < currMinDistance:

currMinDistance = dist

indexOfMin = i

return indexOfMin

def getDistance(point\_1, point\_2):

# Евклидово расстояние с поправочным на масштаб коэффициентом

return sqrt((pow(point\_2[0] - point\_1[0], 2) + pow(ncf\*(point\_2[1] - point\_1[1]),2)))

def findClosestCluster(point, centers):

ds = [getDistance(point, center) for center in centers]

return ds.index(min(ds))

def recalcClusterCenter(cluster, cl\_index, centers):

mid\_x = sum([elem[0] for elem in cluster]) / len(cluster)

mid\_y = sum([elem[1] for elem in cluster]) / len(cluster)

centers[cl\_index] = [mid\_x, mid\_y]

def recalcAllCenters(clusters, centers):

for i, cluster in enumerate(clusters):

recalcClusterCenter(cluster, i, centers)

def initData(sample\_density, sample\_elastic):

# Инициализируем

sample\_2D = list(zip(sample\_density, sample\_elastic))

im\_centers = list(zip(calcFirstCenters(sample\_density), calcFirstCenters(sample\_elastic)))

pointsPull = sample\_2D[:]

centers = []

for im\_center in im\_centers:

indexOfRealPoint = findRealClosestPointIndex(im\_center, pointsPull)

print(indexOfRealPoint)

centers.append(sample\_2D[indexOfRealPoint])

del pointsPull[indexOfRealPoint]

clusters = [[centers[i]] for i in range(K\_num)]

return clusters, centers, pointsPull

def compareClusters(clusters\_1, clusters\_2):

if len(clusters\_1) != len(clusters\_2):

return False

for i in range(K\_num):

if len(clusters\_1[i]) != len(clusters\_2[i]):

return False

for j in range(len(clusters\_2[i])):

if clusters\_1[i][j] != clusters\_2[i][j]:

return False

return True

def F\_1(clusters, centers):

result = 0

for i, cluster in enumerate(clusters):

for elem in cluster:

result += getDistance(elem, centers[i])

return result

def F\_2(clusters):

result = 0

for cluster in clusters:

for i in range(len(cluster)):

for j in range(i, len(cluster)):

result += getDistance(cluster[i], cluster[j])

return result

def F\_3(clusters, centers):

result = 0

for i, cluster in enumerate(clusters):

sigma = 0

for elem in cluster:

sigma += pow(getDistance(elem, centers[i]), 2)

sigma /= len(cluster)

result += sigma

return result

def createColorGenerator():

colors = ["red", "blue", "black", "green", "cyan", "magenta", "yellow"]

i = 0

while (True):

yield colors[i]

i += 1

if i == len(colors):

i = 0

def k\_algo\_1(sample\_density, sample\_elastic):

# Первый вариант - пересчитываем центр после каждого обновления кластера

clusters, centers, pointsPull = initData(sample\_density, sample\_elastic)

print("Начальные центры:")

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

old\_clusters = []

for i in range(20):

for point in pointsPull:

index = findClosestCluster(point, centers)

clusters[index].append(point)

recalcClusterCenter(clusters[index], index, centers)

pointsPull = list(zip(sample\_density, sample\_elastic))

if i == 19:

print("Найдено устойчивое состояние! {0} совпадает с {1}".format(i, i+1))

# colors = ["blue", "red", "yellow", "black", "green", "cyan", "grey", "blue", "red", "yellow"]

fig, ax = plt.subplots()

colors = createColorGenerator()

print(colors)

for j, cluster in enumerate(clusters):

clusterColor = next(colors)

for point in cluster:

ax.plot(point[0], point[1], 'o', color=clusterColor, alpha=0.5)

colors = createColorGenerator()

for j, center in enumerate(centers):

clusterColor = next(colors)

ax.plot(center[0], center[1], 'x', color=clusterColor)

plt.title("Финал алгоритма с пересчётом на каждом шаге")

plt.show()

break

else:

print("{} шаг процедуры:".format(i+1))

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

print("F1: {:.2f}".format(F\_1(clusters, centers)))

print("F2: {:.2f}".format(F\_2(clusters)))

print("F3: {:.2f}".format(F\_3(clusters, centers)))

print(end="\n")

old\_clusters = copy.deepcopy(clusters)

clusters.clear()

for \_ in range(len(old\_clusters)):

clusters.append([])

def k\_algo\_2(sample\_density, sample\_elastic):

print("Начало алгоритма с пересчётом центра после просмотра всех данных")

clusters, centers, pointsPull = initData(sample\_density, sample\_elastic)

print("Начальные центры:")

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

old\_clusters = []

for i in range(20):

for point in pointsPull:

index = findClosestCluster(point, centers)

clusters[index].append(point)

recalcAllCenters(clusters, centers) # Пересчитываем центры после подсчёт статистики

if compareClusters(clusters, old\_clusters):

print("Найдено устойчивое состояние! {0} совпадает с {1}".format(i, i+1))

colors = ["blue", "red", "yellow", "black", "green", "cyan", "grey", "blue", "red", "yellow"]

fig, ax = plt.subplots()

for j, cluster in enumerate(clusters):

for point in cluster:

ax.plot(point[0], point[1], 'o', color=colors[j], alpha=0.5)

for j, center in enumerate(centers):

ax.plot(center[0], center[1], 'x', color=colors[j])

plt.title("Финал алгоритма с пересчётом после всей итерации")

plt.show()

break

else:

print("{} шаг процедуры:".format(i+1))

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

print("F1: {:.2f}".format(F\_1(clusters, centers)))

print("F2: {:.2f}".format(F\_2(clusters)))

print("F3: {:.2f}".format(F\_3(clusters, centers)))

print(end="\n")

pointsPull = list(zip(sample\_density, sample\_elastic))

old\_clusters = copy.deepcopy(clusters)

clusters.clear()

for \_ in range(len(old\_clusters)):

clusters.append([])

n = lab1.selection\_size

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

sample\_density = lab1.get\_sample\_first(general\_population, n)

sample\_elastic = lab1.get\_sample\_second(general\_population, n)

print(len(sample\_density) == len(sample\_elastic))

print(len(sample\_elastic))

min\_elem\_x, max\_elem\_x = min(sample\_density), max(sample\_density)

min\_elem\_y, max\_elem\_y = min(sample\_elastic), max(sample\_elastic)

ncf = (max\_elem\_x - min\_elem\_x) / (max\_elem\_y - min\_elem\_y)

print(ncf)

k\_algo\_1(sample\_density, sample\_elastic)

k\_algo\_2(sample\_density, sample\_elastic)

### lab\_\_\_\_\_\_\_7

from math import sqrt

import matplotlib.pyplot as plt

from matplotlib.lines import Line2D

import copy

import numpy as np

import lab1

def getDistance(point\_1, point\_2):

# Евклидово расстояние с поправочным на масштаб коэффициентом

return sqrt((pow(point\_2[0] - point\_1[0], 2) + pow(ncf\*(point\_2[1] - point\_1[1]),2)))

def compareClusters(list\_1, list\_2):

if len(list\_1) != len(list\_2):

return False

for i in range(len(list\_1)):

if len(list\_1[i]) != len(list\_2[i]):

return False

for j in range(len(list\_2[i])):

if list\_1[i][j] != list\_2[i][j]:

return False

return True

def recalcClusterCenter(cluster):

mid\_x = sum([elem[0] for elem in cluster]) / len(cluster)

mid\_y = sum([elem[1] for elem in cluster]) / len(cluster)

return [mid\_x, mid\_y]

def findNewPopularPoint(points, delta):

if len(points) == 1:

return 0

index = -1

maxNeig = 1

for i, cur\_point in enumerate(points):

num = len([point for point in points if getDistance(cur\_point, point) <= delta])

if num > maxNeig:

maxNeig = num

index = i

return index

def F\_1(clusters, centers):

result = 0

for i, cluster in enumerate(clusters):

for elem in cluster:

result += getDistance(elem, centers[i])

return result

def F\_2(clusters):

result = 0

for cluster in clusters:

for i in range(len(cluster)):

for j in range(i, len(cluster)):

result += getDistance(cluster[i], cluster[j])

return result

def F\_3(clusters, centers):

result = 0

for i, cluster in enumerate(clusters):

sigma = 0

for elem in cluster:

sigma += pow(getDistance(elem, centers[i]), 2)

sigma /= len(cluster)

result += sigma

return result

def makeClusters(data1, data2, R):

size = len(data1)

history = []

points = list(zip(data1, data2))

clusters = []

iters = 0

delta = 1

points\_to\_process = points[:]

while len(points\_to\_process) > 0:

iters += 1

index = np.random.randint(0, len(points\_to\_process))

while True:

index = findNewPopularPoint(points\_to\_process, delta)

if index == -1:

delta += 1

if (delta > R1):

index = 0

break

else:

break

center = points\_to\_process[index]

points\_in\_cluster = [point for point in points\_to\_process if getDistance(point, center) < R]

iter = 0

while True:

iter += 1

center\_new = recalcClusterCenter(points\_in\_cluster)

if (abs(center[0] - center\_new[0]) + abs(center[1] - center\_new[1])) < 0.001:

break

points\_in\_cluster = [point for point in points\_to\_process if getDistance(point, center\_new) < R]

center = center\_new

if iter == 100:

break

for cl\_point in points\_in\_cluster:

for point in points\_to\_process:

if (abs(cl\_point[0] - point[0]) + abs(cl\_point[1] - point[1])) < 0.001:

points\_to\_process.remove(cl\_point)

clusters.append(points\_in\_cluster)

return clusters

def createColorGenerator():

colors = ["red", "blue", "black", "green", "cyan", "grey", "magenta", "yellow", "lime"]

i = 0

while (True):

yield colors[i]

i += 1

if i == len(colors):

i = 0

def plotClusters(clusters, clusterMeans, R):

ax = plt.axes(aspect='equal')

colors = createColorGenerator()

for i, cluster in enumerate(clusters):

x, y = list(zip(\*cluster))

clusterColor = next(colors)

plt.scatter(x, y, marker=".", color=clusterColor, alpha=0.5)

ax.add\_artist(plt.Circle(clusterMeans[i], R, color=clusterColor, alpha=0.1))

xs = [mean[0] for mean in clusterMeans]

ys = [mean[1] for mean in clusterMeans]

plt.scatter(xs[i], ys[i], color = clusterColor, marker='o')

plt.show()

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

n = lab1.selection\_size

general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

sample\_density = lab1.get\_sample\_first(general\_population, n)

sample\_elastic = lab1.get\_sample\_second(general\_population, n)

# n = len(sample\_density)

sample\_2D = list(zip(sample\_density, sample\_elastic))

min\_elem\_x, max\_elem\_x = min(sample\_density), max(sample\_density)

min\_elem\_y, max\_elem\_y = min(sample\_elastic), max(sample\_elastic)

ncf = (max\_elem\_x - min\_elem\_x) / (max\_elem\_y - min\_elem\_y)

minR = np.Inf

maxR = -np.Inf

for p1 in sample\_2D:

for p2 in sample\_2D:

d = getDistance(p1, p2)

if 0 < d < minR:

minR = d

if d > maxR:

maxR = d

print("Минимальный радиус:", minR)

print("Максимальный радиус:", maxR)

R1 = 59.5

# clusters = makeClusters(sample\_density, sample\_elastic, R1, maxIters=10)

# centers = [recalcClusterCenter(cluster) for cluster in clusters]

# print("Clusters num: {0}".format(len(clusters)))

# print(F\_1(clusters, centers))

# print(F\_2(clusters))

# print(F\_3(clusters, centers))

# print(sorted(centers))

# plotClusters(clusters, centers, R1)

# exit(0)

old\_clusters = []

for R in np.linspace(minR, maxR, 143): # 68 400 200

clusters = makeClusters(sample\_density, sample\_elastic, R)

centers = [recalcClusterCenter(cluster) for cluster in clusters]

print("Clusters num: {0}".format(len(clusters)))

print("F1: {:.2f}".format(F\_1(clusters, centers)))

print("F2: {:.2f}".format(F\_2(clusters)))

print("F3: {:.2f}".format(F\_3(clusters, centers)))

print("\n")

if compareClusters(clusters, old\_clusters):

# if len(clusters) == 5:

# Устойчивое!

print("success! ", R)

print(sorted(centers))

print("R: {:.2f}".format(R))

plotClusters(clusters, centers, R)

break

old\_clusters = copy.deepcopy(clusters)

plt.show()