Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Механико-математический факультет
Кафедра математического анализа и теории функций

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
канд.физ.-мат.наук, доцент
_____ Л.В. Гензе
«13» ____06_____ 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

О НЕКОТОРЫХ ВОПРОСАХ УТОЧНЕНИЯ ГРАНИЦ В КРИТЕРИИ ВАЛЬДА

по направлению подготовки 01.03.01 "Математика"

профиль "Основы научно-исследовательской деятельности в области математики"

Васьковский Алексей Сергеевич

Руководитель ВКР
доцент, канд. физ.-мат. наук
_____ Т. В. Емельянова
«10» июня 2023 г.

Автор работы
студент группы № 041903
_____ А. С. Васьковский

Томск - 2023

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Механико-математический факультет

УТВЕРЖДАЮ
Руководитель ООП
доцент, декан
_____ Л.В. Гензе
подпись
« 16 » 02 2023 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра обучающейся
Васьковский Алексей Сергеевич
Фамилия Имя Отчество обучающегося
по направлению подготовки 01.03.01 - Математика

1 Тема выпускной квалификационной работы
О некоторых вопросах уточнения границ в критерии Вальда
2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:

а) в учебный офис / деканат - июнь 2023 г. б) в ГЭК - июнь 2023 г.

3 Исходные данные к работе:
Объект исследования – критические границы SPRT
Предмет исследования – последовательный критерий Вальда
Цель исследования - разработка эффективного алгоритма для вычисления критических границ последовательного критерия Вальда.

Задачи:
1. изучить теоретические аспекты вычисления критических границ;
2. исследовать более детально подход, предложенный С. Н. Постоваловым;
3. рассмотреть итерационные способы решения поставленной задачи;
4. разработать более эффективный алгоритм, вычисляющий критические границы с заданной точностью;
5. провести сравнительный анализ полученных алгоритмов.

Методы исследования:
Изучение литературы, моделирование
Организация или отрасль, по тематике которой выполняется работа, -
Томский Государственный университет

4 Краткое содержание работы
В данной работе рассматриваются вопросы эффективного вычисления критических границ последовательного критерия Вальда при помощи компьютерного моделирования. Предложены эффективные и устойчивые итерационные алгоритмы, а также алгоритмы с использованием R-деревьев, которые позволяют находить границы с заданной точностью при меньших временных затратах по сравнению с ранее известными алгоритмами. На основе приведенных алгоритмов реализовано программное обеспечение на языке C++. На конкретных примерах продемонстрирована эффективность предложенных алгоритмов.

Руководитель выпускной квалификационной работы
доцент ММФ ТГУ _____ / Т. В. Емельянова
должность, место работы *подпись* *И.О. Фамилия*

Задание принял к исполнению
студент ММФ ТГУ _____ / А. С. Васьковский
должность, место работы *подпись* *И.О. Фамилия*

АННОТАЦИЯ

Тема выпускной квалификационной работы: «О некоторых вопросах уточнения границ в критерии Вальда»

Автор работы: Васьковский А. С.

Руководитель работы: доцент, кандидат физико-математических наук Емельянова Т. В.

Год защиты: 2023 г.

В данной работе рассматриваются вопросы эффективного вычисления критических границ последовательного критерия Вальда при помощи компьютерного моделирования. Предложены эффективные и устойчивые итерационные алгоритмы, а также алгоритмы с использованием R-деревьев, которые позволяют находить границы с заданной точностью при меньших временных затратах по сравнению с ранее известными алгоритмами. На основе приведенных алгоритмов реализовано программное обеспечение на языке `C++`. На конкретных примерах продемонстрирована эффективность предложенных алгоритмов.

Целью выпускной квалификационной работы является разработка эффективного алгоритма для вычисления критических границ последовательного критерия Вальда.

Изучены теоретические аспекты вычисления критических границ, проанализированы ранее известные алгоритмы, а также приведён пример показывающий недостаток ранее известного итерационного алгоритма. На основании исследования алгоритмов, разработаны более эффективные способы вычисления критических границ, работа которых продемонстрирована на конкретных примерах.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В наше время неотъемлемой частью эксперимента является статистическая обработка результатов наблюдений. Эти результаты представляют собой значения случайных велин X, распределения которых $\mathcal{F}_\theta$, как правило, хотя бы частично нам неизвестно. Следствием этого незнания, есть не возможность в выборе наилучшего поведения. Поэтому одной из основных задач первичной обработки эмпирических наблюдений является определение закона распределения, наиболее хорошо описывающего случайную величину, выборку которой мы наблюдали.

Для решения данной задачи мы выдвигаем ряд гипотез о принадлежности наблюдаемой выборки конкретному теоретическому закону. После чего используя статистические критерии, а именно критерии согласия, выполняем проверку насколько хорошо описывается тем или иным теоретическим законом, наблюдаемая выборка. Необходимость такой проверке обусловлена стремлением удостовериться в том, что предполагаемая модель теоретического закона не противоречит наблюдаемым данным, и использование её в дальнейшем не приведет к значительным ошибкам.

Представленный в 1933 году в работе [19] и популярный по сей день – критерий Неймана-Пирсона. Он относится к классическим методам математической статистики, в которых количество наблюдений, то есть объем выборки, на которых основывается проверка, считается постоянным для каждой конкретной задачи. К настоящему времени, также известно множество других классических критериев согласия [5, 6], предназначенных для проверки как простых, так и сложных гипотез.

Однако в данной работе нас будет интересовать предложенный А. Вальдом в 1945 году подход последовательной проверки гипотез [21]. Его отличительной чертой от классического является то, что количество наблюдений, необходимых для принятия решения, зависит от значений наблюдаемых данных и, следовательно, является не определенным заранее, а случайной величиной. А. Вальд также описал последовательный метод, названый им критерием последовательных отношений вероятностей (SPRT). Позже в работе [22] была доказана оптимальность указанного критерия, как в случае принятия основной гипотезы, так и альтернативы в классе всех последовательных методов.

К преимуществам последовательного подхода можно отнести значительное сокращение количества наблюдений, необходимое для принятия решения с заданными α и β, соответственно ошибками первого и второго рода. Так для SPRT А. Вальд отмечал «It will be seen that these sequential tests usually lead to average savings of about 50% in the number of trials as compared with the current most powerful test» [21]. Но следует быть предельно осторожным обобщая данное высказывание. Так, например, в работе [9] было показано, что при определенных ситуациях последовательный подход не только не экономит в 2 раза объем требуемой выборки, а даже наоборот требует большее количество наблюдений.

Также последовательный подход обладает недостатком, заключающийся в отсутствии

ограничения верхней границы наблюдений, следовательно процедуры последовательной проверки гипотез могут продолжаться неопределённо долго. Теоретически доказано, что одна из конкурирующих гипотез принимается за конечное число шагов [5, с. 115.], но для практики это неприемлемо.

Более того, для того чтобы получить критерий силы (α, β) необходимо определить критические границы – постоянные $A(\alpha, \beta)$ и $B(\alpha, \beta)$. Их точное аналитическое определение обычно чрезвычайно сложно[1], поэтому на практике пользуются их оценками. Но использование оценочных значений, приводит к уменьшению фактических вероятностей ошибок первого и второго рода, следовательно, к увеличению требуемого количества испытаний.

С развитием компьютерного моделирования, стало возможно с заданной точностью вычислить точные критические границы. Так С. Н. Постовалов [7] разработал алгоритм, позволяющий это сделать и на приведенных им примерах получить сокращение объема выборки от 3% до 17%, что в случае проведения дорогостоящих экспериментов является существенным фактором. Этот алгоритм основан на переборе с достаточно малым шагом значений критических границ, построением линий равного уровня и дальнейшим определением их точек пересечения.

Однако по приведенным вычислительным затратам можно сделать вывод о необходимость значительных вычислительных ресурсов. Конечно, как отмечает автор, время работы можно уменьшить при «распараллеливании» процессов вычислений, однако это лишь незначительно позволит сократить время работы.

Таким образом, возникает потребность в создание более эффективного алгоритма для определение с заданной точностью критических границ в SPRT.

Целью данной выпускной квалификационной работы является разработка алгоритма, позволяющего с меньшими вычислительными затратами и более высокой точности находить критических границ последовательного критерия отношения правдоподобия Вальда, что обеспечит при заданных вероятностях ошибок первого и второго рода минимальное количество требуемых экспериментов при проверке простой и сложной гипотез относительно заданной альтернативы.

Для достижения указанной цели в работе необходимо решить следующие задачи:

1) изучить теоретические аспекты вычисления критических границ;

2) исследовать более детально подход предложенный С. Н. Постоваловым;

3) рассмотреть итерационные способы решения поставленной задачи;

4) разработать более эффективный алгоритм, вычисляющий критические границы с заданной точностью;

5) провести сравнительный анализ полученных алгоритмов.

[1] В частном случае известны точные формулы для $A(\alpha, \beta)$, $B(\alpha, \beta)$ и среднего размера выборки. Например, смотреть [15]

1. Постановка задачи

Пусть в результате экспериментов наблюдаются значения случайной величины X: x_1, x_2, $x_3, \ldots$, которые являются независимыми и одинаково распределенными. Поскольку X может иметь как непрерывное распределение, так и дискретное. Поэтому ради краткости иногда будем использовать термин «вероятность» для обозначения плотности вероятности в непрерывном случае, если это не приведет к недоразумением [2]. Соответственно, тогда $f(x, \theta)$ – означает функцию плотности вероятности, если эта функция существует. Если X имеет дискретное распределение, то $f(x, \theta)$ означает вероятность того, что рассматриваемая случайная величина принимает значение x.

Пусть основная гипотеза о виде распределения случайной величины имеет вид:

$$H_0 : \theta = \theta_0, \tag{1.1}$$

и альтернативная гипотеза имеет вид:

$$H_1 : \theta = \theta_1 \tag{1.2}$$

следовательно, распределение X задается выражением $f_0(x) \equiv f(x, \theta_0)$, когда справедлива H_0, и выражением $f_1(x) \equiv f(x, \theta_1)$, когда справедлива H_1.

Учитывая выше сказанное, для любого натурального целого числа m вероятность получения выборки $x_1, \ldots, x_m$ определяется выражением

$$p_{1,m} = f_1(x_1) \cdot \ldots \cdot f_1(x_m), \tag{1.3}$$

когда справедлива гипотеза H_1, и выражением

$$p_{0,m} = f_0(x_1) \cdot \ldots \cdot f_0(x_m), \tag{1.4}$$

когда справедлива гипотеза H_0.

Последовательный критерий отношения вероятностей для проверки гипотезы H_0 против H_1 строится следующим образом. Выбираются две положительные величины $A(\alpha, \beta)$, $B(\alpha, \beta)$ $(A(\alpha, \beta) > B(\alpha, \beta))$, так, чтобы критерий имел наперед заданную силу (α, β). На каждом стадии эксперимента (в m-м испытании для любого натурального значения m) вычисляется отношение $\dfrac{p_{1,m}}{p_{0,m}}$. Тогда, если

$$B(\alpha, \beta) < \frac{p_{1,m}}{p_{0,m}} < A(\alpha, \beta), \tag{1.5}$$

то эксперимент продолжается и производиться дополнительное наблюдение. Если

$$\frac{p_{1,m}}{p_{0,m}} \geq A(\alpha, \beta), \tag{1.6}$$

то процесс оканчивается отклонением гипотезы H_0 (принятием H_1). Если

$$\frac{p_{1,m}}{p_{0,m}} \leq B(\alpha, \beta), \tag{1.7}$$

то процесс оканчивается принятием гипотезы H_0.

Если для некоторой выборки $(x_1, \ldots, x_m)$ справедливо $p_{1,m} = p_{0,m} = 0$, то будем приравнивать величину $\dfrac{p_{1,m}}{p_{0,m}}$ единице. Если для некоторой выборки $(x_1, \ldots, x_m)$ получим $p_{1,m} > 0$, но $p_{0,m} = 0$, то неравенство (1.6) считается выполненным, и H_0 отвергается.

Эксперимент продолжается до тех пор, пока не будет принято решение о виде распределения случайной величины. Объем выборки заранее не известен и определяется в процессе последовательной процедуры.

Вместо работы напрямую с отношением $\dfrac{p_{1,m}}{p_{0,m}}$ можно использовать $\ln\left(\dfrac{p_{1,m}}{p_{0,m}}\right)$, то есть

$$\ln\left(\frac{p_{1,m}}{p_{0,m}}\right) = \ln\left(\frac{f_1(x_1)}{f_0(x_1)}\right) + \cdots + \ln\left(\frac{f_1(x_m)}{f_0(x_m)}\right) \tag{1.8}$$

Обозначим i-й член этой суммы через z_i:

$$z_i = \ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right), \tag{1.9}$$

тогда процедуру испытаний можно записать следующим образом. На каждой стадии эксперимента вычисляется сумма $z_1 + \cdots + z_m$. Если

$$\ln(B(\alpha,\beta)) < z_1 + \cdots + z_m < \ln(A(\alpha,\beta)),$$

то эксперимент продолжается и производится дополнительное наблюдение. Если

$$z_1 + \cdots + z_m \geq \ln(A(\alpha,\beta)),$$

то процесс оканчивается принятием H_1; если

$$z_1 + \cdots + z_m \leq \ln(B(\alpha,\beta)),$$

то процесс оканчивается принятием H_0.

Пусть

$$Z_n = \sum_{i=1}^{n} z_i = \sum_{i=1}^{n} \ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right) \tag{1.10}$$

Известно [5, с. 115], что с вероятностью, равной единице, процесс оканчивается выбором одной из гипотез, то есть на некотором n-ом шаге статистика λ_n выйдет за интервал $\big(\ln(A(\alpha;\beta)); \ln(B(\alpha;\beta))\big)$.

Однако поскольку точное определение величин $A(\alpha,\beta)$ и $B(\alpha,\beta)$ обычно чрезвычайно трудоемко, то в случае проверки простой гипотезы пользуются следующими оценками [2, с. 71]:

$$A(\alpha,\beta) \leq \frac{1-\beta}{\alpha} = a(\alpha,\beta), \quad B(\alpha,\beta) \geq \frac{\beta}{1-\alpha} = b(\alpha,\beta) \tag{1.11}$$

или с использованием $\ln$:

$$\begin{aligned} lA(\alpha,\beta) &= \ln(A(\alpha,\beta)) \leq \ln\left(\frac{1-\beta}{\alpha}\right) = la(\alpha,\beta), \\ lB(\alpha,\beta) &= \ln(B(\alpha,\beta)) \geq \ln\left(\frac{\beta}{1-\alpha}\right) = lb(\alpha,\beta) \end{aligned} \tag{1.12}$$

Таким образом, основная задача при помощи методов компьютерного моделирования с заданной точностью определить $A(\alpha,\beta)$, $B(\alpha,\beta)$ при заданных α,β.

Для удобства, простоты и демонстрации результатов, если не оговорено противное в алгоритмах будет предполагаться, что используются формулы с использованием $\ln$, а также

$$H_0 : f_0(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}; \quad H_1 : f_1(x) = \frac{\pi}{\sqrt{3}} \exp\left(-\frac{\pi x}{\sqrt{3}}\right) / \left(1 + \exp\left(-\frac{\pi x}{\sqrt{3}}\right)\right)^2, \quad (1.13)$$

то есть основная гипотеза о том, что наблюдаемая случайная величина имеет стандартное нормальное распределение, против конкурирующей гипотезы о логистическом распределении с параметрами 0 и $\sqrt{3}/\pi$.

2. Алгоритм предложенный С. Н. Постоваловым

Рассмотрение различных алгоритмов, решающих поставленную задачу начнем с предложенного С. Н. Постоваловым [7], на основе которого будут рассмотрены основные моменты моделирования.

Схема данного алгоритма состоит из трех этапов:

1) *Моделирование функций вероятности ошибок первого и второго рода.* Методом Монте-Карло моделируется зависимость $\alpha\,(lA, lB)$ и $\beta\,(lA, lB)$, путем перебора всех значений lB из интервала $[lb; 0)$, lA из интервала $(0; la]$ с малым шагом δ, где в случае простой гипотезы la, lb соответственно определяются как в (1.12) при выбранных изначально α_0, β_0.

   Для определения вероятности ошибки α по заданным значениям критических границ lA, lB применяется следующим алгоритм:

   - моделируется последовательность реализаций случайной величины по закону, соответствующему гипотезе H_0, до тех пор, пока не произойдет принятие одной из конкурирующих гипотез. Запоминается, какая гипотеза была принята;
   - моделирование повторяется N раз;
   - эмпирическая вероятность ошибки первого рода будет равна отношению числа случаев, когда ошибочно была принята гипотеза H_1 к числу N.

   Пользуясь методом Монте-Карло [3, с. 328], мы, произведя большое число опытов (реализаций), приближенно заменяем вероятность события его частотой, а математическое ожидание – средним арифметическим.

Естественно появляется вопрос – насколько велика будет ошибка, возникающая от такой приближенной замены? И каково должно быть число реализаций N для того, чтобы эта ошибка с практической достоверностью не вышла за данные пределы? Другими словами, возникает вопрос об оценке точности характеристик случайного явления, полученных методом Монте-Карло.

При ответе на эти вопросы мы будем основываться на центральной предельной теореме из теории вероятности [1, с. 179]. Согласно этой теореме, при большом числе опытов N их средний результат (частота P^* события A или среднее арифметическое $\overline{X}$ наблюденных значений случайной величины X) распределяется приближенно по нормальному закону. Приведем относящиеся к нашей работе формулы.

Теорема 1. ***Закон распределения частоты события при большом числе опытов.***

Если производится большое число N независимых опытов, в каждом из которых событие A проявляется с вероятностью p, то частоты события A

$$P^* = \frac{M_A}{N}, \qquad (2.1)$$

где M_A – число появлений события A в N опытах, причем P^* распределяется приближенно по нормальному закону, с математическим ожиданием:

$$\mathbf{E}\left(P^*\right) = p \tag{2.2}$$

и средним квадратическим отклонением:

$$\sigma_{P^*} = \sqrt{\frac{p\left(1-p\right)}{N}} \tag{2.3}$$

Тогда основываясь на этом законе распределения и формулах, мы можем сформулировать следующее утверждение, относящиеся к точности метода Монте-Карло.

Утверждение 1. Пусть производится ряд независимых опытов, в каждом из которых событие A появляется с вероятностью p. Тогда число опытов N (реализаций) для того, чтобы с заданной, достаточно высокой вероятностью Q можно было ожидать, что частота P^* события A отклонится от его вероятности p меньше, чем на ε вычисляется по следующей формуле:

$$N = \left\lceil \frac{p\left(1-p\right)}{\varepsilon^2} \cdot \left(\Phi^{-1}\left(\frac{1}{2}\cdot Q\right)\right)^2 \right\rceil, \tag{2.4}$$

где Φ^{-1} – функция, обратная функции Лапласа:

$$\Phi\left(x\right) = \frac{1}{\sqrt{2\pi}} \int\limits_0^x e^{-\frac{t^2}{2}}\, dt$$

В нашем случае p – неизвестная заранее вероятность появления события A, то есть выбора гипотезы H_0 или H_1. Поскольку изначально мы предполагаем, что вероятность выбора гипотезы H_0 совпадает с вероятностью выбора гипотезы H_1, то $p = 0.5$. Ряд практически важных значений N при различных ε, Q приведён в приложение (1).

2) *Построение линий уровня.* Для заданных значений вероятностей ошибок первого рода $\alpha_i = 0.15, 0.1, 0.05, 0.01$ и второго рода $\beta_j = 0.15, 0.1, 0.05, 0.01$ строятся линии равного уровня $\alpha\left(lA, lB\right) = \alpha_i$ и $\beta\left(lA, lB\right) = \beta_j$.

3) *Вычисление критических границ* Пересечение линий равного уровня дает искомые критические значения для заданных вероятностей ошибок первого и второго уровня (α_i, β_j).

Рассмотрим одну из возможных реализаций приведенной выше схемы. В качестве входных данных мы можем положить $\varepsilon, Q, step, \alpha_0, \beta_0, \alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m$, где $\alpha_1, \ldots, \alpha_m$ и $\beta_1, \ldots, \beta_m$ – аргументы для второго этапа. Причем для простоты можно считать, что вместо Q, ε на вход подается уже заранее рассчитанное число N.

$\alpha_0, \beta_0, step$ – это обязательные аргументы, поскольку они служат для определения расчетной сетки. Исходя из теоретических соображений, всегда можно считать $\alpha_0 = \min\limits_{i=1...m}(\alpha_i)$, $\beta_0 = \min\limits_{i=1...m}(\beta_i)$.

Сперва определяются $la(\alpha_0, \beta_0)$, $lb(\alpha_0, \beta_0)$ по формулам (1.12), затем вычисляется количество узлов сетки:

$$nMaxB = \left\lceil \frac{-lb(\alpha_0, \beta_0)}{step} \right\rceil; \quad nMaxA = \left\lceil \frac{la(\alpha_0, \beta_0)}{step} \right\rceil;$$

Соответственно, тогда программу реализующую первый этап, рассмотренной выше схемы, можно описать следующим образом:

Алгоритм 1: Моделирование зависимости $\alpha(lA, lB)$

Input: $nMaxA$, $nMaxB$, N, $step$

Output: $M\alpha$

```
1:  Mα ← Matrix[0, {0, nMaxB}, {0, nMaxA}]
2:  lA ← 0
3:  for i = 0 ... nMaxA − 1 do
4:      lA += step
5:      lB ← 0
6:      for j = 0 ... nMaxB − 1 do
7:          lB −= step
8:          for k = 0 ... N − 1 do
9:              Z ← 0
10:             while lB < Z and Z < lA do
11:                 x ← RandomVariate[NormalDistribution[0, 1]]
12:                 Z += ln(f1(x) / f0(x))
13:             end while
14:             if Z ≥ lA then
15:                 Mα[i][j] += 1
16:         end for
17:     end for
18: end for
19: return Mα
```

Ln1: создаем нулевую матрицу $M\alpha$ размера $nMaxA \times nMaxB$, которая будет содержать результат.

Ln4, Ln7: устанавливаем новые значения границ областей принятия гипотез.

Ln8: в теле данного цикла для выбранных границ проводим эксперимент N раз.

Ln10: пока не попали в одну из областей принятия гипотезы, генерируем x – значение случайной величины $X \sim \mathcal{N}(0, 1)$ и вычисляем статистику.

Ln14: проверяем, что мы попали именно в область принятия гипотезы H_1.

Очевидно, что не считая изменения названий, для моделирования зависимости $\beta(lA, lB)$,

необходимо изменить Ln11 на $x \leftarrow$ RandomVariate[LogisticDist[0, Sqrt[3]/Pi]] и условие в Ln14 на $Z \leq lB$. Полный код с внесёнными изменениями приведён в приложение (2)

В результате работы программы мы получим две матрицы – $M\alpha$, $M\beta$. Элементами матриц являются натуральные числа и можно продолжить работу именно с ними, не производя деление на N, однако на данный момент мы следуем рассмотренной выше схеме, а поэтому необходимо каждый элемент матрицы поделить на N. Визуализацию, получившихся результатов при конкретных $nMaxA$, $nMaxB$, N, $step$ можно найти в приложении (3).

Следующим этапом является построение линий уровня. Для этого при помощи точность ε, выбранной на этапе расчета числа N, и интересующих нас значений α_j, β_j выделим из матриц $M\alpha$, $M\beta$ подматрицы элементы которых соответственно принадлежат $(a_j - \varepsilon; a_j + \varepsilon)$, $(b_j - \varepsilon; b_j + \varepsilon)$. Визуализацию полученных результатов смотреть в (4), (5). Из них видно, что при малом числе N относительно выбранной точности ε могут возникать значительные области разрывов, что приведет к невозможности определения точек пересечений, как например в приложении (6, 6.13b).

Затем необходимо найти точки пересечения линий уровня, а именно такие i, j, что элементы $a_{i,j} \in M\alpha$, $b_{i,j} \in M\beta$ принадлежат также и соответствующим подматрицам $M\alpha_j$, $M\beta_j$, которые задают выбранные линии уровня. Результаты данной процедуры приведены в (6).

Таким образом, рассмотрев изначально предложенную С. Н. Постоваловым схему, мы можем приступить к её модификации. Сперва сосредоточимся на первом этапе.

2.1 Модифицированное моделирование функций вероятностей ошибок первого и второго рода

Исходя из рассмотрения алгоритма (1) отметим, что его вычислительная сложность квадратичным образом зависит от размера шага сетки. Поэтому первым делом попытаемся изменить это. Для этого рассмотрим график (2.1) изменения значения переменной Z (из Ln12) при расчетных границах в данный момент $lA_1 = 1$, $lB_1 = lb$, что соответствует некоторому узлу (i, j). Мы можем заключить, что на 8-й итерации цикла (Ln10) мы выйдем из него и примем гипотезу H_1. Более того, для любой $lA \in (Z_5; Z_8)$ и $lB = lb$ мы выйдем из цикла на 8-й итерации.

Откуда можно сделать вывод что при вычисление цикла (Ln10), мы не только можем заключать о принятие гипотез H_0, H_1 для границ lA, lB, но и для всех границ лежащих в полуинтервалах $(0; lA]$, $[lB; 0)$.

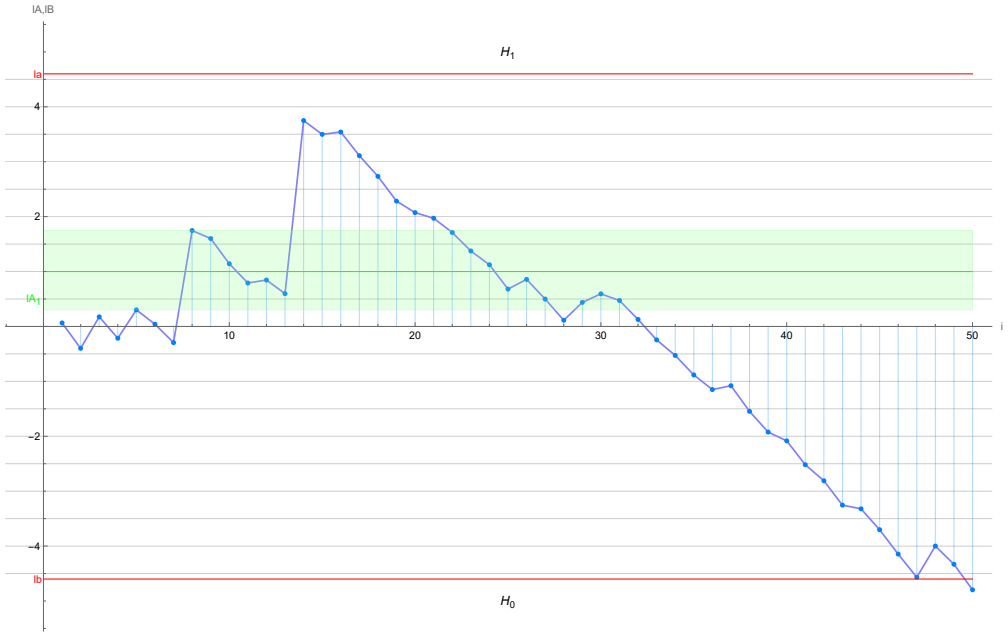


Рис. 2.1: График Z_i при $i \in \left[1; 50\right]$

Таким образом, одна сгенерированная последовательность Z_n при выбранных $lB = lb$, $lA = la$ позволит дать ответ о принятие основной или альтернативной гипотезы для всех интересующих границ. Однако возникает вопрос о том как сохранить информацию о том, какую из гипотез для конкретных lA, lB мы приняли? Наиболее простое решение – это сохранять результаты в виде матрицы V, а именно заранее создать нулевую матрицу размера $1/step \times 1/step$ и при вычислении зависимости $\alpha\left(lA, lB\right)$ помечать 1 значение принятия гипотезы H_1, а при $\beta\left(lA, lB\right)$ – нулём принятие гипотезы H_0. После чего сложить, получившуюся матрицу с итоговой.

Так для примера Z_i, изображенного на (2.1), матрица V при $step = 0.1$ будет выглядеть:

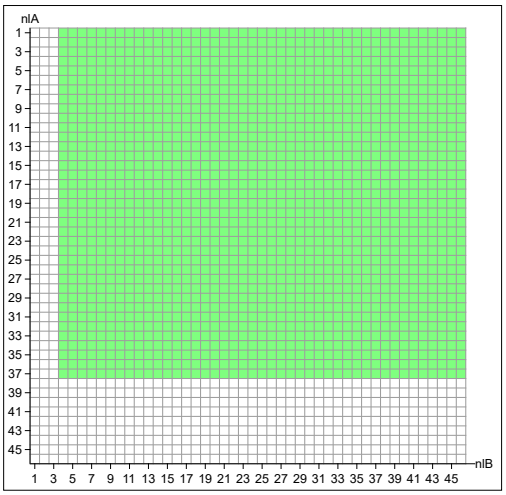


Рис. 2.2: Матрица V при $step = 0.1$

Матрица V при $step = 0.01$ приведена в приложении (7).

Более эффективный способ сохранения требуемой информации мы рассмотрим позже. Сейчас же приведем одну из возможных реализаций модифицированной версии модели-

рования зависимости $\alpha\,(lA, lB)$:

Алгоритм 2: Моделирование зависимости $\alpha\,(lA, lB)$ версия №2

Input: N, $step$, lb, la, $nMaxlA$, $nMaxlB$

Output: $M\alpha$

1: $M\alpha \leftarrow Matrix[0, \{0, nMaxlB\}, \{0, nMaxlA\}]$
2: **for** $i = 0 \ldots N - 1$ **do**
3: $\quad Z \leftarrow 0$
4: $\quad V \leftarrow Matrix[0, \{0, nMaxlB\}, \{0, nMaxlA\}]$
5: $\quad maxlB = 0; nlB = 0; nlA = 0; j = 0; m = 0;$
6: $\quad$ **repeat**
7: $\quad\quad$ **if** $Z > 0$ **then**
8: $\quad\quad\quad nlB = -IntegerPart[maxlB/step]$
9: $\quad\quad\quad nlA = IntegerPart[Z/step]$
10: $\quad\quad\quad$ **while** $m < nlA$ **do**
11: $\quad\quad\quad\quad$ **for** $j = nlB \ldots nMaxlB$ **do**
12: $\quad\quad\quad\quad\quad V[m, j] = 1$
13: $\quad\quad\quad\quad$ **end for**
14: $\quad\quad\quad\quad m \mathrel{+}= 1$
15: $\quad\quad\quad$ **end while**
16: $\quad\quad$ **else**
17: $\quad\quad\quad maxlB = Min[maxlB, Z]$
18: $\quad\quad x \leftarrow$ RandomVariate[NormalDistribution[0, 1]]
19: $\quad\quad Z \mathrel{+}= \ln\left(\dfrac{f_1(x)}{f_0(x)}\right)$
20: $\quad$ **until** $Z > lb$ *and* $Z < la$
21: $\quad$ **if** $Z \geq la$ **then**
22: $\quad\quad nlB = -IntegerPart[maxlB/step]$
23: $\quad\quad nlA = nMaxlA$
24: $\quad\quad$ **while** $m \leq nlA$ **do**
25: $\quad\quad\quad$ **for** $j = nlB \ldots nMaxlB$ **do**
26: $\quad\quad\quad\quad V[m, j] = 1$
27: $\quad\quad\quad$ **end for**
28: $\quad\quad\quad m \mathrel{+}= 1$
29: $\quad\quad$ **end while**
30: $\quad M\alpha \mathrel{+}= V$
31: **end for**
32: **return** $M\alpha$

Ln2: теперь цикл по количеству испытаний в методе Монте-Карло является внешним. В начале каждой итерации обнуляется вспомогательная матрица V и все переменные необходимые для сохранения информации о принятие гипотез.

Ln6: в цикле `repeat` первая итерация проигнорирует вложенный `if` и повторно запишет в `maxlB` значение 0. После чего сгенерируется новое значение x и обновиться значение статистики Z. Далее проверяется условие принятия одной из гипотез. Такого рода конструкция позволит гарантировать, что в Ln9 не произойдет запись в nlB значения большого, чем размер матрицы V.

Ln21: после выхода из цикла мы должны проверить какая гипотеза принялась. Если H_1, то необходимо заполнить оставшиеся $nlA - m + 1$ строк в матрице V, по аналогичной процедуре, рассмотренной ранее.

С незначительными изменениями аналогичным образом реализуется модификация моделирования зависимости $\beta\left(lA, lB\right)$ полный код которого приведён в приложении (8)

Далее рассмотрим модификацию второго и третьего этапов, а именно выделение линий уровня и поиск их пересечений.

2.2 Изменённый метод выделения точек пересечений линий уровня

Пусть мы вычислили матрицы $M\alpha$ и $M\beta$ и хотим найти точку пересечения линий уровня $\alpha_j = 0.01$, $\beta_j = 0.01$. Ранее мы решили данную задачу поэлементным перебор матриц $M\alpha$, $M\beta$, что является очень затратным процессом. Предложим более эффективный подход основанный на итерационном способе выбора интересующей нас точки. Для этого рассмотрим ранее приведенные зависимости $\alpha\left(lA, lB\right)$, $\beta\left(lA, lB\right)$ в приложении (3).

Пусть α_1, β_1 соответствует lA_1, lB_1, а α_2, β_2 – lA_2, lB_2. Мы хотим понять, как измениться α_1, β_1 относительно α_2, β_2. Обозначим за $<$ незначительное уменьшение (незначительно меньше), $>$ незначительное увеличение (незначительно больше), $\ll$ значительное уменьшение (значительно меньше), $\gg$ значительное увеличение (значительное больше) α_1. Тогда можно сделать следующие наблюдения:

- если $lA_1 = lA_2, lB_1 > lB_2$, то $\alpha_1 < \alpha_2$ и $\beta_1 \gg \beta_2$, то есть, если нижняя граница принятия гипотезы H_0 сдвигается вниз, то вероятность принятия гипотезы H_1 незначительно повышается, однако при этом вероятность принятие гипотезы H_0 существенно уменьшается.

- если $lA_1 = lA_2, lB_1 < lB_2$, то $\alpha_1 > \alpha_2$ и $\beta_1 \ll \beta_2$, то есть, если нижняя граница принятия гипотезы H_0 сдвигается вверх, то вероятность принятия гипотезы H_1 незначительно уменьшается, однако при этом вероятность принятие гипотезы H_0 существенно увеличивается.

- если $lA_1 > lA_2, lB_1 = lB_2$, то $\alpha_1 \ll \alpha_2$ и $\beta_1 > \beta_2$, то есть, если верхняя граница принятия гипотезы H_1 сдвигается вниз, то вероятность принятия гипотезы H_0 существенно уменьшается, однако при этом вероятность принятие гипотезы H_1 незначительно увеличивается.

- если $lA_1 < lA_2, lB_1 = lB_2$, то $\alpha_1 \gg \alpha_2$ и $\beta_1 < \beta_2$, то есть, если верхняя граница принятия гипотезы H_1 сдвигается вверх, то вероятность принятия гипотезы H_0 существенно

увеличивается, однако при этом вероятность принятие гипотезы H_1 незначительно уменьшается.

Таким образом, на основе этих наблюдений можно сформулировать правила поиска элементов $a_{r,c}$, $b_{r,c}$ в матрицах $M\alpha$, $M\beta$, соответствующих точке (α_j, β_j). Для этого договоримся о нумерации строк и столбцов в матрицах $M\alpha$, $M\beta$. Пусть a_0, b_0, соответствующие $lA = lB = 0$, при шаге $step = 0.01$ располагаются по координатам $r = 459$, $c = 459$, а a_{la}, b_{lb} по координатам $r = 0, b = 0$, то есть нумерация координат происходит слева направо, сверху вниз, как это было, например, в приложении (7). Тогда:

- если $\alpha > \alpha_j$, $\beta > \beta_j$, то необходимо увеличить области принятия гипотезы H_0 и H_1, то есть уменьшить значения lA и увеличить lB, что соответствует, уменьшить r и c;
- если $\alpha < \alpha_j$, $\beta < \beta_j$, то необходимо уменьшить области принятия гипотезы H_0 и H_1, то есть увеличить lA и уменьшить lB, что соответствует, увеличить r и c;
- если $\alpha \leq \alpha_j$, $\beta > \beta_j$, то необходимо увеличить область принятия гипотезы H_0 и уменьшить область принятия гипотезы H_1, то есть увеличить lA и lB, что соответствует увеличению r и уменьшению c.
- если $\alpha > \alpha_j$, $\beta \leq \beta_j$, то необходимо уменьшить область принятия гипотезы H_0 и увеличить область принятия гипотезы H_1, то есть уменьшить lA и lB, что соответствует уменьшению r и увеличению c.

В приведённых, правилах необходимо уточнить «на сколько уменьшить (увеличить) r, c». В зависимости от этого мы будем получать различных алгоритмы решения задачи. В данной работе мы рассмотрим алгоритм идейно схожий с бинарным поиском. Для этого, в выше изложенных наблюдениях, введём допущение, а именно мы не будем учитывать все незначительные изменения. Более подробно в чем это выражается мы рассмотрим ниже.

Алгоритм 3: Алгоритм нахождения точки пересечения заданных линий уровня α_j, β_j с точностью ε

Input: $M\alpha$,$M\beta$ $step$, α_j, β_j, ε
Output: $\hat{\alpha}, \hat{\beta}$

$la_j \leftarrow \ln((1-\beta_j)/\alpha_j)$; $lb_j \leftarrow \ln(\beta_j/(1-\alpha_j))$
$R \leftarrow Ceiling[la_j/step]$; $C \leftarrow Ceiling[-lb_j/step]$
$br \leftarrow 1$; $tr \leftarrow R$
$lc \leftarrow 1$; $rc \leftarrow C$
$r \leftarrow Ceiling[(br+tr)/2]$; $c = C$
$a \leftarrow M\alpha[r,c]$; $b \leftarrow M\beta[r,c]$
$\Delta_\alpha \leftarrow a - a_j$; $\Delta_\beta \leftarrow b - b_j$
if $\Delta_\alpha > 0$ **then**
    $br = r$
else
    $tr = r - 1$

Алгоритм 3: Продолжение

```
12: while (Abs[Δ_α] > ε || Abs[Δ_β] > ε) do
13:     tr = R; rc = C
14:     while Abs[Δ_β] > ε && lc ≠ rc do
15:         c = Ceiling[(lc + rc)/2];
16:         a = Mα[r, c]; b = Mβ[r, c]
17:         Δ_α = a − a_j; Δ_β = b − b_j
18:         if Δ_β > 0 then
19:             lc = c
20:         else
21:             rc = c − 1
22:     end while
23:     if Abs[Δ_β] > ε && lc == rc then
24:         a = Mα[r, lc]; b = Mβ[r, lc]
25:         Δ_α = a − a_j; Δ_β = b − b_j
26:         if Abs[Δ_β] > ε then
27:             return not found suitable Δ_β < ε
28:     end if
29:     while Abs[Δ_α] > ε && br ≠ tr do
30:         r = Ceiling[(br + tr)/2];
31:         a = Mα[r, c]; b = Mβ[r, c]
32:         Δ_α = a − a_j; Δ_β = b − b_j
33:         if Δ_α > 0 then
34:             br = r
35:         else
36:             tr = r − 1
37:     end while
38:     if Abs[Δ_α] > ε && br == tr then
39:         a = Mα[tr, c]; b = Mβ[tr, c]
40:         Δ_α = a − a_j; Δ_β = b − b_j
41:         if Abs[Δ_α] > ε then
42:             return not found suitable Δ_α < ε
43:     end if
44: end while
45: α̂ ← a; β̂ ← b
46: return α̂, β̂
```

Ln1-Ln4: Определяем по формулам (1.12) границы la_j, $l\beta_j$ при заданных α_j, β_j. Определим строку и столбец в матрицах $M\alpha$, $M\beta$, соответствующие вычисленным границам. Инициализируем вспомогательные переменные br (нижняя строчка), tr (верхняя строчка), lc (левый столбец), rc (правый столбец). Переменные br, tr участвуют в бинарном поиске для

определения r, а lc, rc для c.

Ln5-Ln11: Выполняем подготовительный этап к запуску основного цикла, а именно уменьшаем в два раза r и выбираем нужную половину интервала. Однако, на самом деле в силу того, что изначально мы определяем r и c исходя из формул (1.12), то при любых α_j, β_j изначально мы будем находиться выше требуемой точке, соответственно, Δ_α всегда больше нуля.

Ln12: Это основной цикл алгоритма, условие выхода из которого является решение задачи, то есть определения таких r, c, по которым найдутся такие $a = M\alpha\big[r, c\big], b = M\beta\big[r, c\big]$, что $\Delta_\alpha = a - \alpha_j$ и $\Delta_\beta = b - \beta_j$ будут меньше, чем заданная точность ε.

Ln14-Ln21: Осуществляется классический бинарный поиск по c. Условием выхода из цикла Ln14 является либо достижение заданной точность, либо сужение интервала поиска до единственной точки.

Ln23-Ln27: попасть в тело данного `if` можно только в том случае, если при выполнение цикла Ln14 мы не достигли заданной точности, но границы интервала поиска стали равными. Тогда, либо при $c = lc = rc$ достигается требуемая точность, либо мы заключаем не возможность достижения заданной точности.

Ln29-Ln42: аналогичным образом выполняем классический бинарный поиск по r.

Продемонстрируем действие данного алгоритмы на примере поиска точки пересечения линий уровня $\alpha_j = 0.01$, $\beta_j = 0.15$ при $\varepsilon = 0.001$. Как и в приложении (5) раскрасим и выведем и другие линии уровня. Матрицы $M\alpha$, $M\beta$ были получены, рассмотренными ранее алгоритмами (2), (8) при $N = 6 \cdot 10^6$ и $step = 0.01$.

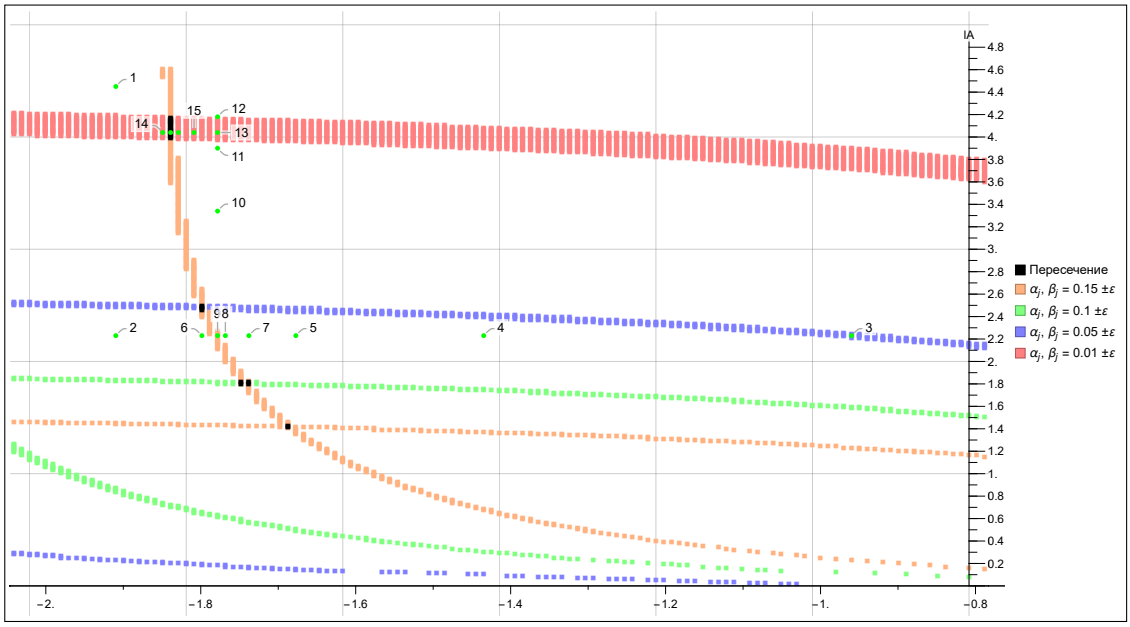


Рис. 2.3: Пример работы алгоритма (3)

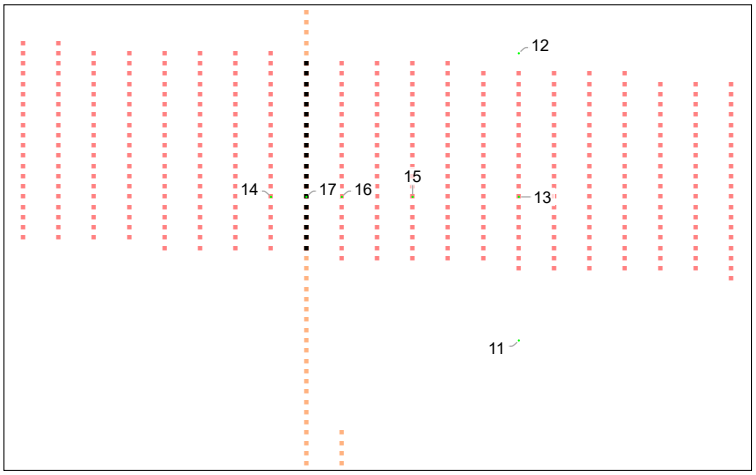


Рис. 2.4: Пример работы алгоритма (3) в увеличенном масштабе

Видим, что итерационный алгоритм успешно нашел точку пересечения линий уровня за 17 итераций, точнее за 16 поскольку точка 1 – это изначальное положение, которое соответствует границам la_j, lb_j из Ln1.

Однако несмотря на то что алгоритм (3) справляется с довольно большим количеством тестов в нём не решена следующая проблема, заключающаяся в том, что при выполнение внутренних циклов Ln14 Ln29 или `if` после них, мы всегда предполагаем достижение заданной точности, что бывает не всегда.

Пусть $a_j = 0.1, b_j = 0.01, N = 6 \cdot 10^6, step = 0.01, \varepsilon = 1/2200$. Согласно формуле (2.4) с $0.97 < Q < 0.98$ нам хватит указанного N. И действительно, полным перебором элементов матриц $M\alpha$, $M\beta$ мы убеждаемся, что существует 7 точек удовлетворяющие заданной точности. Однако алгоритм (3) сталкивается с проблемой невозможности при фиксированном c, найти такое r, чтобы Δ_α удовлетворяла заданной точности. Причем при том c, которое мы получаем с 8 итерации в принципе не существует такого r. Для наглядности изобразим выше сказанное, где по прежнему всё за исключением итерационных точек было получено при помощи ранее рассмотренного алгоритма полного перебора.

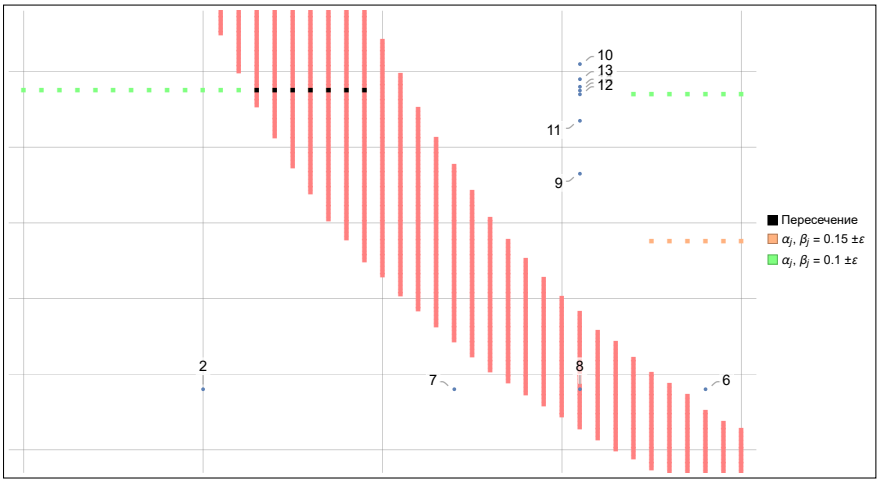
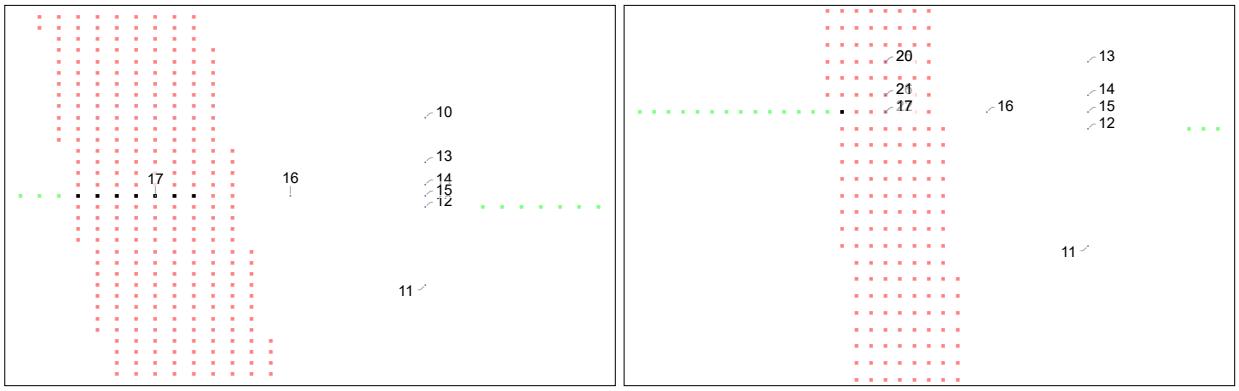


Рис. 2.5: Проблема алгоритма (3)

Частичным решением данной проблемы является удалением (27), (42). Простое их

удаление приведет к бесконечному циклу, поэтому введем вспомогательные переменные `or`, `oc` – (старые значения r и c), которые будем сравнивать в каждой итерации цикла (12), а именно, если по окончанию выполнения итерации цикла (12) значения r, c не изменились, то значит и на следующей итерации оно не измениться, то есть мы попали в бесконечный цикл, следовательно, при первом же повторе необходимо выйти из данного цикла и выдать результат. Полный вариант исправленной версии приведен в приложении (9) – алгоритм (9). Результатом его работы будет:



(a) $\varepsilon = 1/2200$ (b) $\varepsilon = 1/2488$

Рис. 2.6: Результат работы алгоритма (9)

При $\varepsilon = 1/2488$ мы можем наблюдать эффект связанный с ранее введенном допущением, а именно поскольку во внутренних циклах мы осуществляем поиск либо по Δ_α, либо по Δ_β, тем самым мы не учитываем, что при изменении, например c, по мимо существенного изменения Δ_β незначительно изменяется и Δ_α. Однако, как мы видели выше, при умеренных ε, когда $Q \geq 0.99$ алгоритмы успешно находят требуемую точку.

По мимо малого ε на поиск точки, удовлетворяющий поставленным требованиям, значительную роль играет шаг расчетной сетки. Это хорошо можно проследить, если опустить связанные с ε условия в циклах алгоритма (3). Тогда в каждом внутренним цикле при одной из фиксированной координаты r или c мы будем минимизировать Δ_β или Δ_α соответственно. Это в ряде случаев позволит находить точки с меньшей суммой $|\Delta_\alpha| + |\Delta_\beta|$, но это также повысит число совершаемых итераций и по прежнему, в частных случаях, например, в (2.6b), не будет находить точки удовлетворяющие заданной точности. Алгоритм данного подхода приведен в приложении (10).

Также отметим, что можно объединить внутренние циклы `while` в один, тем самым, совершая не два шага – один по горизонтали, другой по вертикали, а один по диагонали. Это также позволит всегда за полное выполнение внешнего цикла попадать в центр некоторого квадрата Sq_3 размера 3 на 3 ячеек матриц $M\alpha$, $M\beta$, а значит внешний цикла `while` значительно упрощается. После попадания в квадрат Sq_3 по рассчитанным Δ_α, Δ_β необходимо определить, какая из ячеек квадрата Sq_3 доставляет минимум суммы $|\Delta_\alpha| + |\Delta_\beta|$. В приложении (11) приведен код алгоритма, однако без корректной обработки после выхода из цикла `while` и анализа расчетного квадрата Sq_3, поэтому найденная точка будет лежать в более объемлющем квадрате Sq_δ, где $\delta = \{3, 5, 7\}$, размера δ на δ, в котором при заданном $step$

находится оптимальная по сумме точка. Более большой квадрат Sq_δ при достаточно маленьком шаге не является проблемой, поскольку чем меньше $step$ тем незначительней является разница между Sq_3 и Sq_δ. Однако при уменьшении $step$ существенно возрастает количество вычислений. Поэтому именно попытке избавиться от зависимости по $step$ и будет посвящена следующая глава.

3. Итерационный метод уточнения границ в SPRT

Во время написания алгоритма по поиску точек пересечения линий уровня мы описали ряд правил в терминах изменения r и c, которые однозначным образом переносятся на lA, lB. Это на водит на мысль о том, что аналогично алгоритмам (3), (9), (11), (12) можно реализовать алгоритмы, которые не будут вычислять значения α и β во всех узлах сеток и строить матрицы $M\alpha$, $M\beta$ целиком, а будут определять значения α и β лишь в определенных узлах, в результате чего возвращать точку (α, β) не отличающуюся от заданной (α_j, β_j) с точностью ε. В качестве основы возьмем алгоритм (12) из приложения (11), который видоизмениться следующим образом.

Алгоритм 4: Алгоритм нахождения критических границ lA, lB итерационным способом, при заданных ошибках α_j первого, β_j второго рода с точностью ε

Input: α_j, β_j, ε, n

Output: lB, lA

$la \leftarrow \ln\left((1-\beta_j)/\alpha_j\right)$; $lb \leftarrow \ln\left(\beta_j/(1-\alpha_j)\right)$

$blA = 0$; $tlA = la$

$llB = 0$; $rlB = lb$

$\Delta_\alpha = \varepsilon$; $\Delta_\beta = \varepsilon$

while $Abs[\Delta_\alpha] + Abs[\Delta_\beta] > \varepsilon$ **do**

  $lB = (llB + rlB)/2$

  $lA = (blA + tlA)/2$

  $a = FunCalculatesAlpha[n, lB, lA]/n$ `// смотреть приложение (12) алгоритм (13)`

  $b = FunCalculatesAlpha[n, lB, lA]/n$ `// смотреть приложение (12) алгоритм (14)`

  $\Delta_\alpha = a - a_j$; $\Delta_\beta = b_j$

  if $\Delta_\beta > 0$ **then** $LLB = lB$

  else $rlB = lB$

  if $\Delta_\alpha > 0$ **then** $blA = lA$

  else $tlA = lA$

end while

return $\{lB, lA\}$

Без ограничения на $step$, как отмечалось ранее, мы можем достичь требуемую точность, двигаясь по диагоналям, то есть выбирая lB, lA как середины полуинтервалов $[llB; rlB)$, $[blA; tlA)$.

Другой итерационный подход вычислений критических границ lB, lA был рассмотрен С. Я. Гродзенским, А. Н. Чесалиным в работе [4]. С учетом ранее нами сделаными обозначениями, они предлагают ввести следующие относительные величины:

$$\delta_\alpha = \frac{\Delta_\alpha}{\hat{\alpha}}, \quad \delta_\beta = \frac{\Delta_\beta}{\hat{\beta}} \tag{3.1}$$

и на каждой i- итерации их алгоритма определять новые критические границы по формулам

$$\begin{cases} lB^{[i+1]} = lB^{[i]} - \Delta_C \left(\lambda \delta_{\beta}^{[i]} - (1-\lambda)\,\delta_{\alpha}^{[i]}\right) \\ lA^{[i+1]} = lB^{[i]} + \Delta_C \left(\lambda \delta_{\alpha}^{[i]} - (1-\lambda)\,\delta_{\beta}^{[i]}\right) \end{cases} \tag{3.2}$$

где Δ_C, λ – константы алгоритма, причем Δ_C определяет шаг изменения lB, lA, то есть является своего рода скоростью, а λ – весомость каждой из невязок Δ_α, Δ_β на lB, lA.

На основании проведенных моделирований они устанавливают, что $\lambda \in \left[0.9; 1\right]$, а $\Delta_C = 1$, тем самым получая:

$$\begin{cases} lB^{[i+1]} = lB^{[i]} - \dfrac{\Delta_{\beta}^{[i]}}{\hat{\beta}^{[i]}} \\ lA^{[i+1]} = lA^{[i]} + \dfrac{\Delta_{\alpha}^{[i]}}{\hat{\alpha}^{[i]}} \end{cases} \tag{3.3}$$

В формулах (3.3) присутствует деление на $\hat{\alpha}$, $\hat{\beta}$, которые вычисляются методом Монте-Карло по алгоритмам аналогичным (13), (14). Соответственно, если $\hat{\alpha}$ или $\hat{\beta}$ будут достаточно малы, то $lB^{[i+1]}$, $lA^{[i+1]}$ могут выйти за пределы lb, la. В качестве подтверждения рассмотрим следующим пример:

Пример 3.0.1. Пусть $N = 6634897$, $a_j = 0.01$, $b_j = 0.01$, тогда согласно таблицы (6.1) из приложения (1) с достоверностью $Q = 0.99$ имеем $\varepsilon = 0.0005$. Пусть

$$H_0 : f_0(x) = \begin{cases} e^{-x}, & x \geq 0 \\ 0, & \text{иначе} \end{cases} \qquad H_1 : f_1(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-1)^2}; \tag{3.4}$$

то есть основная гипотеза о том, что наблюдаемая случайная величина имеет показательное распределение с параметром $\lambda = 1$, против конкурирующей гипотезы о нормальном распределение со средним 1 и среднеквадратическим отклонением $\sigma = 1$. Тогда

$$z_i = \ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right) = \begin{cases} \infty, & x_i < 0 \\ -\dfrac{1}{2}(x_i - 1)^2 + x_i - \dfrac{1}{2}\ln(2\pi), & \text{иначе} \end{cases}$$

Согласно алгоритмам приведённым в [4] реализуем программу (16). Мы сохраним обозначения в коде, используемые в рассматриваемой статье, а именно $C_0 \equiv lB$, $C_1 \equiv lA$, $\gamma \equiv Z$, $\alpha_j \equiv \alpha$, $\beta_j = \beta$.

Сперва соберём и запустим без `-D Slow_speed`. Тогда на первой итерации мы получим $\hat{\alpha}^{[0]} = 3.52681 \cdot 10^{-5}$, $\hat{\beta}^{[0]} = 0.00610153$, соответственно, подставляя данные значения в формулы (3.3) получим:

$$C_0^{[1]} = C_0^{[0]} - \frac{\hat{\beta}^{[0]} - \beta}{\hat{\beta}^{[0]}} = -4.59512 - \left(1 - \frac{0.01}{0.00610153}\right) \approx -3.95604$$

$$C_1^{[1]} = C_1^{[0]} + \frac{\hat{\alpha}^{[0]} - \alpha}{\hat{\alpha}^{[0]}} = 4.59512 + \left(1 - \frac{0.01}{3.526 \cdot 10^{-5}}\right) \approx -277.947$$

но C_1 должно быть всегда больше 0, поэтому в дальнейшем уже после 3 итерации имеем $C_0 = \infty$, $C_1 = -\infty$. Причиной такого поведения, является слишком большая скорость Δ_C, которая в формулах (3.3) равна 1. Если же мы положим в формулах (3.2) $\lambda = 1$ и $\Delta_C = \frac{1}{70}$, то алгоритм отрабатывает правильно, но очень медленно. Так, собрав и запустив программу (16) с флагом `-D Slow_speed` мы установим $\Delta_C = \frac{1}{70}$ и за 219 итераций получим требуемую точность $\varepsilon = 0.0005$, смотреть таблицу (3.1). Процесс изменения $|\Delta_\alpha|$, $|\Delta_\beta|$, $|\Delta_\alpha| + |\Delta_\beta|$, C_0 и C_1 приведены в приложении (14).

Стоит также отметить, что скорость Δ_C можно слегка увеличить, так, например, из 10 запусков при $\Delta_C = \frac{1}{60}$ всего один раз мы получаем $C_1 < 0$, а при $\Delta_C = \frac{1}{55}$ все 10 запусков приводят к $C_1 < 0$. Безусловно на это также влияет выбранный генератор случайных чисел.

Для сравнения, реализуем программу (17) согласно приведенному в данной работе алгоритму (4). Для наглядности и прозрачности функции `funA`, `funB` с точностью до названия переменных остались без изменения. Результаты работы программы приведены в сравнительной таблице (3.1). Процесс изменения $|\Delta_\alpha|$, $|\Delta_\beta|$, $|\Delta_\alpha| + |\Delta_\beta|$, lB и lA приведены в приложениях (14), (15).

	Результаты программы (16)	**Результаты программы (17)**	**Результаты программы (17)**
n	6634897	6634897	26539587
i	219	6	8
$a_j(\alpha)$	0.01	0.01	0.01
$b_j\,(\beta)$	0.01	0.01	0.01
$lB\,(C_0)$	-4.096957	-4.092529	-4.11048
$lA\,(C_1)$	1.934589	1.938566	1.95652
$\lvert\Delta_\alpha\rvert$	0.000451858	0.000374	0.000006137
$\lvert\Delta_\beta\rvert$	0.00000843	0.00001417	0.000120984
$\lvert\Delta_\alpha\rvert + \lvert\Delta_\beta\rvert$	0.00046	0.000388	0.000127121
ε	0.0005	0.0005	0.00025

Таблица 3.1: Сравнительная таблица результатов работы программ (16), (17)

Поскольку функции `funA`, `funB` реализованы единым образом, то время их выполнения одинаково. При достаточно больших n оставшиеся части, выполняемые на каждой итерации циклов относительно времени выполнения функций `funA`, `funB` незначительны, поэтому время исполнения одной итерации у программ (16), (17) совпадают.

При рассмотрении примера, мы можем заметить, что несмотря на то, что алгоритм (4) совершает всего 6, 8 итераций, каждая из них занимает продолжительное время, а увеличение требуемой точности повышает количество итераций. Поэтому можно задаться вопросом, а нельзя ли на первой итерации обработать полученные данные и сохранить их в такую структуру, что время обращения к ней будет значительно меньше чем повторное проведение n экспериментов для вычисления новых a и b.

4. Выбор структуры данных

При выборе и построение структуры данных воспользуемся результатом, полученным при анализе графика (2.1), а именно тем, что сгенерированная последовательность Z_n при выбранных $lB = lb, lA = la$ позволяет сразу дать ответ о принятие одной из гипотез для всех границ $lB \in [lb; 0)$, $lA \in (0; la]$. Однако теперь мы откажемся от сохранения результатов в матрицы, а попытаемся построить более эффективную структуру данных.

Первым делом отметим, что результаты о принятие гипотез для границ lB, lA можно однозначно задать ломанной линией. Действительно, рассмотрим следующий график одного из эксперимента

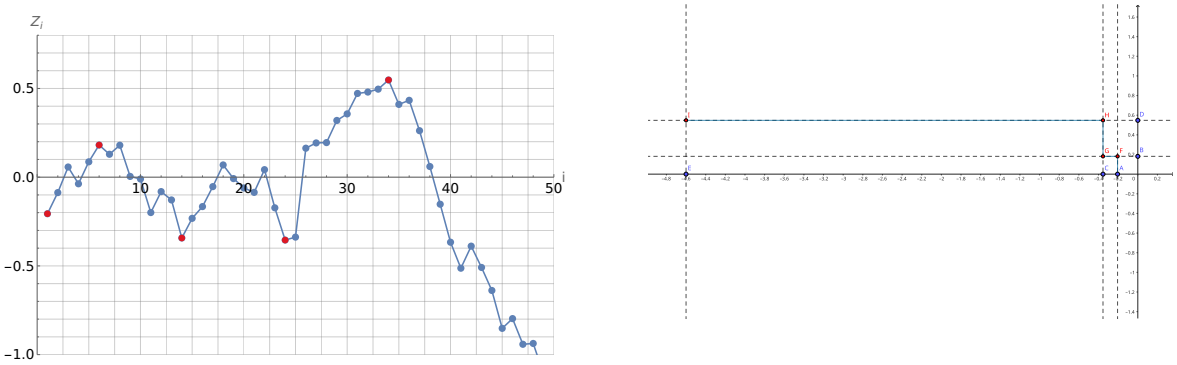


Рис. 4.1: График Z_i при $i \in \overline{[1; 50]}$, и ломанная $AFGHI$

Достигая изначально $Z_1 = -0.205955$ мы заключаем, что для точек $(lB, lA) \in [Z_1; 0) \times (0; la] = S_1$ принимается гипотеза H_0. При $Z_1 < Z_2 < 0$ мы заключаем, что для $(lB, lA) \in [Z_2; 0) \times (0; la] = S_2$, но $S_2 \subset S_1$, поэтому мы не получаем никакой дополнительной информации. При $Z_3 = 0.0573671$ заключаем, что для $(lB, lA) \in [lb; 0) \times [Z_3; la]$ принимается гипотеза H_1, однако $S_1 \cap S_3 \neq \varnothing$ и поскольку точки из S_1 были раньше, то есть при меньших i, то значит при установленных границах из S_1 мы бы закончили рассматривать Z_i принятием гипотезы H_0, поэтому заключаем, что мы принимаем гипотезу H_1 только для точек $S_3 \setminus S_1$. Продолжая данный процесс аналогичным образом, после чего переобозначив значимые точки в алфавитном порядке мы получим точки A, B, C, D, E, где четные точки лежат на оси lA, а не четные на оси lB. В каждой нечетной точки проведём перпендикуляр к оси lB, а в каждой четной перпендикуляр к lA, после чего соединив точки пересечения этих кривых мы получим ступенчатую кривую $AFGHI$, причем для любых точек лежащих ниже данной кривой мы принимаем гипотезу H_1, а для всех, что выше или на кривой гипотезу H_0.

Если изначально мы начинаем с точки A лежащей на оси lA, то для всех нечетных точек проводим перпендикуляр к оси lA, а для всех четных к оси lB.

Ряд других возможных вариантов приведены ниже (4.2), при этом по прежнему для точек выше кривой мы принимаем гипотезу H_0, а для точек ниже кривой H_1.

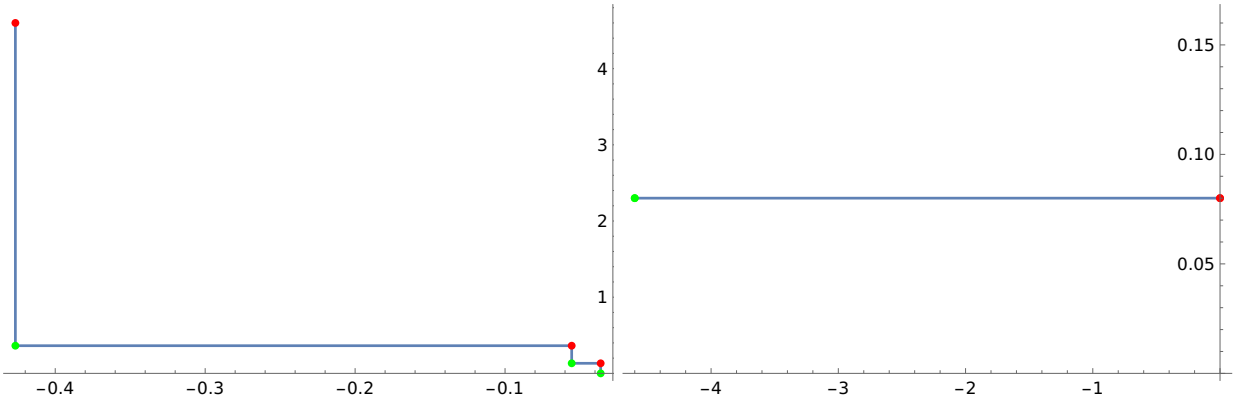


Рис. 4.2: Различные виды, получающихся ломанных.

Таким образом, при проведении n испытаний мы получим n ломанных при моделирование $\hat{\alpha}$ и n ломанных при моделирование $\hat{\beta}$. Причем каждая ломанная однозначно задается либо её горизонтальными звеньями, либо вертикальными. Далее мы будем считать, что ломанные задаются горизонтальными звеньями.

Поскольку каждая ломаная, рассчитанная при изначально заданных границах lb, la, содержит информацию о принятие той или иной гипотезы для любой точки $(lB, lA) \in [lb; 0) \times (0; la]$, то изначально рассчитав и сохранив в некую структуру данных n таких ломаных, а также предоставив алгоритм, который для заданных $(lB, lA) \in [lb; 0) \times (0; la]$ мог бы подсчитать, сколько ломанных ниже точки (lB, lA), мы могли бы решить поставленную изначально задачу по замене моделирования в некой точке (lB, lA) на выполнения запроса к структуре данных.

Однако в формулировке через ломанные данная задача уже близка к классическим, хорошо изученным задачам вычислительной геометрии. Причем у нас есть большая вариативность в том, к какой именно из известных задач свести нашу проблему.

Так например, мы можем её переформулировать следующим образом [20]: предварительно обработать набор S, возможно цветных геометрических объектов (точек, ортогональных сегментов, прямоугольников) в $\mathbb{R}^d$, $d \geq 2$, так чтобы при заданном некотором диапазоне запроса q (например, точка, гипербокс), мы смогли эффективно сообщать для каждого отдельного цвета $S \cap q$ набор $\langle c, \mathcal{F}(c) \rangle$, где $\mathcal{F}(c)$ – функция (например, взвешенная сумма, максимум и так далее) объектов цвета c в q.

При различных конкретизациях: выбор модели вычисления, допускается ли добавление и/или удаление из структуры, работаем в основном с оперативной памятью или с внешней памятью, строим ли мы структуру эффективную для различных d или же для конкретных и так далее, данная задача принимает различные уточнённые варианты и соответственно каждый такой вариант представляет из себя отдельную задачу. В данный момент, несмотря на то что мы решаем не четко поставленную задачу, мы можем выделить ряд свойств, которые позволят определиться с дальнейшим направлением решения.

Сперва сразу отметим, что в нашей задаче $d = 2$, то есть сегменты – отрезки, причём все отрезки осепараллельны. Это значительно упрощает работу [8], поскольку в нашем случае S – множество осепараллельных отрезков.

Также в задаче изначально можно считать, что данные принимают вещественные значения, поэтому мы можем либо продолжить работать с вещественными числами, тем самым отказаться от ряда структур и алгоритмов, которые предполагают работу с целочисленными данными, например смотреть [12, 16, 17, 18], или же как предложено в работе [13] и более подробно описано в [16] перейти к пространству рангов. В данной работе было принято решение не переходить к пространству рангов.

Соответственно, если задать однозначное соответствие между отрезками и граничными точками, то можно свести задачу к поиску в ортогональном диапазоне. Тогда для решения данной задачи существует множество структур данных: дерево диапазонов, дерево интервалов, kd-дерево, R-дерево, B-дерево и ряд других структур, большая часть из которых описала в [8, 11]. Но нам требуется не просто посчитать количество точек в некотором прямоугольном диапазоне, поскольку одна ломанная может состоять из нескольких отрезков, множество граничных точек которых попадет несколько раз в оконный запрос, например на рисунке (4.3) красные сегменты.

Однако наши отрезки не просто осепараллельные, а еще и обладают следующим свойством: начало каждого следующего отрезка совпадает с концом предыдущего, то есть если говорить про горизонтальные отрезки, то правая точка отрезка i имеет тоже значение координаты x, что и левая точка отрезка $i-1$. Более наглядно это можно увидеть на рисунке (4.3). Этот факт хоть и не позволяет полностью отказаться от запоминания значений правой или левой точки отрезка, но позволяет вместо четырех значений для одного отрезка хранить три. А именно мы можем хранить уникальный идентификатор ломанной к которому данный отрезок и соответственно граничные точки принадлежат, тем самым мы получим задачу поиска в ортогональном диапазоне размерности d цветных точек. Данная задача является классической для вычислительной геометрии и рассматривалась во множестве работ, причем как при эффективной работе только, используя оперативную память, так и внешнюю, а также возможность достижения асимптотически оптимальных вычислительных границ как по памяти, так и по времени, например смотреть работу [10] и приведенный в ней список литературы с историей решения данной задачи.

Вместо принадлежности к ломанной, мы можем хранить только координаты левой границы и лишь координату x правой точки. Тем самым, используя три вещественных числа, мы можем свести задачу к поиску в ортогональном диапазоне размерности $d = 3$ и конкретизировать запрос q, а именно, если требуется найти значения $\hat{\alpha}$ в точке $P = (x_1, x_2, y)$, то запрос q должен вернуть количество точек лежащих в $D = [x_1, x_2] \times [x_2; +\infty) \times (-\infty; y]$. Такого рода диапазон D очень похож на двухмерный аналог 3-стороннего прямоугольника.

Именно последний вариант мы и будем реализовывать. Дополнительно потребуем, чтобы структура S поддерживал добавление элементов. Тогда $S \subset \mathbb{R}^3$ – набор некоторых точек в $\mathbb{R}^3$, причем мы будем добавлять в структуру S точки $z = (x'_1, x'_2, y')$, где (x'_1, y') – это левая точка горизонтального отрезка, получаемого в ходе моделирования ломанных, а x'_2 – координата x правой точки того же отрезка.

Уточним как при помощи запроса q мы будем получать $\hat{\alpha}$ и $\hat{\beta}$. Аргументами запроса

всегда будут являться lb, lB, lA, то есть $x_1 = lb$ всегда фиксированный, но можно взять и меньший, поскольку в нашей структуре просто не может быть точек с координатой $x < lb$[1], $x_2 = lB$, $y = lA$, причем в силу свойств lB, lA диапазон поиска примет окончательный вид:

$$D = \left\{(x'_1, x'_2, y') \in \mathbb{R}^3 \mid x'_1 \in [lb, lB], x'_2 \in [lB; 0), y' \in (0; lA]\right\} \tag{4.1}$$

Соответственно запрос q должен вернуть $\mathrm{card}\,(S \cap D)$, а именно необходимо эффективно подсчитать количество точек удовлетворяющих следующему условию:

$$(lb \leq x'_1) \text{ and } (x'_1 \leq lB) \text{ and } (y' \leq lA) \text{ and } (lB \leq x'_2)$$

где $(x'_1, x'_2, y') \in S$.

Следующий рисунок иллюстрирует запрос q для точки $(-10.5, -5, 3.5)$ к структуре S, состоящей из точек, первые две координаты которых совпадают с зелеными точками на рисунке, а третья со значением координаты x правой точки относительно зеленой. Таким образом, всего две точки – $(-7.5, 0.5, 0)$, $(-6, 2.5, -3)$ удовлетворяют условиям (4.1), а значит запрос вернет число 2, это соответствует тому, что только две ломанные находиться ниже или содержат точку P.

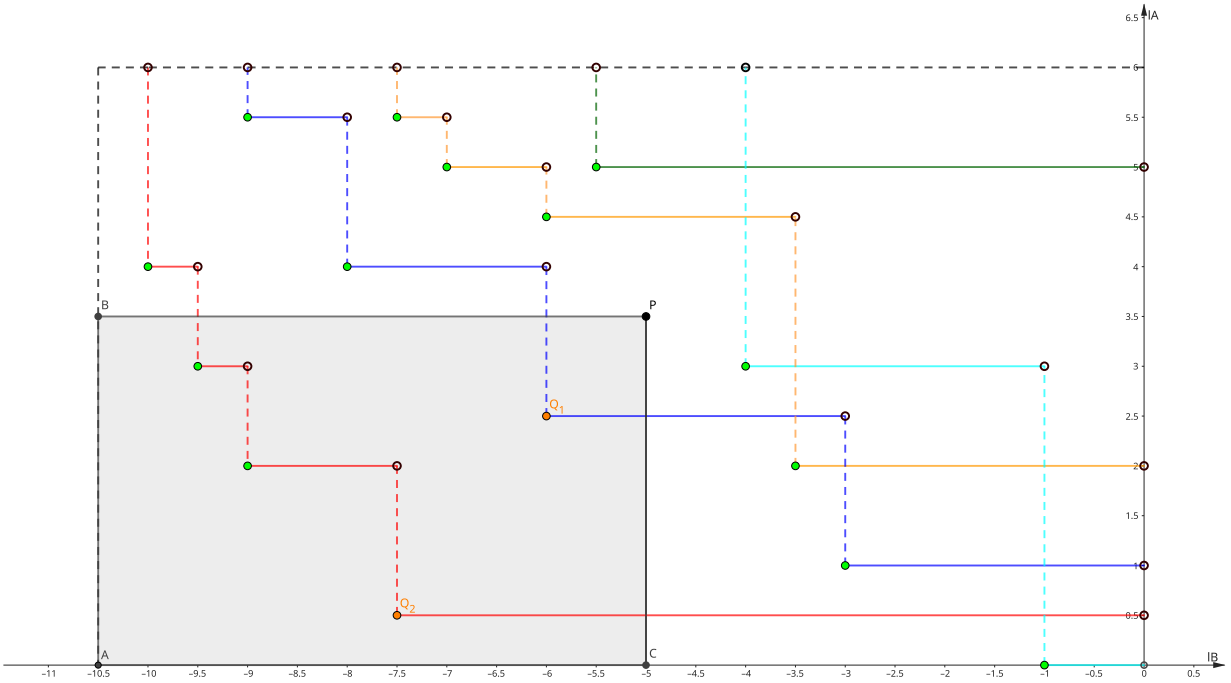


Рис. 4.3: Иллюстрация к процессу выполнения запроса q

Таким образом, по запросу q мы можем получить количество ломанных ниже или содержащих указанную в запросе q точку (lB, lA), то есть количество испытаний, завершившихся принятием гипотезы H_0, тогда если мы моделировали $\hat{\beta}$, то нам достаточно просто вернуть значение, полученное из запроса q. Однако если мы моделировали $\hat{\alpha}$, то нас интересует число испытаний, завершившихся принятием гипотезы H_1, а не H_0. Поскольку мы

[1] Возможно, если положить вместо lb |DBL_MIN|, то можно будет избавиться от лишних условных операторов в коде?!

знаем, что всего было проведено n испытаний и все они завершились принятием либо гипотезы H_0 либо H_1, то для получения интересующего нас числа нам достаточно вычислить разницу между n и числом, полученным из запроса q.

Рассмотрев основные моменты ранее поставленной абстрактной задачи мы можем подойти более осознано к выбору структуры данных S, а именно структуры с учетом возможности добавление элементов и эффективным поиском по «трех стороннему прямоугольнику». Похожим поиском в июне 2016 года занимались Kasper Green Larsen и Gerth Stolting Brodal, проделов хорошую работу по сравнению различных структур для решения задачи о трехстороннем отчете в ортогональном диапазоне [14]. Согласно их заключению во внутренней памяти R-дерево из библиотеки `boost` отлично справляется с поставленной задачей, чего нельзя сказать если дело касается внешней. Поскольку пробные измерения показывали, что из одного испытания мы не будем получать более 10 точек для сохранения информации о ломаной и, располагаемой оперативной памяти достаточно для проведения испытаний при $n = 10^6$, $5 * 10^7$, то было принято решение остановиться на выборе R-дерева в качестве структуры данных S.

5. Реализация программы с использованием структуры S

Первым делом реализуем вспомогательный класс поверх R-дерева из библиотеки `boost`.

Алгоритм 5: Вспомогательные класс boost_r_tree

```
#ifndef BOOST_R_TREE_HPP
#define BOOST_R_TREE_HPP
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/point.hpp>
#include <boost/geometry/index/rtree.hpp>
#include <boost/iterator/function_output_iterator.hpp>

#include "common/point.hpp"
#include  <iostream>

namespace bg = boost::geometry;
namespace bgi = boost::geometry::index;

using pt = bg::model::point<double, 3, bg::cs::cartesian>;

namespace internal {
    class boost_r_tree{
    public:
        boost_r_tree();
        boost_r_tree(const size_t buffer_size);
        ~boost_r_tree();
        void insert(const point &p);
        void remove(const point &p);
        void report(const double x1, const double x2, const double y);
        size_t report_count(const double x1, const double x2, const double y);
        size_t size();
        size_t get_buffer_size();
        void print();
    private:
        bgi::rtree<pt, bgi::quadratic<16> > rtree;
        unsigned next_id;
        size_t buffer_size;
    };

    boost_r_tree::boost_r_tree() {
        next_id = 0;
        buffer_size = 1024;
    }

    boost_r_tree::boost_r_tree(size_t buffer_size) {
```

```
41          next_id = 0;
42          this->buffer_size = buffer_size;
43      }
44
45      boost_r_tree::~boost_r_tree() {
46          rtree.clear();
47      }
48
49      void boost_r_tree::insert(const point &p) {
50          rtree.insert(pt(p.x1,p.y,p.x2));
51      }
52
53      void boost_r_tree::remove(const point &p) {
54          rtree.remove(pt(p.x1,p.y,p.x2));
55      }
56
57      size_t boost_r_tree::size() {
58          return rtree.size();
59      }
60      size_t boost_r_tree::get_buffer_size() {
61          return buffer_size;
62      }
63
64      void boost_r_tree::report(const double x1,const double x2,const double y) {
65  #ifdef DEBUG
66          std::cout <<"\n{x1,x2,y} = {"<< x1 <<", " << x2 <<", " << y<<"}\n";
67  #endif
68          rtree.query(bgi::satisfies([&](pt const& v) {
69              double x1p = v.get<0>();
70              double yp = v.get<1>();
71              double x2p = v.get<2>();
72  #ifdef DEBUG
73              std::cout << "{x1p, yp, x2p} = {" << x1p <<", " << yp  <<", " << x2p <<"}
                ↪  "<<
74              "In_range? " << (x1 <= x1p && x1p <= x2 && yp <= y && (x2 <= x2p)) <<
                ↪  std::endl;
75  #endif
76              return x1 <= x1p && x1p <= x2 && yp <= y && (x2 <= x2p);
77              }),
78              boost::make_function_output_iterator([](pt const& val) {
79                  std::cout  << "{x1,y,x2} = {" << val.get<0>() << ", " <<
                    ↪  val.get<1>() << ", " << val.get<2>() << "}" << std::endl;
80              })
81          );
82      }
83
```

```
84      size_t boost_r_tree::report_count(const double x1,const double x2,const double y)
        ↪  {
85  #ifdef DEBUG
86          std::cout <<"\n{x1,x2,y} = {"<< x1 <<", " << x2 <<", " << y<<"}\n";
87  #endif
88
89          auto r = boost::make_iterator_range(bgi::qbegin(rtree, bgi::satisfies([&](pt
            ↪  const& v){
90              double x1p = v.get<0>();
91              double yp = v.get<1>();
92              double x2p = v.get<2>();
93              return x1 <= x1p && x1p <= x2 && yp <= y && (x2 <= x2p) ;
94          })
95          ), {});
96          return boost::distance(r);
97      }
98
99      void boost_r_tree::print() {
100     }
101
102 };
103
104 #endif
```

Основными методами данного класса являются `report`, `report_count` и `insert`, первый совершая запрос `rtee.query(...)` проверяет выполнение условий (4.1) после чего выводит в поток `std::cout` координаты точек, удовлетворяющих условиям (4.1). Второй аналогичным образом проверяет выполнения условий (4.1) после чего подсчитывает количество удовлетворяющих запросу точек. Именно этот метод будет активно использоваться дла расчета $\hat{\alpha}$ и $\hat{\beta}$. Метод `insert` как понятно из названия добавляет в дерево точку новую точку.

Перейдем непосредственно к основной функции решающей изначальную задачу (3.1).

Алгоритм 6: Вычисление критических границ с использованием структуры R-дерева

```
1  #include <iostream>
2  #define _USE_MATH_DEFINES
3  #include <cmath>
4  #include <cfloat>
5  #include <random>
6
7  #include "settings.h"
8  constexpr size_t nMax = 20;
9
```

```
10 #include "boost_r_tree.hpp"
11
12 #ifdef test_speed_memory
13         #include <numeric>
14         #include <vector>
15         #include "utility/memory.h"
16         #include <tbb/tick_count.h>
17 #endif
18
19 using brt = internal::boost_r_tree;
20
21 std::random_device rd{};
22 std::mt19937 gen{rd()};
23
24 #ifdef DEBUG_COUNT
25 std::vector<std::pair<double, double>> points = {}; // для проверки
26 #endif
27
28 inline double fA(const double x)
29 {
30     // более быстрое упрощение специально посчитанное в wolfram
31     return  (M_PI*x/sqrt(3.0)
32         -2 * log(1 + pow(M_E, M_PI*x/ sqrt(3.0)))
33         + 1.0/2.0 * (x*x + log(2.0*M_PI*M_PI*M_PI/3.0))
34     );
35 }
36 inline double fB(const double x)
37 {
38     // более быстрое упрощение специально посчитанное в wolfram
39     return (log(
40             -sqrt( (2.0/ 3.0))* (-1.0 + x)  *x\
41             * pow(M_E,((6.0*  atanh(1.0 - 2.0 *x)*atanh(1.0 - 2.0 *x)
                ↪  )/(M_PI*M_PI)))\
42             *sqrt(M_PI*M_PI*M_PI))
43     );
44 }
45
46 void funA(const double lB, const double lA, brt& rtree)
47 {
48     std::normal_distribution<> d{0, 1};
49     gen.seed(rd());
50     double Z;
51     // для добавления узлов в дерево
52     double olB, olA, max_lA, min_lB, prev_min_lB;
53
54     for (size_t i = 0; i < N; ++i)
55     {
```

```
56          olB = DBL_MIN;
57          olA = DBL_MAX;
58          max_lA = 0.0;
59          min_lB = 0.0;
60          prev_min_lB = 0.0;
61          Z = 0;
62          while (lB < Z && Z < lA){
63              Z += fA(  d(gen));
64              if(max_lA < Z){
65                  max_lA = Z; olB = min_lB;
66              }
67              if(min_lB > Z){
68                  min_lB = Z; olA = max_lA;
69              }
70              if(min_lB < olB){
71                  olB = DBL_MIN;
72              }
73
74              if(max_lA > olA){
75                  rtree.insert(point(min_lB,olA,prev_min_lB));
76                  prev_min_lB = min_lB;
77                  olA = DBL_MAX;
78              }
79          }
80          if (Z < lA){
81              rtree.insert(point(lB,max_lA,prev_min_lB));
82          }
83      }
84  }
85
86  void  funB(const double lB, const double lA,  brt& rtree)
87  {
88      std::uniform_real_distribution<> d{0, 1};
89      gen.seed(rd());
90      double Z;
91      // для добавления узлов в дерево
92      double olB, olA, max_lA, min_lB, prev_min_lB;
93
94      for (size_t i = 0; i < N; ++i)
95      {
96          Z = 0;
97          olB = DBL_MIN;
98          olA = DBL_MAX;
99          max_lA = 0.0;
100         min_lB = 0.0;
101         prev_min_lB = 0.0;
102         while (lB < Z && Z < lA) {
```

```
            Z += fB(d(gen));
            if(max_lA < Z){
                max_lA = Z; olB = min_lB;
            }
            if(min_lB > Z){
                min_lB = Z; olA = max_lA;
            }
            if(min_lB < olB){
                olB = DBL_MIN;
            }
            if(max_lA > olA){
                rtree.insert(point(min_lB,olA,prev_min_lB));
                prev_min_lB = min_lB;
                olA = DBL_MAX;
            }
        }
        if (Z < lA){
            rtree.insert(point(lB,max_lA,prev_min_lB));
        }
    }
}

int main(void)
{
#ifdef test_speed_memory
    std::vector<double> timings(nMax);
        double timing_build_rTree_alpha;
    double timing_build_rTree_beta;
    tbb::tick_count start;
    tbb::tick_count stop;
        double all_time;
           flushDataCache();
#endif
    constexpr double lb = log(bj / (1 - aj));
    constexpr double la = log((1 - bj) / (aj));
    double blA = 0;
    double tlA = la;
    double llB = 0;
    double rlB = lb;
#ifdef DEBUG_COUNT
    points.emplace_back(lb, la);
#endif
    double lA, lB, a, b;

    double Da = eps;
    double Db = eps;

```

```
150     size_t i = 0;
151
152 #ifdef Debug
153     std::cout << "N = " << N << "; eps = " << eps << "; lb = " << lb << "; la = " <<
        ↪  la << std::endl;
154     std::cout << "Start blA = " << blA << "; tlA = " << tlA << "; llB = " << llB <<
        ↪  "; rlB = "<< rlB << std::endl;
155     std::cout << "Da = " << Da << "; Db = " << Db << "; |Da|+|DB| = " <<
        ↪  fabs(Da)+fabs(Db) << std::endl;
156 #endif
157
158 #ifdef test_speed_memory
159     start = tbb::tick_count::now();
160     double memory_rTree_alpha = getCurrentMemorySize();
161 #endif
162     brt rA(1024);
163     funA(lb,la,rA); /// Строим дерево для alpha
164 #ifdef test_speed_memory
165     stop = tbb::tick_count::now();
166         timing_build_rTree_alpha = (stop-start).seconds() * 1000.0 * 1000.0;
167     memory_rTree_alpha = getCurrentMemorySize() - memory_rTree_alpha;
168  #endif
169  #ifndef wolfram_info
170     std::cout << "Build rTree Alpha time [us 10^-6]: " << timing_build_rTree_alpha <<
        ↪  std::endl;
171     std::cout << "Memory rTree Alpha [mb]: " << memory_rTree_alpha/1024 << std::endl;
172     std::cout << "Size rTree Alpha: " << rA.size() << std::endl;
173     std::cout << "Buffer rTree Alpha: " << rA.get_buffer_size() << std::endl;
174 #endif
175 #ifdef test_speed_memory
176     start = tbb::tick_count::now();
177     double memory_rTree_beta = getCurrentMemorySize();
178 #endif
179     brt rB(1024);
180     funB(lb,la,rB); /// Строим дерево для beta
181 #ifdef test_speed_memory
182     stop = tbb::tick_count::now();
183         timing_build_rTree_beta = (stop-start).seconds() * 1000.0 * 1000.0;
184     memory_rTree_beta = getCurrentMemorySize()-memory_rTree_beta;
185 #endif
186
187 #ifndef wolfram_info
188     std::cout << "Build rTree Beta time [us 10^-6]: " << timing_build_rTree_beta <<
        ↪  std::endl;
189     std::cout << "Memory rTree Beta [mb]: " << memory_rTree_beta/1024.0 << std::endl;
190     std::cout << "Size rTree Beta: " << rB.size() << std::endl;
191     std::cout << "Buffer rTree Beta: " << rB.get_buffer_size() << std::endl;
```

```
192 #endif
193
194     while(fabs(Da)+fabs(Db) > eps && i<nMax){
195 #ifdef test_speed_memory
196                 tbb::tick_count start_while = tbb::tick_count::now();
197 #endif
198         lB = (llB + rlB)/2.0;
199         lA = (blA + tlA)/2.0;
200 #ifdef DEBUG_COUNT
201         points.emplace_back(lB, lA);
202 #endif
203
204         a = ((double)(N - rA.report_count(lb,lB,lA))/(double)N);
205         b =  ((double)(rB.report_count(lb,lB,lA))/(double)N);
206         Da = a - aj; Db = b - bj;
207         if(Db > 0)
208             {llB = lB;}
209         else
210             {rlB = lB;}
211         if(Da > 0)
212             {blA = lA;}
213         else
214             {tlA = lA;}
215         #ifdef test_speed_memory
216                 tbb::tick_count stop_while = tbb::tick_count::now();
217                 timings[i] = (stop_while-start_while).seconds() * 1000.0 * 1000.0;
218         #endif
219 #ifdef Debug
220         std::cout << i <<") blA = " << blA << "; tlA = " << tlA << "; llB = " << llB
        ↪  << "; rlB = "<< rlB << std::endl;
221         std::cout << "a = " << a << "; b = " << b << "; lB = " << lB << "; lA = " <<
        ↪  lA << "\n" <<
222             "Da = " << Da << "; Db = " << Db << "; |Da|+|DB| = " << fabs(Da)+fabs(Db)
                ↪  << std::endl;
223 #endif
224         i++;
225     }
226
227 #ifdef Debug
228     std::cout << i <<") blA = " << blA << "; tlA = " << tlA << "; llB = " << llB <<
    ↪  "; rlB = "<< rlB << std::endl;
229     std::cout << "Da = " << Da << "; Db = " << Db << "; |Da|+|DB| = " <<
    ↪  fabs(Da)+fabs(Db) << std::endl;
230 #endif
231 #ifdef DEBUG_COUNT
232     // вывести все пары
233     std::cout<<"{";
```

```
234     for (auto p: points) {
235         std::cout << "{" << p.first << ", " << p.second << "}, ";
236     }
237         std::cout<<"}" << std::endl;
238 #endif
239 
240 #ifdef test_speed_memory
241                 all_time = accumulate(timings.begin(), timings.end(), 0);
242         all_time += timing_build_rTree_alpha;
243         all_time += timing_build_rTree_beta;
244 #endif
245 #ifdef wolfram_info
246     std::cout<< std::setprecision(16)<< "{" << N << ", " <<  aj << ", " << bj << ", "
    ↪  << eps << ", " << all_time << ", {";
247     std::vector<double>::const_iterator j = timings.begin();
248     for ( ;j != std::prev(timings.end()); ++j){
249         std::cout<< std::setprecision(16) << *j << ", ";
250     }
251     std::cout<< std::setprecision(16) << *j << "}, " << a << ", " << b << ", " << lB
    ↪  << ", " << lA << ", " << Db << ", " << Da << ", " <<  fabs(Da)+fabs(Db) << ",
    ↪  " << i<< ", " << rA.size() << ", " << rB.size()<< ", " <<
    ↪  memory_rTree_alpha/1024.0 << ", " << memory_rTree_beta/1024.0 << ", " <<
    ↪  timing_build_rTree_alpha << ", " << timing_build_rTree_beta <<   "}" <<
    ↪  std::endl;
252 #elif defined(test_speed_memory)
253                 std::cout << "timings [us 10^-6]: \n{";
254                 for (auto const &j: timings) {
255                         std::cout << j << ", ";
256                 }
257                 std::cout << "}" << std::endl;
258                 std::cout << "All time [us 10^-6]: " << all_time << std::endl;
259 #endif
260 
261 #ifndef wolfram_info
262     std::cout << "N = " << N << "; lb = " << lb << "; la = " << la << "\naj = " << aj
    ↪  << "; bj = " << bj << "; eps = " << eps << std::endl;
263         std::cout << "lB = " << lB << "; lA = " << lA << ";\na = " << a << "; b = "
        ↪  << b << ";\nDa = " << Da << "; Db = " << Db << "; |Da|+|DB| = " <<
        ↪  fabs(Da)+fabs(Db) << std::endl;
264 #endif
265 
266     return 0;
267 }
```

Для дополнительной скорости мы отдельно посчитали и упростили отношение

$$\ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right)$$

с учетом того, как из равномерно непрерывного распределения на $(0; 1)$ получить требуемое логистическое из (3.1).

Основным отличием от предыдущей итерационной программы (17) помимо большего числа выводимой информации, является процесс добавления точек в функциях `funA`, `funB`. Для этого дополнительно придется сохранять предыдущие значения lB и lA – `olB` и `olA` соответственно, а также дабы сохранять только значимые точки хранить `max_lA`, `min_lB`, `prev_min_lB`. Также отметим, что изменилось условие `if` после цикла `while` в `funA`, это связано с тем, что раньше мы сразу подсчитывали количество принятых гипотез H_1 в `funA`, а сейчас мы лишь добавляем точки в структуру, и только после вычисления разницы N и результата запроса к дереву, мы узнаем требуемое число. Таким образом, функции `funA` и `funB` отличаются лишь генерируемой последовательностью и вызовом либо `fA` либо `fB`, однако для единообразия с предыдущем кодом мы оставим как две отдельные функции.

В функции `main` мы предварительно строим R-деревья для вычисления $a = \hat{\alpha}$ и $b \equiv \hat{\beta}$, после чего уже известным итерационным методом выполняем поиск точки удовлетворяющей требуемой точности.

5.1 Реализация программы с раздельным построением нескольких поддеревьев

Прежде чем переходить к результатам вычислений отметим, что данная программа довольно легко распараллеливается, поскольку построение R_α – R-дерево для дальнейшего вычисления a ни как не связано с процедурой построения R_β – R-дерево для вычисления b. Более того, сами деревья R_α и R_β можно разбить на несколько поддеревьев R_{α_1}, R_{α_2}, R_{α_3}, ... R_{β_1}, R_{β_2}, R_{β_3}, ..., после чего для вычисления $a = \hat{\alpha}$ необходимо будет выполнить запрос к каждому поддереву и сложить получившийся результат. Аналогичным образом осуществляется процесс вычисления $b = \hat{\beta}$.

Мы реализуем раздельное построение простейшим образом, а именно при помощи `std::thread` создадим три исполняемых потока для R_{α_1}, R_{α_2}, R_{α_3} и три исполняемых потока для R_{β_1}, R_{β_2}, R_{β_3}, поскольку у нас на вычислительном устройстве 6 ядер (18). Процесс управления этими исполняющими потоками мы оставим на операционную систему. Так как время выполнения запросов ничтожно мало относительно времени построения мы для простаты выполним запросы к поддеревьям последовательно в основном потоке. Также нам необходимо для каждого исполняющего потока выделить собственный генератор случайных чисел, поскольку каждый исполняющий поток модифицирует состояние генератора.

6. Результаты численного моделирования

Сперва мы сравним время вычисления критических границ различными алгоритмами и для некоторых занимаемую ими память. После чего продемонстрируем число требуемых испытаний в последовательном критерии Вальда при различных критических границах.

В качестве примеров проверяемых гипотез, мы рассмотрим уже ранее упомянутые (3.1), (3.4), а также, рассматриваемые в работе [4]:

$$H_0 : f_0(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-1)^2}; \quad H_1 : f_1(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-2)^2}; \tag{6.1}$$

то есть основная гипотеза о том, что наблюдаемая случайная величина имеет нормальное распределение со средним 1 и среднеквадратическим отклонением $\sigma = 1$, против конкурирующей гипотезы о нормальном распределение со средним 2 и среднеквадратическим отклонением $\sigma = 1$. Тогда

$$z_i = \ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right) = x - \frac{3}{2} \tag{6.2}$$

Последним мы рассмотрим

$$H_0 : f_0(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad H_1 : f_1(x) = \frac{1}{\pi(x^2+1)} \tag{6.3}$$

то есть основная гипотеза о том, что наблюдаемая случайная величина имеет стандартное нормальное распределение, против конкурирующей гипотезы о распределение Коши с параметрами 0, 1. Тогда

$$z_i = \ln\left(\frac{f_1(x_i)}{f_0(x_i)}\right) = \frac{1}{2}\left(x^2 - 2\ln\left(x^2+1\right) + \ln\left(\frac{2}{\pi}\right)\right)$$

Критические границы мы будем рассчитывать при $\alpha = 0.01$, $\beta = 0.01$ и следующем количестве итераций моделирования:

N	9604	19699	37843	60023	103671	414683	1658725	6634897	26539587
ε	0.01	0.01	0.01	0.004	0.004	0.002	0.001	0.005	0.0025

где точность ε определяется согласно таблице (6.1), так чтобы достоверность Q была не меньше 0.95. Характеристики вычислительного устройства приведены в приложение (18).

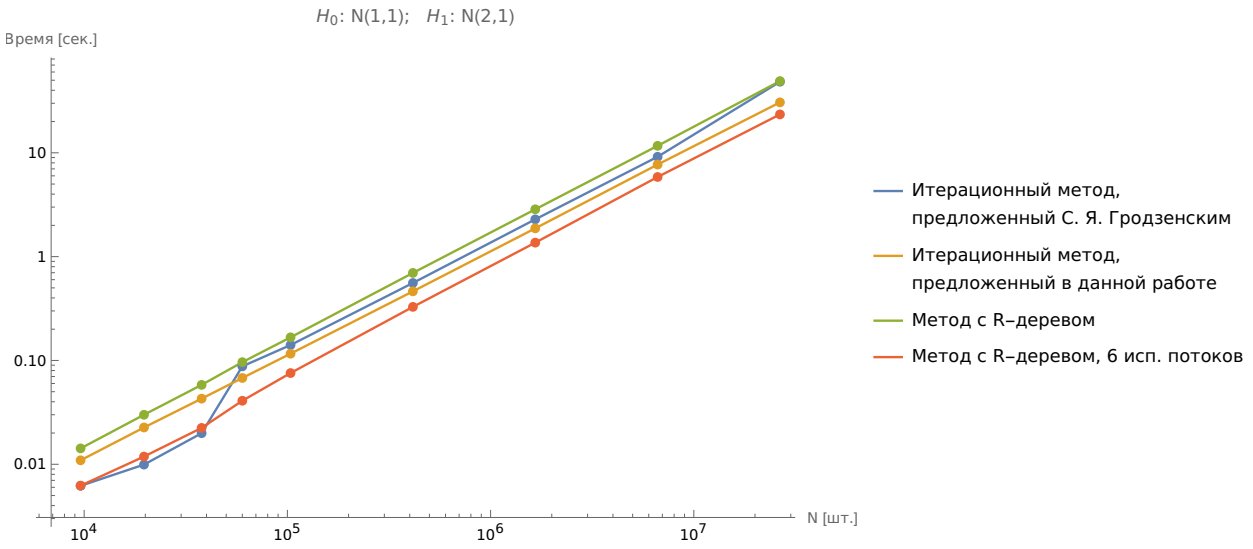


Рис. 6.1: Зависимость времени достижения заданной точности от количества испытаний

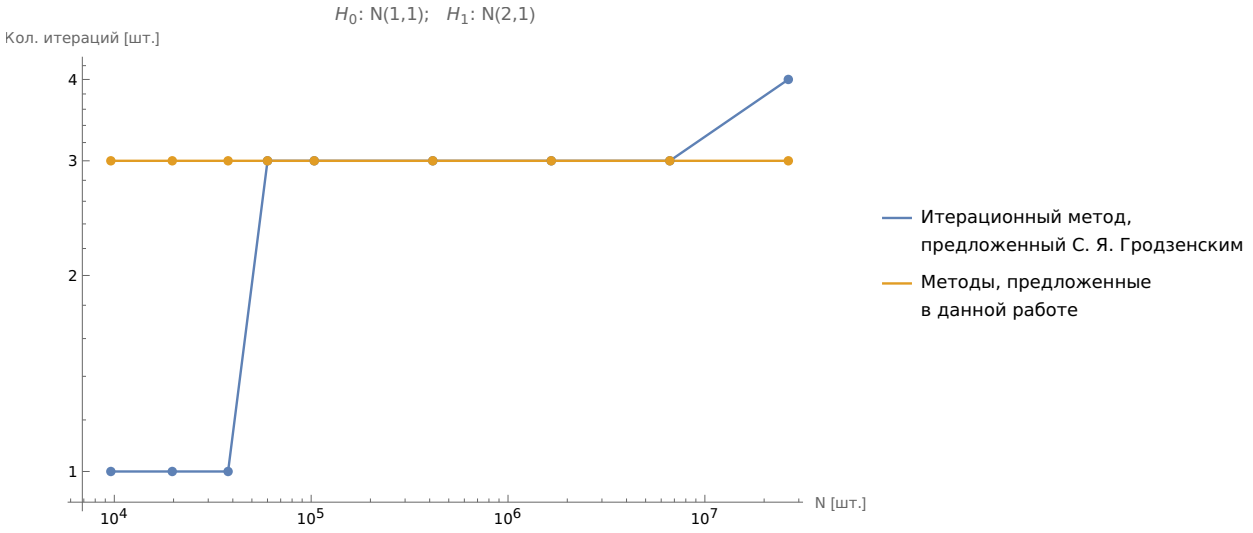


Рис. 6.2: Зависимость количества итераций от количества испытаний

Из графика (6.1) мы можем видеть, что итерационный метод, предложенный С. Я. Гродзенским (далее $M1$) до $N = 37843$ работает быстрее, однако если обратиться к подробной таблице (6.3), то видим, что метод проводит численное моделирование при критических границах, предложенные А. Вальдом и сразу же выходит, поскольку достигается, заданная точность. При больших N методу $M1$ как и остальным требуется, совершить более три и более итераций. Для проверки гипотез (6.1) в единственной точке $(\alpha, \beta) = (0.01, 0.01)$ слишком затратно использовать структуру данных (метод с R-деревом, далее $M3$), поскольку проведение моделирования методом Монте-Карло в силу простаты формулы (6.2) осуществляется быстрее.

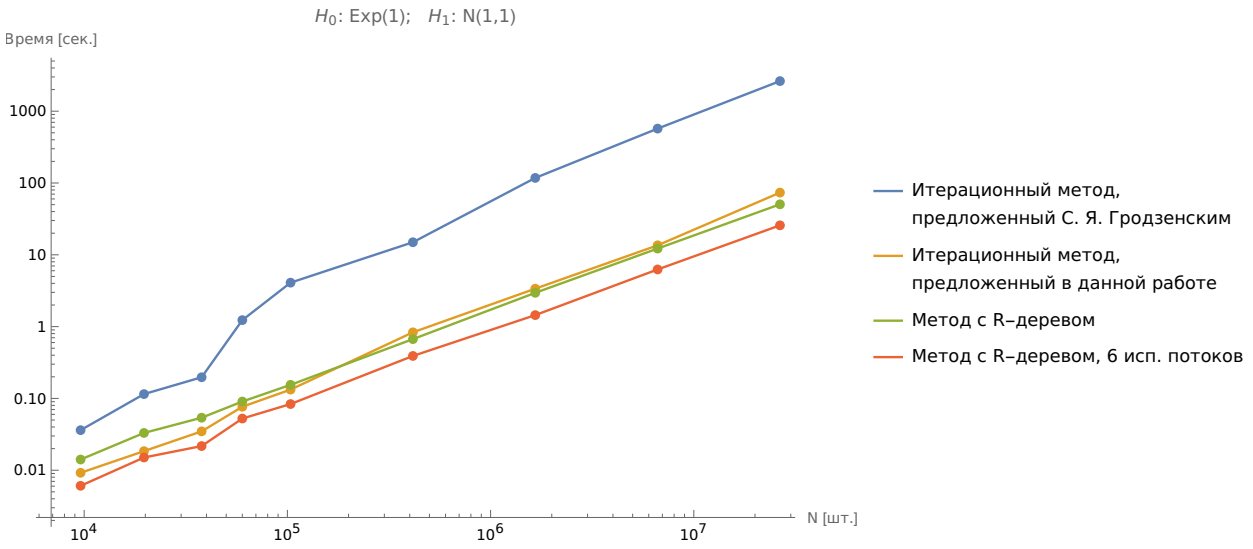


Рис. 6.3: Зависимость времени достижения заданной точности от количества испытаний

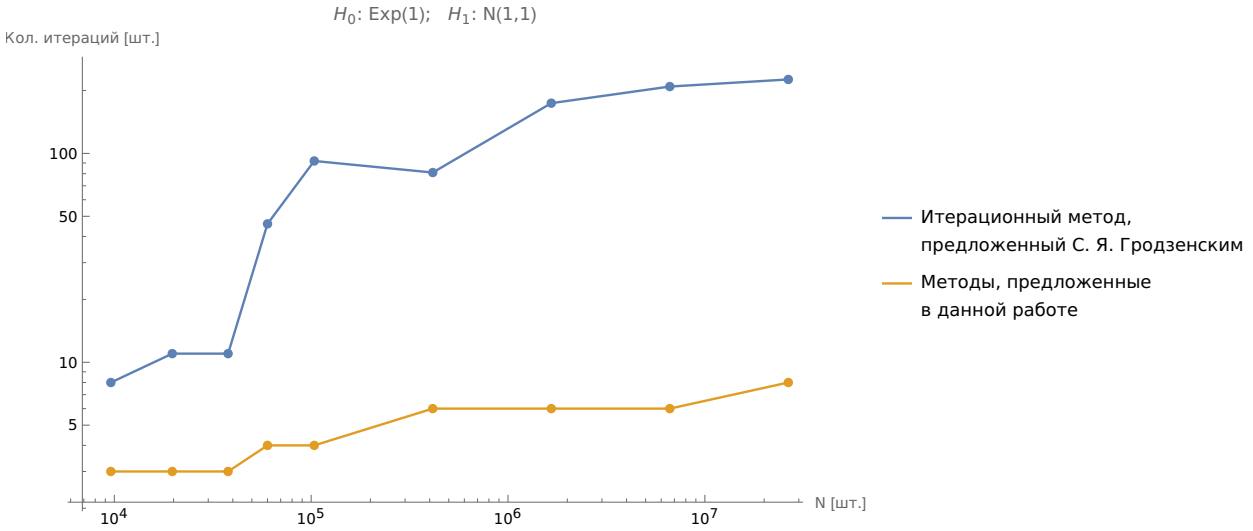


Рис. 6.4: Зависимость количества итераций от количества испытаний

Из графика (6.3) мы можем видеть, что поскольку для корректной работы алгоритмы $M1$ необходимо взять Δ_C довольно маленькое, то методы приведенные в данной работе работают быстрее. При N, начиная с $N > 10^5$ метод $M3$ работает быстрее чем итерационный метод (далее $M2$), даже для проверки в единственной точке. Более детальная информация приведена в таблице (6.4).

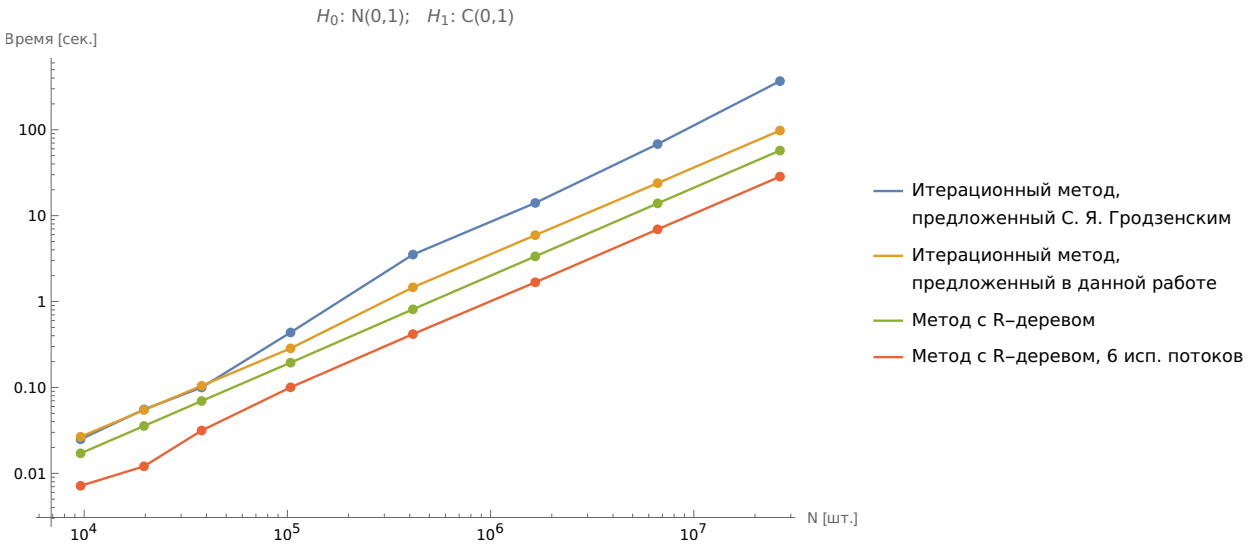


Рис. 6.5: Зависимость времени достижения заданной точности от количества испытаний

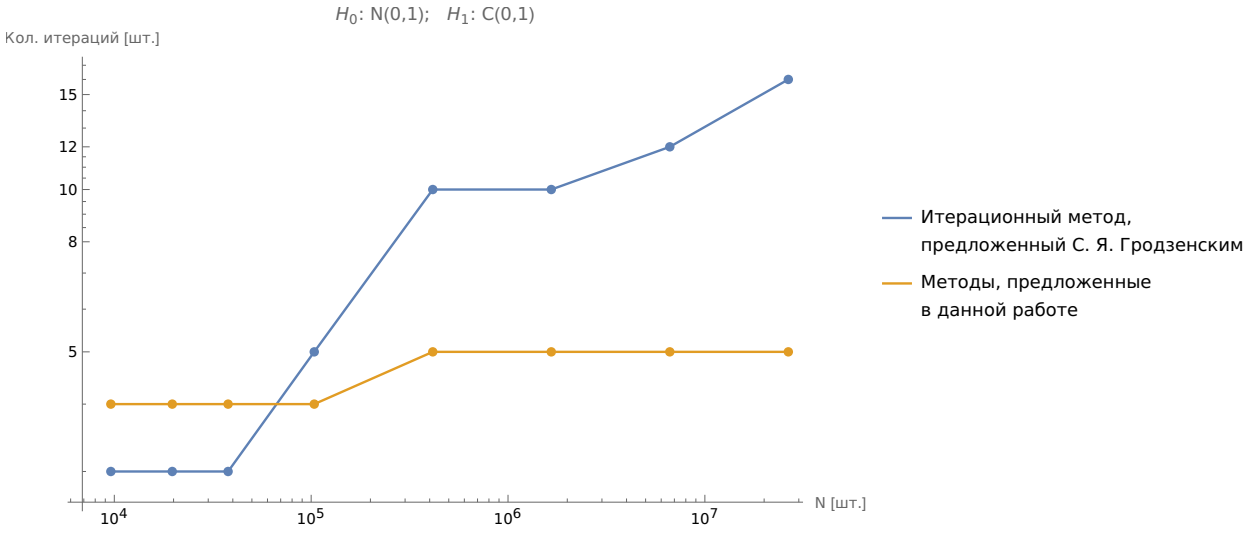


Рис. 6.6: Зависимость количества итераций от количества испытаний

При проведения данного испытания метод $M1$ при $\Delta_C = 1$ также выдавал некорректные результаты, поэтому было принято решение положить $\Delta_C = 1/5$. На графике (6.5) мы наблюдаем стабильное превосходство по скорости алгоритма $M3$ над $M2$ и $M1$. Более детальная информация приведена в таблице (6.5).

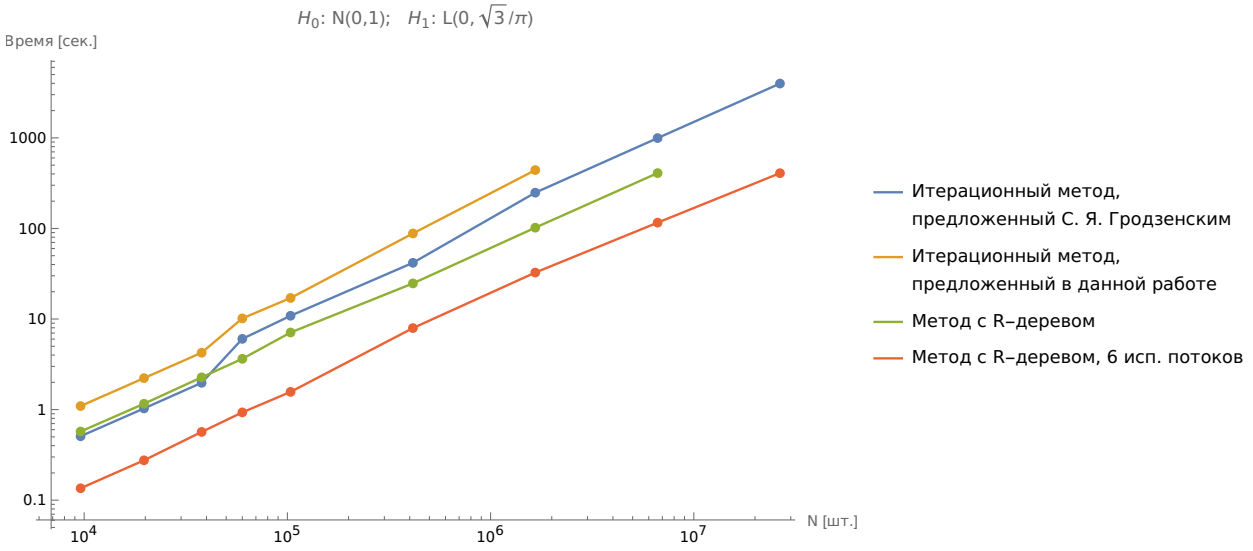


Рис. 6.7: Зависимость времени достижения заданной точности от количества испытаний

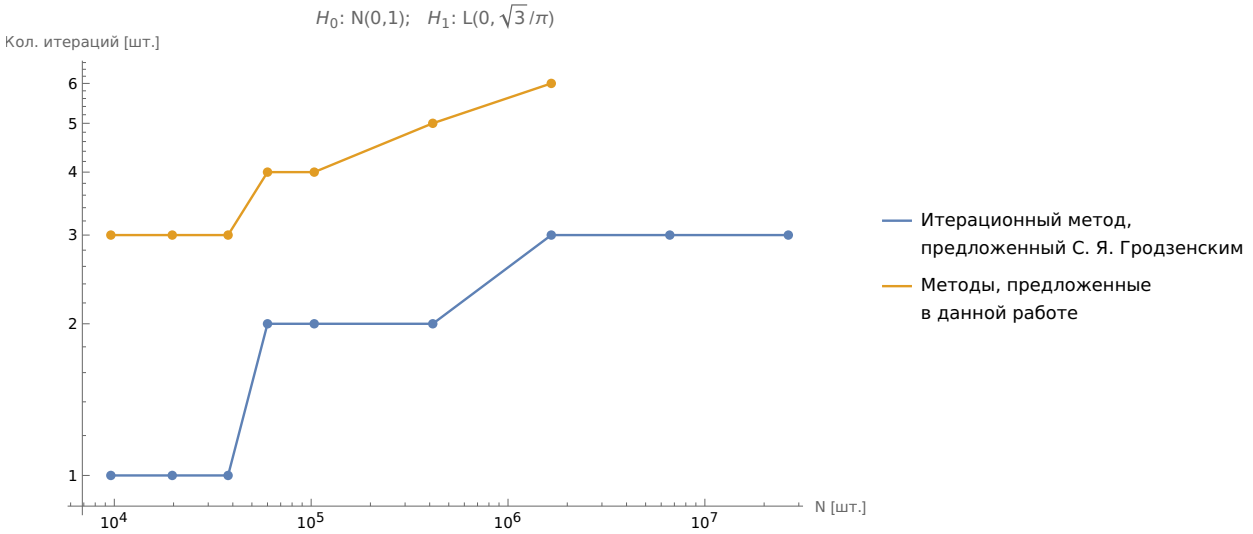


Рис. 6.8: Зависимость количества итераций от количества испытаний

Из графиков (6.7), (6.7) видно, что в данном случае стратегия выбора следующей точки в методе $M1$ лучше, поскольку количество необходимых итераций меньше. Также мы видим, что метод $M3$, начиная с $N = 40000$ быстрее чем итерационные, а при меньших сопоставим с $M1$ и быстрее $M2$. Подробная информация приведена в таблице (6.6).

Также отметим, что количество используемой памяти во всех четырёх испытаниях примерно одинаково. Она линейным образом зависит от N и всегда на дерево R_α необходимо больше чем на дерево R_β. Приведен график демонстрирующий это для последнего испытания:

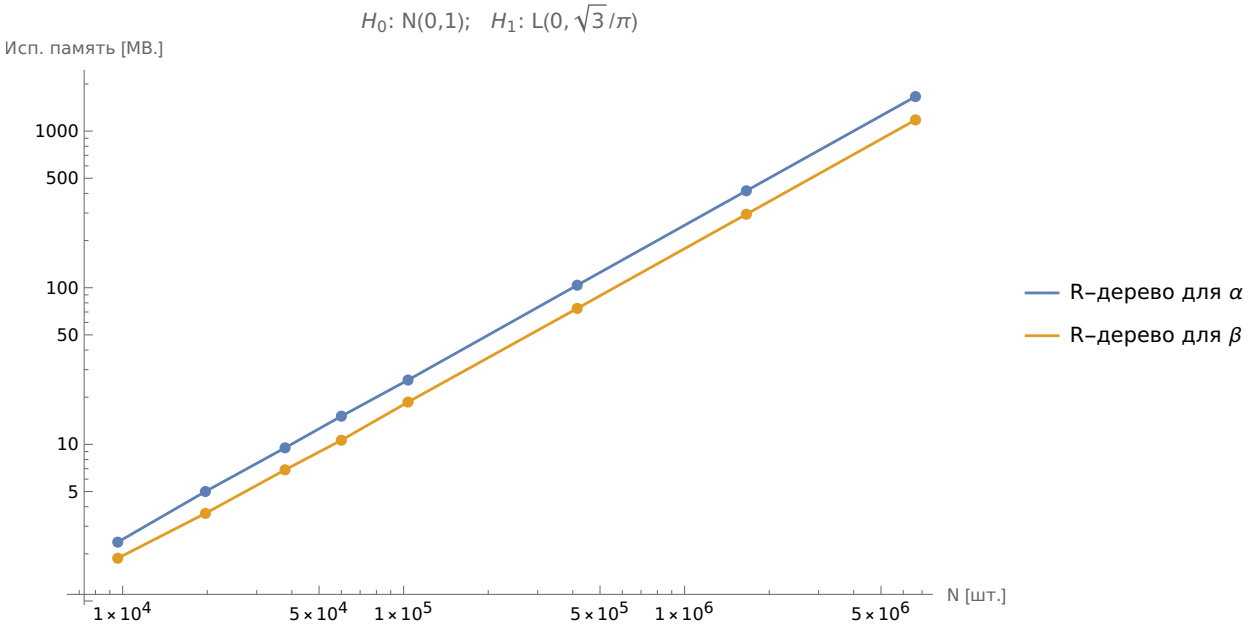


Рис. 6.9: Зависимость используемой памяти от количества испытаний

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе мы изучили теоретические аспекты вычисления критических границ в последовательном критерии Вальда, рассмотрели и проанализировали предложенный ранее С. Н. Постоваловым и С. Я. Гродзенским подходы к численному моделированию границ. Привели пример, показывающий необходимость в ряде случаев дополнительных изменений в итерационном методе С. Я. Гродзенского. Разработали иной итерационный алгоритм без необходимости дополнительных настроек и реализовали его в виде программы на языке C++.

Помимо итерационного метода в данной работе представлен алгоритм с использованием структуры данных, эффективность которого в ходе проведения тестов оказалась выше ранее предложенных методов. Его реализация также представлена в работе.

Таким образом, были выполнены все поставленные задачи и достигнута цель данной работы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Боровков А. А. Теория вероятностей: Учеб. пособие для вузов. – 2-е изд., перераб. и доп. – М.З Наука. Гл. ред. физ.-мат. лит. 1986. – 432 с.

[2] Вальд А. Последовательный анализ. – Гос. изд-во физико-математической лит-ры, 1960.

[3] Вентцель Е. С. Исследование операций. М., Советское радио, 1972. 552 с.

[4] Гродзенский С. Я., Чесалин А. Н. Уточнение границ последовательных статистических критериев с помощью компьютерного моделирования //Метрология. – 2019. – №. 3. – С. 30-45.

[5] Леман Э. «Проверка статистических гипотез» Издательство: Главная редакция физико-математической литературы издательства. – 1979.

[6] Лемешко Б. Ю., Лемешко С. Б., Постовалов С. Н. Сравнительный анализ мощности критериев согласия при близких конкурирующих гипотезах. I. Проверка простых гипотез //Сибирский журнал индустриальной математики. – 2008. – Т. 11. – №. 2. – С. 96-111.

[7] Постовалов С. Н. Проверка простых и сложных гипотез с использованием последовательного критерия Вальда //Доклады Академии наук высшей школы Российской Федерации. – 2011. – №. 2. – С. 140-150.

[8] Чеонг О. [и др.]. Вычислительная геометрия. Алгоритмы и приложения / О. Чеонг, М. де Берг, М. ван Кревельд, М. Овермарс, Litres, 2022. 440 с.

[9] Bechhofer R. A note on the limiting relative efficiency of the Wald sequential probability ratio test //Journal of the American Statistical Association. – 1960. – Т. 55. – №. 292. – С. 660-663.

[10] Blelloch G. E. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting //SODA. – 2008. – Т. 8. – С. 894-903.

[11] Brass P. Advanced Data Structures / P. Brass, Cambridge University Press, 2019. 472 с.

[12] Chazelle B. A Functional Approach to Data Structures and Its Use in Multidimensional Searching // SIAM Journal on Computing. 1988. № 3 (17). С. 427–462.

[13] Gabow H. N., Bentley J. L., Tarjan R. E. Scaling and related techniques for geometry problems Not Known: ACM Press, 1984.С. 135–143.

[14] Gabrielsen P. Three-sided Range Reporting in External Memory : дис. – Aarhus Universitet, Datalogisk Institut, 2016.

[15] Girshick M. A. Contributions to the theory of sequential analysis, II, III //The Annals of Mathematical Statistics. – 1946. – С. 282-298.

[16] Ishiyama K., Sadakane K. Orthogonal Range Search Data Structures под ред. N. Katoh [и др.]., Singapore: Springer Singapore, 2022.C. 121–147.

[17] JaJa J., Mortensen C. W., Shi Q. Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting Lecture Notes in Computer Science / под ред. R. Fleischer, G. Trippen, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.C. 558–568.

[18] Kejlberg-Rasmussen C. [и др.]. I/O-efficient planar range skyline and attrition priority queues New York New York USA: ACM, 2013.C. 103–114.

[19] Neyman J., Pearson E. S. IX. On the problem of the most efficient tests of statistical hypotheses //Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character. – 1933. – Т. 231. – №. 694-706. – С. 289-337.

[20] Rahul S., Gupta P., Rajan K. S. DATA STRUCTURES FOR RANGE-AGGREGATION OVER CATEGORIES // International Journal of Foundations of Computer Science. 2011. № 07 (22). С. 1707–1728.

[21] Wald A. Sequential Tests of Statistical Hypotheses //The Annals of Mathematical Statistics. – 1945. – Т. 16. – №. 2. – С. 117-186.

[22] Wald A., Wolfowitz J. Optimum character of the sequential probability ratio test //The Annals of Mathematical Statistics. – 1948. – С. 326-339.

ПРИЛОЖЕНИЕ А

Q	ε							
	0.1	0.01	0.004	0.002	0.001	0.0005	0.0004	0.00025
0.8	42	4106	25663	102649	410594	1642375	2566211	6569498
0.85	52	5181	32379	129516	518063	2072251	3237892	8289004
0.9	68	6764	42275	169097	676386	2705544	4227412	10822174
0.95	97	9604	60023	240092	960365	3841459	6002280	15365836
0.96	106	10545	65905	263618	1054472	4217885	6590445	16871539
0.97	118	11774	73583	294331	1177324	4709293	7358270	18837169
0.98	136	13530	84561	338244	1352974	5411895	8456086	21647578
0.99	166	16588	103671	414682	1658725	6634897	10367026	26539587
0.995	197	19699	123117	492465	1969860	7879439	12311623	31517755
0.999	271	27069	169181	676723	2706892	10827567	16918073	43310265
0.9995	303	30290	189308	757230	3028917	12115666	18930727	48462661
0.9999	379	37842	236512	946045	3784177	15136706	23651102	60546821

Таблица 6.1: Необходимое число итераций N для обеспечения точности ε при достоверности Q

ПРИЛОЖЕНИЕ Б

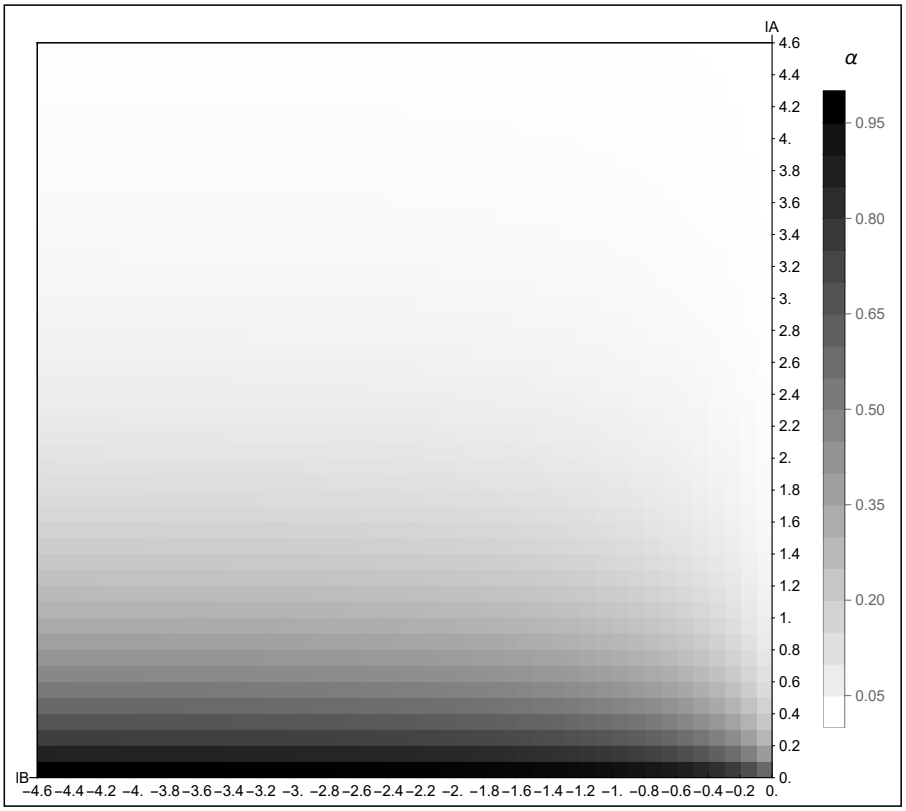
Алгоритм 7: моделирование зависимости $\beta\left(lA, lB\right)$

Input: $nMaxA$, $nMaxB$, N, $step$

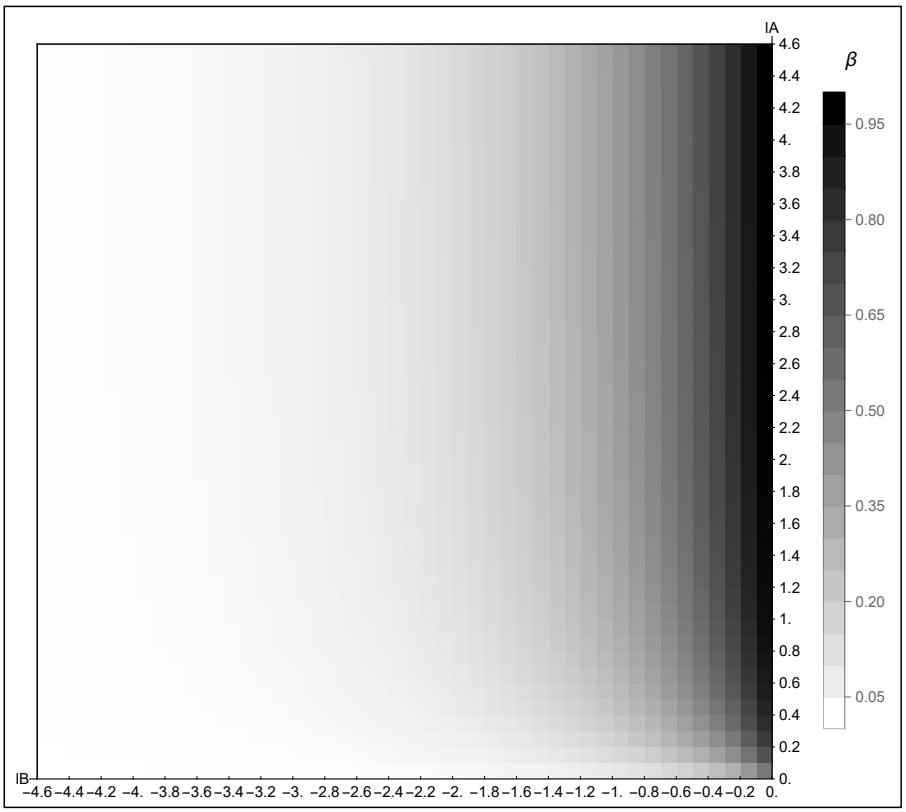
Output: $M\beta$

$M\beta \leftarrow Matrix[0, \{0, nMaxB\}, \{0, nMaxA\}]$
$lA \leftarrow 0$
for $i = 1 \ldots nMaxA$ **do**
    $lA \mathrel{+}= step$
    $lB \leftarrow 0$
    for $j = 1 \ldots nMaxB$ **do**
        $lB \mathrel{-}= step$
        for $k = 1 \ldots N$ **do**
            $Z \leftarrow 0$
            while $lB < Z$ **and** $Z < lA$ **do**
                $x \leftarrow$ RandomVariate[LogisticDistribution[0, Sqrt[3]/Pi]]
                $Z \mathrel{+}= \ln\left(\dfrac{f_1\left(x\right)}{f_0\left(x\right)}\right)$
            end while
            if $Z \leq lB$ **then**
                $M\beta[i][j] \mathrel{+}= 1$
        end for
    end for
end for
return $M\beta$

ПРИЛОЖЕНИЕ В



(a) Зависимость $\alpha\,(lA, lB)$, при $step = 0.1$, $N = 6 \cdot 10^6$



(b) Зависимость $\beta\,(lA, lB)$, при $step = 0.1$, $N = 6 \cdot 10^6$

Рис. 6.10: Зависимости $\alpha\,(lA, LB)$, $\beta\,(lA, lB)$

ПРИЛОЖЕНИЕ Г

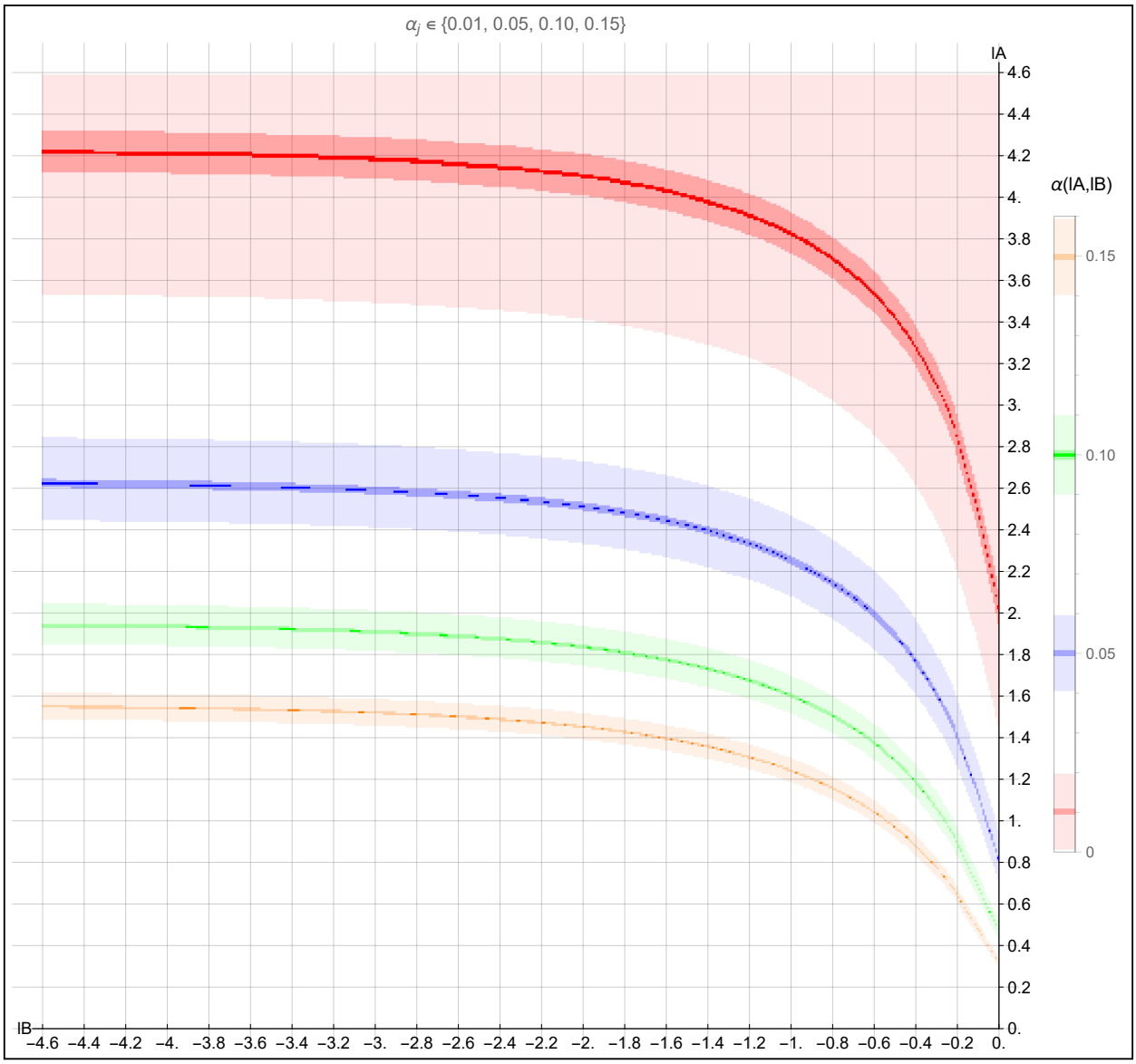


Рис. 6.11: Линии уровня для $a_j \in \{0.01, 0.05, 0.10, 0.15\}$ и $\varepsilon \in \{0.01, 0.001, 0.0001\}$ при $step = 0.01$, $N = 1.2 \cdot 10^6$

ПРИЛОЖЕНИЕ Д

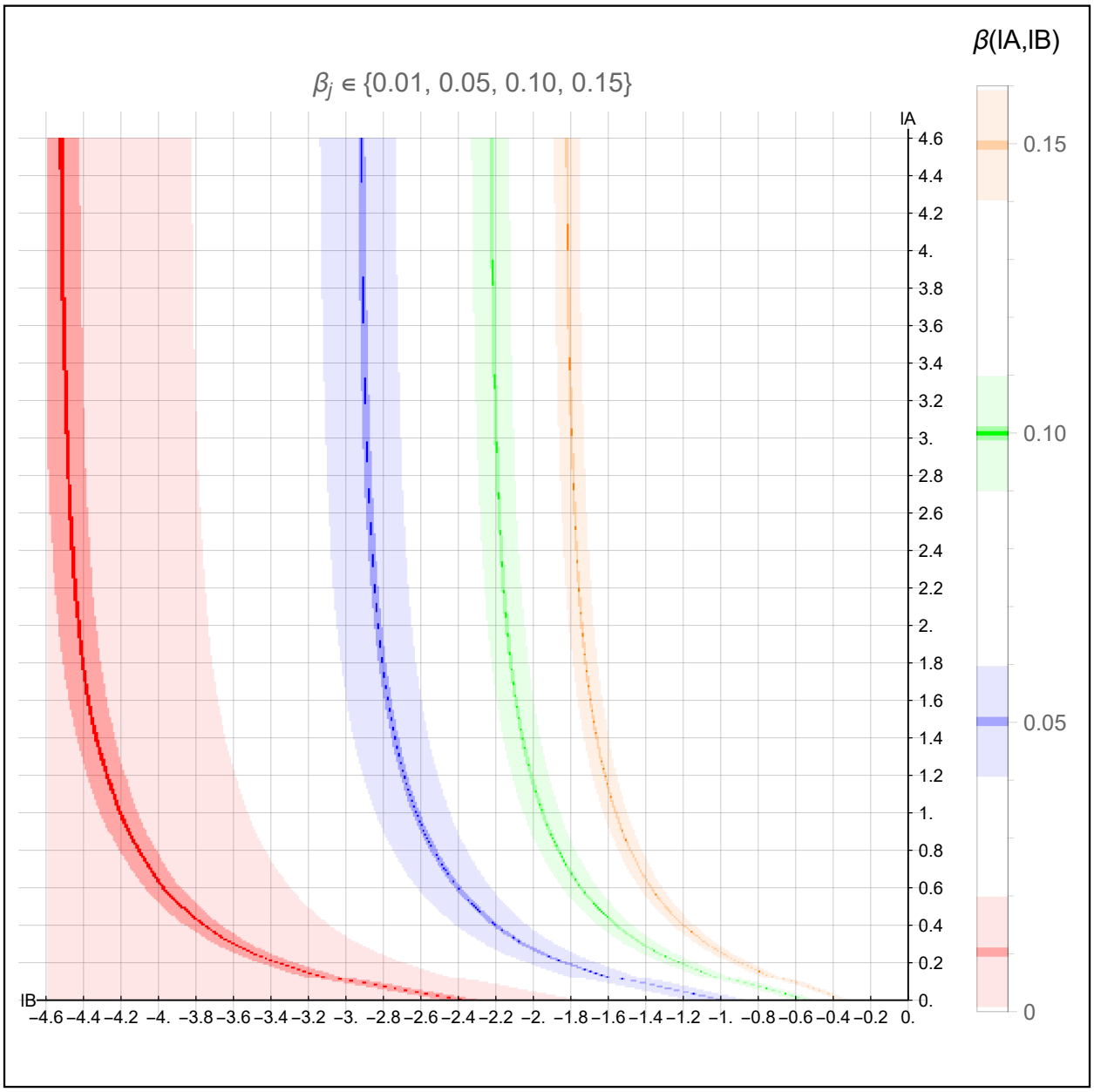
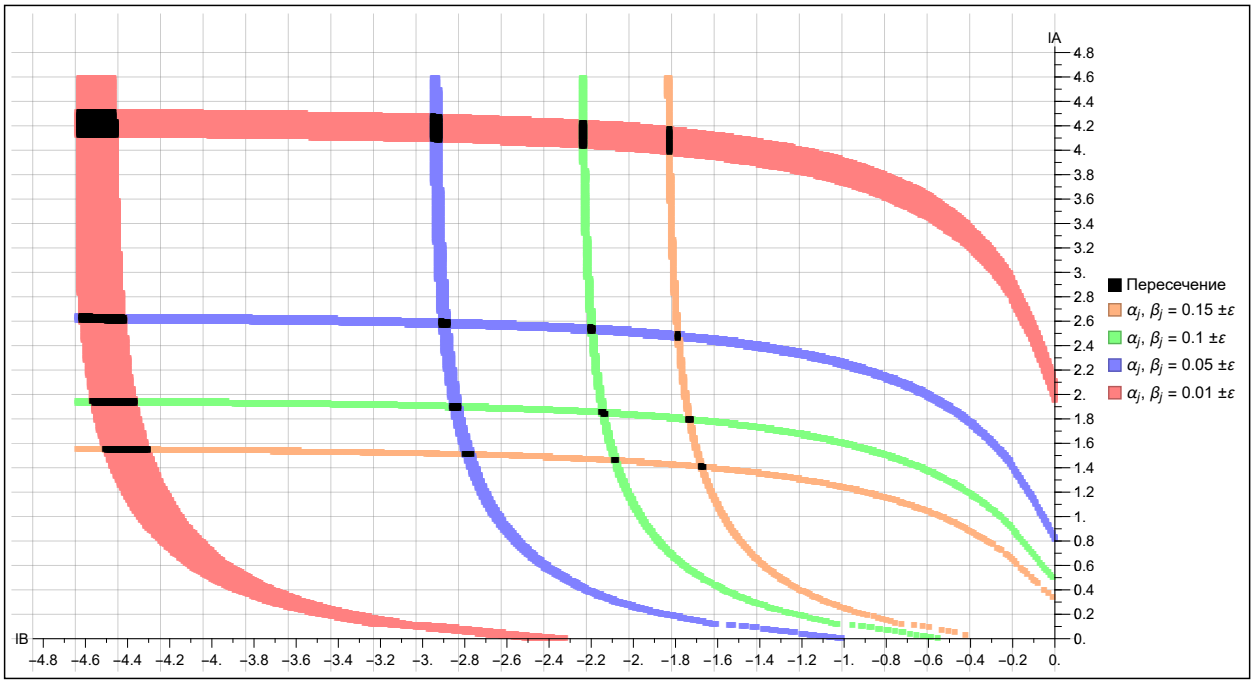
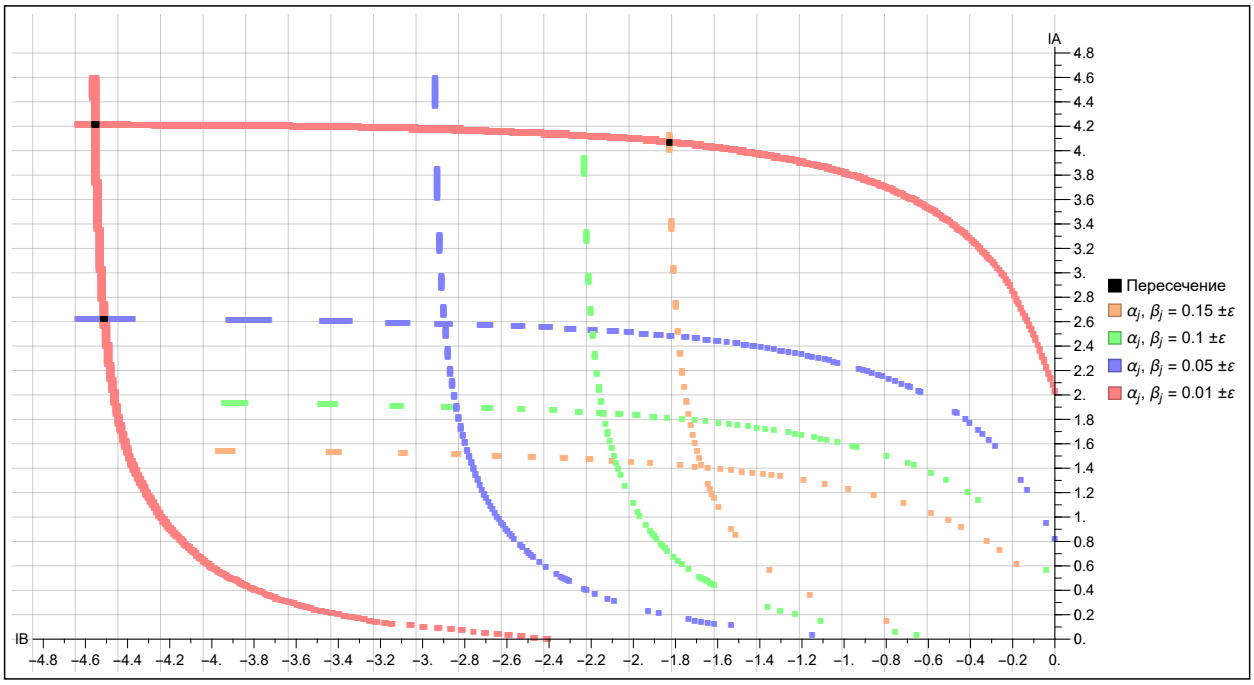


Рис. 6.12: Линии уровня для $b_j \in \{0.01, 0.05, 0.10, 0.15\}$ и $\varepsilon \in \{0.01, 0.001, 0.0001\}$ при $step = 0.01$, $N = 1.2 \cdot 10^6$

ПРИЛОЖЕНИЕ Е



(a) $\varepsilon = 0.001$



(b) $\varepsilon = 0.0001$

Рис. 6.13: Пересечение линий уровня для $a_j, b_j \in \{0.01, 0.05, 0.10, 0.15\}$ и $\varepsilon \in \{0.001, 0.0001\}$ при $step = 0.01$, $N = 1.2 \cdot 10^6$

ПРИЛОЖЕНИЕ Ж

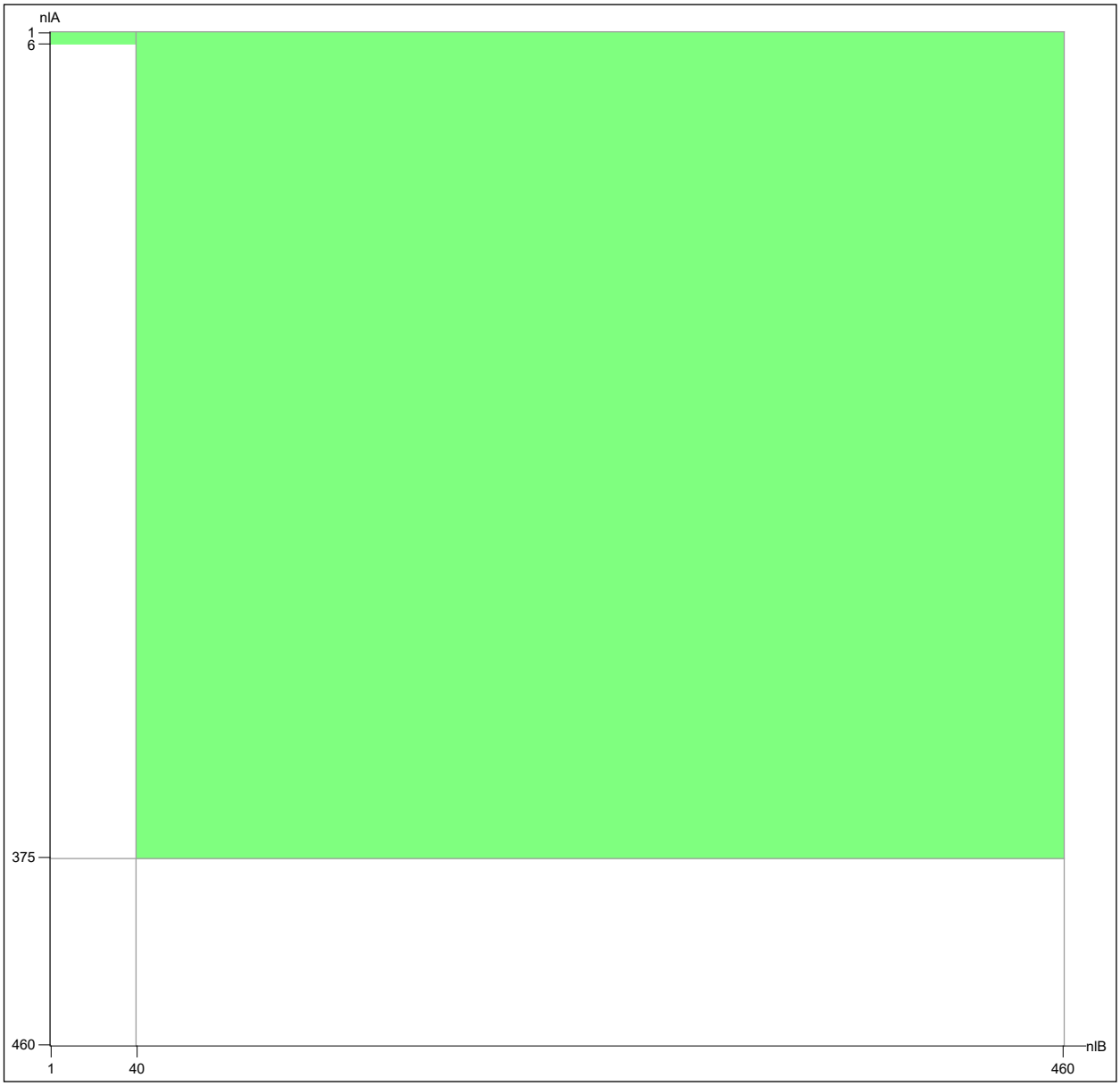


Рис. 6.14: Матрица V при $step = 0.01$

ПРИЛОЖЕНИЕ З

Алгоритм 8: Моделирование зависимости $\beta\left(lA, lB\right)$ версия №2

Input: N, $step$, lb, la, $nMaxlA$, $nmaxlA$

Output: $M\beta$

$M\alpha \leftarrow Matrix[0, \{0, nmaxlA\}, \{0, nMaxlA\}]$
for $i = 0 \ldots N-1$ **do**
  $Z \leftarrow 0$
  $V \leftarrow Matrix[0, \{0, nmaxlA\}, \{0, nMaxlA\}]$
  $maxlA = 0$; $nlB = 0$; $nlA = 0$; $j = 0$; $m = 0$;
  repeat
    if $Z < 0$ **then**
      $nlB = -IntegerPart[Z/step]$
      $nlA = IntegerPart[maxlA/step]$
      while $m < nlB$ **do**
        for $j = nlA \ldots nmaxlA$ **do**
          $V[j, m] = 1$
        end for
        $m \mathrel{+}= 1$
      end while
    else
      $maxlA = Max[maxlA, Z]$
    $x \leftarrow$ RandomVariate[LogisticDistribution[0, Sqrt[3]/Pi]]
    $Z \mathrel{+}= \ln\left(\dfrac{f_1\left(x\right)}{f_0\left(x\right)}\right)$
  until $Z > lb$ *and* $Z < la$
  if $Z \leq lb$ **then**
    $nlB = nMaxlB$
    $nlA = IntegerPart[maxlA/step]$
    while $m \leq nlB$ **do**
      for $j = nlA \ldots nmaxlA$ **do**
        $V[j, m] = 1$
      end for
      $m \mathrel{+}= 1$
    end while
  $M\alpha \mathrel{+}= V$
end for
return $M\beta$

ПРИЛОЖЕНИЕ И

Алгоритм 9: Алгоритм нахождения точки пересечения заданных линий уровня α_j, β_j с точностью ε

Input: $M\alpha$,$M\beta$ *step*, α_j, β_j, ε

Output: $\hat{\alpha}, \hat{\beta}$

$la_j \leftarrow \ln((1-\beta_j)/\alpha_j)$; $lb_j \leftarrow \ln(\beta_j/(1-\alpha_j))$

$R \leftarrow Ceiling[la_j/step]$; $C \leftarrow Ceiling[-lb_j/step]$

$br \leftarrow 1$; $tr \leftarrow R$

$lc \leftarrow 1$; $rc \leftarrow C$

$or = -1$; $oc = -1$

$r \leftarrow Ceiling[(br+tr)/2]$; $c = C$

$a \leftarrow M\alpha[r,c]$; $b \leftarrow M\beta[r,c]$

$\Delta_\alpha \leftarrow a - a_j$; $\Delta_\beta \leftarrow b - b_j$

if $\Delta_\alpha > 0$ **then**

    $br = r$

else

    $tr = r - 1$

while *(*$Abs[\Delta_\alpha] > \varepsilon$ || $Abs[\Delta_\beta] > \varepsilon$*) && (*$or \neq r$ || $oc \neq c$*)* **do**

    $or = r$; $oc = c$

    $tr = R$; $rc = C$

    while $Abs[\Delta_\beta] > \varepsilon$ *&&* $lc \neq rc$ **do**

        $c = Ceiling[(lc+rc)/2]$;

        $a = M\alpha[r,c]$; $b = M\beta[r,c]$

        $\Delta_\alpha = a - a_j$; $\Delta_\beta = b - b_j$

        if $\Delta_\beta > 0$ **then**

            $lc = c$

        else

            $rc = c - 1$

    end while

    if $Abs[\Delta_\beta] > \varepsilon$ *&&* $lc == rc$ **then**

        $a = M\alpha[r,lc]$; $b = M\beta[r,lc]$

        $\Delta_\alpha = a - a_j$; $\Delta_\beta = b - b_j$

    end if

    ↓

Алгоритм 10: Моделирование зависимости $\beta\,(lA, lB)$ версия №2

```
30:
31:     while Abs[Δ_α] > ε && br ≠ tr do
32:         r = Ceiling[(br + tr)/2];
33:         a = Mα[r, c]; b = Mβ[r, c]
34:         Δ_α = a − a_j; Δ_β = b − b_j
35:         if Δ_α > 0 then
36:             br = r
37:         else
38:             tr = r − 1
39:     end while
40:     if Abs[Δ_α] > ε && br == tr then
41:         a = Mα[tr, c]; b = Mβ[tr, c]
42:         Δ_α = a − a_j; Δ_β = b − b_j
43:     end if
44: end while
45: α̂ ← a; β̂ ← b
46: return α̂, β̂
```

ПРИЛОЖЕНИЕ К

Алгоритм 11: Алгоритм нахождения точки пересечения заданных линий уровня α_j, β_j с точностью ε

Input: $M\alpha$,$M\beta$ *step*, α_j, β_j, ε

Output: $\hat{\alpha}, \hat{\beta}$

1: $la_j \leftarrow \ln((1-\beta_j)/\alpha_j)$; $lb_j \leftarrow \ln(\beta_j/(1-\alpha_j))$
2: $R \leftarrow Ceiling[la_j/step]$; $C \leftarrow Ceiling[-lb_j/step]$
3: $br \leftarrow 1$; $tr \leftarrow R$
4: $lc \leftarrow 1$; $rc \leftarrow C$
5: $or = -1$; $oc = -1$
6: $r \leftarrow Ceiling[(br+tr)/2]$; $c = C$
7: $a \leftarrow M\alpha[r,c]$; $b \leftarrow M\beta[r,c]$
8: $\Delta_\alpha \leftarrow a - a_j$; $\Delta_\beta \leftarrow b - b_j$
9: **if** $\Delta_\alpha > 0$ **then**
10: $\quad br = r$
11: **else**
12: $\quad tr = r - 1$
13: **do** *(*$or \neq r$ || $oc \neq c$*)* **while**
14: $\quad or = r$; $oc = c$
15: $\quad tr = R$; $rc = C$
16: $\quad$ **while** $lc \neq rc$ **do**
17: $\quad\quad c = Ceiling[(lc+rc)/2]$;
18: $\quad\quad a = M\alpha[r,c]$; $b = M\beta[r,c]$
19: $\quad\quad \Delta_\alpha = a - a_j$; $\Delta_\beta = b - b_j$
20: $\quad\quad$ **if** $\Delta_\beta > 0$ **then**
21: $\quad\quad\quad lc = c$
22: $\quad\quad$ **else**
23: $\quad\quad\quad rc = c - 1$
24: $\quad$ **end while**
25: $\quad a = M\alpha[r,lc]$; $b = M\beta[r,lc]$
26: $\quad$ **if** $Abs[b - b_j] < Abs[\Delta_\beta]$ **then**
27: $\quad\quad c = lc$

Алгоритм 11: Продолжение

```
28:
29:     while br ≠ tr do
30:         r = Ceiling[(br + tr)/2];
31:         a = Mα[r, c]; b = Mβ[r, c]
32:         Δ_α = a − a_j; Δ_β = b − b_j
33:         if Δ_α > 0 then
34:             br = r
35:         else
36:             tr = r − 1
37:     end while
38:     a = Mα[tr, c]; b = Mβ[tr, c]
39:     if Abs[a − a_j] < Abs[Δ_α] then
40:         r = tr
41:     return {r, c}
42: end while
```

ПРИЛОЖЕНИЕ Л

Алгоритм 12: Алгоритм нахождения точки пересечения заданных линий уровня α_j, β_j с точностью ε

Input: $M\alpha$,$M\beta$ *step*, α_j, β_j, ε

Output: $\hat{\alpha}, \hat{\beta}$

$la_j \leftarrow \ln((1-\beta_j)/\alpha_j)$; $lb_j \leftarrow \ln(\beta_j/(1-\alpha_j))$

$R \leftarrow Ceiling[la_j/step]$; $C \leftarrow Ceiling[-lb_j/step]$

$br \leftarrow 1$; $tr \leftarrow R$

$lc \leftarrow 1$; $rc \leftarrow C$

$r = tr$; $c = rc$

while $(lc < rc)$ || $(br < tr)$ **do**

    $c = Ceiling[(lc + rc)/2]$;

    $r = Ceiling[(br + tr)/2]$;

    $a = M\alpha[r, c]$; $b = M\beta[r, c]$

    $\Delta_\alpha = a - a_j$; $\Delta_\beta = b - b_j$

    if $\Delta_\beta > 0$ **then** $lc = c$

    else $rc = c - 1$

    if $\Delta_\alpha > 0$ **then** $br = r$

    else $tr = r - 1$

end while

if $\Delta_\beta > 0$ **then** $rc\ ++$

else $rc\ --$

if $\Delta_\alpha > 0$ **then** $tr\ ++$

else $tr\ --$

$a = M\alpha[tr, rc]$; $b = M\beta[tr, rc]$

if $Abs[b - b_j] - Abs[a - a_j] < Abs[\Delta_\alpha] + Abs[\Delta_\beta]$ **then**

    $r = tr$; $c = rc$

return $\{r, c\}$

ПРИЛОЖЕНИЕ М

Алгоритм 13: Алгоритм вычисления α методом Монте-Карло, при заданных n, lB и lA

Input: n, lB, lA

Output: nH_1

$z \leftarrow 0$; $nH_1 \leftarrow 0$

for $i = 0 \dots N-1$ **do**

  $z = 0$

  while $lB < z$ && $z < lA$ **do**

    $x \leftarrow$ RandomVariate[NormalDistribution[0, 1]]

    $Z \mathrel{+}= \ln\left(\frac{f_1(x)}{f_0(x)}\right)$

  end while

  if $z > lB$ **then**

    $nH_1 ++$

end for

return nH_1

Алгоритм 14: Алгоритм вычисления β методом Монте-Карло, при заданных n, lB и lA

Input: n, lB, lA

Output: nH_0

$z \leftarrow 0$; $nH_0 \leftarrow 0$

for $i = 0 \dots N-1$ **do**

  $z = 0$

  while $lB < z$ && $z < lA$ **do**

    $x \leftarrow$ RandomVariate[LogisticDistribution[0, Sqrt[3]/Pi]]

    $z \mathrel{+}= \ln\left(\frac{f_1(x)}{f_0(x)}\right)$

  end while

  if $z < lA$ **then**

    $nH_0 ++$

end for

return nH_0

ПРИЛОЖЕНИЕ Н

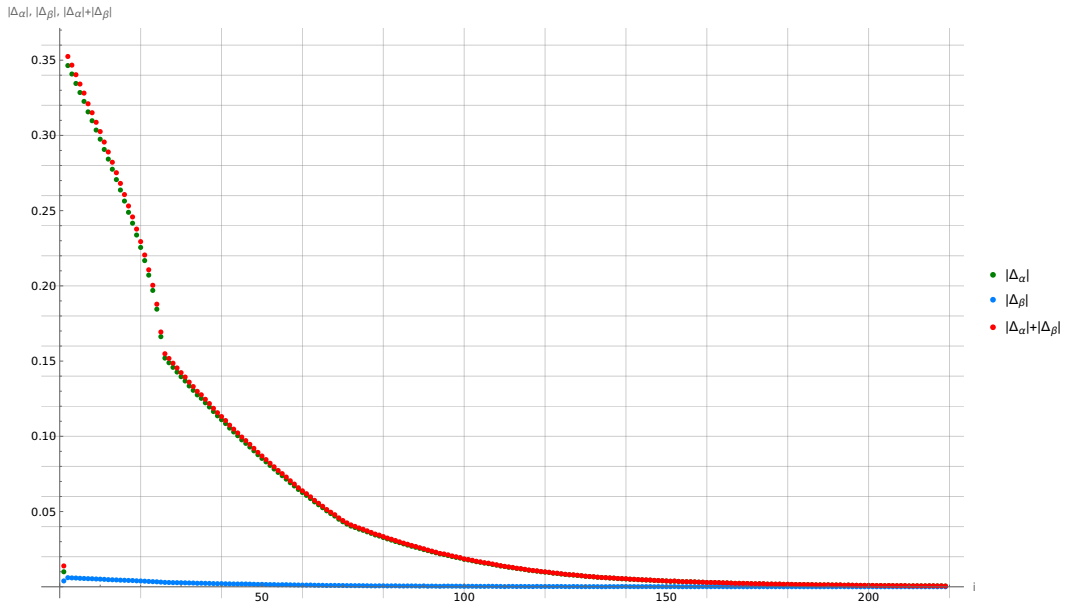


Рис. 6.15: Значения $|\Delta_\alpha|$, $|\Delta_\beta|$, $|\Delta_\alpha| + |\Delta_\beta|$ на i-й итерации, при $n = 6634897$, $\alpha = 0.01$, $\beta = 0.01$, $\varepsilon = 0.0005$

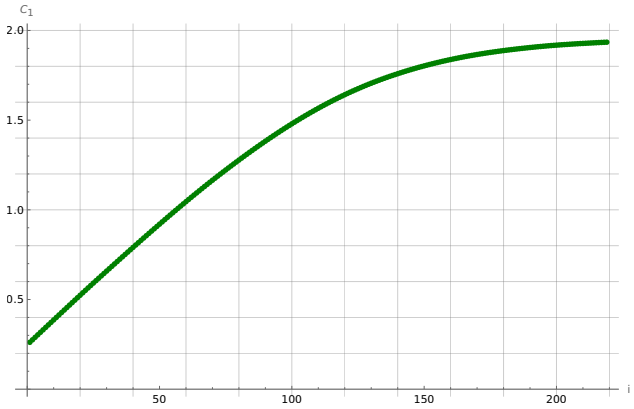


Рис. 6.16: Значения C_1 на i-й итерации, при $n = 6634897$, $\alpha = 0.01$, $\beta = 0.01$, $\varepsilon = 0.0005$

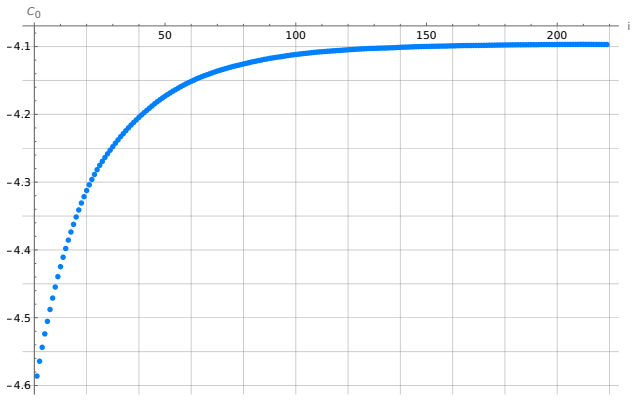


Рис. 6.17: Значения C_0 на i-й итерации, при $n = 6634897$, $\alpha = 0.01$, $\beta = 0.01$, $\varepsilon = 0.0005$

ПРИЛОЖЕНИЕ О

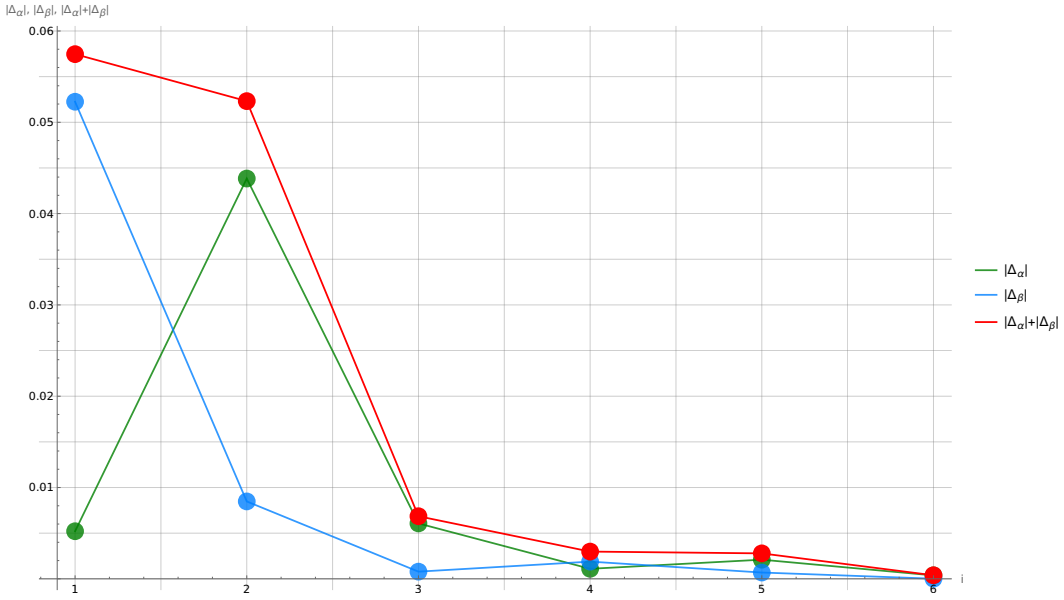


Рис. 6.18: Значения $|\Delta_\alpha|$, $|\Delta_\beta|$, $|\Delta_\alpha| + |\Delta_\beta|$ на i-й итерации, при $n = 6634897$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.0005$

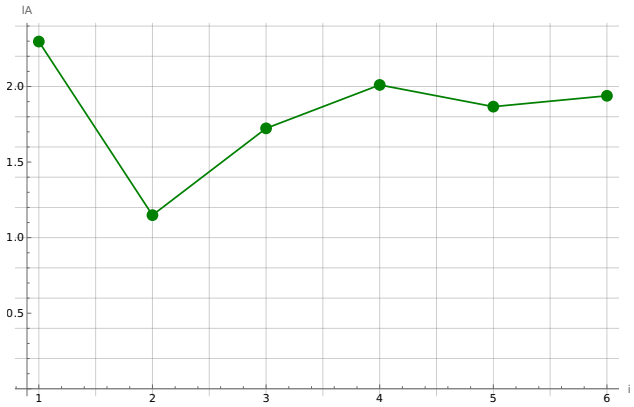


Рис. 6.19: Значения lA на i-й итерации, при $n = 6634897$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.0005$

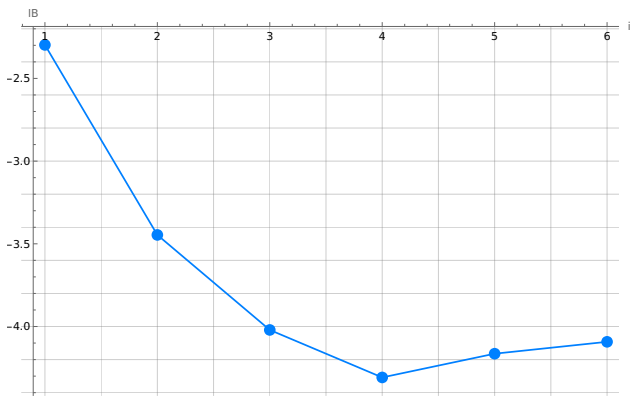


Рис. 6.20: Значения lB на i-й итерации, при $n = 6634897$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.0005$

ПРИЛОЖЕНИЕ П

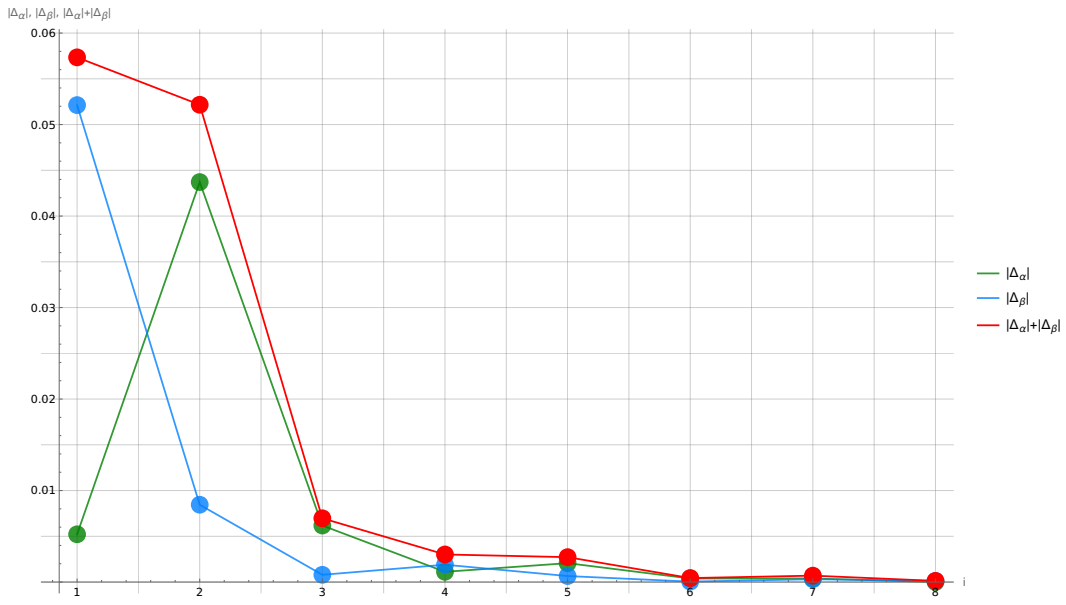


Рис. 6.21: Значения $|\Delta_\alpha|$, $|\Delta_\beta|$, $|\Delta_\alpha| + |\Delta_\beta|$ на i-й итерации, при $n = 26539587$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.00025$

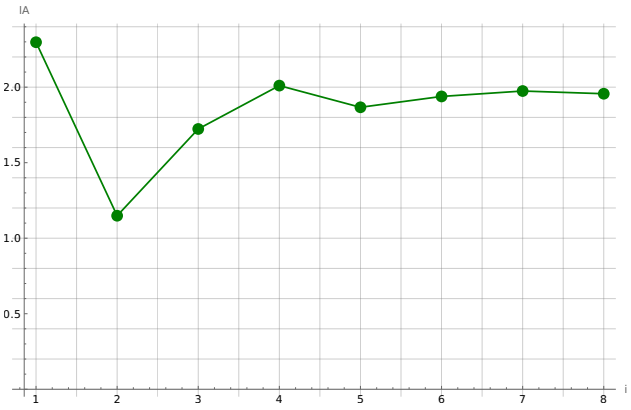


Рис. 6.22: Значения lA на i-й итерации, при $n = 26539587$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.00025$

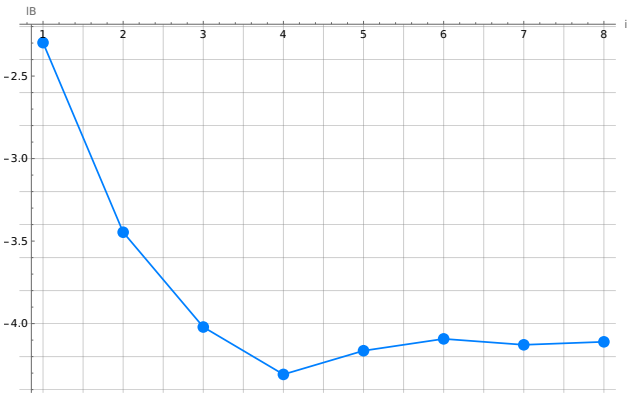


Рис. 6.23: Значения lB на i-й итерации, при $n = 26539587$, $a_j = 0.01$, $b_j = 0.01$, $\varepsilon = 0.00025$

ПРИЛОЖЕНИЕ Р

Алгоритм 15: Итерационный метод вычисления критических границ C_0, C_1 при заданных n, α, β, ε

```cpp
/***
 *      Контрпример к алгоритму предложенному в статье С. Я.Гродзенского,
 * А. Н. Чесалина "Уточнение границ последовательных статистических критериев
 * с помощью компьютерного моделирования".
 *
 *  Васьковский А. С. 05.05.2023
***/
#define _USE_MATH_DEFINES

#include <iostream>
#include <cmath>
#include <random>
#include <cfloat>

constexpr size_t N = 6634897;
constexpr double alpha = 0.01; // ошибка первого рода
constexpr double beta = 0.01;  // ошибка второго рода
constexpr double eps = 0.0005; // точность

#ifdef Slow_speed
    constexpr size_t MaxIterations = 500;
#else
    constexpr size_t MaxIterations = 5;
#endif
#ifdef Save_C0_C1_Da_Db
    #include <vector>
    #include <iomanip>
    using VectorPoints = std::vector<std::pair<double, double>>;
    VectorPoints points_Da_Db;
    VectorPoints points_C0_C1;

    void print_VectorPoints_Wolfram(VectorPoints& points)
    {
        VectorPoints::const_iterator j = points.begin();
        std::cout << '{';
        for ( ;j != std::prev(points.end()); ++j){
            std::cout<< std::setprecision(12) <<  '{' << (j->first) << ", " <<
            ↪  (j->second) << "}, ";
        }
        std::cout<< std::setprecision(12) <<  '{' << (j->first) << ", " <<
        ↪  (j->second) << "}}" << std::endl;

```

```
    }
#endif

std::random_device rd{};
std::mt19937 gen{rd()};

double logf1_f0(const double x){// f0 ~ Exp[1]; f1 ~ Normal[1,1]
    return (x < 0) ? DBL_MAX : (-1.0/2.0 * (-1.0+ x)*(-1.0+ x) + x -1.0/2.0 *
    ↪  log(2*M_PI));
}

size_t funA(const double C0, const double C1){
    double gamma;
    size_t nH1 = 0, i;

    std::exponential_distribution<> d{1};
    gen.seed(rd());
    for (i = 0; i < N; ++i){
        gamma = 0;
        while (C0 < gamma && gamma < C1)
            gamma += logf1_f0(d(gen));
        // Поскольку нам важен в funA учесть лишь принятие nH1, то
        // если она принялась мы должны увеличить счетчик.
        if (gamma > C0)
            ++nH1;
    }
    return nH1;
}

size_t funB(const double C0, const double C1){
    double gamma;
    size_t nH0 = 0, i;

    std::normal_distribution<> d{1, 1};
    gen.seed(rd());
    for (i = 0; i < N; ++i){
        gamma = 0;
        while (C0 < gamma && gamma < C1){
            gamma += logf1_f0(d(gen));
        }
        // Поскольку нам важен в funB учесть лишь принятие nH0, то
        // если она принялась мы должны увеличеть счетчик
        if (gamma < C1)
            nH0++;
    }
    return nH0;
}
```

```
87
88 int main(void)
89 {
90     // начальные значения
91     double C0 = log(beta / (1 - alpha));
92     double C1 = log((1 - beta) / (alpha));
93
94     double alpha_hat, beta_hat, delta_alpha, delta_beta;
95
96     size_t i = 0; // счетчик для ограничения числа итераций
97 #ifdef Details
98     std::cout << "N = " << N << "; eps = "<< eps
99     <<";\nalpha = " << alpha <<"; beta = "<< beta
100    << ";\nC0 = " << C0 << "; C1 = " << C1 << std::endl;
101 #endif
102    do {
103        alpha_hat = ((double)(funA(C0, C1))) / (double)N;
104        beta_hat = ((double)funB(C0, C1)) / (double)N;
105
106        delta_alpha = alpha_hat - alpha;
107        delta_beta = beta_hat - beta;
108
109 #ifdef Slow_speed
110        C0 -= 1.0/70.0*delta_beta / beta_hat;
111        C1 += 1.0/70.0*delta_alpha / alpha_hat;
112 #else
113        C0 -= delta_beta / beta_hat;
114        C1 += delta_alpha / alpha_hat;
115 #endif
116 #ifdef Save_C0_C1_Da_Db
117    points_Da_Db.push_back(std::make_pair(delta_alpha, delta_beta));
118    points_C0_C1.push_back(std::make_pair(C0, C1));
119 #endif
120        i++;
121 #ifdef Details
122        std::cout << "\ni = " << i << "; C0 = " << C0 << "; C1 = " << C1
123        << ";\nalpha_hat = " << alpha_hat << "; beta_hat = " << beta_hat
124        << ";\n|delta_alpha| = " << fabs(delta_alpha) << "; |delta_beta| = " <<
        ↪  fabs(delta_beta)
125        << ";\n|Delta(alpha)| + |Delta(beta)| = " << fabs(delta_alpha) +
        ↪  fabs(delta_beta) << std::endl;
126 #endif
127    } while ((fabs(delta_alpha) + fabs(delta_beta) > eps) && (i < MaxIterations));
128
129 #ifdef Save_C0_C1_Da_Db
130    print_VectorPoints_Wolfram(points_C0_C1);
131    print_VectorPoints_Wolfram(points_Da_Db);
```

```
#endif
    std::cout << "C0 = " << C0 << "; C1 = " << C1
    << ";\nAlpha_hat = " << alpha_hat << "; Beta_hat = " << beta_hat
    << ";\n|Delta(alpha)| + |Delta(beta)| = " << fabs(delta_alpha) + fabs(delta_beta)
    ↪  << std::endl;
    return 0;
}
```

ПРИЛОЖЕНИЕ С

Алгоритм 16: Итерационный метод вычисления критических границ lB, lA при заданных n, a_j, b_j, ε

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <cmath>
#include <random>
#include <cfloat>
#include "utility/forWolfram.h"

constexpr size_t N = 6634897;
constexpr double aj = 0.01; // ошибка первого рода
constexpr double bj = 0.01;  // ошибка второго рода
constexpr double eps = 0.0005; // точность
constexpr size_t MaxIterations = 10;

std::random_device rd{};
std::mt19937 gen{rd()};

double logf1_f0(const double x){// f0 ~ Exp[1]; f1 ~ Normal[1,1]
    return (x < 0) ? DBL_MAX : (-1.0/2.0 * (-1.0+ x)*(-1.0+ x) + x -1.0/2.0 *
    ↪  log(2*M_PI));
}

size_t funA(const double lB, const double lA){
    double Z;
    size_t nH1 = 0, i;

    std::exponential_distribution<> d{1};
    gen.seed(rd());
    for (i = 0; i < N; ++i){
        Z = 0;
        while (lB < Z && Z < lA)
            Z += logf1_f0(d(gen));
        if (Z > lB)
            ++nH1;
    }
    return nH1;
}

size_t funB(const double lB, const double lA){
    double Z;
    size_t nH0 = 0, i;

```

```
    std::normal_distribution<> d{1, 1};
    gen.seed(rd());
    for (i = 0; i < N; ++i){
        Z = 0;
        while (lB < Z && Z < lA){
            Z += logf1_f0(d(gen));
        }
        if (Z < lA)
            nH0++;
    }
    return nH0;
}

int main(void)
{
    constexpr double lb = log(bj / (1 - aj));
    constexpr double la = log((1 - bj) / (aj));
    double blA = 0;
    double tlA = la;
    double llB = 0;
    double rlB = lb;

    double lB, lA, a, b;
    // Инициализация такая, чтобы попасть в цикл for
    // с условием fabs(Da)+fabs(Db) > eps
    double Da = eps;
    double Db = eps;

    size_t i = 0;

#ifdef Details
    std::cout << "N = " << N << "; eps = "<< eps
    <<";\na = " << aj <<"; bj = "<< bj
    << ";\nStart blA = " << blA << "; tlA = " << tlA << "; llB = " << llB << "; rlB =
    ↪  "<< rlB << std::endl;
#endif
#ifdef F_lb_la
std::cout << "Для проверки значений a и b при lb la\nfunA(lb,la)/N = " <<
  ↪  ((double)funA(lb,la)/(double)N) << '\n'
  << "funB(lb,la)/N = "  <<  ((double)funB(lb,la)/(double)N) <<std::endl;
#endif

    while(fabs(Da)+fabs(Db) > eps && i<MaxIterations){
        lB = (llB + rlB)/2.0;
        lA = (blA + tlA)/2.0;

        a = ((double)funA(lB,lA)/(double)N);
```

```
87          b =  ((double)funB(lB,lA)/(double)N);
88          Da = a - aj; Db = b - bj;
89  #ifdef Save_lB_lA_Db_Da
90      points_Db_Da.push_back(std::make_pair(Db, Da));
91      points_lB_lA.push_back(std::make_pair(lB, lA));
92  #endif
93          if(Db > 0)
94              {llB = lB;}
95          else
96              {rlB = lB;}
97          if(Da > 0)
98              {blA = lA;}
99          else
100             {tlA = lA;}
101         i++;
102 #ifdef Details
103         std::cout << "\ni = " << i << "; lB = " << lB << "; lA = " << lA
104         << ";\na = " << a << "; b = " << b
105         << ";\nblA = " << blA << "; tlA = " << tlA << "; llB = " << llB << "; rlB =
            ↪  "<< rlB
106         << ";\nDa = " << Da << "; Db = " << Db << "; |Da|+|Db| = " <<
            ↪  fabs(Da)+fabs(Db) << std::endl;
107 #endif
108     }
109
110 #ifdef Save_lB_lA_Db_Da
111     print_VectorPoints_Wolfram(points_lB_lA);
112     print_VectorPoints_Wolfram(points_Db_Da);
113 #endif
114         std::cout << "lB = " << lB << "; lA = " << lA
115     << ";\na = " << a << "; b = " << b
116     << ";\nDa = " << Da << "; Db = " << Db << "; |Da|+|Db| = " << fabs(Da)+fabs(Db)
        ↪  << std::endl;
117
118     return 0;
119 }
```

ПРИЛОЖЕНИЕ Т

Parameters name	Parameters value	Parameters name	Parameters value
Operating System	Arch Linux	**CPU**	Intel(R) Core(TM) i7-8750H CPU 2.20GHz
Kernel	Linux 6.2.12-arch1-1	**CPU current speed**	2100 MHz
Architecture	x86-64	**CPU core count**	6
Hardware Model	GL63 8RC	**CPU L1 cache**	384 kB, 8-way
Firmware Version and Date	E16P6IMS.107 09/05/2018	**CPU L2 cache**	1536 kB, 4-way
Disk	ST1000LM049-2GH172	**CPU L3 cache**	9 MB, 12-way
Parameters name	**Parameters value**		
RAM 1 (product, size, width, clock)	MSI26D4S9S8ME-8 DDR4, 8 Gib, 64 bits, 2667 MHz (0.4ns)		
RAM 2 (product, size, width, clock)	MSI26D4S9S8ME-8 DDR4, 8 Gib, 64 bits, 2667 MHz (0.4ns)		
Compiler	g++ 13.1,1 20230429, -std=c++17 -O3 -m64 -march=x86-64		
Wolfram Mathematica	13.1.0 for Linux x86 (64-bit) (June 16, 2022)		

Таблица 6.2: Характеристики вычислительного устройства для проведения экспериментов.

ПРИЛОЖЕНИЕ У

M	N	ε	$\hat{\alpha}$	$\hat{\beta}$	lA	lB	Δ_α	Δ_β	$\lvert\Delta_\alpha\rvert + \lvert\Delta_\beta\rvert$	i	время	sR_α	sR_β	mR_α	mR_β	mAll
$M1$	9604	0.01	0.00531	0.00656	4.59512	−4.59512	−0.00469	−0.00344	0.00813	1	0.0062	–	–	–	–	–
$M2$	9604	0.01	0.009788	0.0101	4.02073	−4.02073	−0.0002124	0.00009996	0.0003124	3	0.0109	–	–	–	–	–
$M3$	9604	0.01	0.009892	0.01041	4.02073	−4.02073	−0.0001083	0.0004123	0.0005206	3	0.0142	11705	4985	1.5	0.625	2.125
$M4$	9604	0.01	0.009788	0.009475	4.02073	−4.02073	−0.0002124	−0.0005248	0.0007372	3	0.00623	11594	4979	–	–	2.25
$M1$	19699	0.01	0.005584	0.005838	4.59512	−4.59512	−0.004416	−0.004162	0.008578	1	0.00993	–	–	–	–	–
$M2$	19699	0.01	0.009747	0.009442	4.02073	−4.02073	−0.0002533	−0.0005579	0.0008112	3	0.0226	–	–	–	–	–
$M3$	19699	0.01	0.01091	0.009341	4.02073	−4.02073	0.0009143	−0.0006594	0.001574	3	0.03	23768	10362	3.25	1.25	4.5
$M4$	19699	0.01	0.008833	0.009899	4.02073	−4.02073	−0.001167	−0.000101	0.001268	3	0.0119	23756	10313	–	–	4.836
$M1$	37843	0.01	0.00547	0.005919	4.59512	−4.59512	−0.00453	−0.004081	0.008611	1	0.0199	–	–	–	–	–
$M2$	37843	0.01	0.01039	0.01036	4.02073	−4.02073	0.000385	0.0003586	0.0007436	3	0.0429	–	–	–	–	–
$M3$	37843	0.01	0.00946	0.009196	4.02073	−4.02073	−0.0005399	−0.0008041	0.001344	3	0.0582	45752	19838	6.25	2.5	8.75
$M4$	37843	0.01	0.01023	0.01031	4.02073	−4.02073	0.0002265	0.0003057	0.0005322	3	0.0224	45870	19734	–	–	8.926
$M1$	60023	0.004	0.011	0.01005	3.920945	−4.006986	0.0009958	0.00004615	0.001042	3	0.0878	–	–	–	–	–
$M2$	60023	0.004	0.00998	0.01035	4.02073	−4.02073	−0.00002049	0.000346	0.0003665	3	0.0678	–	–	–	–	–
$M3$	60023	0.004	0.00998	0.009896	4.02073	−4.02073	−0.00002049	−0.0001038	0.0001243	3	0.0963	72610	31210	9.625	4	13.63
$M4$	60023	0.004	0.009696	0.01015	4.02073	−4.02073	−0.0003037	0.0001461	0.0004498	3	0.0409	72433	31389	–	–	13.64
$M1$	103671	0.004	0.01037	0.009771	3.968853	−3.96987	0.0003693	−0.0002287	0.000598	3	0.141	–	–	–	–	–
$M2$	103671	0.004	0.01014	0.01024	4.02073	−4.02073	0.0001378	0.0002439	0.0003818	3	0.116	–	–	–	–	–
$M3$	103671	0.004	0.01041	0.01033	4.02073	−4.02073	0.0004079	0.0003308	0.0007387	3	0.167	125299	54245	16.75	7	23.75
$M4$	103671	0.004	0.009916	0.01045	4.02073	−4.02073	−0.00008402	0.0004465	0.0005305	3	0.0756	125175	54014	–	–	23.82
$M1$	414683	0.002	0.01024	0.01049	3.985166	−3.99986	0.0002416	0.0004948	0.0007363	3	0.558	–	–	–	–	–
$M2$	414683	0.002	0.009986	0.009991	4.02073	−4.02073	−0.00001406	−0.000009236	0.00002329	3	0.463	–	–	–	–	–
$M3$	414683	0.002	0.009887	0.01031	4.02073	−4.02073	−0.0001129	0.0003067	0.0004196	3	0.697	500943	216312	66.5	27.88	94.38
$M4$	414683	0.002	0.00974	0.01017	4.02073	−4.02073	−0.00026	0.0001692	0.0004292	3	0.328	501381	216915	–	–	94.63
$M1$	1658725	0.001	0.01004	0.01016	3.997579	−3.99469	0.00003964	0.0001572	0.0001968	3	2.29	–	–	–	–	–
$M2$	1658725	0.001	0.009923	0.01002	4.02073	−4.02073	−0.00007672	0.00001794	0.00009465	3	1.87	–	–	–	–	–
$M3$	1658725	0.001	0.009985	0.009893	4.02073	−4.02073	−146200.	−0.0001075	0.0001221	3	2.86	2004561	866876	265.8	112	377.8
$M4$	1658725	0.001	0.009951	0.009965	4.02073	−4.02073	−0.00004898	−0.00003512	0.0000841	3	1.36	2004721	867889	–	–	378.
$M1$	6634897	0.0005	0.01018	0.01018	3.999187	−3.994926	0.000183	0.000176	0.000359	3	9.16	–	–	–	–	–
$M2$	6634897	0.0005	0.009877	0.00988	4.02073	−4.02073	−0.0001233	−0.0001203	0.0002436	3	7.71	–	–	–	–	–
$M3$	6634897	0.0005	0.01003	0.009964	4.02073	−4.02073	0.00002638	−0.00003556	0.00006195	3	11.7	8018847	3468801	1063	447.9	1511.
$M4$	6634897	0.0005	0.00991	0.00994	4.02073	−4.02073	−0.00008967	−0.00006028	0.00015	3	5.85	8016440	3467366	–	–	1511.
$M1$	26539587	0.00025	0.01005	0.009954	4.012697	−4.018247	0.00004669	−0.00004581	0.0000925	4	48.3	–	–	–	–	–
$M2$	26539587	0.00025	0.009959	0.009946	4.02073	−4.02073	−0.00004076	−0.00005373	0.00009449	3	30.6	–	–	–	–	–
$M3$	26539587	0.00025	0.009969	0.009941	4.02073	−4.02073	−0.00003112	−0.00005859	0.00008971	3	49.1	32067551	13867192	4251.	1791.	6042.
$M4$	26539587	0.00025	0.009935	0.009952	4.02073	−4.02073	−0.00006465	−0.00004804	0.0001127	3	23.4	32065540	13869974	–	–	6042.

Таблица 6.3: Результаты различных методов при $H_0 : N(1,1)$, $H_1 : N(2,1)$ и $\alpha = 0.01$, $\beta = 0.01$

где $M1$ – итерационный метод, предложенный С. Я. Гродзенским; $M2$ – итерационный метод, предложенный в данной работе; $M3$ – метод с R-деревом; $M4$ – метод с R-деревом, 6 исполняемых потоков; i – количество итераций; время приведено в секундах; sR_α, sR_β – количество сохраненных точек соответственно в R_α, R_β, причем для $M4$ приведена сумма по sR_{α_1}, sR_{α_2}, sR_{α_3}, и сумма по sR_{β_1}, sR_{β_2}, sR_{β_3}; mR_α, mR_β – используемая в мегабайтах память для хранения соответственно R_α и R_β, для $M4$ указана лишь mAll; mAll – общая используемая в мегабайтах память.

ПРИЛОЖЕНИЕ Ф

M	N	ε	$\hat{\alpha}$	$\hat{\beta}$	lA	lB	Δ_α	Δ_β	$\lvert\Delta_\alpha\rvert + \lvert\Delta_\beta\rvert$	i	время	sR_α	sR_β	mR_α	mR_β	mAll
$M1$	9604	0.01	0.002395	0.00833	2.657671	−4.533434	−0.007605	−0.00167	0.009275	8	0.0361	–	–	–	–	–
$M2$	9604	0.01	0.01374	0.01135	1.72317	−4.02073	0.003744	0.001349	0.005094	3	0.00921	–	–	–	–	–
$M3$	9604	0.01	0.01645	0.009996	1.72317	−4.02073	0.006451	−0.000004165	0.006456	3	0.0141	11058	5786	1.75	0.75	2.5
$M4$	9604	0.01	0.01645	0.0126	1.72317	−4.02073	0.006451	0.002599	0.00905	3	0.00607	11056	5741	–	–	2.563
$M1$	19699	0.01	0.003503	0.006599	2.513377	−4.514267	−0.006497	−0.003401	0.009898	11	0.115	–	–	–	–	–
$M2$	19699	0.01	0.01558	0.01025	1.72317	−4.02073	0.005585	0.0002543	0.005839	3	0.0184	–	–	–	–	–
$M3$	19699	0.01	0.01528	0.01086	1.72317	−4.02073	0.00528	0.0008635	0.006143	3	0.0331	22686	11729	3	1.5	4.5
$M4$	19699	0.01	0.01533	0.01107	1.72317	−4.02073	0.005331	0.001067	0.006397	3	0.015	22677	11809	–	–	4.668
$M1$	37843	0.01	0.003462	0.006897	2.502994	−4.5109	−0.006538	−0.003103	0.009641	11	0.197	–	–	–	–	–
$M2$	37843	0.01	0.0163	0.01065	1.72317	−4.02073	0.006304	0.0006493	0.006953	3	0.0348	–	–	–	–	–
$M3$	37843	0.01	0.01699	0.01068	1.72317	−4.02073	0.006991	0.0006757	0.007667	3	0.0539	43550	22513	6	2.875	8.875
$M4$	37843	0.01	0.01591	0.01115	1.72317	−4.02073	0.005908	0.001151	0.007059	3	0.0217	43375	22637	–	–	8.918
$M1$	60023	0.004	0.008513	0.007697	2.063853	−4.331625	−0.001487	−0.002303	0.00379	46	1.23	–	–	–	–	–
$M2$	60023	0.004	0.00933	0.008547	2.010365	−4.307925	−0.0006702	−0.001453	0.002124	4	0.0766	–	–	–	–	–
$M3$	60023	0.004	0.00888	0.008097	2.010365	−4.307925	−0.00112	−0.001903	0.003023	4	0.0905	69164	35924	9.25	4.625	13.88
$M4$	60023	0.004	0.008397	0.00833	2.010365	−4.307925	−0.001603	−0.00167	0.003273	4	0.0525	69054	35796	–	–	13.88
$M1$	103671	0.004	0.01327	0.00954	1.814407	−4.201153	0.003273	−0.0004602	0.003733	92	4.09	–	–	–	–	–
$M2$	103671	0.004	0.009086	0.008006	2.010365	−4.307925	−0.0009136	−0.001994	0.002907	4	0.133	–	–	–	–	–
$M3$	103671	0.004	0.009193	0.009106	2.010365	−4.307925	−0.0008075	−0.0008943	0.001702	4	0.155	119354	62520	16	8	24
$M4$	103671	0.004	0.00871	0.008652	2.010365	−4.307925	−0.00129	−0.001348	0.002637	4	0.0835	119594	62340	–	–	24.13
$M1$	414683	0.002	0.01064	0.008857	1.921256	−4.236368	0.0006395	−0.001143	0.001782	81	15.	–	–	–	–	–
$M2$	414683	0.002	0.01043	0.01038	1.938566	−4.092529	0.0004297	0.0003766	0.0008063	6	0.833	–	–	–	–	–
$M3$	414683	0.002	0.01022	0.009971	1.938566	−4.092529	0.000215	−0.00002853	0.0002436	6	0.67	476879	247921	63.38	32	95.38
$M4$	414683	0.002	0.01043	0.009769	1.938566	−4.092529	0.0004321	−0.0002311	0.0006632	6	0.39	477383	248657	–	–	95.75
$M1$	1658725	0.001	0.01073	0.009799	1.918655	−4.126018	0.0007329	−0.0002009	0.0009338	174	118.	–	–	–	–	–
$M2$	1658725	0.001	0.01035	0.01002	1.938566	−4.092529	0.0003489	0.00001854	0.0003675	6	3.38	–	–	–	–	–
$M3$	1658725	0.001	0.01037	0.01004	1.938566	−4.092529	0.0003658	0.00003904	0.0004048	6	2.96	1909946	994203	253.6	128.5	382.1
$M4$	1658725	0.001	0.01042	0.01026	1.938566	−4.092529	0.0004164	0.0002561	0.0006725	6	1.45	1909531	993904	–	–	382.1
$M1$	6634897	0.0005	0.01029	0.009838	1.94104	−4.113143	0.0002852	−0.0001617	0.0004469	209	572.	–	–	–	–	–
$M2$	6634897	0.0005	0.01036	0.01002	1.938566	−4.092529	0.000361	0.00001779	0.0003788	6	13.6	–	–	–	–	–
$M3$	6634897	0.0005	0.01045	0.01003	1.938566	−4.092529	0.0004469	0.00002683	0.0004737	6	12.2	7636126	3977349	1014.	514.4	1528.
$M4$	6634897	0.0005	0.01045	0.009987	1.938566	−4.092529	0.0004504	−0.00001296	0.0004633	6	6.25	7637186	3976325	–	–	1528.
$M1$	26539587	0.00025	0.01007	0.00983	1.951546	−4.113074	0.00007393	−0.0001704	0.0002443	226	2630.	–	–	–	–	–
$M2$	26539587	0.00025	0.009994	0.009852	1.956516	−4.110478	−5873000.	−0.0001478	0.0001537	8	73.5	–	–	–	–	–
$M3$	26539587	0.00025	0.009972	0.009836	1.956516	−4.110478	−0.0000278	−0.0001638	0.0001916	8	50.5	30545956	15905149	4055.	2057.	6112.
$M4$	26539587	0.00025	0.01003	0.009841	1.956516	−4.110478	0.00003	−0.0001586	0.0001886	8	25.7	30546436	15907422	–	–	6112.

Таблица 6.4: Результаты различных методов при $H_0 : Exp\,(1)$, $H_1 : N\,(1,1)$ и $\alpha = 0.01$, $\beta = 0.01$

где $M1$ – итерационный метод, предложенный С. Я. Гродзенским, в данном случае $\Delta_C = 1/70$; $M2$ – итерационный метод, предложенный в данной работе; $M3$ – метод с R-деревом; $M4$ – метод с R-деревом, 6 исполняемых потоков; i – количество итераций; время приведено в секундах; sR_α, sR_β – количество сохраненных точек соответственно в R_α, R_β, причем для $M4$ приведена сумма по sR_{α_1}, sR_{α_2}, sR_{α_3}, и сумма по sR_{β_1}, sR_{β_2}, sR_{β_3}; mR_α, mR_β – используемая в мегабайтах память для хранения соответственно R_α и R_β, для $M4$ указана лишь mAll; mAll – общая используемая в мегабайтах память.

ПРИЛОЖЕНИЕ X

M	N	ε	$\hat{\alpha}$	$\hat{\beta}$	lA	lB	Δ_α	Δ_β	$\lvert\Delta_\alpha\rvert + \lvert\Delta_\beta\rvert$	i	время	sR_α	sR_β	mR_α	mR_β	mAll
$M1$	9604	0.01	0.008017	0.00885	2.262195	−4.516349	−0.001983	−0.00115	0.003132	3	0.0249	–	–	–	–	–
$M2$	9604	0.01	0.01281	0.0102	2.010365	−4.307925	0.002807	0.0002041	0.003011	4	0.0266	–	–	–	–	–
$M3$	9604	0.01	0.01187	0.01031	2.010365	−4.307925	0.00187	0.0003082	0.002178	4	0.0171	10546	7489	1.375	1.125	2.5
$M4$	9604	0.01	0.0126	0.01208	2.010365	−4.307925	0.002599	0.002078	0.004677	4	0.00716	10576	7406	–	–	2.5
$M1$	19699	0.01	0.01031	0.009036	2.16931	−4.505817	0.0003051	−0.000964	0.001269	3	0.0555	–	–	–	–	–
$M2$	19699	0.01	0.01122	0.01097	2.010365	−4.307925	0.001219	0.000965	0.002184	4	0.0546	–	–	–	–	–
$M3$	19699	0.01	0.01284	0.01061	2.010365	−4.307925	0.002843	0.0006097	0.003453	4	0.0356	21678	15395	3	2.125	5.125
$M4$	19699	0.01	0.01076	0.01208	2.010365	−4.307925	0.000762	0.002082	0.002844	4	0.0121	21711	15297	–	–	5.125
$M1$	37843	0.01	0.01128	0.008879	2.03989	−4.512396	0.001283	−0.001121	0.002405	3	0.0999	–	–	–	–	–
$M2$	37843	0.01	0.01221	0.01136	2.010365	−4.307925	0.002208	0.001363	0.003571	4	0.104	–	–	–	–	–
$M3$	37843	0.01	0.01213	0.0116	2.010365	−4.307925	0.002129	0.001601	0.00373	4	0.0695	41508	29595	5.5	4	9.5
$M4$	37843	0.01	0.01181	0.01205	2.010365	−4.307925	0.001812	0.00205	0.003862	4	0.0315	41395	29582	–	–	9.5
$M1$	103671	0.004	0.01285	0.009202	1.991927	−4.474639	0.002848	−0.0007978	0.003646	5	0.436	–	–	–	–	–
$M2$	103671	0.004	0.0114	0.01104	2.010365	−4.307925	0.001401	0.001045	0.002446	4	0.285	–	–	–	–	–
$M3$	103671	0.004	0.01184	0.01124	2.010365	−4.307925	0.001836	0.001237	0.003073	4	0.194	113788	80804	15.13	10.75	25.88
$M4$	103671	0.004	0.01111	0.01132	2.010365	−4.307925	0.001112	0.001324	0.002436	4	0.1	113865	80897	–	–	25.75
$M1$	414683	0.002	0.01188	0.01001	2.054362	−4.455894	0.001879	0.000007644	0.001887	10	3.52	–	–	–	–	–
$M2$	414683	0.002	0.00993	0.009747	2.153962	−4.451522	−0.00006952	−0.0002528	0.0003223	5	1.46	–	–	–	–	–
$M3$	414683	0.002	0.01003	0.009805	2.153962	−4.451522	0.00003417	−0.0001949	0.0002291	5	0.812	455342	323243	60.5	42.5	103.
$M4$	414683	0.002	0.009943	0.009704	2.153962	−4.451522	−0.00005747	−0.0002962	0.0003537	5	0.418	455550	323757	–	–	103.3
$M1$	1658725	0.001	0.01068	0.009806	2.105509	−4.44635	0.0006829	−0.0001943	0.0008772	10	14.	–	–	–	–	–
$M2$	1658725	0.001	0.0101	0.009677	2.153962	−4.451522	0.00009812	−0.0003227	0.0004208	5	5.92	–	–	–	–	–
$M3$	1658725	0.001	0.009884	0.009887	2.153962	−4.451522	−0.0001159	−0.0001129	0.0002288	5	3.35	1820840	1293890	241.9	169.6	411.5
$M4$	1658725	0.001	0.01009	0.009847	2.153962	−4.451522	0.00009149	−0.0001533	0.0002448	5	1.67	1821431	1294310	–	–	412.
$M1$	6634897	0.0005	0.01033	0.009952	2.130715	−4.445069	0.0003349	−0.00004762	0.0003825	12	68.1	–	–	–	–	–
$M2$	6634897	0.0005	0.01006	0.009822	2.153962	−4.451522	0.00005818	−0.000178	0.0002362	5	23.9	–	–	–	–	–
$M3$	6634897	0.0005	0.009984	0.009957	2.153962	−4.451522	−0.00001597	−0.0000428	0.00005877	5	13.9	7286444	5174009	968.3	677.8	1646.
$M4$	6634897	0.0005	0.01002	0.009832	2.153962	−4.451522	0.00002412	−0.0001683	0.0001925	5	6.92	7285629	5173427	–	–	1646.
$M1$	26539587	0.00025	0.01019	0.009953	2.143691	−4.437741	0.000189	−0.00004713	0.0002362	16	368.	–	–	–	–	–
$M2$	26539587	0.00025	0.01002	0.009822	2.153962	−4.451522	0.00002276	−0.0001777	0.0002004	5	97.9	–	–	–	–	–
$M3$	26539587	0.00025	0.01002	0.009842	2.153962	−4.451522	0.00002355	−0.0001577	0.0001813	5	57.3	29140942	20695579	3872.	2711.	6583.
$M4$	26539587	0.00025	0.01002	0.009843	2.153962	−4.451522	0.0000176	−0.0001571	0.0001747	5	28.5	29145715	20700073	–	–	6584.

Таблица 6.5: Результаты различных методов при $H_0 : N(0,1)$, $H_1 : C(0,1)$ и $\alpha = 0.01, \beta = 0.01$

где $M1$ – итерационный метод, предложенный С. Я. Гродзенским, в данном случае $\Delta_C = 1/5$; $M2$ – итерационный метод, предложенный в данной работе; $M3$ – метод с R-деревом; $M4$ – метод с R-деревом, 6 исполняемых потоков; i – количество итераций; время приведено в секундах; sR_α, sR_β – количество сохраненных точек соответственно в R_α, R_β, причем для $M4$ приведена сумма по sR_{α_1}, sR_{α_2}, sR_{α_3}, и сумма по sR_{β_1}, sR_{β_2}, sR_{β_3}; mR_α, mR_β – используемая в мегабайтах память для хранения соответственно R_α и R_β, для $M4$ указана лишь mAll; mAll – общая используемая в мегабайтах память.

ПРИЛОЖЕНИЕ Ц

M	N	ε	$\hat{\alpha}$	$\hat{\beta}$	lA	lB	Δ_α	Δ_β	$\lvert\Delta_\alpha\rvert + \lvert\Delta_\beta\rvert$	i	время	sR_α	sR_β	mR_α	mR_β	mAll
$M1$	9604	0.01	0.007289	0.009267	4.59512	−4.59512	−0.002711	−0.000733	0.003444	1	0.508	–	–	–	–	–
$M2$	9604	0.01	0.01343	0.01479	4.02073	−4.02073	0.003432	0.004786	0.008217	3	1.09	–	–	–	–	–
$M3$	9604	0.01	0.01343	0.01635	4.02073	−4.02073	0.003432	0.006347	0.009779	3	0.573	18961	14524	2.375	1.875	4.25
$M4$	9604	0.01	0.0101	0.0176	4.02073	−4.02073	0.00009996	0.007597	0.007697	3	0.135	18902	14364	–	–	4.316
$M1$	19699	0.01	0.007158	0.009391	4.59512	−4.59512	−0.002842	−0.0006087	0.003451	1	1.03	–	–	–	–	–
$M2$	19699	0.01	0.01147	0.01604	4.02073	−4.02073	0.001473	0.006041	0.007514	3	2.23	–	–	–	–	–
$M3$	19699	0.01	0.01056	0.01604	4.02073	−4.02073	0.0005589	0.006041	0.0066	3	1.16	38551	29605	5	3.625	8.625
$M4$	19699	0.01	0.01284	0.01538	4.02073	−4.02073	0.002843	0.005381	0.008225	3	0.275	38691	29717	–	–	8.625
$M1$	37843	0.01	0.00769	0.009196	4.59512	−4.59512	−0.00231	−0.0008041	0.003114	1	1.98	–	–	–	–	–
$M2$	37843	0.01	0.01118	0.01572	4.02073	−4.02073	0.001178	0.005723	0.006901	3	4.25	–	–	–	–	–
$M3$	37843	0.01	0.01308	0.01638	4.02073	−4.02073	0.00308	0.006383	0.009464	3	2.27	74594	56754	9.5	6.875	16.38
$M4$	37843	0.01	0.01128	0.01712	4.02073	−4.02073	0.001283	0.007123	0.008407	3	0.567	74406	56931	–	–	16.63
$M1$	60023	0.004	0.0113	0.00998	4.056069	−4.538377	0.001296	−0.00002049	0.001316	2	6.04	–	–	–	–	–
$M2$	60023	0.004	0.008897	0.01273	4.307925	−4.307925	−0.001103	0.002728	0.003832	4	10.2	–	–	–	–	–
$M3$	60023	0.004	0.00913	0.01248	4.307925	−4.307925	−0.0008702	0.002479	0.003349	4	3.64	117596	90487	15.13	10.63	25.75
$M4$	60023	0.004	0.009196	0.01268	4.307925	−4.307925	−0.0008035	0.002678	0.003482	4	0.932	118453	90530	–	–	26.23
$M1$	103671	0.004	0.01134	0.009569	4.195687	−4.484571	0.001344	−0.0004313	0.001775	2	10.9	–	–	–	–	–
$M2$	103671	0.004	0.008865	0.01247	4.307925	−4.307925	−0.001135	0.002472	0.003608	4	17.1	–	–	–	–	–
$M3$	103671	0.004	0.009144	0.01291	4.307925	−4.307925	−0.0008557	0.002906	0.003762	4	7.1	203568	156423	25.75	18.63	44.38
$M4$	103671	0.004	0.009086	0.01193	4.307925	−4.307925	−0.0009136	0.001932	0.002846	4	1.57	203920	156055	–	–	44.74
$M1$	414683	0.002	0.01116	0.01023	4.151238	−4.518301	0.001163	0.0002271	0.00139	2	41.8	–	–	–	–	–
$M2$	414683	0.002	0.01059	0.01069	4.164327	−4.451522	0.0005912	0.0006877	0.001279	5	88.	–	–	–	–	–
$M3$	414683	0.002	0.01107	0.0108	4.164327	−4.451522	0.001074	0.0007986	0.001872	5	24.7	814560	624389	103.8	73.75	177.5
$M4$	414683	0.002	0.01073	0.01101	4.164327	−4.451522	0.0007287	0.001013	0.001742	5	7.94	814662	625664	–	–	177.9
$M1$	1658725	0.001	0.01001	0.01011	4.222874	−4.504228	0.00001492	0.0001054	0.0001203	3	248.	–	–	–	–	–
$M2$	1658725	0.001	0.009931	0.01013	4.236126	−4.523321	−0.00006888	0.0001313	0.0002002	6	441.	–	–	–	–	–
$M3$	1658725	0.001	0.009891	0.01015	4.236126	−4.523321	−0.0001093	0.000153	0.0002622	6	102.	3261140	2496613	415.8	294.5	710.3
$M4$	1658725	0.001	0.009826	0.01001	4.236126	−4.523321	−0.0001744	0.0000113	0.0001857	6	32.6	3260174	2495755	–	–	710.
$M1$	6634897	0.0005	0.01001	0.009984	4.221228	−4.523906	0.00001493	−0.00001597	0.0000309	3	995.	–	–	–	–	–
$M3$	6634897	0.0005	0.009884	0.01004	4.236126	−4.523321	−0.0001159	0.00004175	0.0001577	6	409.	13036003	9989969	1661.	1179.	2839.
$M4$	6634897	0.0005	0.009868	0.009991	4.236126	−4.523321	−0.000132	−0.000009039	0.0001411	6	116.	13039767	9988943	–	–	2841.
$M1$	26539587	0.00025	0.01002	0.009974	4.220284	−4.525217	0.00002469	−0.00002577	0.00005045	3	3980.	–	–	–	–	–
$M4$	26539587	0.00025	0.009848	0.01005	4.236126	−4.523321	−0.0001517	0.0000517	0.0002034	6	408.	52161724	39954050	–	–	11360.

Таблица 6.6: Результаты различных методов при $H_0 : N(0,1)$, $H_1 : L\left(0, \sqrt{3}/\pi\right)$ и $\alpha = 0.01$, $\beta = 0.01$

где $M1$ – итерационный метод, предложенный С. Я. Гродзенским; $M2$ – итерационный метод, предложенный в данной работе; $M3$ – метод с R-деревом; $M4$ – метод с R-деревом, 6 исполняемых потоков; i – количество итераций; время приведено в секундах; sR_α, sR_β – количество сохраненных точек соответственно в R_α, R_β, причем для $M4$ приведена сумма по sR_{α_1}, sR_{α_2}, sR_{α_3}, и сумма по sR_{β_1}, sR_{β_2}, sR_{β_3}; mR_α, mR_β – используемая в мегабайтах память для хранения соответственно R_α и R_β, для $M4$ указана лишь mAll; mAll – общая используемая в мегабайтах память.

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Томский Государственный Университет

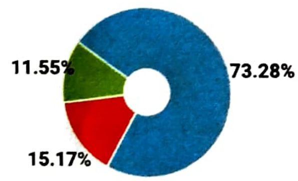
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы:	Васьковский Алексей Сергеевич
Самоцитирование рассчитано для:	Васьковский Алексей Сергеевич
Название работы:	О НЕКОТОРЫХ ВОПРОСАХ УТОЧНЕНИЯ ГРАНИЦ В КРИТЕРИИ ВАЛЬДА
Тип работы:	Выпускная квалификационная работа
Подразделение:	Механико-математический факультет

РЕЗУЛЬТАТЫ

■ **ВНИМАНИЕ, ДОКУМЕНТ ПОДОЗРИТЕЛЬНЫЙ:** ОБНАРУЖЕНЫ ПОПЫТКИ МАСКИРОВКИ ЗАИМСТВОВАНИЙ. РЕКОМЕНДУЕТСЯ ПРОВЕРИТЬ ПОЛНЫЙ ОТЧЕТ

СОВПАДЕНИЯ	15.17%
ОРИГИНАЛЬНОСТЬ	73.28%
ЦИТИРОВАНИЯ	11.55%
САМОЦИТИРОВАНИЯ	0%



ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 10.06.2023

Структура документа:	Проверенные разделы: основная часть с.2, 4-40, приложение с.43-70
Модули поиска:	ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс*; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по коллекции Гарант: аналитика; Переводные заимствования по коллекции Интернет в английском сегменте; Переводные заимствования по Интернету (EnRu); Переводные заимствования по коллекции Интернет в русском сегменте; Переводные заимствования издательства Wiley ; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика; Перефразирования по Интернету; Перефразирования по Интернету (EN); Перефразированные заимствования по коллекции Интернет в английском сегменте; Перефразированные заимствования по коллекции Интернет в русском сегменте; Перефразирования по коллекции

науч. рук. к. ф.-м. н. доцент каф. матем. анализа и теории функций

[подпись] / Емельянова Т.В. /

Работу проверил: Лазарева Елена Геннадьевна

ФИО проверяющего

Дата подписи:

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.