**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕНЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

**Кафедра теории вероятностей и математической статистики**

П

Лев П

**Эффективность последовательных статистических решающих правил для наблюдений из дискретных распределений вероятностей с искажениями**

Дипломная работа

Научный руководитель:

доктор физико-математических наук,

профессор Харин Алексей Юрьевич

Допущена к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022 г

Зав. кафедрой теории вероятностей и математической

статистики, доктор физико-математических наук, профессор Харин А.Ю.

Минск, 2022

**Оглавление**

[ВВЕДЕНИЕ 7](#_Toc105400893)

[ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ. ВВОДНЫЕ ОПРЕДЕЛЕНИЯ 8](#_Toc105400894)

[Постановка задачи 8](#_Toc105400895)

[Актуальность и практическая значимость 9](#_Toc105400896)

[Общие теоретические положения 9](#_Toc105400897)

[Общая модель задачи 10](#_Toc105400898)

[ГЛАВА 2 ПОСЛЕДОВАТЕЛЬНЫЕ СТАТИСТИЧЕСКИЕ ТЕСТЫ В АНАЛИЗЕ СЛУЧАЙНЫХ ВЕЛИЧИН, РАСПРЕДЕЛЕННЫХ ПО ЗАКОНУ ПУАССОНА 12](#_Toc105400899)

[ГЛАВА 3 ПОСЛЕДОВАТЕЛЬНЫЕ ТЕСТЫ ДЛЯ НАБЛЮДЕНИЙ С ИСКАЖЕНИЯМИ ИЗ РАСПРЕДЕЛЕНИЙ ТОГО ЖЕ КЛАССА И С ФИКСИРОВАННЫМ ВЫБРОСОМ 15](#_Toc105400900)

[3.1. Искажения первого типа 15](#_Toc105400901)

[3.2. Искажения второго типа 17](#_Toc105400902)

[ГЛАВА 4 ПОСЛЕДОВАТЕЛЬНЫЕ ТЕСТЫ С ИСКЖЕНИЯМИ, РАСПРЕДЕЛЕННЫМИ ПО ДРУГИМ ЗАКОНАМ 19](#_Toc105400903)

[4.1. Искажения первого типа 20](#_Toc105400904)

[4.2. Искажения второго типа 21](#_Toc105400905)

[ГЛАВА 5 ПОСЛЕДОВАТЕЛЬНЫЕ СТАТИСТИЧЕСКИЕ ТЕСТЫ В АНАЛИЗЕ ДАННЫХ, РАСПРЕДЕЛЁННЫХ ПО ПОЛИНОМИАЛЬНОМУ ЗАКОНУ. ЭФФЕКТИВНОСТЬ ПОСЛЕДОВАТЕЛЬНЫХ ТЕСТОВ 24](#_Toc105400906)

[5.1 Исследование полиномиального распределения 24](#_Toc105400907)

[5.2 Эффективность последовательных тестов 30](#_Toc105400908)

[ЗАКЛЮЧЕНИЕ 34](#_Toc105400909)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc105400910)

[ПРИЛОЖЕНИЕ А 37](#_Toc105400911)

[ПРИЛОЖЕНИЕ Б 43](#_Toc105400912)

**РЕФЕРАТ**

**Дипломная работа**, 33 страницы, 11 таблиц, 29 рисунков, 11 источников.

**Ключевые слова** –простая гипотеза, последовательная проверка, распределение Пуассона, метод Монте-Карло, тест Вальда, искажения, Геометрическое распределение, Полиномиальное распределение.

**Объекты исследования** –теоретические и фактические вероятности ошибки при принятии одной из простых гипотез; результаты тестов Вальда в случае появления искажений во входных данных, а также зависимость полученных результатов от модели и объёма искажений; способы модификации последовательного теста.

**Методы исследования** –изучение теории, составление тестирующего алгоритма и анализ полученных в ходе экспериментов данных.

**Цели работы** –изучить связь оценок ошибок при принятии гипотез с фактической вероятностью ошибки с помощью последовательных тестов; изучить поведение тестов Вальда в случае появления искажений; изучить способы модификации тестов с целью повышения их эффективности.

**Области применения** –области, нуждающиеся в статистической обработке дискретных числовых данных, а также данных, распределённых по полиномиальному закону.

**Результаты** –алгоритм, сравнительные эксперименты, вопросы для дальнейших исследований.

**ABSTRACT**

**Diploma Thesis**, 33 pages, 11 tables, 29 pictures, 11 sources.

**Keywords** – simple hypothesis, sequential testing, Poisson distribution, Monte Carlo method, Wald test, distortion, Geometric distribution, Polynomial distribution.

**Objects of research** – theoretical and actual probabilities of error when making one of the simple hypotheses; the results of the Wald tests in case of distortions in the input data, as well as the dependence of the obtained results on the model and the number of distortions; ways to modify a sequential test.

**Research methods** – studying the theory, compiling a testing algorithm and analyzing the data obtained during the experiments.

**Work Purpose** – to study the relationship of error estimates when accepting hypotheses with the actual probability of error using sequential tests; to study the behavior of Wald tests in case of distortions; explore ways to modify tests to improve their performance.

**Application areas** – areas requiring statistical processing of discrete numerical data, as well as data distributed according to a polynomial law.

**Results** – algorithm, comparative experiments, questions for further research.

**РЭФЕРАТ**

**Дыпломная работа**, 33 старонкі, 11 табліц, 29 малюнкаў, 11 крыніц.

**Ключавыя словы** – простая гіпотэза, паслядоўны тэст, размеркаванне Пуасона, метад Монтэ-Карла, тэст Вальда, скажэнні, Геаметрычнае размеркаванне, Паліномнае размеркаванне.

**Аб'екты даследавання** – тэарэтычныя і фактычныя імавернасці абмылы пры прыняцці адной з простых гіпотэз; вынікі тэстаў Вальда ў выпадку з'яўлення перакручванняў ва ўваходных дадзеных, а таксама залежнасць атрыманых вынікаў ад мадэлі і аб'ёму перакручванняў; спосабы мадыфікацыі паслядоўнага тэсту.

**Метады даследавання** – вывучэнне тэорыі, складанне які тэстуе алгарытму і аналіз атрыманых падчас эксперыментаў дадзеных.

**Мэты работы** – вывучыць сувязь ацэнак абмылаў пры прыняцці гіпотэз з фактычнай імавернасцю абмылы з дапамогай паслядоўных тэстаў; вывучыць паводзіны тэстаў Вальда ў выпадку з'яўлення перакручванняў; вывучыць спосабы мадыфікацыі тэстаў дзеля падвышэння іх эфектыўнасці.

**Вобласці ўжывання** – вобласці, што патрабуюць у статыстычнай апрацоўцы дыскрэтных лікавых дадзеных, а таксама дадзеных, размеркаваных па паліномным законе.

**Атрыманыя вынікі** – алгарытм, параўнальныя эксперыменты, пытанні для дальнейшых даследаванняў.

ВВЕДЕНИЕ

Статистические методы используются во многих областях науки. И многие дисциплины, имеющие дело с реальной жизнью (вроде экономики, медицины и т.п.), требуют обработки различных числовых данных. Числовые данные в свою очередь являются объектом изучения математической статистики. Так вот одним из методов работы с такими данными является проверка гипотез.

В данной работе мною был рассмотрен последовательный тест для проверки двух простых гипотез. В последовательной проверке, в отличие от других способов, количество наблюдений, необходимых для принятия решения, зависит от результатов самих наблюдений, и, по факту, является случайной величиной. Работает данный метод следующим образом: после каждого наблюдения требуется принять решение – принять одну из гипотез или провести дополнительное наблюдение. Если одна из двух гипотез принята – проверка заканчивается, если нет – производится следующее наблюдение. На основе уже произведённых наблюдений принимается решение в каждом следующем случае повторного наблюдения до принятия одной из гипотез.

Преимущество последовательного анализа заключается в отсутствии необходимости работать со всей выборкой целиком, поскольку она формируется в ходе самого эксперимента. Таким образом, для работы со случайными величинами достаточно лишь знать, в соответствии с каким распределением приходят случайные величины, что может быть полезно при синхронной обработке данных (т.е. сразу при их поступлении), чтобы можно было уточнять параметры распределения в процессе работы чего-либо, не дожидаясь её завершения. В частности, такая ситуация может сложиться в теории массового обслуживания. Можно также отметить, что простейший поток в теории называется потоком Пуассона (число событий, выпадающих на интервал некоторой длины распределено по Закону Пуассона), а в данной работе были рассмотрены как раз простые гипотезы о параметре распределения Пуассона.

Кроме того, в последствии было исследовано поведение теста для полиномиально распределённой случайной величины, и, на её примере, подробнее рассмотрено, каким образом исходные параметры влияют на работу теста, а также какие существуют способы повышения эффективности работы последовательного теста.

ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ. ВВОДНЫЕ ОПРЕДЕЛЕНИЯ

1.1 Постановка задачи

1. Реализовать на компьютере последовательный тест для проверки двух простых гипотез о параметре распределения Пуассона.

2. Оценить методом Монте-Карло вероятности ошибочных решений реализованного теста по схеме:

* задали ;
* вычислили значения порогов;
* запустили 1000 раз тест при справедливой гипотезе ();
* подсчитали число неверных решений, разделили на 1000.

3. Оценить методом Монте-Карло математические ожидания числа случайного наблюдений до остановки теста при справедливых нулевой и первой гипотезах по схеме:

* задали;
* вычислили значения порогов;
* запустили 1000 раз тест при справедливой гипотезе ();
* подсчитали суммарное число наблюдений, разделили на 1000.

4. Построить таблицу полученных результатов при разных α, β из множества {0.1, 0.05, 0.01}. Сравнить полученные результаты.

5. Реализовать модель тестов с искажениями во входных данных, построить графики, отражающие поведение тестов в условиях данных искажений.

1) Искажения вида

а)

б)

где ;

2) Искажения вида

а)

б)

таким образом математическое ожидание распределений, генерирующих помехи, совпадает с таковым у тестируемого распределения.

6. По полученным графикам проанализировать поведение тестов в зависимости от параметров, влияющих на степень искажения входных данных, сравнить две модели помех.

7. Применить данные методы исследования к другим, более сложным распределениям, например, полиномиальному.

## 1.2 Актуальность и практическая значимость

Статистические методы применяются в огромном числе различных видах деятельности, подразумевающих обработку числовых данных. И данная работа позволяет получить представление о том, каким образом повысить вероятность принять верную из гипотез.

В данной работе исследовались гипотезы о параметре распределения Пуассона, которое находит применение в т.ч. и в изучаемой нами теории массового обслуживания, а также во множестве других сфер научной и практической деятельности.

## 1.3 Общие теоретические положения

Поскольку мы исследуем последовательный тест двух простых гипотез, начнём с основных определений:

Простая гипотеза – статистическая гипотеза, однозначно определяющая распределение (предположение о значении некоторого параметра; в нашем случае – параметр распределения Пуассона).

Уровень значимости теста — вероятность отклонить гипотезу , если на самом деле она верна.

Последовательная проверка (тест) двух простых гипотез – метод, при котором после каждого испытания принимается, по ранее сформулированному правилу, одно из трёх решений: 1) принять гипотезу ; 2) отклонить гипотезу ; 3) продолжить наблюдение. Причём каждое следующее наблюдение основывается на всех предыдущих. В дальнейшем при подсчёте наблюдений учитывались количества именно этих дополнительных наблюдений.

нулевая гипотеза, конкурирующая. Тогда:

Ошибка первого рода – отвержение верной гипотезы .

Ошибка второго рода – принятие ложной гипотезы (отвержение верной ).

Искажения – часть входных данных, распределённых не по предполагаемому закону, а по некоторой другой модели, будь то другой параметр распределения или всё распределение вовсе.

## 1.4 Общая модель задачи

В данной работе сначала рассматривается Пуассоновская модель распределения случайных величин, поэтому в первых трёх главах работа проводится с дискретными случайными величинами. Далее было рассмотрено поведение последовательных тестов на полиномиальном распределении случайных величин, применены схожие методы исследования с поправкой на отличия вида отдельно взятых случайных величин.

На вход подаются гипотезы (некоторые их значения) и их уровни значимости. Затем эти гипотезы проверяются (по правилу\*) на некоторой сгенерированной последовательности случайных величин, распределённых по закону Пуассона с параметром, равным одной из проверяемых гипотез.

Далее с помощью метода Монте-Карло (описан в условии) получаем оценки числа неверно принятых решений и математического ожидания числа наблюдений (всего проводится 1000 испытаний).

И наконец, строим таблицу полученных значений вероятностей ошибок первого и второго рода для (всего 9 значений).

\*Правило(алгоритм) проверки гипотез:

Пусть есть гипотеза, что , – гипотеза, что . Тогда:

1. По заданным вычисляем значения порогов (:

2. Вычисляем значения функции плотности вероятности от проверяемых гипотез и очередной случайной величины:

3. Считаем

где – вероятность получения выборки, когда справедлива гипотеза соответственно (.

4. Принимаем решение:

Если – отклоняем гипотезу ;

Если – принимаем гипотезу ;

Иначе (если ) – проводим дополнительное наблюдение. Далее в работе под количеством наблюдений подразумевается именно количество этих дополнительных наблюдений. Чтобы перейти от них к общему числу – достаточно лишь добавить число запусков теста.

Данный процесс и называется последовательным критерием отношений вероятности для проверки гипотезы относительно . Он был рассмотрен во второй части работы.

В третьей части было рассмотрено поведение тестов Вальда в случае появления искажений двух видов. В первом случае искажения были представлены двумя моделями (либо распределены по тому же закону с другим параметром, либо сразу шли в тест):

а)

б)

где ;

Чтобы понять, есть в наблюдении искажение или нет, была введена вспомогательная случайная величина . Если , то искажений нет. Если – есть искажение. Очевидно, что чем больше , тем больше искажений.

В четвёртой части был рассмотрен другой тип искажений, для их генерации использовались другие целочисленные распределения – Биномиальное и Геометрическое:

а)

б)

В последней части были рассмотрена проблематика прошлых частей, где в качестве основного распределения было взято полиномиальное распределение и проведено через аналогичные этапы исследования.

ГЛАВА 2 ПОСЛЕДОВАТЕЛЬНЫЕ СТАТИСТИЧЕСКИЕ ТЕСТЫ В АНАЛИЗЕ СЛУЧАЙНЫХ ВЕЛИЧИН, РАСПРЕДЕЛЕННЫХ ПО ЗАКОНУ ПУАССОНА

Для каждого набора параметров было проведено по 5 тестов, – полученные вероятности ошибки первого (второго) рода соответственно; – матожидания числа дополнительных наблюдений до принятия одной из гипотез.

Итак, перейдём к полученным результатам:

1) , результаты в таблице 2.1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Тест 1 | 0.026 | 0.016 | 0.436 | 0.594 |
| Тест 2 | 0.022 | 0.019 | 0.471 | 0.558 |
| Тест 3 | 0.022 | 0.023 | 0.453 | 0.568 |
| Тест 4 | 0.019 | 0.019 | 0.446 | 0.587 |
| Тест 5 | 0.019 | 0.016 | 0.463 | 0.552 |
| **средние** | **0.0216** | **0.0186** | **0.4538** | **0.5718** |

Таблица 2.1

Отклонения уровней значимости от реальных вероятностей ошибок:

Стоит отметить, что вероятность ошибки понижается с уменьшением причём зачастую независимо (таблица 2.2):

|  |  |  |  |
| --- | --- | --- | --- |
| α β | 0.1 | 0.05 | 0.01 |
| 0.1 |  |  |  |
| 0.05 |  |  |  |
| 0.01 |  |  |  |

Таблица 2.2

2) – увеличили разницу м-у :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Тест 1 | 0.019 | 0.035 | 0.277 | 0.582 |
| Тест 2 | 0.015 | 0.045 | 0.288 | 0.587 |
| Тест 3 | 0.016 | 0.035 | 0.292 | 0.547 |
| Тест 4 | 0.015 | 0.045 | 0.312 | 0.586 |
| Тест 5 | 0.011 | 0.042 | 0.313 | 0.62 |
| **средние** | **0.0152** | **0.0404** | **0.2964** | **0.5844** |

Таблица 2.3

Отклонения уровней значимости от реальных вероятностей ошибок:

Интересно, что в первом случае как и здесь, но фактическая вероятность ошибки первого рода в первом пункте несколько выше, чем у ошибок второго рода; тогда как здесь разница уже почти 3 раза и имеет тот же знак, что и между оценками. Также стоит отметить, что с увеличением выросло и его отклонение от реального значения.

3) – здесь мы уменьшили разницу между гипотезами (оценки оставили как в пункте 1):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Тест 1 | 0.059 | 0.095 | 68.395 | 77.506 |
| Тест 2 | 0.064 | 0.083 | 71.242 | 78.627 |
| Тест 3 | 0.064 | 0.073 | 73.081 | 78.28 |
| Тест 4 | 0.057 | 0.086 | 71.042 | 76.33 |
| Тест 5 | 0.061 | 0.094 | 69.198 | 78.993 |
| **средние** | **0.061** | **0.0862** | **70.5916** | **77.9472** |

Таблица 2.4

Отклонения уровней значимости от реальных вероятностей ошибок:

Что интересно, здесь фактические значения стали ближе к теоретическим. Видим также существенный скачок среднего числа наблюдений до принятия одной из гипотез – поскольку при меньшей разнице между параметрами, сумма делает меньшие шаги из области, при получении значения в которой требуется произвести очередное наблюдение.

Затем мы снова рассмотрели влияние уровней значимости – здесь наибольшая точность оценок получилась при , а не 0.01 как в первом тесте. Но разница с фактическим значением уменьшается по мере уменьшения , это легко видеть хотя бы потому, что наибольшая разница всегда для , а наименьшая получилась при меньших их значениях в обоих тестах.

|  |  |  |  |
| --- | --- | --- | --- |
| α β | 0.1 | 0.05 | 0.01 |
| 0.1 |  |  |  |
| 0.05 |  |  |  |
| 0.01 |  |  |  |

Таблица 2.5

Сравним также отклонения диагональных элементов этой таблицы

с соответствующими из первого теста:

Легко видеть, что во всех трёх случаях отклонения оценок от реальных результатов в первом тесте больше, чем во втором. Значит уменьшение разницы между гипотезами существенно повышает точность оценок (уровней значимости).

ГЛАВА 3 ПОСЛЕДОВАТЕЛЬНЫЕ ТЕСТЫ ДЛЯ НАБЛЮДЕНИЙ С ИСКАЖЕНИЯМИ ИЗ РАСПРЕДЕЛЕНИЙ ТОГО ЖЕ КЛАССА И С ФИКСИРОВАННЫМ ВЫБРОСОМ

Рассмотрим теперь поведение тестов, в случае появления искажений.

Общие параметры, взятые для проведения всех тестов этого раздела:

Поскольку шаг , то для каждого тест запускался 50000 раз, поэтому накопленные ошибки не могут превысить этот порог.

**3.1. Искажения первого типа**

Здесь распределение имеет следующий вид:

Как и ранее, рассмотрим два случая, когда верна одна из гипотез:

* Верна гипотеза

Количество ошибок первого рода растёт с увеличением тем быстрее, чем больше разница (рисунок 3.1, справа).

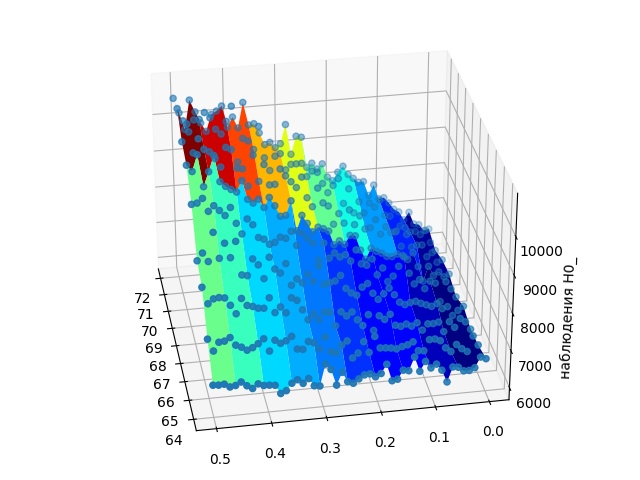
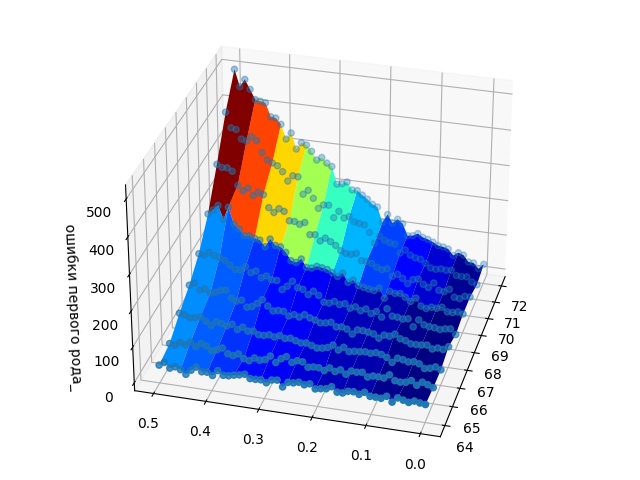
**

Рисунок 3.1 – результаты работы тестов при верной первой гипотезе

Причём, если , то, по сути, у нас в модели нет искажений, поэтому вне зависимости от , количество ошибок не меняется. Также можно обратить внимание на накопленные наблюдения и ошибки – с увеличением зависимость количества ошибок от постепенно переходит от линейной к экспоненциальной (рисунок 3.2, справа):

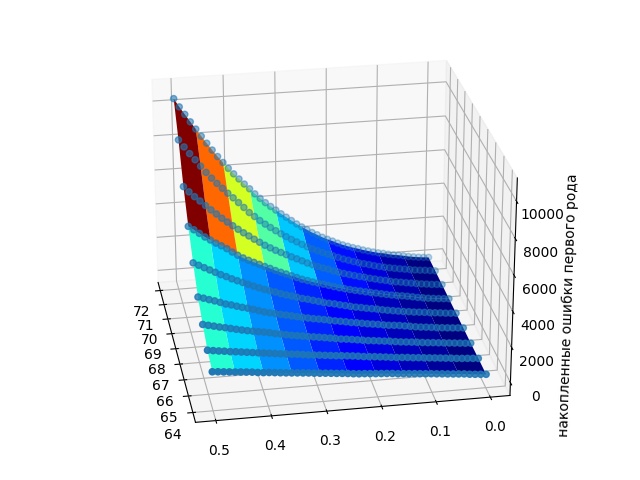
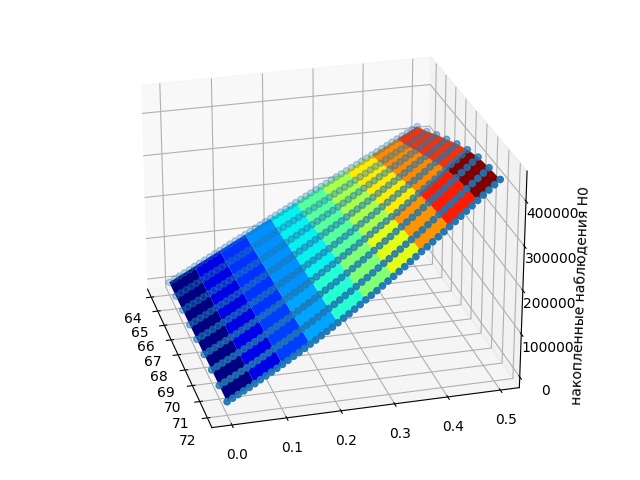


Рисунок 3.2 – накопленные результаты работы тестов при верной первой гипотезе

Количество же наблюдений растёт по тем быстрее, чем больше . Зависимость количества наблюдений от близка к линейной, с увеличением меняется, по сути, угол наклона прямой, что также можно видеть и по накопленным наблюдениям.

* Верна конкурирующая гипотеза

Имеем в целом симметричную ситуацию относительно : можно наблюдать те же тренды, количества наблюдений и ошибок практически совпадают:

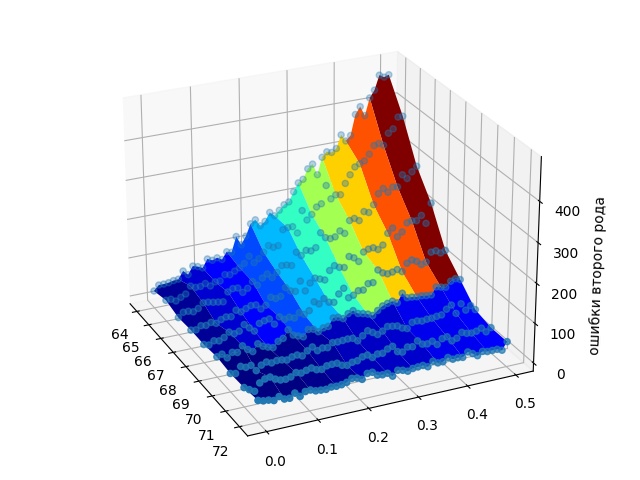
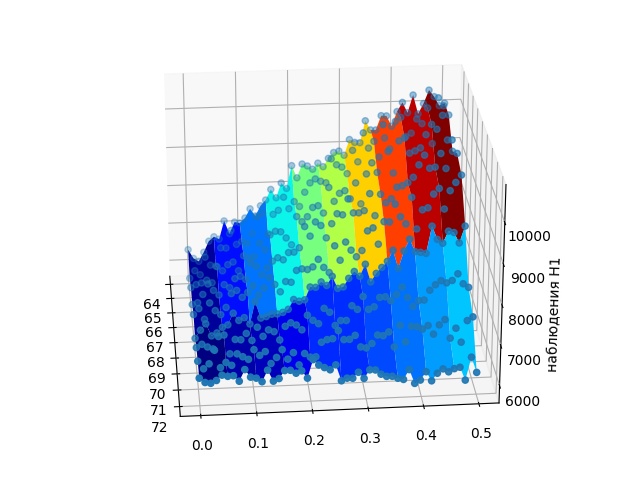


Рисунок 3.3 – результаты работы тестов при верной второй гипотезе

**3.2. Искажения второго типа**

Здесь распределение имеет следующий вид:

т.е. с вероятностью в качестве плотности засоряющего распределения добавляется дельта-функция Дирака в точке .

* Верна гипотеза , имеем:

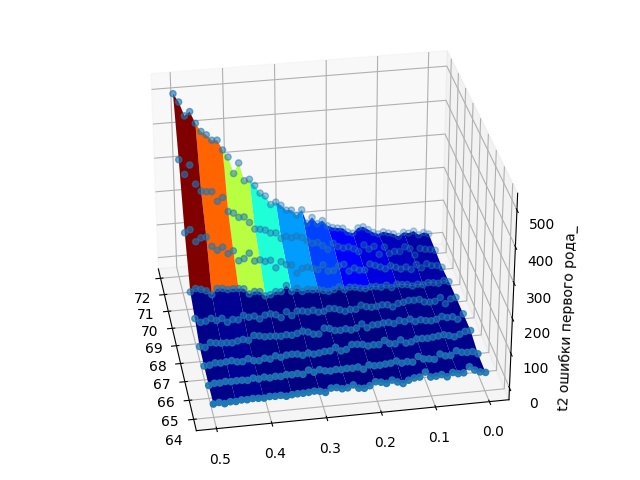
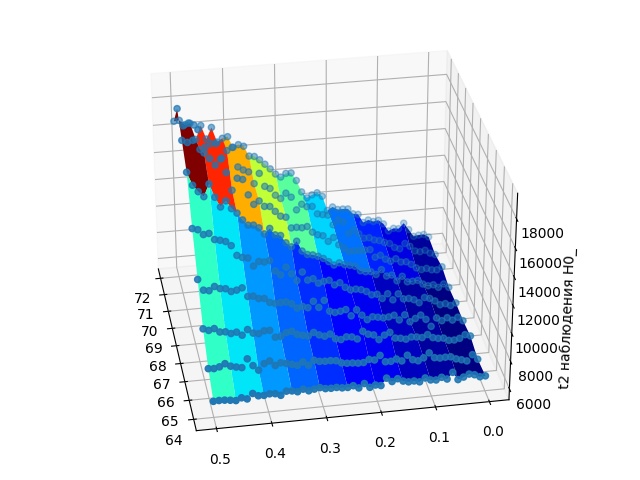
****

Рисунок 3.4 – результаты тестов при верной первой гипотезе

Ситуация схожая, однако здесь для принятия решения потребовалось гораздо больше наблюдений (порой почти вдвое больше), а также количество ошибок возрастает заметно резче, чем в предыдущей модели искажений.

* Верна конкурирующая гипотеза , имеем:

Ситуация снова симметрична относительно в обоих случаях (рисунок 3.5).

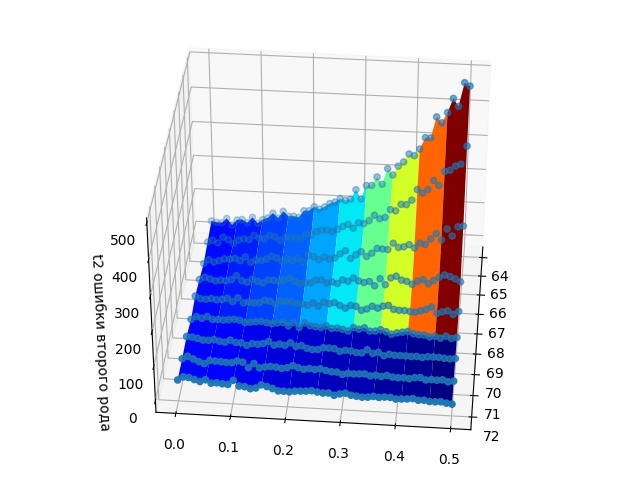
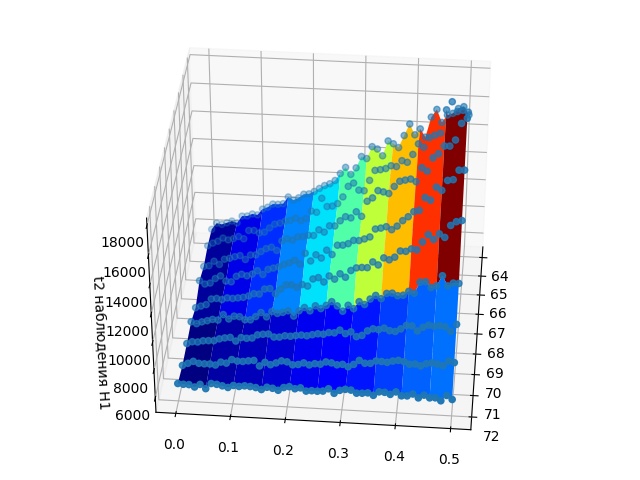
****

Рисунок 3.5 – результаты тестов при верной второй гипотезе

Однако здесь уже просматривается некоторый «изгиб» графика количеств наблюдений от линейного к экспоненциальному, а также ярче выражено аналогичное явление для ошибок (рисунок 3.6):

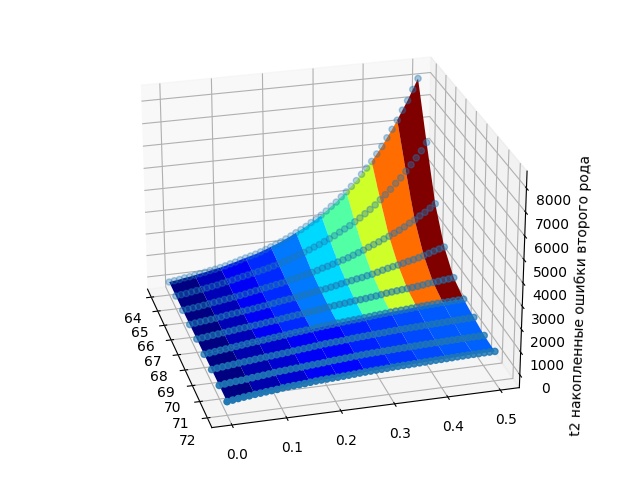
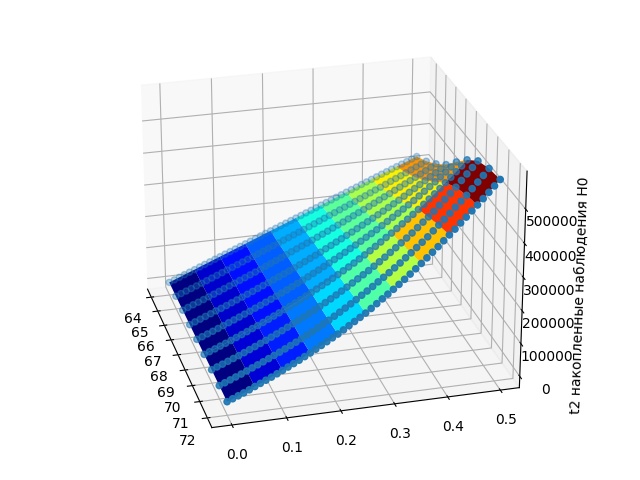
****

Рисунок 3.6 – накопленные результаты тестов при верной второй гипотезе

В накопленных значениях также можно увидеть подтверждение того, что потребовалось большее число наблюдений и что количество ошибок растёт более резко. Однако, стоит также отметить, что, во сколько раз накопленное число наблюдений больше, чем для первой модели искажений, во столько раз накопленное число ошибок меньше во второй (~1/4), таким образом в данной модели тест реже ошибался за счёт увеличения числа наблюдений.

Можно также заметить, что здесь тест Вальда достаточно устойчив к искажениям во входных данных — количество ошибок серьёзно увеличивается только по приближении количества искажений к половине от всех данных, что также можно наблюдать и по средним значениям параметров (таблица 3.1):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Модель искажений 1 | | | | Модель искажений 2 | | | |
| θ\* | верна H0: θ=64 | | верна H1: θ=72 | | верна H0: θ=64 | | верна H1: θ=72 | |
| наблю  дения | ошибки 1  рода | наблю  дения | ошибки 2 рода | наблю  дения | ошибки 1 рода | наблю  дения | ошибки 2 рода |
| 64 | 6412.08 | 29.88 | 8582.26 | 197.62 | 6454.02 | 15.1 | 10778.7 | 162.2 |
| 65 | 6761.74 | 34.94 | 8407.42 | 166.7 | 6937.14 | 15.22 | 10692.46 | 115.34 |
| 66 | 7110.68 | 46.04 | 8180.1 | 132.06 | 7558.18 | 18.24 | 10318.48 | 71.18 |
| 67 | 7513.72 | 61.84 | 8006.14 | 100.04 | 8297.98 | 22.2 | 9602.16 | 43.74 |
| 68 | 7927.88 | 80.78 | 7675.8 | 76.58 | 9246.76 | 29.48 | 8842.74 | 28.34 |
| 69 | 8138.2 | 107.98 | 7254.64 | 58.42 | 10156.26 | 44.44 | 7960.12 | 21.72 |
| 70 | 8435.64 | 137.74 | 6898.3 | 46.36 | 10874.98 | 75.88 | 7340.92 | 18.3 |
| 71 | 8641.14 | 175.5 | 6548.3 | 36.1 | 11220.42 | 119.08 | 6711.48 | 15.4 |
| 72 | 8676.28 | 212.3 | 6242.58 | 29.56 | 11372.58 | 170.92 | 6255.74 | 14.8 |

Таблица 3.1 - средние количества наблюдений и ошибок 1/2 рода

ГЛАВА 4 ПОСЛЕДОВАТЕЛЬНЫЕ ТЕСТЫ С ИСКАЖЕНИЯМИ, РАСПРЕДЕЛЕННЫМИ ПО ДРУГИМ ЗАКОНАМ

Здесь мы проверим 2 гипотезы в условиях наличия искажений, распределённых по некоторому другому закону. Для начала вспомним, как ведёт себя модель без искажений во входных данных. Как и ранее, тест был запущен 1000 раз, для каждого запуска считались ошибки и дополнительные наблюдения, рассмотренные гипотезы и заданные уровни значимости:

Результаты для каждой из гипотез приведены на рисунках 4.1 и 4.2:

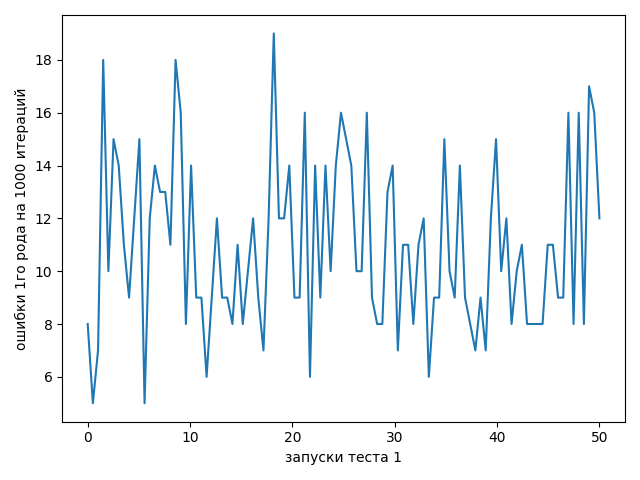
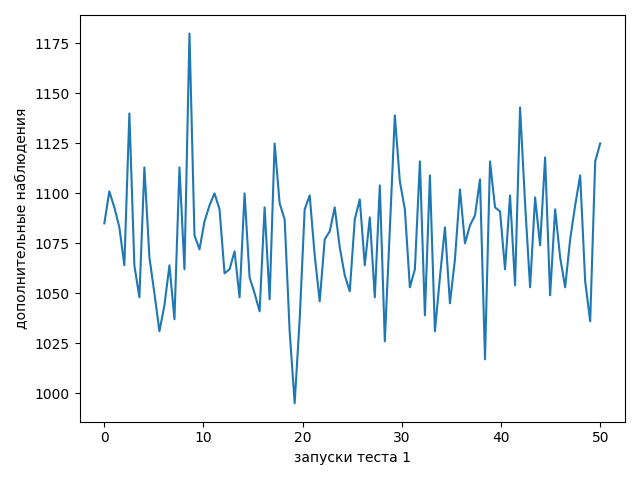


Рисунок 4.1 – результаты при первой верной гипотезе

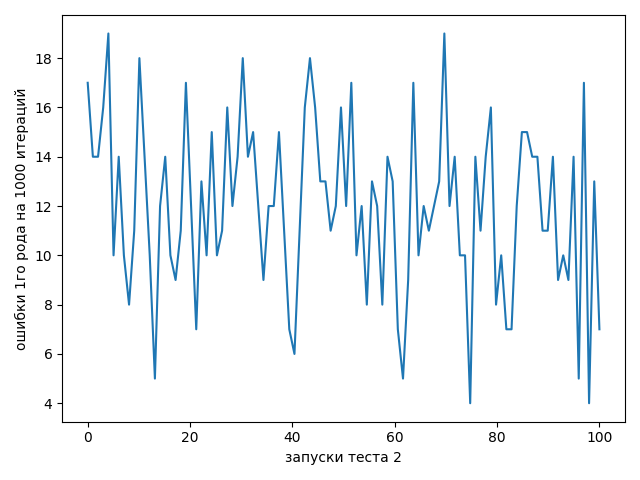
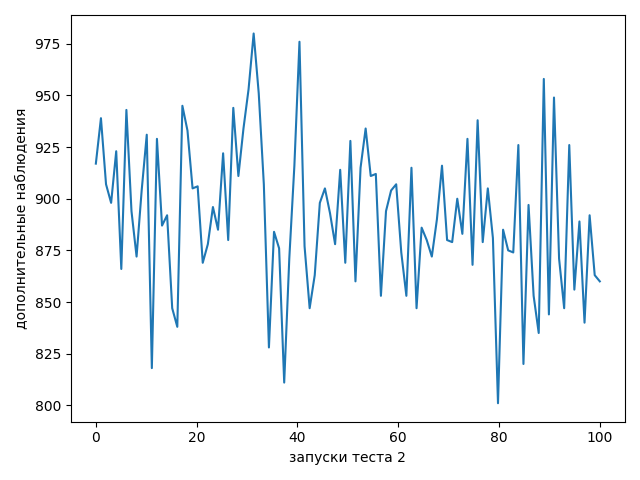


Рисунок 4.2 – результаты при верной второй гипотезе

Видим, что процент ошибочных решений от общего числа проведённых тестов примерно соответствует уровню значимости, как и было установлено во второй главе.

## 4.1. Искажения первого типа

В этом случае распределение моделируется следующим образом:

В первой рассмотренной модели с искажениями засоряющие данные были распределены по биномиальному закону, результаты на рисунке 4.3:

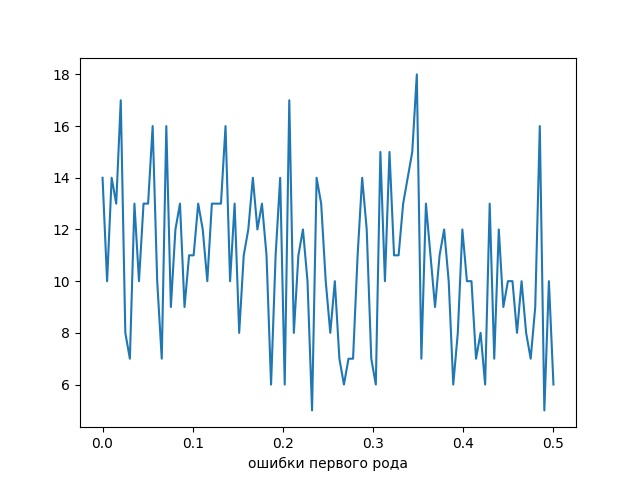
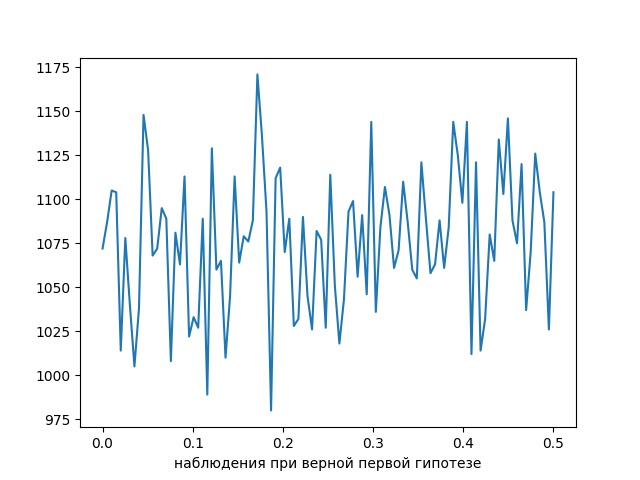


Рисунок 4.3 – зависимость результатов от числа искажений, распределенных биномиально

В целом видим аналогичную картину (если верна другая гипотеза, результаты такие же), это можно объяснить схожестью оригинального распределения (Пуассона) и генерирующего помехи (Биномиального):

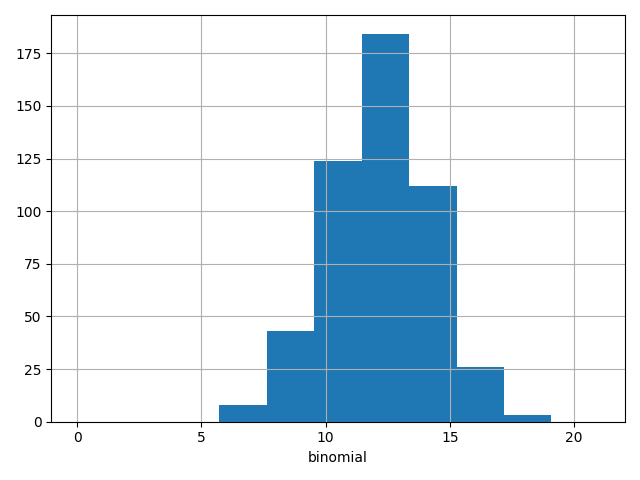
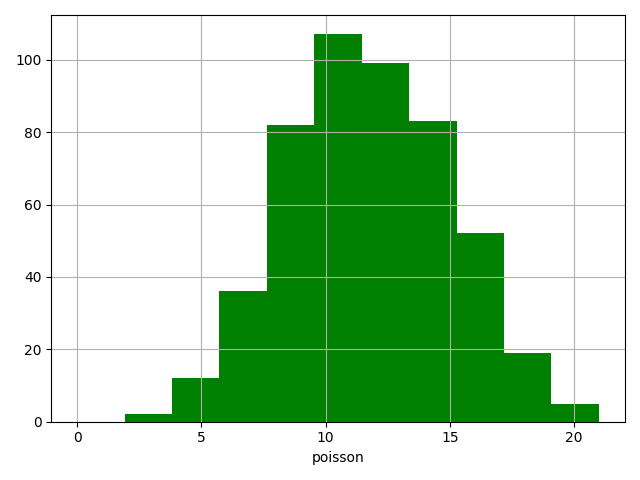


Рисунок 4.4 – сравнение выборок распределений Пуассона (слева) и биномиального (справа)

В данном случае для простоты и наглядности сравнения, у рассматриваемых распределений (и геометрического), определено матожидание, равное 12, выборка – 1000 случайных значений.

А вот с геометрическим распределением помех дела обстоят несколько интереснее (рисунок 4.5):

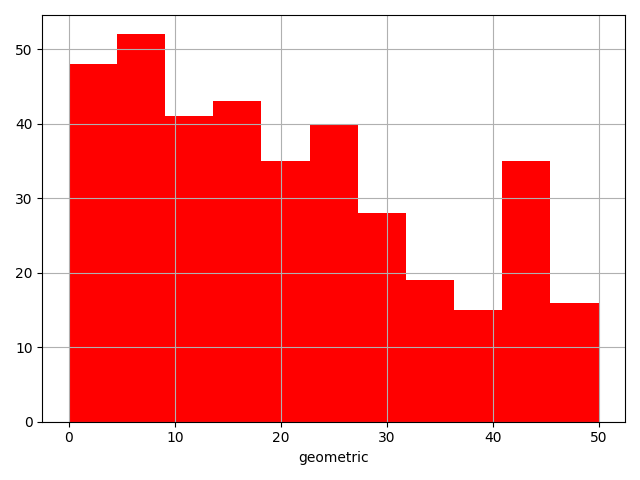


Рисунок 4.5 – графическое представление выборки, распределённой по геометрическому закону

## 4.2. Искажения второго типа

Распределение имеет вид

Рассмотрим поведение тестов при значениях параметров

.

Если верная первая гипотеза:

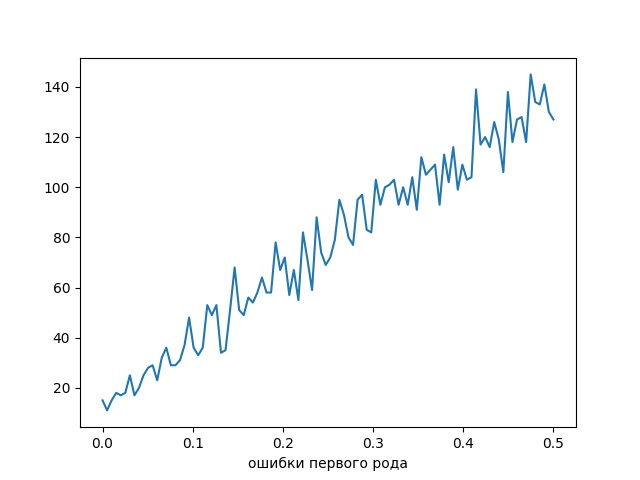
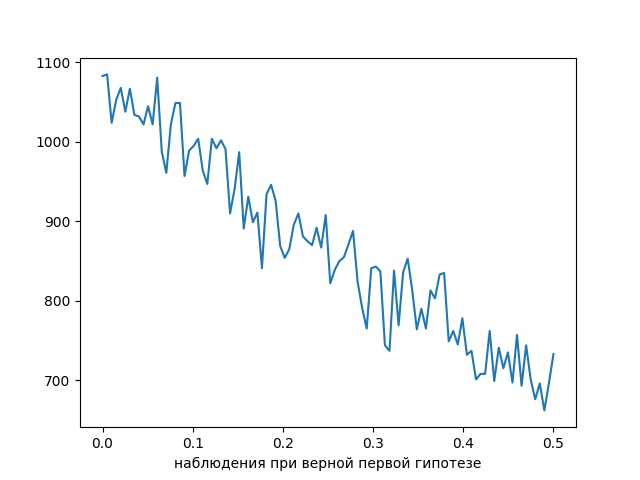


Рисунок 4.6 – зависимость значений параметров от количества искажений, верна

Здесь видим тренд снижения количества наблюдений и роста количества ошибок с ростом количества искажённых случайных величин, таким образом тест принимал решение раньше и заметно чаще ошибался.

И при верной второй гипотезе:

Имеем схожий тренд при меньшем числе наблюдений и большем числе ошибок (рисунок 3.7).

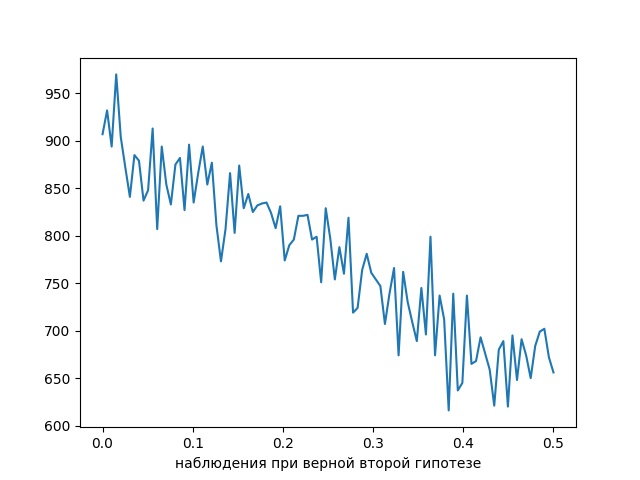
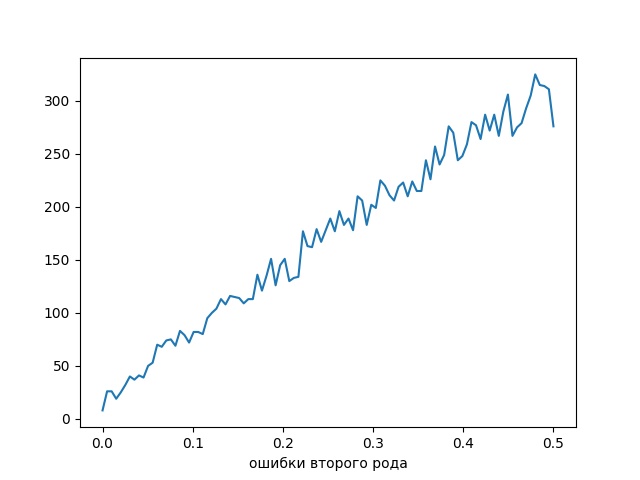


Рисунок 4.7 – зависимость значений параметров от количества искажений, верна

Как видим, кроме тренда по количеству искажённых данных, есть также тренд по значению второй гипотезы – чем оно больше, тем больше ошибок второго рода:

При значениях параметров и верной первой гипотезе:

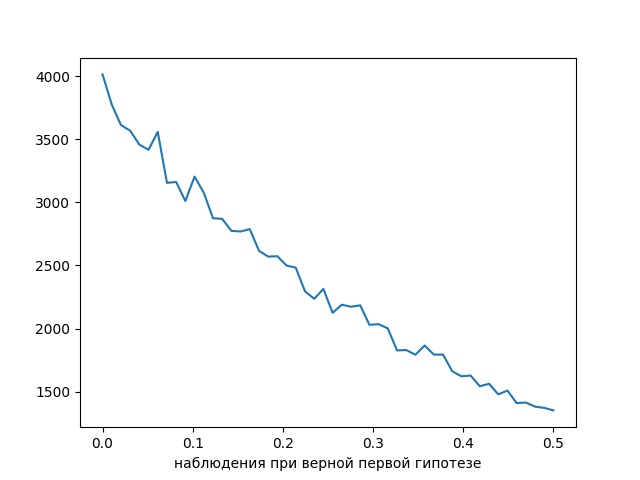
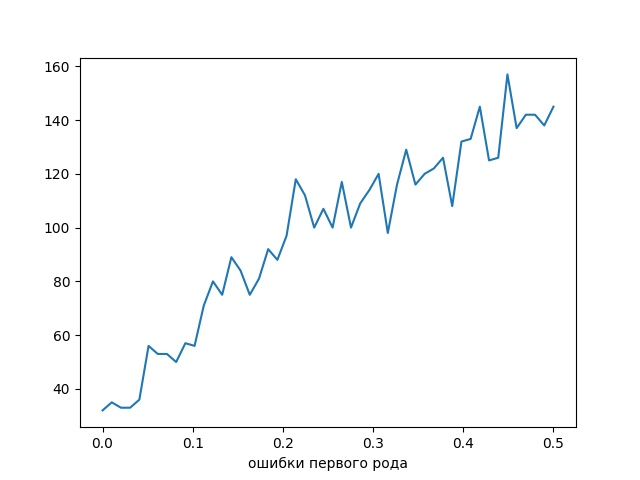
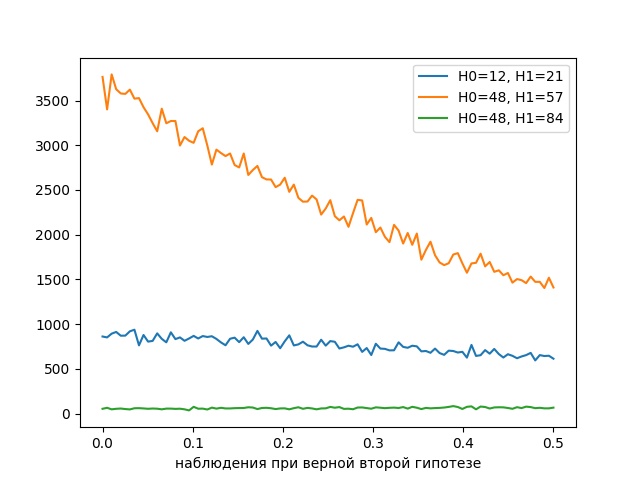


Рисунок 4.8 – зависимость значений параметров от количества искажений, верна

На рисунке 4.9 видим, что для первой гипотезы изменилось лишь число наблюдений, в то время как для второй, при относительно таком же числе наблюедний, резко выросло число ошибочных решений (жёлтая линия). Если же увеличить их пропорционально, то резко упадёт количество наблюдений при относительном сохранении количества ошибочных решений (зелёная линия), а как ранее было установлено – это и ожидается при увеличении разницы между гипотезами:



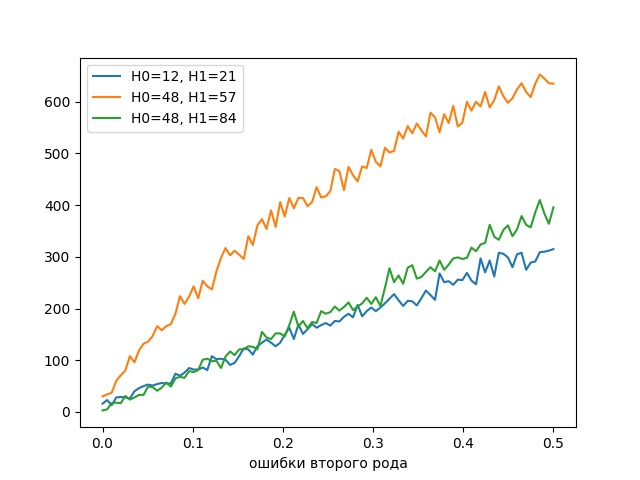


Рисунок 4.9 – сравнительные результаты для различных наборов параметров

Такое поведение можно объяснить относительной «скученностью» около нуля случайных значений, распределённых по геометрическому закону.

ГЛАВА 5 ПОСЛЕДОВАТЕЛЬНЫЕ СТАТИСТИЧЕСКИЕ ТЕСТЫ В АНАЛИЗЕ ДАННЫХ, РАСПРЕДЕЛЁННЫХ ПО ПОЛИНОМИАЛЬНОМУ ЗАКОНУ. ЭФФЕКТИВНОСТЬ ПОСЛЕДОВАТЕЛЬНЫХ ТЕСТОВ

В данном разделе для проведения дальнейший исследований было выбрано полиномиальное распределение и на его примере подробно изучено, какие параметры распределения сказываются на поведении последовательных тестов и каким образом это влияние выражено. За основу для проведения исследования была взята модель распределения “игральная кость” – вектор из 6 значений, каждое задаёт вероятность выпадения каждой из 6 граней кубика. В качестве первой гипотезы был выбран “идеальный” кубик, конкурирующая гипотеза варьировалась.

## 5.1 Исследование полиномиального распределения

Для проведения исследований в этом классе случайных величин, нам нужно несколько модифицировать полученный ранее алгоритм, ведь сейчас случайная величина представлена вектором, а не числом. Теперь, плотность вероятности считается следующим образом:

где – очередная случайная величина и гипотеза соответственно, представленные векторами длины 6, по количеству граней кости; – число бросков кости, задаётся перед проведением теста. Остальные формулы те же. Аналогично для конкурирующей гипотезы.

Итак, на первом шаге были рассмотрены различные варианты конкурирующей гипотезы, как и ранее, согласно методу Монте-Карло, тест запускался 1000 раз, в результате каждого теста записывалось число дополнительных наблюдений до принятия одной из гипотез, а также число ошибочных решений. Во всех случаях нулевая гипотеза имела вид

Полученные в первом случае результаты отображены в таблице 4.1:

| Средние значения характеристик, набор A | | | |
| --- | --- | --- | --- |
| **Верна гипотеза** | **Наблюдения** | **Ошибки** | **Вероятность ошибки** |
| 1 | 3693.746154 | 22.307692 | 0.022308 |
| 2 | 3410.823077 | 25.707692 | 0.025708 |

Таблица 5.1 – Характеристики для 1-го набора параметров

Здесь, в связи с увеличением разницы между гипотезами, закономерно падает число ошибочных решений и дополнительных наблюдений. Полученные в этом случае результаты отображены далее в таблице 4.2:

| Средние значения характеристик, набор Б | | | |
| --- | --- | --- | --- |
| **Верна гипотеза** | **Наблюдения** | **Ошибки** | **Вероятность ошибки** |
| 1 | 1128.676923 | 16.538462 | 0.016538 |
| 2 | 931.246154 | 16.092308 | 0.016092 |

Таблица 5.2 – Характеристики для 2-го набора параметров

В данном случае были получены результаты близкие набору параметров а), однако с меньшим числом дополнительных наблюдений, приведены в таблице 4.3:

| Средние значения характеристик, набор В | | | |
| --- | --- | --- | --- |
| **Верна гипотеза** | **Наблюдения** | **Ошибки** | **Вероятность ошибки** |
| 1 | 2411.876923 | 22.061538 | 0.022062 |
| 2 | 2380.084615 | 23.100000 | 0.023100 |

Таблица 5.3 – Характеристики для 3-го набора параметров

Как и можно было предположить, от порядка вероятностей выпадения полученный результат работы теста не зависит, приведён в таблице 4.4:

| Средние значения характеристик, набор Г | | | |
| --- | --- | --- | --- |
| **Верна гипотеза** | **Наблюдения** | **Ошибки** | **Вероятность ошибки** |
| 1 | 2419.607692 | 22.146154 | 0.022146 |
| 2 | 2371.615385 | 22.746154 | 0.022746 |

Таблица 5.4 – Характеристики для 4-го набора параметров

Таким образом, в дальнейшем за основную была взята модель из случая а), т.к. характеристики устойчивости теста в ней были выражены наиболее ярко.

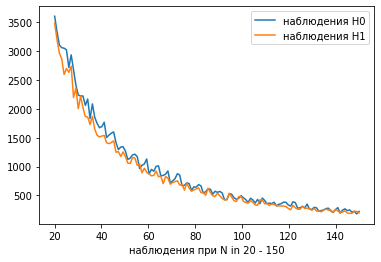
На втором шаге было рассмотрено влияние уровней значимости на поведение теста, результаты представлены в таблице 4.5:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Уровни значимо-сти | Верна первая гипотеза | | | Верна вторая гипотеза | | |
| наблюдения | ошибки | вероятность  ошибки | наблюдения | ошибки | вероятность  ошибки |
| a = 0.05,  b = 0.05 | 3691.369 | 22.054 | 2.2% | 3420.269 | 26.085 | 2.6% |
| a = 0.05,  b = 0.1 | 2747.369 | 23.4 | 2.34% | 3111.661 | 54.99 | 5.499% |
| a = 0.1,  b = 0.05 | 3421.438 | 44.285 | 4.43% | 2620.876 | 27.885 | 2.79% |
| a = 0.05,  b = 0.01 | 5667.384 | 22.485 | 2.25% | 3571.907 | 5.531 | 0.55% |
| a = 0.01,  b = 0.05 | 3849.492 | 4.862 | 0.49% | 5142.876 | 26.48 | 2.65% |
| a = 0.1,  b = 0.01 | 5437.492 | 46.37 | 4.6% | 2783.946 | 5.1 | 0.51% |

Таблица 5.5 – Средние значения характеристик при различных уровнях значимости

Как и ожидалось, количество принимаемых ошибочных решений непосредственно связано с уровнями значимости и сразу реагирует на малейшие их изменения – во сколько раз отличаются уровни значимости, примерно во столько же раз отличаются полученные вероятности принятия ошибочного решения.

На третьем шаге была рассмотрена зависимость поведения тестов от количества бросков кубика для получения отдельной случайной величины, полученные характеристики представлены на рисунке 5.1:

****

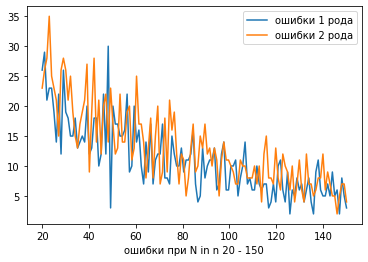
****

Рисунок 5.1 – Зависимость характеристик от числа бросков кости

Видим явную тенденцию к повышению точности и эффективности теста при увеличении количества бросков кубика. Таким образом здесь мы можем влиять на эффективность принятия решений тестом, не меняя как гипотез, так и самого теста. На практике это значит, что чем лучше подготовим данные – тем лучше будет работать последовательный тест. В данном случае можно установить примерную планку для хорошей работы теста в 80-100 бросков кубика – далее количество ошибок снижается не так значительно, зато заметно возрастает количество вычислений и риск переполнения, ведь формула плотности вероятности использует факториал.

На четвёртом шаге было рассмотрено поведение теста в условиях наличия искажений, распределеных со значением параметра, пробегающим значения из отрезка между двумя гипотезами:

Данный отрезок разбивался на 10 частей, количество искажений изменялось от 0 до половины наблюдаемых значений за 50 шагов. Количество бросков кости – . Также в силу большого количества вычислений, итераций было взято 500 вместо 1000, результаты представлены на рисунке 5.2:

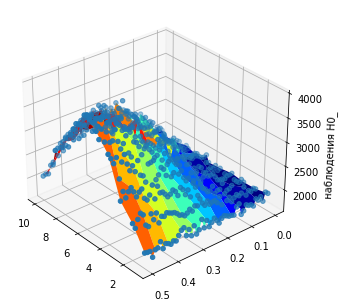
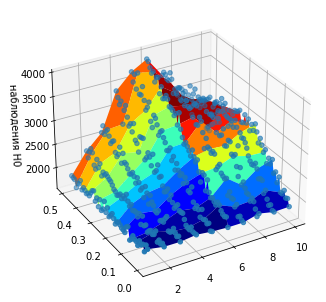
****

Рисунок 5.2 – Зависимость числа наблюдений от числа помех и значения параметра искажающего распределения

Здесь, видим отличный от распределения Пуассона тренд на увеличение количества наблюдений с увеличением числа искажений: тем ярче он выражен, чем ближе к середине отрезка между значениями проверяемых гипотез значение параметра искажающего распределения. В случае распределения Пуассона, количество наблюдений увеличивалось по мере приближения значения параметра искажающего распределения к значению параметра конкурирующей гипотезы.

Среднее число ошибок растёт достаточно равномерно по мере отклонения значения параметра распределения помех от , рисунок 5.3:

Рисунок 5.3 – Измеренная средняя вероятность ошибки 1-го рода

Общее число ошибок первого рода в целом соответствует полученным ранее результатам – имеем те же тренды как по количеству, так и по «силе» искажений, полученные характеристики представлены на рисунке 5.4:

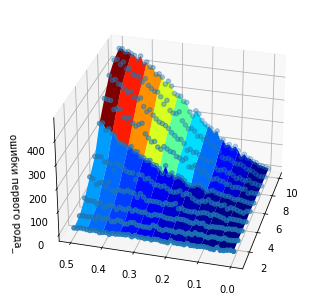
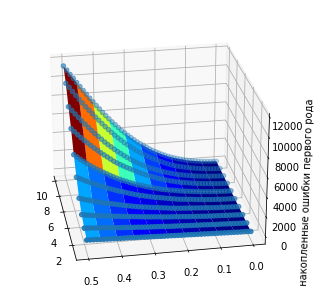
 

Рисунок 5.4 – Количественные характеристики ошибок 1-го рода на 500 запусков теста

Стоит заметить, что несмотря на аналогичные распределению Пуассона результаты в отсутствии искажений (количество ошибок ~2.5% при уровнях значимости ), здесь в худшем случае имеем до 90% ошибочных решений – из 500 итераций было получена максимально 461 ошибка, что свидетельствует о низкой устойчивости последовательного теста к искажениям в случае полиномиального распределения.

Для верной второй гипотезы ситуация в целом аналогичная, с несколько большим числом наблюдений и, что важнее и выражено ярче, большим числом ошибочных решений, рисунок 5.5:

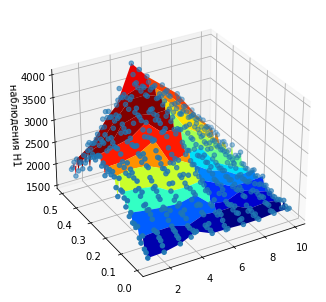
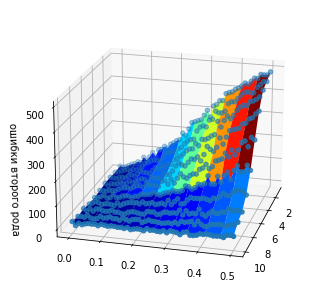
 

Рисунок 5.5 – Полученные характеристики при верной второй гипотезе, N = 20

Также стоит отметить, что увеличение количества бросков здесь уже не помогает, ведь оно влияет и на число искажений. При и 1000 запусках получили (рисунок 5.6):

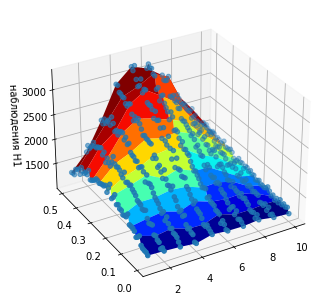
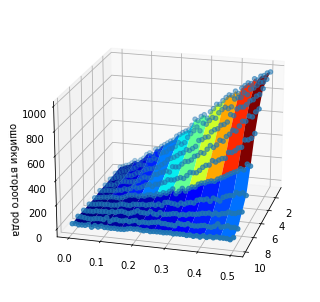
 

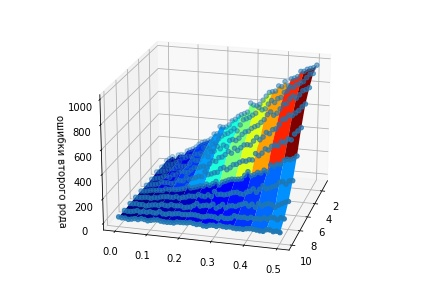
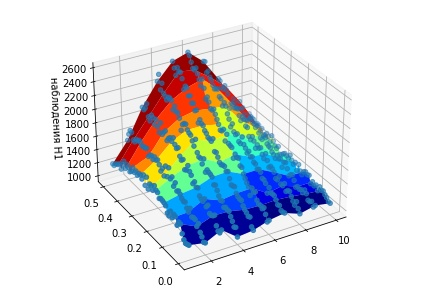
Рисунок 5.6 – Полученные характеристики при верной второй гипотезе, N = 50

Таким образом встаёт вопрос поиска других способов повышения эффективности теста.

## 5.2 Эффективность последовательных тестов

На последнем шаге предыдущей части была получена достаточно низкая эффективность теста, а значит, стоит рассмотреть способы модификации теста. Для начала, поскольку количество наблюдений растёт достаточно резко, а серьёзного эффекта от этого не наблюдается, попробуем ввести «ускоритель теста» – коэффициент, на который будем домножать каждое очередное значение в формуле из первой главы и который ускорит выход значения расчётного параметра из зоны безразличия. Причём этот коэффициент будет становиться тем больше, чем больше дополнительных наблюдений проведено. Будем проводить эксперименты при и числе итераций равном 1000, верна .

Попробуем для начала взять геометрическую прогрессию в качестве этого коэффициента с множителем 1.1, полученные результаты на рисунке 5.7:

Рисунок 5.7 – Полученные характеристики с «ускоряющим коэффициентом»

Видим уменьшение числа наблюдений (до 400 для самого большого количества наблюдений) при относительном сохранении ситуации в отношении ошибок. Значит всё-таки имеет смысл рассмотреть зависимость количества наблюдений от «ускоряющего коэффициента.

Зададим параметры модели для проведения эксперимента. Зафиксируем в качестве искажающей величины значение – середина отрезка , т.е. на предыдущем рисунке это линии из точек в районе значений 5-6 по нижней целочисленной оси. Там мы имели до 2500 наблюдения и до 400 ошибок второго рода в зависимости от числа искажений. Рассмотрим теперь вместо изменения значения параметра, искажающего распределение изменение «ускоряющего коэффициента». Для проверки был взят вектор таких коэффициентов

В полученных результатах отчётливо видно снижение дополнительных наблюдений, в нашем случае заметный эффект достигается для значения коэффициента 2.5. Значит с точки зрения наблюдений, коэффициент работает как нужно, результаты представлены на рисунке 5.8:

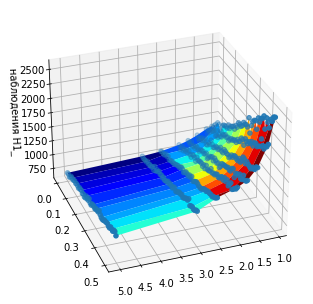
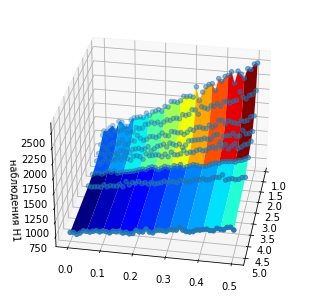
****

Рисунок 5.8 – Зависимость числа наблюдений от значения «ускоряющего коэффициента»»

А что же у нас по ошибкам? Здесь ситуация интереснее: чем больше «ускоряющий коэффициент», тем больше у нас становится ошибочных решений в отсутствии искажений. Однако при увеличении числа «искажённых» случайных величин, эта разница нивелируется и при половине искажённых данных почти не заметна, рисунок 5.9:

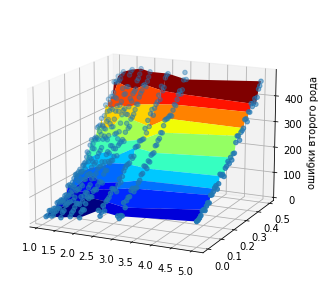
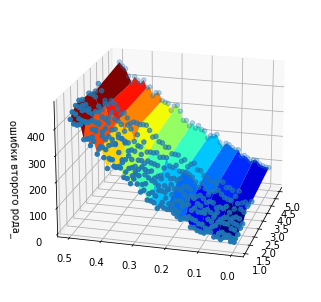


Рисунок 5.9 – Зависимость числа ошибок от значения «ускоряющего коэффициента»

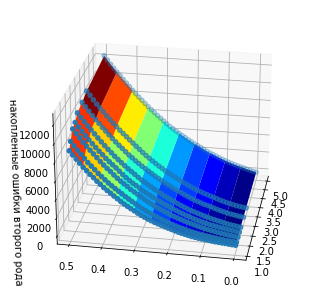
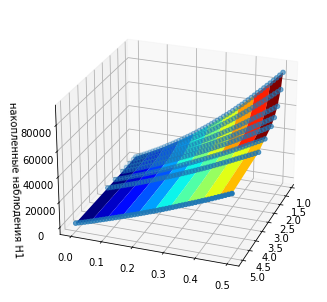


Рисунок 5.10 – Накопленные характеристики в зависимости от значения коэффициента

По представленным на рисунке 5.10 накопленным результатам видно, что при последних 3-х значениях коэффициентах достаточно резко возрастает общее число ошибок второго рода, в то время как количество дополнительных наблюдений снижается не настолько сильно. Значит в нашем случае т.н. «оптимальным» значением коэффициента будет что-то в районе 2.5, возможно даже число .

Поскольку мы убедились, что с помощью этого коэффициента можно снизить количество наблюдений, существенно не повлияв на число ошибок, то можно также поставить задачу о нахождении оптимального значения этого коэффициента, однако сейчас мы это пропустим.

С другой стороны, в отсутствии искажений мы заметили некоторый скачок ошибочных решений, если значение этого коэффициента было достаточно велико, причём это существенно не влияло на ошибки в случае половины случайных величин, распределённых по «засоряющему» закону. Таким образом, вероятно, с помощью данного коэффициента с некоторыми доработками можно узнавать о наличии искажений в обрабатываемых нами данных. Это также может стать темой для дальнейших исследований.

ЗАКЛЮЧЕНИЕ

В ходе проведённых экспериментов было установлено, что теоретические вероятности ошибок в выборе верной гипотезы (уровни значимости) практически всегда отличаются от практических, причём чаще в большую сторону. Также они влияют на число наблюдений (при большей оценке происходит больше наблюдений), требуемых для принятия одной из гипотез. Например, при изменении значений уровней значимости в разные стороны, фактические вероятности ошибки в большинстве случаев изменятся в таком же ключе.

Также было установлено, что уровни значимости дают тем более точную оценку, чем ближе значения гипотез друг к другу, но практически никогда не совпадают с фактическими вероятностями ошибок. Важно отметить, что зачастую точность повышается за счёт значительного увеличения числа наблюдений до принятия одной из гипотез, что может повлиять на время вычислений.

Таким образом, для повышения шанса принять верную гипотезу о параметре некоторого распределения (в нашем случае – Пуассона, а также Полиномиального), гораздо важнее подобрать сами гипотезы, нежели их уровни значимости. Если же высокая точность нам не особо важна, то можно ограничится лишь выбором последних не трогая гипотезы. В таком случае лучше всё-таки стараться задавать уровни значимости как можно меньше – тогда (как было установлено для обоих распределений) и фактическая вероятность ошибки будет меньше.

Стоит отметить и тот факт, что с уменьшением разницы между гипотезами повышается точность оценок возможной ошибки – т.е. уменьшается средняя разница между уровнем значимости и фактической вероятностью ошибки. Но также значительно повышается требуемое число наблюдений, что может существенно повлиять на скорость вычислений.

Что касается искажений во входных данных, то для первого типа, когда искажения распределены по некоторому закону, это вызывает больше ошибок, чем если они идут напрямую в тест. Но такое получается за счёт увеличения числа наблюдений, как видно на примере модели 1.2. Для второго типа искажений было установлено, что разница законов распределения напрямую отражается на разнице в принятии решений. Стоит также отметить и устойчивость теста к искажениям в первом случае – если их не более 20%, то тест работает с достаточно высокой точностью для обеих моделей искажений, серьёзные неточности возникают только при сильных искажениях. А поскольку повышенное число наблюдений дало более высокую точность, возможно стоит искусственно повысить «помехоустойчивость» теста, проводя дополнительные наблюдения после принятия первого решения (например, принимаем решения 3 раза и по ним делаем вывод). Для второго типа искажений ещё значительную роль играет характер помех – так, для искажений, распределённых по геометрическому закону, уже при 10% помех количество ошибочных решений было около 20%.

Также было установлено, что тест Вальда достаточно устойчив к искажениям во входных данных, кроме случаев, когда искажённые данные серьёзно отличаются от реальных (например, принадлежат другому распределению с совершенно другими свойствами) – тогда это сразу отражается на результатах теста. Серьёзные ошибки в принятии решений возникают в достаточно небольшом интервале от количества искажений, сопоставимом с количеством неискажённых данных. Если искажений ещё больше, то, возможно, стоит рассматривать модель с искажениями в качестве основной. Таким образом тест может эффективно принимать решения примерно для половины значений . А в случае серьёзного отличия искажённых данных справедливо предположить, что, проведя ряд наблюдений для разных входных параметров, можно установить характер искажений во входных данных. В таком случае мы сможем с высокой точностью «отфильтровывать» нужные данные, игнорируя искажённые.

На последнем шаге исследования был разработан механизм, позволяющий несколько повысить эффективность последовательного теста при наличии искажений путем ускорения принятия решения с помощью «ускоряющего коэффициента». Также было установлено, что этот коэффициент тем меньше влияет на получаемые результаты, чем больше искажений есть во входных данных, таким образом добавлением этого коэффициента можно примерно определить количество искаженных данных. Определение эффективности отдельных значений и попытку найти оптимальное значение такого коэффициента можно вынести в отдельный объект дальнейших исследований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А. Вальд «Последовательный анализ» / А. Вальд – Москва, 1960. – 329 с.
2. Wald, A.  Sequential Analysis/ A. Wald. – John Wiley & Sons, New York-London-Sydney, 1947.
3. Харин, А. Ю. Методы анализа эффективности последовательных статистических тестов / А. Ю. Харин// Доклады национальной академии наук Беларуси. – Минск, 2017. – Т. 6, №5. –С. 22 - 27.
4. Ширяев, А. Н. Статистический  последовательный анализ / А. Н. Ширяев. – М.: Наука, 1976. – 272 с.
5. Ермаков С.М. «Метод Монте-Карло и смежные вопросы» / С.М. Ермаков – М.: Наука, 1975. – 472 c.
6. Распределение Пуассона [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Распределение\_Пуассона.
7. Функция вероятности [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Функция\_вероятности.
8. Использование библиотеки Matplotlib. Как рисовать трехмерные графики [Электронный ресурс]. – Режим доступа: <https://jenyay.net/Programming/Python3d>
9. Matplotlib. Урок 5. Построение 3D графиков. Работа с mplot3d Toolkit [Электронный ресурс]. – Режим доступа: <https://devpractice.ru/matplotlib-lesson-5-1-mplot3d-toolkit>.
10. Мультиномиальное распределение [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Мультиномиальное_распределение>.
11. Mukhopadhyay N. Applied Sequential Methodologies / N. Mukhopadhyay, S. Datta, S. Chattopadhyay.–  New York: MarcelDekker, 2004. – 480 p.

ПРИЛОЖЕНИЕ А

Код программы для глав 1-4:

import math

import matplotlib.pyplot as plt

from matplotlib import cm

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import statistics as stats

import time

def calcA(a, b):

return (1. - b) / a

def calcB(a, b):

return b / (1. - a)

def pfx(x, lam):

return lam \*\* x / (math.factorial(x) \* math.exp(lam))

def calcZ(x, theta0, theta1):

# f.write("\nlatest div = " + str(pfx(x, teta1)) + ' ' + str(pfx(x, teta0)) + " for x = " + str(x) + " for t0 = " + str(teta0) + " for t1 = " + str(teta1))

return math.log(pfx(x, theta1) / pfx(x, theta0))

n = 1000

obses = 1000

steps = 200

teta0 = 12

teta1 = 21

teta0s = 48

teta1s = 57

teta0ss = 48

teta1ss = 84

a = 0.05

b = 0.05

A = calcA(a, b)

B = calcB(a, b)

workNum = steps // 2

def drawplot(mas, rotate, name):

mas = np.array(mas)

mas = mas.transpose()

fig = plt.figure()

axes = Axes3D(fig)

xval = np.linspace(teta0, teta1, teta1 - teta0 + 1)

yval = np.linspace(0, 0.5, workNum)

x, y = np.meshgrid(xval, yval)

z = mas

axes.plot\_surface(x, y, z, rstride=5, cstride=5, cmap=cm.jet, label=name)

axes.set\_zlabel(name)

axes.scatter(x, y, z, cmap=cm.jet)

axes.view\_init(rotate[0], rotate[1])

plt.savefig(name + '.jpg')

plt.show()

def drawplot2(mas, rotate, name):

mas = np.array(mas)

mas = mas.transpose()

fig = plt.figure()

axes = Axes3D(fig)

xval = np.linspace(teta0, teta1, 2)

yval = np.linspace(0, 0.5, workNum)

x, y = np.meshgrid(xval, yval)

z = mas

axes.plot\_surface(x, y, z, rstride=5, cstride=5, cmap=cm.jet, label=name)

axes.set\_zlabel(name)

axes.scatter(x, y, z, cmap=cm.jet)

axes.view\_init(rotate[0], rotate[1])

plt.savefig(name + '.jpg')

plt.show()

def drawplot2d\_h1(mas, name, isWrongs):

mas = np.array(mas)

print(mas)

yval = np.linspace(0, 0.5, workNum)

index = 1 if isWrongs else 0

z = mas # [mas[0][i] for i in range(len(mas[0]))]

plt.plot(yval, z)

plt.xlabel(name)

plt.savefig(name + '.jpg')

plt.show()

def drawplot2d\_h2(mas, name):

mas = np.array(mas)

print(mas)

yval = np.linspace(0, 0.5, workNum)

z = mas # [mas[1][i] for i in range(len(mas[0]))]

plt.plot(yval, z)

plt.xlabel(name)

plt.savefig(name + '.jpg')

plt.show()

def drawplot2d\_h2s(mas0, mas1, name):

mas0 = np.array(mas0)

yval = np.linspace(0, 0.5, workNum)

z0 = mas0 # [mas[1][i] for i in range(len(mas[0]))]

z1 = mas1 # [mas[1][i] for i in range(len(mas[0]))]

plt.plot(yval, z0, label='H0=12, H1=21')

plt.plot(yval, z1, label='H0=48, H1=57')

plt.legend(loc='best')

plt.xlabel(name)

plt.savefig(name + '.jpg')

plt.show()

def drawplot2d\_h2ss(mas0, mas1, mas2, name):

mas0 = np.array(mas0)

yval = np.linspace(0, 0.5, workNum)

z0 = mas0 # [mas[1][i] for i in range(len(mas[0]))]

z1 = mas1 # [mas[1][i] for i in range(len(mas[0]))]

z2 = mas2 # [mas[1][i] for i in range(len(mas[0]))]

plt.plot(yval, z0, label='H0=12, H1=21')

plt.plot(yval, z1, label='H0=48, H1=57')

plt.plot(yval, z2, label='H0=48, H1=84')

plt.legend(loc='best')

plt.xlabel(name)

plt.savefig(name + '.jpg')

plt.show()

def test(var, k=10):

wrongs = [[]] \* 2 # (teta1 - teta0 + 1)

observations = [[]] \* 2 # (teta1 - teta0 + 1)

cumWrongs = [[]] \* 2 # (teta1 - teta0 + 1)

cumObservations = [[]] \* 2 # (teta1 - teta0 + 1)

index = 0

for tetaR in [teta0, teta1]:

cumWrong = 0.

cumObs = 0.

wrongs[index] = [0.] \* workNum

observations[index] = [0.] \* workNum

cumObservations[index] = [0.] \* workNum

cumWrongs[index] = [0.] \* workNum

for eps in range(0, workNum):

wrong = 0.

obs = 0.

for i in range(0, n + 1):

result = testNoisyBin(index, tetaR, float(eps / steps), k)

obs += result[1]

if not result[0]:

wrong += 1

print("eps = " + str(eps / steps) + "\twrongs = " + str(wrong) + "\tobservations = " + str(obs))

wrongs[index][eps] = wrong

observations[index][eps] = obs

cumObs += obs

cumWrong += wrong

cumObservations[index][eps] = cumObs

cumWrongs[index][eps] = cumWrong

# print("При eps = " + str(eps / steps) + "\t wrong = " + str(wrong))

# +"\tres[0] = "+str(result[0])+"\tobs = "+str(result[1]))

minW = min(wrongs[index])

maxW = max(wrongs[index])

avgW = stats.mean(wrongs[index])

minO = min(observations[index])

maxO = max(observations[index])

avgO = stats.mean(observations[index])

index += 1

# for t in range(10):

# print(wrongs[tetaR - teta0][t\*5:(t+1)\*5])

# print("eps = "+str(eps / steps)+"\tmin wrong = "+str(minW))

# write\_res("task #" + str(var), tetaR, minO, minW, maxO, maxW, avgO, avgW)

print("\nПри teta = " + str(tetaR))

print("min obs = " + str(minO) + "\nmin wrong = " + str(minW))

print("max obs = " + str(maxO) + "\nmax wrong = " + str(maxW))

print("\navg obs = " + str(avgO) + "\navg wrong = " + str(avgW))

return observations, wrongs, cumObservations, cumWrongs

def test4(var, sub, k=10):

tetaR = teta0 if var == 0 else teta1

if sub == 1:

tetaR = teta0s if var == 0 else teta1s

if sub == 2:

tetaR = teta0ss if var == 0 else teta1ss

# result = [False, 0.]

cumWrong = 0.

cumObs = 0.

wrongs = [0.] \* workNum

observations = [0.] \* workNum

cumObservations = [0.] \* workNum

cumWrongs = [0.] \* workNum

# 50000 запуск теста, к ним считается число ошибок

for eps in range(0, workNum):

# f.write("\nfor "+str(eps))

# result = [False, 0.]

wrong = 0.

obs = 0.

for i in range(0, n + 1):

result = testNoisyBin(var, tetaR, float(eps / steps), sub, k)

obs += result[1]

if not result[0]:

wrong += 1

print("eps = " + str(eps / steps) + "\twrongs = " + str(wrong) + "\tobservations = " + str(obs))

f.write("\neps = " + str(eps / steps) + "\twrongs = " + str(wrong) + "\tobservations = " + str(obs))

wrongs[eps] = wrong

observations[eps] = obs

cumObs += obs

cumWrong += wrong

cumObservations[eps] = cumObs

cumWrongs[eps] = cumWrong

minW = min(wrongs)

maxW = max(wrongs)

avgW = stats.mean(wrongs)

minO = min(observations)

maxO = max(observations)

avgO = stats.mean(observations)

print("\nПри teta = " + str(tetaR))

print("min obs = " + str(minO) + "\nmin wrong = " + str(minW))

print("max obs = " + str(maxO) + "\nmax wrong = " + str(maxW))

print("\navg obs = " + str(avgO) + "\navg wrong = " + str(avgW))

f.write("\n\nПри teta = " + str(tetaR))

f.write("\nmin obs = " + str(minO) + "\nmin wrong = " + str(minW))

f.write("\nmax obs = " + str(maxO) + "\nmax wrong = " + str(maxW))

f.write("\navg obs = " + str(avgO) + "\navg wrong = " + str(avgW))

return observations, wrongs, cumObservations, cumWrongs

def write\_res(task, tetaR, minO, minW, maxO, maxW, avgO, avgW):

f.write(task)

f.write("\nПри teta = " + str(tetaR))

f.write("\nmin obs = " + str(minO) + "\nmin wrong = " + str(minW))

f.write("\nmax obs = " + str(maxO) + "\nmax wrong = " + str(maxW))

f.write("\navg obs = " + str(avgO) + "\navg wrong = " + str(avgW))

f.write("\n\n")

r1\_ = 30

r2\_ = 170

def f1():

print("\n\*\*\*\*Верна первая гипотеза\*\*\*\*\n")

res = test(0)

drawplot(res[0], [45, 80], 'наблюдения H0')

drawplot(res[0], [r1\_, 170], 'наблюдения H0\_')

drawplot(res[1], [25, -15], 'ошибки первого рода')

drawplot(res[1], [r1\_, 195], 'ошибки первого рода\_')

drawplot(res[2], [25, -15], 'накопленные наблюдения H0')

drawplot(res[3], [25, 170], 'накопленные ошибки первого рода')

def f4\_1():

print("\n\*\*\*\*Верна первая гипотеза\*\*\*\*\n")

res = test4(0, 0)

drawplot2d\_h1(res[0], 'наблюдения при верной первой гипотезе', False)

drawplot2d\_h1(res[1], 'ошибки первого рода', True)

def f4\_2():

print("\n\*\*\*\*Верна вторая гипотеза\*\*\*\*\n")

res = test4(1, 0)

res1 = test4(1, 1)

res2 = test4(1, 2)

drawplot2d\_h2ss(res[0], res1[0], res2[0], 'наблюдения при верной второй гипотезе')

drawplot2d\_h2ss(res[1], res1[1], res2[1], 'ошибки второго рода')

def f2():

print("\n\*\*\*\*Верна вторая гипотеза\*\*\*\*\n")

res = test(1)

drawplot(res[0], [25, -5], 'наблюдения H1')

drawplot(res[0], [r1\_, 195], 'наблюдения H1\_')

drawplot(res[1], [25, -25], 'ошибки второго рода')

drawplot(res[1], [r1\_, r2\_], 'ошибки второго рода\_')

drawplot(res[2], [25, -15], 'накопленные наблюдения Н1')

drawplot(res[3], [25, -25], 'накопленные ошибки второго рода')

def f3():

print("\n\*\*\*\*Верна первая гипотеза\*\*\*\*\n")

res = test(2)

drawplot(res[1], [35, 5], 't2 ошибки первого рода')

drawplot(res[1], [r1\_, 170], 't2 ошибки первого рода\_')

drawplot(res[0], [35, 5], 't2 наблюдения H0')

drawplot(res[0], [r1\_, 170], 't2 наблюдения H0\_')

drawplot(res[2], [25, -15], 't2 накопленные наблюдения H0')

drawplot(res[3], [25, -15], 't2 накопленные ошибки первого рода')

def f4():

print("\n\*\*\*\*Верна вторая гипотеза\*\*\*\*\n")

res = test(3)

drawplot(res[1], [35, 5], 't2 ошибки второго рода')

drawplot(res[1], [r1\_, 170], 't2 ошибки второго рода\_')

drawplot(res[0], [35, 5], 't2 наблюдения H1')

drawplot(res[0], [r1\_, 170], 't2 наблюдения H1\_')

drawplot(res[2], [25, -45], 't2 накопленные наблюдения H1')

drawplot(res[3], [25, -15], 't2 накопленные ошибки второго рода')

def second():

print("\n\*\*\*\*\*\*\*\*\*\*\*\*ПУНКТ 1\*\*\*\*\*\*\*\*\*\*\*\*\n")

f1()

f2()

print("\n\*\*\*\*\*\*\*\*\*\*\*\*ПУНКТ 2\*\*\*\*\*\*\*\*\*\*\*\*\n")

f3()

f4()

def third():

print("\"Загрязнение\" биномиальным распределением:")

f4\_1()

f4\_2()

# f2()

def fun2(scenario, tetaR, eps):

global A

global B

global logA

global logB

global teta0

global teta1

tetaW = teta0 if scenario % 2 == 0 else teta1

accept = False # True if scenario % 2 == 0 else

number = 0

sumZ = 0

values = []

Zs = []

while True:

t = 0.

picker = np.random.uniform(0, 1)

if picker <= eps:

if scenario < 2:

t = np.random.poisson(tetaR)

else:

t = tetaR

else:

t = np.random.poisson(tetaW)

values.append(t)

Zs.append(calcZ(t, teta0, teta1))

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

accept = True # отколняем H0

break

if sumZ <= logB:

accept = False # принимаем H0

break

if logB < sumZ < logA:

number += 1

elif scenario % 2 == 1:

if sumZ >= logA:

accept = False # принимаем H1

break

if sumZ <= logB:

accept = True # отклоняем H1

break

if logB < sumZ < logA:

number += 1

return [accept, number]

def testNoisyBin(scenario, tetaR, eps, sub, k=10):

global A

global B

global logA

global logB

global teta0

global teta1

if sub == 1:

teta0 = teta0s

teta1 = teta1s

if sub == 2:

teta0 = teta0ss

teta1 = teta1ss

accept = False # True if scenario % 2 == 0 else

thetaR = teta0 if scenario % 2 == 0 else teta1

number = 0

sumZ = 0

values = []

Zs = []

N = n / k

P = thetaR \* k / n

while True:

xt = 0

isBin = False

picker = np.random.uniform(0, 1)

if picker >= eps:

xt = np.random.poisson(thetaR)

else:

# Binomial

# xt = int(np.random.binomial(N, P, 1)[0])

# while xt == 0:

# xt = int(np.random.binomial(N, P, 1)[0])

# Geometrical

xt = int(np.random.geometric(1 / thetaR))

while xt == 0 or xt > 100:

xt = int(np.random.geometric(1 / thetaR))

isBin = True

values.append(xt)

Zs.append(calcZ(xt, teta0, teta1))

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

accept = False # отклоняем H0

break

if sumZ <= logB:

accept = True # принимаем H0

break

if logB < sumZ < logA:

number += 1

elif scenario % 2 == 1:

if sumZ >= logA:

accept = True # принимаем H1

break

if sumZ <= logB:

accept = False # отклоняем H1

break

if logB < sumZ < logA:

number += 1

return [accept, number]

print("Входные:")

print("H0 = " + str(teta0))

print("H1 = " + str(teta1))

print("a = " + str(a))

print("b = " + str(b))

f = open("results.txt", 'w')

third()

f.close()

ПРИЛОЖЕНИЕ Б

Код программы для главы 5, выполнено в среде jupyter (.ipynb):

#%%

# init part

import math

from math import factorial as f

import matplotlib.pyplot as plt

from matplotlib import cm

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import statistics as stats

from scipy.stats import multinomial

import pandas as pd

def calcA(a, b):

return (1. - b) / a

def calcB(a, b):

return b / (1. - a)

def calcZ(n, X, theta0, theta1):

return math.log(pmf(n, X, theta1) / pmf(n, X, theta0))

def pmf(n, X, P):

xFactM = 1

for x in X:

xFactM \*= f(x)

pDegsM = 1

for i in range(len(X)):

pDegsM \*= P[i] \*\* X[i]

return f(n) \* pDegsM / xFactM

def test1(scenario, theta0, theta1, a, b, N, isPrint):

A = calcA(a, b)

B = calcB(a, b)

logB = math.log(B)

logA = math.log(A)

wrongs = 0

observations = 0

global iterations

# 1000 запусков теста, к ним считается число ошибок

for i in range(iterations):

result = simpleTest(scenario, theta0, theta1, logA, logB, N)

observations += result[1]

if not result[0]:

wrongs += 1

if isPrint:

print("simple test result for " + str(iterations) + " iterations")

print("scenario: " + str(scenario))

print("obeservations = " + str(observations))

print("wrongs = " + str(wrongs))

return observations, wrongs

def test1b(scenario, theta0, theta1, a, b, N, isPrint):

A = calcA(a, b)

B = calcB(a, b)

logB = math.log(B)

logA = math.log(A)

wrongs = 0

observations = 0

global iterations

# 500 запуск теста, к ним считается число ошибок

for i in range(iterations):

result = simpleTestWithCounter(scenario, theta0, theta1, logA, logB, N)

observations += result[1]

if not result[0]:

wrongs += 1

if isPrint:

print("simple test result for " + str(iterations) + " iterations")

print("scenario: " + str(scenario))

print("obeservations = " + str(observations))

print("wrongs = " + str(wrongs))

return observations, wrongs

def test2(scenario, theta0, theta1, thetaNoise, a, b, N):

w = 3

def simpleTest(scenario, theta0, theta1, logA, logB, N):

accept = False # True if scenario % 2 == 0 else

thetaR = theta0 if scenario % 2 == 0 else theta1

number = 0

sumZ = 0

values = []

Zs = []

counter = 0

limit = 1

while True:

X = np.random.multinomial(N, thetaR, size=1)[0]

values.append(X)

Zs.append(calcZ(N, X, theta0, theta1))

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

accept = False # отклоняем H0

break

if sumZ <= logB:

accept = True # принимаем H0

break

if logB < sumZ < logA:

number += 1

elif scenario % 2 == 1:

if sumZ >= logA:

accept = True # принимаем H1

break

if sumZ <= logB:

accept = False # отклоняем H1

break

if logB < sumZ < logA:

number += 1

return [accept, number]

def testNoisy1(scenario, theta0, theta1, logA, logB, N, eps, thetaN):

thetaR = theta0 if scenario % 2 == 0 else theta1

X = np.random.multinomial(N, thetaR, size=1)[0]

number = 0

sumZ = 0

values = []

Zs = []

while True:

picker = np.random.uniform(0, 1)

if picker < eps:

X = np.random.multinomial(N, thetaN, size=1)[0]

else:

X = np.random.multinomial(N, thetaR, size=1)[0]

values.append(X)

Zs.append(calcZ(N, X, theta0, theta1))

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

accept = False # отклоняем H0

break

if sumZ <= logB:

accept = True # принимаем H0

break

if logB < sumZ < logA:

number += 1

elif scenario % 2 == 1:

if sumZ >= logA:

accept = True # принимаем H1

break

if sumZ <= logB:

accept = False # отклоняем H1

break

if logB < sumZ < logA:

number += 1

return [accept, number]

def simpleTestWithCounter(scenario, theta0, theta1, logA, logB, N):

accept = False # True if scenario % 2 == 0 else

thetaR = theta0 if scenario % 2 == 0 else theta1

X = np.random.multinomial(N, thetaR, size=1)[0]

number = 0

sumZ = 0

values = []

Zs = []

counter = 0

limit = 10

while True:

values.append(X)

Zs.append(calcZ(N, X, theta0, theta1))

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

accept = False # отклоняем H0

if counter >= limit:

break

counter += 1

if sumZ <= logB:

accept = True # принимаем H0

if counter >= limit:

break

counter += 1

if logB < sumZ < logA:

number += 1

elif scenario % 2 == 1:

if sumZ >= logA:

accept = True # принимаем H1

if counter >= limit:

break

counter += 1

if sumZ <= logB:

accept = False # отклоняем H1

if counter >= limit:

break

counter += 1

if logB < sumZ < logA:

number += 1

return [accept, number]

def avg(mas):

return sum(mas) / len(mas)

def drawplot2d\_h1(mas, name, isWrongs):

mas = np.array(mas)

print(mas)

# f.write(\*mas)

# f.write("\n")

yval = np.linspace(1, runs, runs)

index = 1 if isWrongs else 0

z = mas # [mas[0][i] for i in range(len(mas[0]))]

# print()

plt.plot(yval, z)

plt.xlabel(name)

plt.savefig(name + '.jpg')

plt.show()

def drawplot2d\_2lines(mas1, mas2, line1, line2, plotname, isWrongs):

mas1 = np.array(mas1)

mas2 = np.array(mas2)

# f.write(\*mas)

# f.write("\n")

yval = np.linspace(1, runs, runs)

index = 1 if isWrongs else 0

z1 = mas1 # [mas[0][i] for i in range(len(mas[0]))]

# print()

plt.plot(yval, mas1, label=line1)

plt.plot(yval, mas2, label=line2)

plt.legend(loc='best')

plt.xlabel(plotname)

plt.savefig(plotname + '.jpg')

plt.show()

def drawplot2d\_2lines\_c3(mas1, mas2, line1, line2, plotname, isWrongs):

mas1 = np.array(mas1)

mas2 = np.array(mas2)

# f.write(\*mas)

# f.write("\n")

yval = np.linspace(N, N + runs, runs)

index = 1 if isWrongs else 0

z1 = mas1 # [mas[0][i] for i in range(len(mas[0]))]

# print()

plt.plot(yval, mas1, label=line1)

plt.plot(yval, mas2, label=line2)

plt.legend(loc='best')

plt.xlabel(plotname)

plt.savefig(plotname + '.jpg')

plt.show()

def visualise\_pt1(result, case):

drawplot2d\_2lines(result[0][0], result[1][0], "верна H0", "верна H1", case + str(" наблюдения"), False)

drawplot2d\_2lines(result[0][1], result[1][1], "первого рода", "второго рода", case + str(" ошибки"), True)

def visualise\_pt1\_c3(result, case):

drawplot2d\_2lines(result[0][0], result[1][0], "верна H0", "верна H1", case + str(" наблюдения"), False)

drawplot2d\_2lines(result[0][1], result[1][1], "первого рода", "второго рода", case + str(" ошибки"), True)

def smallPrint(header, obses, wrongs, wrongsProb):

print(header)

print("Наблюдения|Ошибки|Вероятности ошибки")

print(avg(obses), end='\t|')

print(avg(wrongs), end='\t|')

print(avg(wrongsProb))

def smallTable(result, name):

mas1 = result[0]

mas2 = result[1]

tabledata = [["1"] + mas1, ["2"] + mas2]

table = pd.DataFrame(tabledata, columns=["Верна гипотеза", "Наблюдения", "Ошибки", "Вероятность ошибки"]).style.hide\_index()

table.set\_caption(name)

return table

def simpleTestCase1(theta0, theta1, at, bt, runs, iterations):

print("Входные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

print("a = " + str(at), end=', ')

print("b = " + str(bt))

wrongs0 = []

wrongsProb0 = []

obses0 = []

wrongs1 = []

wrongsProb1 = []

obses1 = []

print("pt1")

for i in range(runs):

res = test1(0, theta0, theta1, at, bt, N, False)

obses0.append(res[0])

wrongs0.append(res[1])

wrongsProb0.append(res[1] / iterations)

print("pt2")

for i in range(runs):

res = test1(1, theta0, theta1, at, bt, N, False)

obses1.append(res[0])

wrongs1.append(res[1])

wrongsProb1.append(res[1] / iterations)

# res0 = [avg(obses0), avg(wrongs0), avg(wrongsProb0)]

# res1 = [avg(obses1), avg(wrongs1), avg(wrongsProb1)]

# tabledata = [["1"] + res0, ["2"] + res1]

# pd.DataFrame(tabledata, columns=["Верна гипотеза", "Наблюдения", "Ошибки", "Вероятность ошибки"]).style.hide\_index()

# print(smallTable(res0, res1, "Среднее значение параметров за "+str(runs)+" запусков"))

smallPrint("Верная первая гипотеза", obses0, wrongs0, wrongsProb0)

smallPrint("Верная вторая гипотеза", obses1, wrongs1, wrongsProb1)

return [obses0, wrongs0, wrongsProb0], [obses1, wrongs1, wrongsProb1]

steps = 100

workNum = steps // 2

segments = 11

def testCase2(theta0, theta1, at, bt):

print("\nТест 2 с помехами\nВходные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

global logAg

global logBg

wrongs = [[]] \* segments # (teta1 - teta0 + 1)

observations = [[]] \* segments # (teta1 - teta0 + 1)

cumWrongs = [[]] \* segments # (teta1 - teta0 + 1)

cumObservations = [[]] \* segments # (teta1 - teta0 + 1)

index = 0

thetaNoise = theta0

delta = [(theta1[i] - theta0[i]) / segments for i in range(len(theta0))]

print("test 2 pt 1")

while thetaNoise[0] > theta1[0]:

cumWrong = 0.

cumObs = 0.

wrongs[index] = [0.] \* workNum

observations[index] = [0.] \* workNum

cumObservations[index] = [0.] \* workNum

cumWrongs[index] = [0.] \* workNum

for eps in range(0, workNum):

wrong = 0.

obs = 0.

for i in range(iterations):

res = testNoisy1(0, theta0, theta1, logAg, logBg, N, float(eps / workNum), thetaNoise)

obs += res[1]

if not res[0]:

wrong += 1

wrongs[index][eps] = wrong

observations[index][eps] = obs

cumObs += obs

cumWrong += wrong

cumObservations[index][eps] = cumObs

cumWrongs[index][eps] = cumWrong

minW = min(wrongs[index])

maxW = max(wrongs[index])

avgW = stats.mean(wrongs[index])

minO = min(observations[index])

maxO = max(observations[index])

avgO = stats.mean(observations[index])

print("\nПри teta = " + str(thetaNoise) + "\nindex = " + str(index))

print("min obs = " + str(minO) + "\nmin wrong = " + str(minW))

print("max obs = " + str(maxO) + "\nmax wrong = " + str(maxW))

print("\navg obs = " + str(avgO) + "\navg wrong = " + str(avgW))

print("\navg chance wrong = " + str(avgW / iterations))

index += 1

for i in range(len(thetaNoise)):

thetaNoise[i] += delta[i]

return observations, wrongs, cumObservations, cumWrongs

def drawplot(mas, rotate, name):

mas = np.array(mas)

mas = mas.transpose()

fig = plt.figure()

axes = Axes3D(fig, auto\_add\_to\_figure=False)

fig.add\_axes(axes)

xval = np.linspace(1, segments, segments)

yval = np.linspace(0, 0.5, workNum)

x, y = np.meshgrid(xval, yval)

z = mas

axes.plot\_surface(x, y, z, rstride=5, cstride=5, cmap=cm.jet, label=name)

axes.set\_zlabel(name)

axes.scatter(x, y, z, cmap=cm.jet)

axes.view\_init(rotate[0], rotate[1])

plt.savefig(name + '.jpg')

plt.show()

def drawplot\_v1(mas, rotate, name, segments\_v1, workNum\_v1):

mas = np.array(mas)

mas = mas.transpose()

fig = plt.figure()

axes = Axes3D(fig, auto\_add\_to\_figure=False)

fig.add\_axes(axes)

xval = np.linspace(1, segments\_v1, segments\_v1)

yval = np.linspace(0, 0.5, workNum\_v1)

x, y = np.meshgrid(xval, yval)

z = mas

axes.plot\_surface(x, y, z, rstride=5, cstride=5, cmap=cm.jet, label=name)

axes.set\_zlabel(name)

axes.scatter(x, y, z, cmap=cm.jet)

axes.view\_init(rotate[0], rotate[1])

plt.savefig(name + '.jpg')

plt.show()

def drawplot\_v2(mas, rotate, name, coeffs, workNum\_v1):

mas = np.array(mas)

mas = mas.transpose()

fig = plt.figure()

axes = Axes3D(fig, auto\_add\_to\_figure=False)

fig.add\_axes(axes)

xval = np.array(coeffs)

yval = np.linspace(0, 0.5, workNum\_v1)

x, y = np.meshgrid(xval, yval)

z = mas

axes.plot\_surface(x, y, z, rstride=5, cstride=5, cmap=cm.jet, label=name)

axes.set\_zlabel(name)

axes.scatter(x, y, z, cmap=cm.jet)

axes.view\_init(rotate[0], rotate[1])

plt.savefig(name + '.jpg')

plt.show()

def simpleTestCase1b(theta0, theta1, at, bt, runs, iterations):

print("Входные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

wrongs0 = []

wrongsProb0 = []

obses0 = []

wrongs1 = []

wrongsProb1 = []

obses1 = []

print("pt1")

for i in range(runs):

res = test1b(0, theta0, theta1, at, bt, N, False)

obses0.append(res[0])

wrongs0.append(res[1])

wrongsProb0.append(res[1] / iterations)

print("pt2")

for i in range(runs):

res = test1b(1, theta0, theta1, at, bt, N, False)

obses1.append(res[0])

wrongs1.append(res[1])

wrongsProb1.append(res[1] / iterations)

drawplot2d\_2lines(obses0, obses1, "наблюдения H0", "наблюдения H1", "наблюдения при N in 100 - 150", False)

drawplot2d\_2lines(wrongs0, wrongs1, "ошибки 1 рода", "ошибки 2 рода", "ошибки при N in 100-150", True)

# smallPrint("Верная первая гипотеза", obses0, wrongs0, wrongsProb0)

# smallPrint("Верная вторая гипотеза", obses1, wrongs1, wrongsProb1)

return [obses0, wrongs0, wrongsProb0], [obses1, wrongs1, wrongsProb1]

def simpleTestCase2(theta0, theta1, at, bt, runs, iterations):

print("Входные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

wrongs0 = []

wrongsProb0 = []

obses0 = []

wrongs1 = []

wrongsProb1 = []

obses1 = []

print("pt1")

for i in range(runs):

res = test1(0, theta0, theta1, at, bt, N + i, False)

obses0.append(res[0])

wrongs0.append(res[1])

wrongsProb0.append(res[1] / iterations)

print("pt2")

for i in range(runs):

res = test1(1, theta0, theta1, at, bt, N + i, False)

obses1.append(res[0])

wrongs1.append(res[1])

wrongsProb1.append(res[1] / iterations)

print("obs 0: " + str(obses0))

print("wrongs 0: " + str(wrongs0))

print("obs 1: " + str(obses1))

print("wrongs 1: " + str(wrongs1))

drawplot2d\_2lines\_c3(obses0, obses1, "наблюдения H0", "наблюдения H1", "наблюдения при N in "+str(N) + " - "+str(N+runs), False)

drawplot2d\_2lines\_c3(wrongs0, wrongs1, "ошибки 1 рода", "ошибки 2 рода", "ошибки при N in n "+str(N) + " - "+str(N+runs), True)

smallPrint("Верная первая гипотеза", obses0, wrongs0, wrongsProb0)

smallPrint("Верная вторая гипотеза", obses1, wrongs1, wrongsProb1)

return [obses0, wrongs0], [obses1, wrongs1]

N = 50

runs = 130

iterations = 1000

theta0 = [1 / 6.] \* 6

theta1 = [1 / 7.] \* 5 + [2 / 7.]

theta1\_a = [1 / 7.] \* 5 + [2 / 7.]

theta1\_b = [1 / 8.] \* 5 + [3 / 8.]

theta1\_c = [2 / 8.] + [1 / 8.] \* 4 + [2 / 8.]

theta1\_d = [1 / 8.] \* 2 + [2 / 8.] + [1 / 8.] \* 2 + [2 / 8.]

a = 0.05

b = 0.05

Ag = calcA(a, b)

Bg = calcB(a, b)

logBg = math.log(Bg)

logAg = math.log(Ag)

print("norm0")

print(sum(theta0))

print("norm1a = " + str(sum(theta1\_a)))

print("norm1b = " + str(sum(theta1\_b)))

print("norm1c = " + str(sum(theta1\_c)))

print("norm1d = " + str(sum(theta1\_d)))

print("a = " + str(a))

print("b = " + str(b))

print("запусков: " + str(runs))

print("итераций теста: " + str(iterations))

def part1():

print("\ncase A\n")

resA1 = simpleTestCase1(theta0, theta1\_a, a, b, runs, iterations)

print("\ncase B\n")

resB1 = simpleTestCase1(theta0, theta1\_b, a, b, runs, iterations)

print("\ncase C\n")

resC1 = simpleTestCase1(theta0, theta1\_c, a, b, runs, iterations)

print("\ncase D\n")

resD1 = simpleTestCase1(theta0, theta1\_d, a, b, runs, iterations)

visualise\_pt1(resA1, "case A")

visualise\_pt1(resB1, "case B")

visualise\_pt1(resC1, "case C")

visualise\_pt1(resD1, "case D")

return [resA1, resB1, resC1, resD1]

def part2():

print("\ncase A1")

resA1 = simpleTestCase1(theta0, theta1\_a, a, b, runs, iterations)

print("\ncase A2")

resA2 = simpleTestCase1(theta0, theta1\_a, a, 2\*b, runs, iterations)

print("\ncase A3")

resA3 = simpleTestCase1(theta0, theta1\_a, 2\*a, b, runs, iterations)

print("\ncase A4")

resA4 = simpleTestCase1(theta0, theta1\_a, a, b/5, runs, iterations)

print("\ncase A5")

resA5 = simpleTestCase1(theta0, theta1\_a, a / 5, b, runs, iterations)

print("\ncase A6")

resA6 = simpleTestCase1(theta0, theta1\_a, a \* 2, b / 5, runs, iterations)

return [resA1,resA2,resA3,resA4,resA5, resA6]

def part3():

print("\ncase A pt 3\n")

resA = simpleTestCase2(theta0, theta1\_a, a, b, runs, iterations)

r1\_ = 30

r2\_ = 170

def part4():

print("pt 4 Noisy Test")

res = testCase2(theta0, theta1, a, b)

print(res[0])

return res

#%%

def testNoisy1\_v1(scenario, theta0, theta1, N, eps, thetaN, coeff):

global logAg

global logBg

logA = logAg

logB = logBg

thetaR = theta0 if scenario % 2 == 0 else theta1

X = np.random.multinomial(N, thetaR, size=1)[0]

number = 0

sumZ = 0

importance = 1.

values = []

Zs = []

while True:

picker = np.random.uniform(0, 1)

X = np.random.multinomial(N, thetaR, size=1)[0]

if picker < eps:

X = np.random.multinomial(N, thetaN, size=1)[0]

else:

X = np.random.multinomial(N, thetaR, size=1)[0]

values.append(X)

newZ = calcZ(N, X, theta0, theta1) \* importance

Zs.append(newZ)

sumZ += Zs[-1]

if scenario % 2 == 0:

if sumZ >= logA:

sumZ -= Zs[-1]

accept = False # отклоняем H0

break

if sumZ <= logB:

accept = True # принимаем H0

break

if logB < sumZ < logA:

number += 1

# importance \*= coeff ускоряющий коэфф

elif scenario % 2 == 1:

if sumZ >= logA:

accept = True # принимаем H1

break

if sumZ <= logB:

accept = False # отклоняем H1

break

if logB < sumZ < logA:

number += 1

# importance \*= coeff ускоряющий коэфф

return [accept, number]

def testCase2\_v1(theta0, theta1, at, bt):

print("\nТест 2 с помехами\nВходные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

global logAg

global logBg

segments\_v1 = 10

iterations\_v1 = 1000

workNum\_v1 = 50

wrongs = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

observations = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

cumWrongs = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

cumObservations = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

index = 0

coeff = 1.5

thetaNoise = [t for t in theta0]

delta = [(theta1[i] - theta0[i]) / segments\_v1 for i in range(len(theta0))]

print("test 2 pt 2")

while thetaNoise[0] > theta1[0] and index < segments\_v1:

cumWrong = 0.

cumObs = 0.

wrongs[index] = [0.] \* workNum\_v1

observations[index] = [0.] \* workNum\_v1

cumObservations[index] = [0.] \* workNum\_v1

cumWrongs[index] = [0.] \* workNum\_v1

for eps in range(0, workNum\_v1):

wrong = 0.

obs = 0.

for i in range(iterations\_v1):

res = testNoisy1\_v1(1, theta0, theta1, N, float(eps / workNum\_v1), thetaNoise, coeff)

obs += res[1]

if not res[0]:

wrong += 1

wrongs[index][eps] = wrong

observations[index][eps] = obs

cumObs += obs

cumWrong += wrong

cumObservations[index][eps] = cumObs

cumWrongs[index][eps] = cumWrong

minW = min(wrongs[index])

maxW = max(wrongs[index])

avgW = stats.mean(wrongs[index])

minO = min(observations[index])

maxO = max(observations[index])

avgO = stats.mean(observations[index])

print("\nПри teta = ", end='[')

for theta in thetaNoise:

print(f'{theta:.4f}', end=', ')

print("]\nindex = " + str(index))

print("min obs = " + str(minO) + ", min wrong = " + str(minW))

print("max obs = " + str(maxO) + ", max wrong = " + str(maxW))

print("avg obs = " + str(avgO) + ", avg wrong = " + str(avgW))

print("\navg chance wrong = " + str(avgW / iterations\_v1))

index += 1

for i in range(len(thetaNoise)):

thetaNoise[i] += delta[i]

return observations, wrongs, cumObservations, cumWrongs, segments\_v1, workNum\_v1

def testCase2\_v2(theta0, theta1, at, bt):

print("\nТест 2 с помехами\nВходные:")

print("H0 = " + str(theta0))

print("H1 = " + str(theta1))

global logAg

global logBg

coeffs = [1.1, 1.25, 1.5, 1.75, 2, 2.5, 3, 5]

segments\_v1 = len(coeffs)

iterations\_v1 = 1000

workNum\_v1 = 50

wrongs = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

observations = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

cumWrongs = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

cumObservations = [[]] \* segments\_v1 # (teta1 - teta0 + 1)

index = 0

thetaNoise = [t for t in theta0]

delta = [(theta1[i] - theta0[i]) / 2 for i in range(len(theta0))]

for i in range(len(thetaNoise)):

thetaNoise[i]+=delta[i]

print("test 2 pt 2")

for index in range(len(coeffs)):

cumWrong = 0.

cumObs = 0.

wrongs[index] = [0.] \* workNum\_v1

observations[index] = [0.] \* workNum\_v1

cumObservations[index] = [0.] \* workNum\_v1

cumWrongs[index] = [0.] \* workNum\_v1

for eps in range(0, workNum\_v1):

wrong = 0.

obs = 0.

for i in range(iterations\_v1):

res = testNoisy1\_v1(1, theta0, theta1, N, float(eps / workNum\_v1), thetaNoise, coeffs[index])

obs += res[1]

if not res[0]:

wrong += 1

wrongs[index][eps] = wrong

observations[index][eps] = obs

cumObs += obs

cumWrong += wrong

cumObservations[index][eps] = cumObs

cumWrongs[index][eps] = cumWrong

minW = min(wrongs[index])

maxW = max(wrongs[index])

avgW = stats.mean(wrongs[index])

minO = min(observations[index])

maxO = max(observations[index])

avgO = stats.mean(observations[index])

print("\nПри teta = ", end='[')

for theta in thetaNoise:

print(f'{theta:.4f}', end=', ')

print("]\ncoeff = " + str(coeffs[index]))

print("min obs = " + str(minO) + ", min wrong = " + str(minW))

print("max obs = " + str(maxO) + ", max wrong = " + str(maxW))

print("avg obs = " + str(avgO) + ", avg wrong = " + str(avgW))

print("\navg chance wrong = " + str(avgW / iterations\_v1))

index += 1

# for i in range(len(thetaNoise)):

# thetaNoise[i] += delta[i]

return observations, wrongs, cumObservations, cumWrongs, segments\_v1, workNum\_v1, coeffs

def part4\_v1():

print("pt 4 Noisy Test")

res = testCase2\_v1(theta0, theta1, a, b)

for r in res:

print(r)

return res

def part5\_v2():

print("pt 4 Noisy Test")

res = testCase2\_v2(theta0, theta1, a, b)

for r in res:

print(r)

return res

#%%

def pt4Draw(res):

drawplot(res[0], [r1\_, -120], 'наблюдения H0')

drawplot(res[0], [r1\_, 140], 'наблюдения H0\_')

drawplot(res[1], [20, 105], 'ошибки первого рода')

drawplot(res[1], [r1\_, 195], 'ошибки первого рода\_')

drawplot(res[2], [25, 170], 'накопленные наблюдения H0')

drawplot(res[3], [25, 170], 'накопленные ошибки первого рода')

def pt4Draw\_v1(resv1):

drawplot\_v1(resv1[0], [r1\_, -120], 'наблюдения H1', resv1[4], resv1[5])

drawplot\_v1(resv1[0], [r1\_, 140], 'наблюдения H1\_', resv1[4], resv1[5])

drawplot\_v1(resv1[1], [20, 15], 'ошибки второго рода', resv1[4], resv1[5])

drawplot\_v1(resv1[1], [r1\_, 145], 'ошибки второго рода\_', resv1[4], resv1[5])

drawplot\_v1(resv1[2], [25, 170], 'накопленные наблюдения H0', resv1[4], resv1[5])

drawplot\_v1(resv1[3], [25, 170], 'накопленные ошибки первого рода', resv1[4], resv1[5])

def pt5Draw\_v2(resv1):

drawplot\_v2(resv1[0], [r1\_, 10], 'наблюдения H1', resv1[6], resv1[5])

drawplot\_v2(resv1[0], [r1\_, 70], 'наблюдения H1\_', resv1[6], resv1[5])

drawplot\_v2(resv1[1], [15, -75], 'ошибки второго рода', resv1[6], resv1[5])

drawplot\_v2(resv1[1], [130, -75], 'ошибки второго рода\_', resv1[6], resv1[5])

drawplot\_v2(resv1[2], [25, 20], 'накопленные наблюдения H1', resv1[6], resv1[5])

drawplot\_v2(resv1[3], [25, 190], 'накопленные ошибки второго рода', resv1[6], resv1[5])

#%%

res\_v1 = part4\_v1()

#%%

pt4Draw\_v1(res\_v1)

#%%

res\_v2 = part5\_v2()

#%%

pt5Draw\_v2(res\_v2)

#%%

def table1(res, name):

mas1 = [avg(res[0][0]),avg(res[0][1]),avg(res[0][2])]

mas2 = [avg(res[1][0]),avg(res[1][1]),avg(res[1][2])]

tabledata = [["1"] + mas1, ["2"] + mas2]

table = pd.DataFrame(tabledata, columns=["Верна гипотеза", "Наблюдения", "Ошибки", "Вероятность ошибки"]).style.hide\_index()

table.set\_caption(name)

return table

def table2(res, name, \_a, \_b):

mas1 = [avg(res[0][0]),avg(res[0][1]),avg(res[0][2])]

mas2 = [avg(res[1][0]),avg(res[1][1]),avg(res[1][2])]

tabledata = [["1"] + mas1, ["2"] + mas2]

table = pd.DataFrame(tabledata, columns=["Верна гипотеза", "Наблюдения", "Ошибки", "Вероятность ошибки"]).style.hide\_index()

table.set\_caption(name+"\na = "+str(\_a)+", b = "+str(\_b))

return table

#%%

pt1 = part1()

#%%

table1(pt1[0], "Средние значения характеристик, набор A")

table1(pt1[1], "Средние значения характеристик, набор B")

table1(pt1[2], "Средние значения характеристик, набор C")

table1(pt1[3], "Средние значения характеристик, набор D")

#%%

pt2 = part2()

#%%

table2(pt2[0], "Средние значения характеристик, набор A1", a, b)

table2(pt2[1], "Средние значения характеристик, набор A2", a, 2\*b)

table2(pt2[2], "Средние значения характеристик, набор A3", 2\*a, b)

table2(pt2[3], "Средние значения характеристик, набор A4", a, b/5)

table2(pt2[4], "Средние значения характеристик, набор A5", a/5, b)

table2(pt2[5], "Средние значения характеристик, набор A6", 2\*a, b/5)

#%%

part3()

#%%

res = part4()

#%%

pt4Draw(res)