ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В.ЛОМОНОСОВА»

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «МАТЕМАТИЧЕСКОЙ ЛОГИКИ И ТЕОРИИ АЛГОРИТМОВ»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(ДИПЛОМНАЯ РАБОТА)
специалиста

Тесты случайности и псевдослучайные числа: теория и практика
Tests of randomness and randomness extraction: theory and practice

Выполнил студент
604 группы
Виноградов Алексей Владимирович

_____
подпись студента

Научный руководитель:
профессор, д.ф.-м.н.
Верещагин Николай Константинович

_____
подпись научного руководителя

Москва
2018

Содержание

1 Введение

Случайные числа имеют большое количество применений в различных сферах жизни: от криптографии до компьютерных игр и лотерей. Существует много способов их получения: подбрасывание монеток и игральных костей, лототроны, генераторы псевдослучайных чисел (ГПСЧ), аппаратные генераторы «истинно» случайных чисел. Последние генерируют последовательность случайных чисел на основе измерения хаотически изменяющихся параметров физических процессов: теплового шума, шума аудиокарты, квантовых эффектов, и других.

Для проверки качества источников случайности используются статистические тесты. В данной работе изложена идея корректировки тестов случайности [1]. Кроме того, представлены способ и результаты реализации этой идеи на практике на основе популярного набора тестов *dieharder* [2].

2 Тесты случайности

Пусть некоторый источник (генератор) выдал строку s длины n ($n \in \mathbb{N}$). Назовем источник (истинно) случайным, если $\forall n \in \mathbb{N}$ s распределено равномерно, т.е. существует равная вероятность $1/2^n$ встретить на месте s любую строку длины n.

2.1 Первичные тесты

Простейший статистический тест случайности, получающий на вход строки длины $n \in \mathbb{N}$, с нулевой гипотезой «все строки длины n равновероятны» устроен следующим образом. Пусть $F \subset \{0,1\}^n$ - небольшое множество, выбранное до проведения измерения (запуска ГПСЧ). Если полученная от источника последовательность попала в множество F, то нулевая гипотеза отвергается. Если распределение на строках равномерно, то вероятность попадания в множество F равна $|F|/2^n$, это число называется уровнем значимости теста. Также это вероятность ошибки первого рода - отвергнуть верную гипотезу. Эта вероятность должна быть мала, чтобы тест имел смысл.

В общем случае, в условиях нулевой гипотезы можно рассмотреть $\mathcal{P} = (\{0,1\}^n, \Omega, P)$ — дискретное вероятностное пространство с сигма алгеброй всех подмножеств и вероятностью случайного исхода $P(\omega) = 1/2^n, \omega \in \{0,1\}^n$. Пусть $\xi : \mathcal{P} \to \mathbb{R}$ — случайная величина на этом пространстве. Тогда для каждого случайного исхода s определено:

$$p_{value}(s) = P(\omega | \xi(\omega) \geq \xi(s)).$$

Для любого $\varepsilon > 0$ мы получаем множество $T_\varepsilon = \{\omega \in \{0,1\}^n | p_{value}(\omega) \leq \varepsilon\}$. При этом, вероятность попадания в множество T_ε равна:

$$P(T_\varepsilon) = \max\{p_{value}(\omega) | p_{value}(\omega) \leq \varepsilon\}.$$

Теперь, предварительно выбрав нужное $\varepsilon > 0$, мы можем посчитать значение p_{value} для s — строки битов, выданной генератором. Далее, если $p_{value}(s) \leq \varepsilon$, нулевая гипотеза отвергается, с вероятностью не менее $(1 - \varepsilon)$ источник не является случайным. Таким образом, построен тест $\mathcal{T} = (\xi, \varepsilon)$.

В качестве примера рассмотрим тест $\mathcal{T}_0 = (\mu, \varepsilon)$, где μ - случайная величина, равная частоте единиц в последовательности длины n. Пусть $h = \mu(s)$ — частота единиц в выпавшей

строке s. Тогда p_{value} можно посчитать следующим образом:

$$p_{value}(s) = \sum_{k=h\times n}^{n} \frac{C_n^k}{2^n}$$

Если p_{value} оказывается меньше выбранного нами уровня значимости ε, то тест не пройден, и источник с вероятностью не менее $(1-\varepsilon)$ не является случайным.

Стоит отметить, что если источник прошел отдельный тест, это не означает, что он в действительности является случайным. Тест из предыдущего примера будет с легкостью пройден очевидно неслучайным источником, выдающим строки, чередующие нули и единицы (0101010101...). Конкретный тест обычно направлен на выявление неслучайностей определенного типа и не может в одиночку оценить качество источника случайности. Поэтому в наиболее известных программах тестирования случайных чисел (*diehard* by G. Marsaglia [4] и ее наследник: *dieharder* by Robert G Brown [2]) используются наборы разнообразных тестов. Каждый из них выдает свой «вердикт» — p_{value}. И только на основе результатов большого количества тестов мы можем судить о качестве источника случайности.

Пусть источником порождена строка длины $100n$ и тест $\mathcal{T}$ запущен 100 раз на последовательных не пересекающихся кусках этой строки длины n каждый. Допустим, все 100 раз тест был пройден, но все p_{value} оказались между 0.5 и 0.6. Такой исход выглядит довольно подозрительным. Для проверки подобных ситуаций используют вторичные тесты.

2.2 Вторичные тесты

Рассмотрим первичный тест (ξ, ε) и случайную величину:

$$\psi = p_{value}(\xi).$$

Тогда, как было показано выше:

$$P(\psi \le x) = P(T_x) = \max\{p_{value}(\omega) | p_{value}(\omega) \le x\}.$$

Поскольку множество значений ξ дискретно, функция распределения ψ ступенчатая, при этом высоты скачков равны вероятностям отдельных значений случайной величины ξ, а значения в точках скачков совпадают со значениями функции распределения равномерномерной случайной величины. Таким образом, если у ξ нет значений, принимаемых с большой вероятностью (например константной по n), то распределение ψ близко к равномерному. Случайная величина μ из нашего примера принимает наиболее вероятное значение $n/2$ (для четных n) с вероятностью:

$$P(\mu = n/2) = \frac{C_n^{n/2}}{2^n} \sim \sqrt{\frac{2}{\pi n}}, n \to \infty$$

В данном случае, взяв достаточно большое n, можно обеспечить близость распределения случайной величины ψ (в условиях нулевой гипотезы) и теоретического равномерного распределения.

Если расстояние между двумя распределениями удалось сделать достаточно малым, используют вторичный тест, проверяющий, насколько эмпирическое распределение случайной величины ψ похоже на равномерное. Для этого в тестах *dieharder* на вход подается pn бит. Далее, первичный тест запускается p раз на непересекающихся, а значит, предположительно,

независимых n-битных кусках входной последовательности. По выборке из полученных p_{value} строится эмпирическая функция распределения и сравнивается с теоретической равномерной с помощью критерия согласия Колмогорова [1] .

Итак, общая идея тестирования довольно проста. Однако, при реализации программы, проводящей тестирование источников случайности, возникают трудности. Рассмотрим их на примере набора тестов *dieharder*.

3 Тесты *dieharder*

Запуск *dieharder* в базовом режиме позволяет проверить встроенный или пользовательский генератор случайных чисел на нескольких десятках различных тестов. Также существует возможность брать данные из файла или со стандартного ввода. Часть тестов с исправлениями и дополнениями унаследована из набора *diehard*, также представлены их обобщения. Кроме того, добавлены тесты из набора STS [3] . Результат работы каждого теста — финальный p_{value}, полученный с помощью критерия согласия Колмогорова. Рассматривается значение

$$\min\{p_{value}, 1 - p_{value}\}$$

и сравнивается с двумя уровнями значимости. В зависимости от результата выдается оценка: PASSED, WEAK или FAILED.

3.1 Корректность тестов

В любой программе, которая тестирует источники случайности все тесты должны быть реализованы корректно. А именно, финальное значение p_{value} должно быть посчитано настолько точно, чтобы фактическая вероятность ошибки первого рода не превышала уровня значимости, заданного в тесте. В противном случае, тест может начать систематически отклонять хорошие источники случайности, что недопустимо. Изучив принцип работы тестов в *dieharder*, приходим к выводу, что для большинства из них корректность по крайней мере не доказана.

Во-первых, как было упомянуто выше, для всех тестов, в случае верности нулевой гипотезы, распределение p_{value} отличается от равномерного. Однако, отклонение обычно можно сделать маленьким по сравнению с разницей между эмпирическим и теоретическим распределением в критерии Колмогорова.

Во-вторых, во многих тестах *dieharder* распределение базовой статистики в точности неизвестно. В некоторых тестах (filltree, feeltree2, opso, oqso, tsang_gsd и др.) для подсчета p_{value} используются статистические параметры, которые были получены эмпирически

[1] Существует два типа тестов Колмогорова-Смирнова: с одной и двумя выборками. В английском языке используется дополнительное уточнение: one (two) sample KS test. В русском языке тип теста обычно понимается из контекста. Поскольку в данной работе речь пойдет об обоих типах, здесь и далее будем обозначать:

- критерий (согласия) Колмогорова — критерий проверки гипотезы о принадлежности выборки некоторому теоретическому закону распределения (one-sample KS test);
- критерий (однородности) Смирнова — критерий проверки гипотезы о принадлежности двух независимых выборок одному закону распределения (two-sample KS test).

на основе данных «хороших» (по мнению общественности) ГПСЧ [2]. Если на самом деле выбранный для этих целей генератор окажется неслучайным, тест может систематически отвергать по-настоящему случайные генераторы (если таковые имеются). Другие тесты (birthdays, min_distr, 2d_sphere, 3d_sphere) используют асимптотические статистики для подсчета p_{value}. Ни тот, ни другой подход не дает оснований доверять финальным значениям p_{value} вторичных тестов. Для первичных тестов мы могли бы оценивать p_{value} сверху, тем самым уменьшая их чувствительность, но сохраняя корректность, для вторичных тестов таких оценок недостаточно.

Некоторые тесты (OPSO, OQSO, DNA) по признанию автора *dieharder* систематически отвергают «хорошие» ГПСЧ и потому помечены в списке тестов как подозрительные.

3.2 Чувствительность и уникальность

Предположим теперь, что новый тест реализован корректно. Для того чтобы принять корректный тест в уже существующий набор тестов, разумно требовать выполнения им следующих условий:

1. Чувствительность. Тест хорошо выявляет тот тип неслучайностей, для борьбы с которым он был предназначен;

2. Уникальность. Тест находит новые неслучайности по сравнению с уже имеющимися тестами в наборе.

Ряд тестов (diehard_squeeze, diehard_count_1s_stream) по мнению автора *dieharder* оказываются нечувствительными к уже известным неслучайностям некоторых ГПСЧ [3].

В базовом наборе есть целые группы схожих тестов, [4] более того, существуют тесты, которые могут быть либо пройдены, либо не пройдены одновременно. [5] Помимо этого, некоторые тесты из набора diehard объединены в более общие тесты [6] , несмотря на это, в базовый набор включены и те, и другие. Все выше перечисленное говорит о том, что тесты в базовом наборе *dieharder* не являются уникальными и их количество можно сильно сократить.

Итак, несмотря на то, что тесты *dieharder* достаточно разнообразны, у нас недостаточно оснований доверять их результатам. Попробуем улучшить эти тесты. Предположим, что у нас есть источник случайных чисел, которому мы можем доверять (т.е. для которого на самом деле верна нулевая гипотеза). Настроим тесты с помощью этой «истинной» случайности.

[2][marsaglia_tsang_gcd test] four «good» RNGs (gfsr4,mt1993_1999,rndlxs2,taus2) were used to construct a simulated table of high precision probabilities for k a process that obviously begs the question as to whether or not THESE generators are «good».

[3]from my limited experimentation, this [diehard_squeeze] test is uselessly insensitive on at least the rng's in the GSL with a few notable exceptions (the worst of the worst).

[4]The tests BITSTREAM, OPSO, OQSO and DNA are all closely related.

[5]diehard_count_1s_stream test will fail whenever rgb_bitdist fails at 40 bits and pass when rgb_bitdist passes

[6]generalized minimum distance test ... utilizes correction terms that are essential in order for the test not to fail for large numbers of trials. It replaces both diehard_2dsphere and diehard_3dsphere

4 «Истинная» случайность

Если надежный источник случайности существует, его можно использовать для исправления и улучшения имеющегося набора тестов.

4.1 Проверка корректности существующих тестов

Запустим тест на «истинно» случайных данных. Если тест их систематически отвергает, то он очевидно не является корректным. Такой проверкой можно было бы отсеять некорректные тесты из списка.

4.2 Улучшение тестов

Допустим, мы хотим оценить качество работы генератора случайных чисел при помощи некоторого теста. Для этого можно сравнить результаты работы теста на битах генератора и битах независимого от него надежного источника случайности. А именно, пусть две строки из pn и qn битов сгенерированы соответственно тестируемым и эталонным источниками случайности. Построим две выборки из p и q элементов — значения статистики теста на непересекающихся n-битных кусках первой и второй строки соответственно. Если для тестируемого генератора верна нулевая гипотеза, то значения в обоих полученных выборках распределены одинаково. Для проверки гипотезы о принадлежности выборок одному закону распределения можно использовать например критерий однородности Смирнова.

Плюсом такого подхода является отсутствие необходимости точного подсчета p_{value}. Теперь необязательно знать распределение статистики теста при условии выполнения нулевой гипотезы. Достаточно проверить, что эмпирическое распределение тестируемого генератора похоже на эмпирическое распределение надежного источника случайности.

Впрочем, у приведенного выше метода существуют и недостатки: увеличение необходимого количества данных и времени работы (теперь программа должна посчитать значение статистики не на одной, а на двух выборках). Кроме того, для одного и того же уровня приближения потребуется больший размер выборок.

Таким образом, если мы имеем «истинный» источник случайности, то преобразованные тесты, использующие данные этого источника как эталон, становятся корректными (с точностью до реализации критерия Смирнова). Однако, даже существование такого источника не доказано, и поэтому эталонным вполне может оказаться плохой источник случайности. В таком случае наш тест может начать систематически принимать плохие источники и, главное, отвергать хорошие, т.е. станет некорректным тестом. Для того чтобы избежать этой ситуации, можно воспользоваться следующим трюком. Выше для получения элементов двух финальных выборок в критерии Смирнова мы использовали p непересекающихся кусков $(x_1, \dots x_p)$ от битовой последовательности тестируемого генератора и q кусков $(e_1, \dots, e_q)$ от битовой последовательности эталонного источника. Теперь рассмотрим удлиненную строку и набор ее кусков $(x_1, \dots x_p, x_{p+1}, \dots x_{p+q})$. Запустим описанный выше двухвыборочный тест на $(x_1, \dots x_p)$ и $(x_{p+1} \oplus e_1, \dots x_{p+q} \oplus e_q)$ (здесь «$\oplus$» — побитовый XOR).

Утверждение 1. *Полученный тест является корректным, если тестируемый и эталонный источники независимы.*

Доказательство. Действительно, предположим, что тестируемый источник истинно случайный. Тогда строки $(x_1, \dots x_p)$ и $(x_{p+1}, \dots x_{p+q})$ независимы и равномерно распределены. Кроме того, из независимости источников следует, что $(e_1, \dots, e_q)$ и $(x_{p+1}, \dots x_{p+q})$ независимы. Значит их XOR равномерно распределен (используется свойство XOR сохранять равномерность). Кроме того, результат XOR не зависит от $(x_1, \dots x_p)$. Далее тест превращается в первоначальный двухвыборочный тест, проверяющий верную гипотезу, используя надежный эталонный источник случайности. Поэтому, оба теста отвергнут верную гипотезу с одинаковой вероятностью, уровни значимости в них тоже совпадают. Первоначальный тест с истинно случайным эталонным источником корректен, значит и новый тест тоже является корректным. □

Следствие 1. *Тест остается корректным для любых фиксированных (заранее вычисленных) эталонных данных, так как в таком случае независимость гарантирована.*

Таким образом, используя в независимые эталонные данные (какими бы плохими они ни были), мы можем больше не беспокоиться о корректности модифицированного теста.

Если же строка $(e_1, \dots, e_q)$ порождена независимым истинно случайным источником, то результат XOR останется равномерно распределенным и будет использован в качестве эталонного. Тогда новый тест превратится в первоначальный двухвыборочный тест с истинным источником случайности (теперь уже вне всяких предположений о тестируемом источнике). В этом случае, при переходе к модифицированному тесту мы не потеряем в чувствительности, поэтому чтобы сохранить все преимущества обычного двухвыборочного теста, в качестве эталонного все равно стоит выбирать наиболее качественный источник случайности.

Ниже будут описаны процесс и результаты реализации идеи корректировки тестов на основе открытого кода программы *dieharder*. Но прежде чем запускать двухвыборочный тест, нужно понять, какие данные принять за эталон.

5 Выбор надежного источника случайности

Наборы тестов во многом важны своим практическим применением для проверки качества генераторов псевдослучайных чисел. Поэтому интересно рассмотреть результаты работы ГПСЧ в сравнении с данными полученными на основе природных источников случайности. Большинство источников, использующих энтропию окружающей среды, работают крайне медленно. При этом, для запуска базового набора тестов *dieharder* требуется около 15 гигабайт данных, поэтому в качестве эталонных были взяты преобразованные данные записи аудиокарты, как один из наиболее быстрых и легко осуществимых методов. Данные с аудиокарты записывались в 2 потока с максимальной частотой дискретизации — 192кГц и максимальной битовой глубиной — 24 бита. В результате запись шла со относительно неплохой скоростью порядка 1 мБ в секунду. Впрочем, полученные данные перед дальнейшим использованием в тестах, конечно, нуждаются в обработке и «просеивании», что приводит к уменьшению их объема. Во время записи, насколько это возможно, поддерживалась субъективная тишина (сведение к минимуму громких звуков с выделенной частотой).

5.1 Последние байты

На первом этапе данные были прорежены: из каждой 24-битной записи были оставлены наименее значимые, младшие 8 бит. Можно предположить, что в полученных данных еще могут

присутствовать зависимости: между близкими по времени записи байтами одного канала, между битами внутри одного байта, и т. д. Однако, по результатам тестирования этих данных на базовом наборе тестов оказывается, что они уже достаточно хороши (см. last bytes таблицы (5) в конце работы).

Увидев такие хорошие результаты, можно было бы остановить поиски и взять за эталон младшие байты записи аудиокарты. Однако, как говорилось выше, нет оснований безоговорочно доверять тестам. Некоторые простые закономерности в данных вполне могли быть пропущены из-за нечувствительности тестов. В поисках подобных неслучайностей нужно исходить из способа получения данных. Необходимо стараться обезопасить себя и изменить данные так, чтобы, по возможности, исключить все потенциальные закономерности, которые можно было бы заподозрить, исходя из природы источника случайности.

Для дополнительного перемешивания данных использовался следующий алгоритм.

5.2 Случайная таблица

Рассмотрим таблицу из $[256 \times 256]$ битов, причем в каждой полосе (строке или столбце) у нее равное количество нулей и единиц.

Такую таблицу можно построить,с помощью некоторого массива данных. Таблица заполняется исходными данными. Далее на каждом следующем шаге в таблице проверяется соотношение количества нулей и единиц во всех полосах. Если везде установилось равенство, процесс заканчивается. В противном случае считывается бит для выбора типа полосы (строки/столбца) и байт для определения ее номера номера. Далее в полосе проверяется соотношение единиц и нулей. Если их не поровну, выбирается позиция в полосе и меняется бит в сторону уменьшения разницы между нулями и единицами в этой полосе. Эти действия выполняются до тех пор, пока во всех полосах не установится равенство. Для генерации таблицы использовались случайные числа, полученные G. Marsaglia [4].

Преобразуем данные аудиокарты с использованием полученной таблицы. Для этого будем каждый раз брать пару последних байт из соответствующих 24-битных записей двух аудио потоков. По этим двум байтам определяем место в таблице и записываем соответствующий бит в выходную последовательность.

Такое табличное преобразование использует идею построения двухвходого экстрактора, но никакого теоретического обоснования не имеет. Слабым местом такого подхода является возможная зависимость соседних байтов в одном аудио потоке.

Тем не менее, полученные таким образом данные также были протестированы на базовом наборе оригинальных тестов dieharder. В результате все тесты оказались пройдены (см. (5)).

Такое может случиться в одном из двух случаев. Либо полученный источник является истинно случайным и тесты *dieharder* выглядят корректными (не отвергают его систематически), либо источник не случаен, но тесты не чувствительны к этому типу неслучайности. Если мы возьмем этот источник в качестве эталонного для двухвыборочного теста, то в первом случае это полностью оправдано, а во втором, мы все равно не сможем доказать, что какой-то другой источник качественнее (без написания новых тестов). Поэтому при реализации двухвыборочного теста в качестве эталонных возьмем именно данные, полученные описанным только что способом.

6 Реализация

За основу реализации двухвыборочных тестов был взят открытый код программы *dieharder* на языке C. Теперь если через командную строку передан флаг --two-sample, выбран тестируемый генератор и передано имя файла с эталонными данными. Программа запускает набор из p первичных тестов на n-битных кусках последовательности порожденной тестируемым генератором и $(p-1)$ первичных тестов с теми же параметрами на n-битных отрезках данных файла. Если через командную строку был передан флаг --xor, то вместо чистых эталонных данных используется их XOR с еще не использованными отрезками последовательности, полученной тестируемым генератором. Как говорилось выше, в оригинальном *dieharder* каждый первичный тест возвращает одно число p_{value}, набор из которых используется в финальном тесте Колмогорова проверки равномерности распределения. Теперь первичные тесты возвращают еще и $s_{value} = \xi(s)$, где ξ — случайная величина первичного теста, s — строка, полученная на вход теста (значение s_{value} является промежуточным, как раз по нему потом в первичном тесте считается p_{value}). После окончания работы первичных тестов, если $p = 1$, выдается p_{value}, посчитанное в единственном первичном тесте. Если же $p > 1$, для двух полученных выборок s_{value} размера p и $(p-1)$ соответственно запускается тест однородности Смирнова.

6.1 Критерий однородности Смирнова

Используется двусторонний тест однородности Смирнова, реализованный на основе теста default two-sided two-sample ks.test библиотеки dgof языка R [5]. Пусть p и q — размеры выборок. Тогда финальное p_{value} подсчитывается точно при $pq < 10000$ и по ассимптотической формуле, в противном случае.

6.2 Процесс тестирования

На всех встроенных ГПСЧ *dieharder* был запущен базовый набор оригинальных тестов. Исходя из полученных результатов было выбрано несколько показательных генераторов:

- «Плохой»: transputer (PASSED: 47, WEAK: 2, FAILED: 65).
- «Плохой»: vax (PASSED: 58, WEAK: 2, FAILED: 54).
- «Средний»: rand48 (PASSED: 110, WEAK: 0, FAILED: 4);
- «Хороший»: mt19937_1999 (PASSED: 112, WEAK: 2, FAILED: 0).

Для них был дважды запущен набор новых двухвыборочных тестов (с использованием XOR и без него). В качестве эталонных брались данные, полученные при помощи таблицы из последних байтов аудиозаписи в стерео режиме [192 кГц, 24 бит] (именно те данные, что прошли все оригинальные тесты *dieharder*).

6.3 Результаты

Назовем вторичные тесты родственными, если они используют один и тот же первичный тест. Для каждого из выбранных генераторов в результате тестирования было получено по 3×114

p_{value} (114 — число первичных тестов в базовой наборе). Ниже представлены графики, сравнивающие результаты работы родственных тестов на каждом генераторе в отдельности. По оси абсцисс отложены p_{value} оригинальных тестов, по оси ординат — p_{value} двухвыборочных тестов (xor или regular). Каждому первичному тесту на графиках сопоставляются два числа — p_{value} оригинального и соответствующего двухвыборочного тестов, использовавших этот первичный тест для получения выборки. В результате каждому первичному тесту, пройденному генератором будет соответствовать точка на графике.

Для того, чтобы лучше было заметно поведение старых и новых тестов при p_{value} близких к нулю (единице), вместо самого значения p_{value} отображается десятичный логарифм

$$\log_{10}(\min\{p_{value}, 1 - p_{value}\}).$$

Dieharder выдает значения p_{value} с точностью 8 знаков после запятой, Коричневая линия на графике соответствует границе, ниже (левее) которой все фактические p_{value} равны нулю. Для наглядности, точки для которых хотя бы одно из p_{value} равно нулю, разнесены на небольшое случайное расстояние, чтобы они полностью не совпадали, и можно было бы оценить их количество.

Далее, красным обозначена используемая в *dieharder* граница FAILED:

$$\min\{p_{value}, 1 - p_{value}\} = 10^{-6},$$

соответствующая уровню значимости $\varepsilon = 2 \times 10^{-6}$. В свою очередь, оранжевым обозначена граница WEAK:

$$\min\{p_{value}, 1 - p_{value}\} = 0.005,$$

соответствующая уровню значимости $\varepsilon = 0.01$.

Общее наблюдения для всех генераторов:

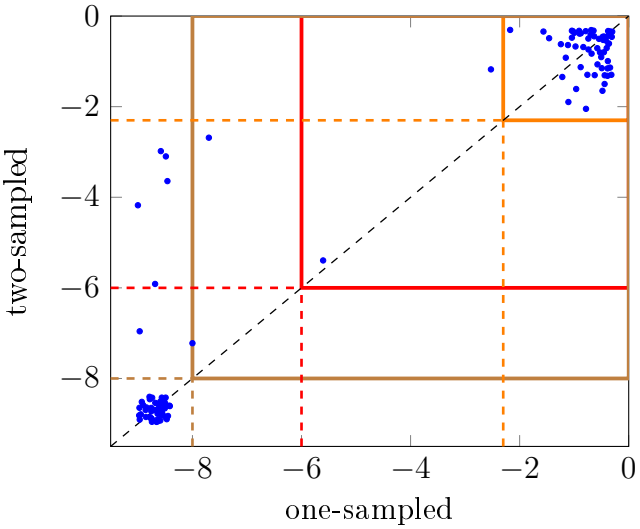
1. Генераторы ведут себя одинаково на большинстве родственных тестов. А именно, почти все тесты либо одновременно пройдены (правый верхний оранжевый квадрат), либо одновременно не пройдены (левый нижний коричневый пунктирный квадрат);

2. Большинство точек, не лежащих в этих квадратах оказывается выше главной диагонали. Это наблюдение подтверждает, что оригинальный, одновыборочный тест должен быть более чувствителен, чем двухвыборочные тесты при одинаковых размерах выборок.

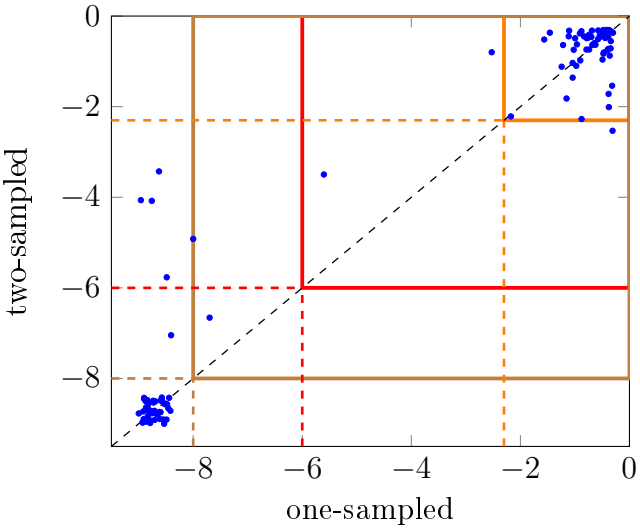
Теперь дополнительно изучим каждый генератор в тех точках, в которых ситуация осталась неопределенной (хотя бы один из двух p_{value} оказался WEAK). Будем запускать оба теста, соответствующие каждой такой точке используя все бОльшую длину выборки p. Это дополнительное тестирование заканчивается, как только тест выдаст PASSED или FAILED. Для каждого генератора результаты дополнительного тестирования приведены в соответствующих таблицах.

Рис. 1: Генератор vax

(a) xor



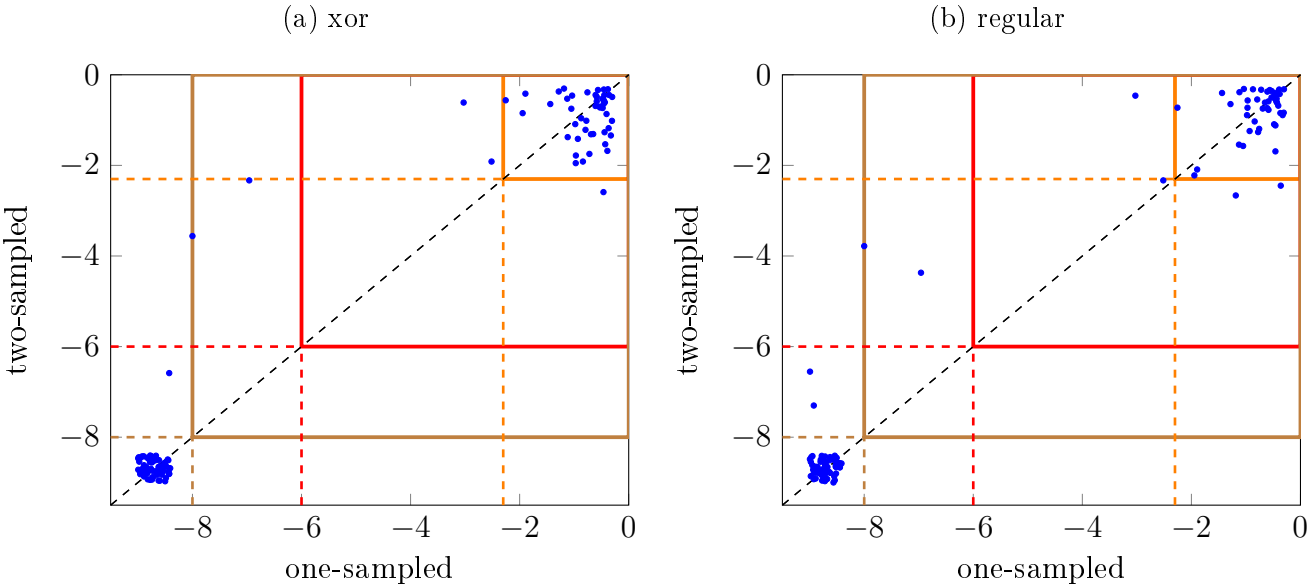
(b) regular



				original one-sample		xor two-sample		regular two-sample	
test name	test type	n	p	pvalue	assessment	pvalue	assessment	pvalue	assessment
sts_serial	2	100000	100	0.00000000	FAILED	0.00104392	WEAK	0.00008327	WEAK
	2	100000	200			0.00000062	FAILED	0.00000305	WEAK
	2	100000	300					0.00000000	FAILED
	3	100000	100	0.00000000	FAILED	0.00206506	WEAK	0.00000000	FAILED
	3	100000	200			0.00000000	FAILED		
	3	100000	100	0.00000002	FAILED	0.00006660	WEAK	0.00000022	FAILED
	3	100000	200			0.00000020	FAILED		
	4	100000	100	0.00000000	FAILED	0.00022726	WEAK	0.00037206	WEAK
	4	100000	200			0.00000004	FAILED	0.00000004	FAILED
	5	100000	100	0.00000000	FAILED	0.00000011	FAILED	0.00008637	WEAK
	5	10000	200					0.00000000	FAILED
rgb_bitdist	9	100000	100	0.00000000	FAILED	0.00079754	WEAK	0.00000009	FAILED
	9	100000	200			0.00010802	WEAK		
	9	100000	300			0.00000447	WEAK		
	9	100000	400			0.00000001	FAILED		
	10	100000	100	0.00297772	WEAK	0.06648670	PASSED	0.15889036	PASSED
	10	100000	200	0.00000164	WEAK	0.00006687	WEAK	0.00243367	WEAK
	10	100000	300	0.00000009	FAILED	0.00000695	WEAK	0.00009907	WEAK
	10	100000	400			0.00000004	FAILED	0.00000027	FAILED
rgb_mindist	2	10000	1000	0.00000001	FAILED	0.00000006	FAILED	0.00001204	WEAK
	2	10000	1100					0.00002259	WEAK
	2	10000	1200					0.00001850	WEAK
	2	10000	1300					0.00000592	WEAK
	2	10000	1400					0.00000450	WEAK
	2	10000	1500					0.00000544	WEAK
	2	10000	1600					0.00000016	FAILED
	4	10000	1000	0.00000000	FAILED	0.00000122	WEAK	0.00000171	WEAK
	4	10000	1100			0.00000046	FAILED	0.00000009	FAILED
	5	10000	1000	0.00000249	WEAK	0.00000403	WEAK	0.00031849	WEAK
	5	10000	1100	0.00000082	FAILED	0.00009744	WEAK	0.00014186	WEAK
	5	10000	1200			0.00002964	WEAK	0.00005238	WEAK
	5	10000	1300			0.00000899	WEAK	0.00002814	WEAK
	5	10000	1400			0.00000488	WEAK	0.00003703	WEAK
	5	10000	1500			0.00000067	FAILED	0.00000218	WEAK
	5	10000	1600					0.00000055	FAILED
rgb_lagged_sum	18	1000000	100	0.51057436	PASSED	0.34723854	PASSED	0.99706176	WEAK
	18	1000000	200					0.97566729	PASSED

Таблица 1: Таблица дополнительного тестирования генератора vax

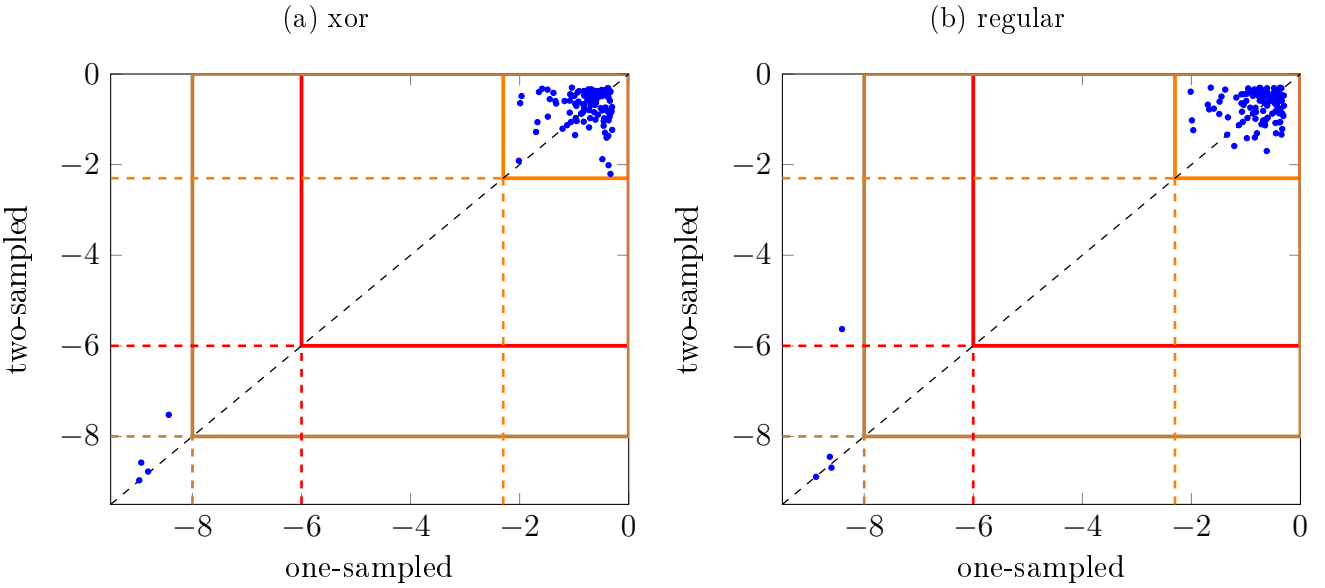
Рис. 2: Генератор transputer



				original one-sample		xor two-sample		regular two-sample	
test name	test type	n	p	pvalue	assessment	pvalue	assessment	pvalue	assessment
diehard_birthdays	0	100	100	0.00000001	FAILED	0.00027546	WEAK	0.00016536	WEAK
	0	100	200			0.00000122	WEAK	0.00000000	FAILED
	0	100	300			0.00000000	FAILED		
diehard_2dsphere	2	8000	100	0.00000011	FAILED	0.00466187	WEAK	0.00004271	WEAK
	2	8000	200			0.00008219	WEAK	0.00000000	FAILED
	2	8000	300			0.00000026	FAILED		
rgb_lagged_sum	6	1000000	100	0.56482504	PASSED	0.64133876	PASSED	0.00356811	WEAK
	6	1000000	200					0.0570784	PASSED
	8	1000000	100	0.65551882	PASSED	0.99742932	WEAK	0.70728762	PASSED
	8	1000000	200			0.50074159	PASSED		
	26	1000000	100	0.99906122	WEAK	0.24381396	PASSED	0.34587801	PASSED
	26	1000000	200	0.48123189	PASSED				
	31	1000000	100	0.00305116	WEAK	0.01215037	PASSED	0.00466191	WEAK
	31	1000000	200	0.00002099	WEAK	0.00030402	WEAK	0.00069002	WEAK
	31	1000000	300	0.00000010	FAILED	0.00008685	WEAK	0.00000591	WEAK
	31	1000000	400			0.00010060	WEAK	0.00000024	FAILED
	31	1000000	500			0.00000298	WEAK		
	31	1000000	600			0.00000012	FAILED		
	32	1000000	100	0.06513085	PASSED	0.49594738	PASSED	0.00216762	WEAK
								0.06004558	PASSED

Таблица 2: Таблица дополнительного тестирования генератора transputer

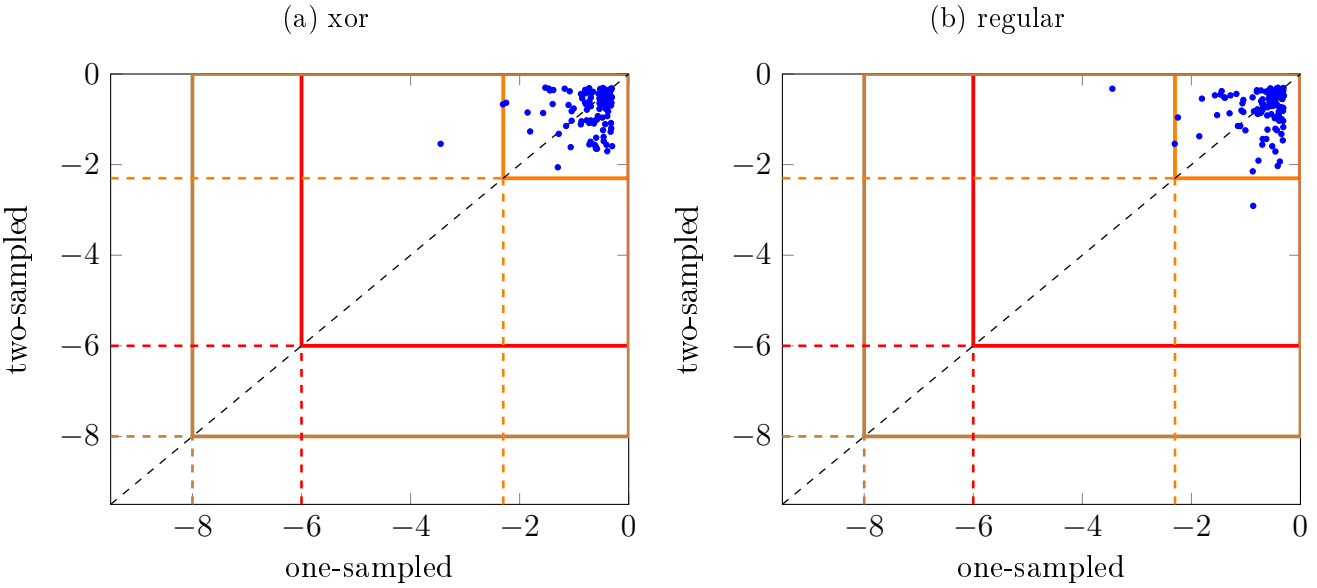
Рис. 3: Генератор rand48



test name	test type	n	p	original one-sample pvalue	assessment	xor two-sample pvalue	assessment	regular two-sample pvalue	assessment
diehard_dna	0	2097152	100	0.00000000	FAILED	0.00000003	FAILED	0.00000234	WEAK
								0.00000000	FAILED

Таблица 3: Таблица дополнительного тестирования генератора rand48

Рис. 4: Генератор mt19937_1999



				original one-sample		xor two-sample		regular two-sample	
test name	test type	n	p	pvalue	assessment	pvalue	assessment	pvalue	assessment
diehard_opso	0	2097152	100	0.13600761	PASSED	0.90976944	PASSED	0.00122795	WEAK
	0	2097152	200					0.23960016	PASSED
rgb_lagged_sum	17	1000000	100	0.99964431	WEAK	0.97566729	PASSED	0.52637947	PASSED
	17	1000000	200	0.79400432	PASSED				
	30	1000000	100	0.99504985	WEAK	0.78559719	PASSED	0.97122355	PASSED
	30	1000000	200	0.87680649	PASSED				

Таблица 4: Таблица дополнительного тестирования генератора mt19937_1999

По итогам дополнительной проверки видно, что нет ни одной пары родственных тестов, для которых интенсивная проверка привела бы к различным результатам (получилось, что либо во всех трех случаях результат PASSED, либо во всех трех случаях — FAILED). С другой стороны, заметно, что для получения окончательного вердикта двухвыборочным тестам, как правило, нужны более длинные выборки.

7 Заключение

В данной работе был изучен новый подход в тестировании источников случайных чисел, позволяющий быть уверенным в корректности результатов тестов. Далее он был реализован на практике на основе набора тестов *dieharder*. В итоге удалось получить корректный набор двухвыборочных тестов. Результаты тестирования каждого из четырех ГПСЧ с помощью оригинальных и новых тестов оказались достаточно похожи. Однако дальнейшее их сравнение показало, что двухвыборочные тесты являются (ожидаемо) менее чувствительными. Тем не менее, при увеличении размера выборки в двухвыборочном тесте, результаты тестирования фактически совпадают. Для запуска двухвыборочных тестов использовались младшие байты записи аудиокарты, обработанные при помощи случайной таблицу. Однако текущая модификация кода *dieharder* позволит в дальнейшем экспериментировать, используя разнообразные данные в качестве эталонных.

h

Таблица 5: Результаты тестирования последних байтов и результатов использования таблицы на базовом наборе тестов *dieharder*

				last bytes		random table	
test name	test type	n	p	pvalue	assessment	pvalue	assessment
diehard_birthdays	0	100	100	0.59704751	PASSED	0.91048792	PASSED
diehard_operm5	0	1000000	100	0.06268735	PASSED	0.96510084	PASSED
diehard_rank_32x32	0	40000	100	0.05711303	PASSED	0.79651562	PASSED
diehard_rank_6x8	0	100000	100	0.38706425	PASSED	0.83012762	PASSED
diehard_bitstream	0	2097152	100	0.34112424	PASSED	0.57517285	PASSED
diehard_opso	0	2097152	100	0.23786175	PASSED	0.12747239	PASSED
diehard_oqso	0	2097152	100	0.6113898	PASSED	0.46222722	PASSED
diehard_dna	0	2097152	100	0.15125412	PASSED	0.22245481	PASSED
diehard_count_1s_str	0	256000	100	0.16661297	PASSED	0.91815776	PASSED
diehard_count_1s_byt	0	256000	100	0.92057893	PASSED	0.3932696	PASSED
diehard_parking_lot	0	12000	100	0.54134122	PASSED	0.79488513	PASSED
diehard_2dsphere	2	8000	100	0.31472646	PASSED	0.97993157	PASSED
diehard_3dsphere	3	4000	100	0.8796619	PASSED	0.90594647	PASSED
diehard_squeeze	0	100000	100	0.49266232	PASSED	0.88236123	PASSED
diehard_sums	0	100	100	0.14973035	PASSED	0.51609116	PASSED
diehard_runs	0	100000	100	0.05207538	PASSED	0.11494505	PASSED
diehard_runs	0	100000	100	0.8340073	PASSED	0.52942308	PASSED
diehard_craps	0	200000	100	0.84402954	PASSED	0.17199293	PASSED
diehard_craps	0	200000	100	0.60416821	PASSED	0.75068905	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.15170163	PASSED	0.32597627	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.85067283	PASSED	0.74776691	PASSED
sts_monobit	1	100000	100	0.30561397	PASSED	0.07630919	PASSED
sts_runs	2	100000	100	0.00473648	WEAK	0.13825694	PASSED
sts_serial	1	100000	100	0.45026012	PASSED	0.07630919	PASSED
sts_serial	2	100000	100	0.71727337	PASSED	0.05009488	PASSED
sts_serial	3	100000	100	0.52754826	PASSED	0.76424704	PASSED
sts_serial	3	100000	100	0.5131546	PASSED	0.18908039	PASSED
sts_serial	4	100000	100	0.90731281	PASSED	0.70963343	PASSED
sts_serial	4	100000	100	0.94289569	PASSED	0.59637656	PASSED
sts_serial	5	100000	100	0.01089412	PASSED	0.39024758	PASSED
sts_serial	5	100000	100	0.29484439	PASSED	0.57189941	PASSED
sts_serial	6	100000	100	0.89352244	PASSED	0.93486206	PASSED
sts_serial	6	100000	100	0.03736523	PASSED	0.98285148	PASSED
sts_serial	7	100000	100	0.9903081	PASSED	0.96328495	PASSED
sts_serial	7	100000	100	0.38533772	PASSED	0.33795566	PASSED
sts_serial	8	100000	100	0.82520917	PASSED	0.865799	PASSED
sts_serial	8	100000	100	0.00219122	WEAK	0.89834129	PASSED
sts_serial	9	100000	100	0.94818525	PASSED	0.7537782	PASSED
sts_serial	9	100000	100	0.78996527	PASSED	0.59921688	PASSED
sts_serial	10	100000	100	0.86892905	PASSED	0.64749877	PASSED
sts_serial	10	100000	100	0.46537675	PASSED	0.76730171	PASSED

Таблица 5: Результаты тестирования последних байтов и результатов использования таблицы на базовом наборе тестов *dieharder*

				last bytes		random table	
sts_serial	11	100000	100	0.8044603	PASSED	0.69513609	PASSED
sts_serial	11	100000	100	0.49469259	PASSED	0.92529968	PASSED
sts_serial	12	100000	100	0.86820169	PASSED	0.81816783	PASSED
sts_serial	12	100000	100	0.31644433	PASSED	0.13522425	PASSED
sts_serial	13	100000	100	0.97845171	PASSED	0.40635603	PASSED
sts_serial	13	100000	100	0.12637731	PASSED	0.90762082	PASSED
sts_serial	14	100000	100	0.478581	PASSED	0.27176003	PASSED
sts_serial	14	100000	100	0.75587196	PASSED	0.53957271	PASSED
sts_serial	15	100000	100	0.69879857	PASSED	0.62967728	PASSED
sts_serial	15	100000	100	0.37030613	PASSED	0.45253469	PASSED
sts_serial	16	100000	100	0.97821158	PASSED	0.99010151	PASSED
sts_serial	16	100000	100	0.98670283	PASSED	0.70930582	PASSED
rgb_bitdist	1	100000	100	0.31185109	PASSED	0.50093562	PASSED
rgb_bitdist	2	100000	100	0.48857632	PASSED	0.8621319	PASSED
rgb_bitdist	3	100000	100	0.83889651	PASSED	0.92106255	PASSED
rgb_bitdist	4	100000	100	0.99702441	WEAK	0.69886074	PASSED
rgb_bitdist	5	100000	100	0.36041696	PASSED	0.94540194	PASSED
rgb_bitdist	6	100000	100	0.88865628	PASSED	0.24891721	PASSED
rgb_bitdist	7	100000	100	0.73102229	PASSED	0.76454683	PASSED
rgb_bitdist	8	100000	100	0.24247897	PASSED	0.73700707	PASSED
rgb_bitdist	9	100000	100	0.90789918	PASSED	0.79774399	PASSED
rgb_bitdist	10	100000	100	0.68332139	PASSED	0.14847339	PASSED
rgb_bitdist	11	100000	100	0.90265534	PASSED	0.65346143	PASSED
rgb_bitdist	12	100000	100	0.94164189	PASSED	0.7428195	PASSED
rgb_minimum_distance	2	10000	1000	0.98840893	PASSED	0.63743138	PASSED
rgb_minimum_distance	3	10000	1000	0.35598821	PASSED	0.7819773	PASSED
rgb_minimum_distance	4	10000	1000	0.52048143	PASSED	0.07832356	PASSED
rgb_minimum_distance	5	10000	1000	0.22511034	PASSED	0.86655063	PASSED
rgb_permutations	2	100000	100	0.99999241	WEAK	0.29794168	PASSED
rgb_permutations	3	100000	100	0.57219024	PASSED	0.93244966	PASSED
rgb_permutations	4	100000	100	0.81727343	PASSED	0.89297478	PASSED
rgb_permutations	5	100000	100	0.86236875	PASSED	0.26694988	PASSED
rgb_lagged_sum	0	1000000	100	0.91174209	PASSED	0.90968012	PASSED
rgb_lagged_sum	1	1000000	100	0.58411368	PASSED	0.00701025	PASSED
rgb_lagged_sum	2	1000000	100	0.12176625	PASSED	0.50783135	PASSED
rgb_lagged_sum	3	1000000	100	0.41412408	PASSED	0.65607831	PASSED
rgb_lagged_sum	4	1000000	100	0.6635429	PASSED	0.77809482	PASSED
rgb_lagged_sum	5	1000000	100	0.3695484	PASSED	0.60894518	PASSED
rgb_lagged_sum	6	1000000	100	0.68083243	PASSED	0.04717042	PASSED
rgb_lagged_sum	7	1000000	100	0.01028799	PASSED	0.79167596	PASSED
rgb_lagged_sum	8	1000000	100	0.38136933	PASSED	0.90905305	PASSED
rgb_lagged_sum	9	1000000	100	0.94334834	PASSED	0.90186072	PASSED
rgb_lagged_sum	10	1000000	100	0.99332932	PASSED	0.92610889	PASSED

Таблица 5: Результаты тестирования последних байтов и результатов использования таблицы на базовом наборе тестов *dieharder*

				last bytes		random table	
rgb_lagged_sum	11	1000000	100	0.54256666	PASSED	0.66162548	PASSED
rgb_lagged_sum	12	1000000	100	0.01475992	PASSED	0.86288925	PASSED
rgb_lagged_sum	13	1000000	100	0.41345133	PASSED	0.42466245	PASSED
rgb_lagged_sum	14	1000000	100	0.88746908	PASSED	0.29657253	PASSED
rgb_lagged_sum	15	1000000	100	0.69859075	PASSED	0.92716277	PASSED
rgb_lagged_sum	16	1000000	100	0.98461825	PASSED	0.3959812	PASSED
rgb_lagged_sum	17	1000000	100	0.2731385	PASSED	0.73352857	PASSED
rgb_lagged_sum	18	1000000	100	0.99226027	PASSED	0.96794017	PASSED
rgb_lagged_sum	19	1000000	100	0.69773067	PASSED	0.28530183	PASSED
rgb_lagged_sum	20	1000000	100	0.87408082	PASSED	0.97411755	PASSED
rgb_lagged_sum	21	1000000	100	0.71101842	PASSED	0.91255316	PASSED
rgb_lagged_sum	22	1000000	100	0.70942348	PASSED	0.98822082	PASSED
rgb_lagged_sum	23	1000000	100	0.22086778	PASSED	0.9193764	PASSED
rgb_lagged_sum	24	1000000	100	0.86091112	PASSED	0.76983467	PASSED
rgb_lagged_sum	25	1000000	100	0.668027	PASSED	0.33950761	PASSED
rgb_lagged_sum	26	1000000	100	0.64624674	PASSED	0.24095135	PASSED
rgb_lagged_sum	27	1000000	100	0.97939048	PASSED	0.4495364	PASSED
rgb_lagged_sum	28	1000000	100	0.62341756	PASSED	0.68208698	PASSED
rgb_lagged_sum	29	1000000	100	0.75509481	PASSED	0.11867416	PASSED
rgb_lagged_sum	30	1000000	100	0.76727526	PASSED	0.65101584	PASSED
rgb_lagged_sum	31	1000000	100	0.92244891	PASSED	0.54885179	PASSED
rgb_lagged_sum	32	1000000	100	0.13741943	PASSED	0.23286497	PASSED
rgb_kstest_test	0	10000	1000	0.66752604	PASSED	0.19821068	PASSED
dab_bytedistrib	0	51200000	1	0.12310245	PASSED	0.03005138	PASSED
dab_dct	256	50000	1	0.64394681	PASSED	0.23658755	PASSED
dab_filltree	32	15000000	1	0.90089377	PASSED	0.60767132	PASSED
dab_filltree	32	15000000	1	0.74466377	PASSED	0.20267544	PASSED
dab_filltree2	0	5000000	1	0.66179043	PASSED	0.56660694	PASSED
dab_filltree2	1	5000000	1	0.05282664	PASSED	0.07164338	PASSED
dab_monobit2	12	65000000	1	0.70897206	PASSED	0.91334675	PASSED

8

Список литературы

[1] Shen Alexander. Making randomness tests more robust. URL: https://hal.archives-ouvertes.fr/hal-01707610/document.

[2] Brown R.G. Dieharder: A Random Number Test Suite. Version 3.31.0. URL: https://webhome.phy.duke.edu/ rgb/General/dieharder.php.

[3] NIST. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. M., 2010. 131 p.

[4] Marsaglia George. diehard cd-rom. URL: http://web.archive.org/web/19971014105330/http://stat.fsu.edu:80/pub/diehard/cdrom/.

[5] ks.test. URL: http://stat.ethz.ch/R-manual/R-devel/library/stats/html/ks.test.html.