ЖУРНАЛ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ, 2020, том 60, № 1, с. 70–79

УДК 519.174.1

ПРАКТИЧЕСКИЙ ПОДХОД К ТЕСТИРОВАНИЮ ГЕНЕРАТОРОВ СЛУЧАЙНЫХ ЧИСЕЛ СИСТЕМ КОМПЬЮТЕРНОЙ АЛГЕБРЫ[1)]

© 2020 г. М. Н. Геворкян[1,*], А. В. Демидова[1,], А. В. Королькова[1,***], Д. С. Кулябов[1,****]**

[1] *117198 Москва, ул. Миклухо-Маклая, 6, РУДН, Россия*

**e-mail: gevorkyan-mn@rudn.ru*

***e-mail: demidova-av@rudn.ru*

****e-mail: korolkova-av@rudn.ru*

*****e-mail: kulyabov-ds@rudn.ru*

Поступила в редакцию 28.03.2018 г.
Переработанный вариант 19.08.2019 г.
Принята к публикации 18.09.2019 г.

Данная работа носит практический характер. Долгое время реализации генераторов последовательностей псевдослучайных чисел в стандартных библиотеках языков программирования и математических пакетов были плохо проработаны. Ситуация начала улучшаться сравнительно недавно. До сих пор большое количество библиотек и слабо поддерживаемых математических пакетов используют в своем составе старые алгоритмы генерации псевдослучайных чисел. Описываем четыре актуальных набора статистических тестов, которые можно применять для проверки генератора, который используется в той или иной программной системе. В работе предлагается использовать для исследования утилиты командной строки, что позволяет избежать низкоуровневого программирования на языках типа C или C++. Кроме того, рассматриваются только свободные системы с открытым программным кодом. Библ. 21. Табл. 2.

Ключевые слова: генерация псевдослучайных чисел, TestU01, PractRand, DieHarder, gjrand.

DOI: 10.31857/S004446692001010X

1. ВВЕДЕНИЕ

В информатике и вычислительной технике случайные числа находят широкое применение: для проведения статистических испытаний, в криптографии, в имитационном моделировании. Однако получение истинно случайных чисел является крайне трудоемким процессом. Это вызвано в первую очередь сложностью и дороговизной генераторов истинно случайных чисел. Также следует учесть, что генерация истинно случайных чисел может занимать много времени, и вполне вероятен вариант, когда при исчерпании источника истинно случайных чисел программа переходит в режим ожидания (время которого тоже случайно). Таким образом, чаще речь идет не об истинно случайных числах, а о псевдослучайных числах. В данной работе мы будем рассматривать программные реализации только генераторов псевдослучайных чисел.

Генераторы псевдослучайных чисел должны удовлетворять следующему ряду критериев [1]:

- для предотвращения зацикливания последовательности псевдослучайных чисел необходимо иметь достаточно длинный период;
- алгоритм должен быть эффективным по скорости работы алгоритма и затрате вычислительных ресурсов;
- алгоритм должен удовлетворять критерию воспроизводимости, то есть должна быть возможность воспроизвести ранее сгенерированную последовательность псевдослучайных чисел любое количество раз;
- алгоритм должен быть переносим по отношению к архитектурам оборудования и операционным окружениям;
- алгоритм должен быть быстрым и ресурсосберегающим.

[1)]Работа выполнена при финансовой поддержке Программы РУДН “5-100”.

Тестирование генераторов псевдослучайных чисел фактически заключается в проверке последовательности псевдослучайных величин на независимость, одинаковую распределенность, равномерность на единичном интервале. Работы по данной тематике активно велись и отечественными учеными (см. [2]–[5]). Также доказательство статистической выявляемости псевдослучайных чисел можно найти, например, в [6].

В основе каждой программной реализации генератора псевдослучайных чисел лежит свой алгоритм. Во многих теоретических исследованиях обсуждаются и сравниваются сами алгоритмы, но мы оставим данный аспект без обсуждения. В работе сосредоточимся на вопросах практической проверки качества генерируемых программным кодом последовательностей псевдослучайных чисел. Исследование проводится с помощью программных инструментов.

В рамках работы затрагиваются следующие проблемы. Во-первых, в свободных системах компьютерной алгебры зачастую используют генераторы псевдослучайных чисел, не прошедшие всех возможных тестов. Во-вторых, желательно использовать наилучший (хотя бы по некоторым критериям) генератор псевдослучайных чисел. Для решения данных проблем нами описывается практическая структура стенда для исследования программных реализаций генераторов псевдослучайных чисел. Это позволяет оценить текущую реализацию генератора, встроенную в ту или иную систему компьютерной алгебры. Кроме того, можно подобрать другую программную реализацию для встраивания ее в свободную систему компьютерной алгебры.

В качестве иллюстрации методика исследования продемонстрирована на некоторых системах компьютерной алгебры.

2. ГЕНЕРАТОРЫ В СОВРЕМЕННЫХ СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

Изобретение первого алгоритма генерации последовательности псевдослучайных чисел приписывают Дж. фон Нейману в 1946 г. После этого в 1949 г. Д.Г. Лехмер (D.H. Lehmer) предложил свой алгоритм, который впоследствии был обобщен и стал известен как линейный конгруэнтный генератор (linear congruential generator – LCG) [7]. Именно этот генератор в различных его модификациях стал основным алгоритмом, реализованным в библиотеках на языках Fortran, ADA, C.

В 1995 г. Джордж Марсальея (George Marsaglia) выпустил набор статистических тестов, который позволял проверить, насколько случайную и равномерно-распределенную последовательность чисел дает тот или иной алгоритм. Применение данного набора тестов показало, что подавляющее большинство генераторов случайных чисел на практике дают некачественную последовательность и проваливают большинство тестов.

Данный набор тестов приобрел широкую известность и побудил специалистов начать поиски более качественных алгоритмов генерации случайных чисел и их программных реализаций.

К настоящему моменту в современных версиях стандартных библиотек активно поддерживаемых языков программирования и систем компьютерной алгебры, таких как Maple [8], Mathematica [9], SymPy [10] применяется реализация алгоритма вихрь Мерсенна (MT – **M**ersenne **T**wister). Именно этот алгоритм стали повсеместно внедрять как качественную замену LCG, так как он был одним из первых алгоритмов, программная реализация которого проходила все имеющиеся на тот момент тесты. Он был разработан в 1997 г. [11] и получил свое название из-за использования простого числа Мерсенна $2^{19937} - 1$. В зависимости от реализации он обеспечивает период вплоть до $2^{216091} - 1$.

Основным недостатком алгоритма является относительная громоздкость и, как следствие, сравнительно медленная работа программного кода. Заметим также, что в настоящее время разработаны намного более эффективные и простые алгоритмы (см. [12]–[14]). В остальном же данный генератор обеспечивает псевдослучайную последовательность хорошего качества и вполне применим для большинства задач.

Перейдем, однако, к основной цели данной работы. Если в исследовательской работе используется генерация псевдослучайных чисел, то какими средствами можно проверить качество последовательностей этих чисел? Это может быть актуально при использовании нестандартной системы компьютерной алгебры или системы старых версий. Даже если используется одно из современных средств, то остается вопрос выбора удачного начального значения.

Очевидным ответом на данный вопрос будет использование какого-либо программного пакета, реализующего набор статистических тестов. Однако все известные авторам пакеты реализованы на языках C или C++ и для применения их функций непосредственно необходимо низко-

уровневое программирование. Так как язык программирования для систем компьютерной алгебры является высокоуровневым проблемно-ориентированным языком программирования, то внедрение функций на языках типа C/C++ может быть невозможно или весьма трудоемко.

На наш взгляд, это затруднение можно обойти, использовав утилиты командной строки. Использование подобных утилит избавит от необходимости внедрять код на языках типа C/C++ в программу и позволит ограничиться созданием скрипта, передающего последовательность анализируемых чисел на вход тестирующей утилите.

3. СТАТИСТИЧЕСКИЕ ТЕСТЫ

Тестирование генераторов псевдослучайных чисел фактически является классической задачей проверки статистических гипотез. Но если в математической статистике обычно ставится задача опровержения нулевой гипотезы, то при тестировании генераторов псевдослучайных чисел, наоборот, нулевую гипотезу пытаются подтвердить. Иными словами, тесты направлены на обоснование того, что сгенерированные последовательности псевдослучайных чисел являются случайными независимо распределенными случайными величинами и не связаны никаким детерминированным законом. Генератор успешно проходит тест, если не удалось найти статистически значимых отклонений от ноль гипотезы.

Так как генератор псевдослучайных чисел является детерминированным алгоритмом, то всегда найдется тот или иной статистический тест, который данный генератор не пройдет. Качество генератора определяется тем, насколько много тестов он может пройти. Поэтому при создании программной реализации тестов их объединяют в наборы тестов и применяют совместно.

Перечислим несколько тестов, которые входят в набор статистических тестов `DieHard`.

• Тест на игру в кости. Генерируется 200000 последовательностей равномерно распределенных псевдослучайных чисел. Каждая из полученных последовательностей используется для симуляции игры в кости. Далее проверяется, насколько теоретические значения математического ожидания и дисперсии согласуются с эмпирическими. Для статистической проверки используется критерий χ-квадрат.

• Парадокс дней рождения. Генерируется последовательность на большом интервале. Расстояния между числами должны быть асимптотически распределены по Пуассону.

• Пересекающиеся перестановки. Генерируется большое число выборок из пяти последовательных случайных чисел. Вероятности появления каждой из всех возможных перестановок должны быть статистически эквивалентны.

Всего в DieHard входило 12 тестов. В современных программных пакетах, реализующих наборы статистических тестов, количество тестов существенно увеличено и превышает несколько десятков.

4. ПРОГРАММНЫЕ ПАКЕТЫ НАБОРОВ СТАТИСТИЧЕСКИХ ТЕСТОВ

Как уже отмечалось, исторически первым программным пакетом, реализующим набор статистических тестов (или просто набор тестов) для тестирования генераторов псевдослучайных чисел, был пакет программ DieHard [15], созданный в 1995 г. Джорджем Марсальей (George Marsaglia). Он распространялся на CD-диске и в настоящее время официальная страница доступна только в виде архива. Пакет DieHard на данный момент не актуален, но тесты, входившие в его состав, сейчас включены в другие программные пакеты статистических тестов.

Из актуальных в настоящее время программных пакетов, реализующих наборы тестов, можно выделить следующие четыре.

• `TestU01` [16], [17] за авторством Пьера Л'Экуйе и Ричарда Симарда (Pierre L'Ecuyer, Richard Simard). Написана на `ANSI C`. На сегодняшний день является самым известным набором тестов. Тестирует генераторы, генерирующие числа из интервала [0,1). Последняя версия `1.2.3` от 18 августа 2009 г.

• `PractRand` [18] за авторством Криса Доти-Хамфри (Chris Doty-Humphrey). Написана на `C++11` с элементами C99. Принимает на вход поток байт, может тестировать 32 и 64 битные генераторы. Способен справляться с очень большими объемами данных. Последняя версия `0.94` от 04 августа 2018 г.

• `gjrand` [19]. Контактов автора на официальном сайте найти не удалось. Написан на `C99`. Принимает на вход поток байт. Поставляется с набором различных генераторов, способных ге-

Таблица 1. Сводная характеристика пакетов программ для тестирования

Пакет	Язык	CMD	Unix	Windows	Версия	Год	Сайт
TestU01	ANSI C	–	++	±	1.2.3	18.08.2009	[16]
PractRand	C99, C++11	+	+	+	0.94	04.08.2018	[18]
gjrand	C99	+	+	±	4.2.1	28.11.2014	[19]
DieHarder	C99	+	++	±	3.31.1	19.06.2017	[20]

нерировать не только равномерно-распределенные последовательности псевдослучайных чисел, но и последовательности, подчиняющиеся нормальному, пуассоновскому и некоторым другим распределениям. Последняя версия 4.2.1 от 28 ноября 2014 г.

• DieHarder [20] за авторством Роберта Брауна. Позиционируется как наследник тестов DieHard. Написан на языке C. Требует для своей работы библиотеки GSL [21] и может тестировать любой генератор, с интерфейсом в стиле интерфейсов генераторов из GSL. Последняя версия 3.31.1 от 19 июня 2017 г.

В табл. 1 дана сводка основных характеристик обозреваемых пакетов программ. В колонке Unix указана возможность установки под *nix системами. В колонке Windows поставлен плюс только если программу возможно собрать без установки CygWin или MinGW. При должном старании любую библиотеку можно скомпилировать и под Windows тоже.

Все перечисленные программные пакеты наборов тестов имеют открытый исходный код. TestU01 и DieHarder доступны для установки через официальные репозитории многих дистрибутивов, в частности Ubuntu 18.10. Два других набора тестов необходимо устанавливать путем сборки из исходных кодов. Каждый из перечисленных программных пакетов позволяет проводить тесты как путем подключения библиотек к C/C++ программе пользователя, так и предоставляет утилиту командной строки. Наиболее функциональная утилита командной строки у программного пакета DieHarder.

4.1. Установка пакетов тестов под ОС типа Unix

Опишем процесс установки в домашний каталог пользователя без прав администратора. Все действия выполнялись в операционной системе GNU/Linux Ubuntu 18.10. Использовался набор компиляторов gcc 8 версии. Из документации к пакетам следует, что подойдет любой компилятор, поддерживающий стандарты языков C до C99 включительно и C++ до C++11 включительно.

В домашнем каталоге пользователя создадим следующую иерархию каталогов:

```
mkdir -p ~/usr/bin ~/usr/lib ~/usr/share ~/usr/include
```

• Каталог ~/usr/bin для исполняемых файлов.

• Каталог ~/usr/lib для файлов разделяемых и статических библиотек (.so и .a).

• Каталог ~/usr/include для заголовочных файлов.

• Каталог ~/usr/share для примеров и документации.

Также в файл ~/.bashrc добавляем следующие переменные окружения:

```
export PATH="$HOME/usr/bin/:$PATH"
export LD_LIBRARY_PATH="$HOME/usr/lib:$LD_LIBRARY_PATH"
export LIBRARY_PATH="$HOME/usr/lib:$LIBRARY_PATH"
export C_INCLUDE_PATH="$HOME/usr/include:$C_INCLUDE_PATH".
```

Это даст возможность командному интерпретатору искать исполняемые файлы в том числе и в каталоге ~/usr/bin, а компилятору автоматически подключать библиотеки и заголовочные файлы.

4.1.1. Установка TestU01. Для установки TestU01 скачаем zip архив с официального сайта [16], распакуем его и перейдем в корневую директорию.

```
wget http://simul.iro.umontreal.ca/testu01/TestU01.zip
uz TestU01.zip
cd TestU01-1.2.3/
```

TestU01 поставляется с набором скриптов Autoconf и Automake, поэтому процесс компиляции и установки сводится к выполнению следующих трех команд:

```
./configure --prefix=$HOME/usr
make
make install
```

Указание опции -prefix=$HOME/usr позволяет установить программу локально в ~/usr.

После установки в директории ~/usr/include будет создано большое количество заголовочных файлов. Для того, чтобы организовать их аккуратней, создадим директорию ~/usr/include/testu01 и перенесем туда все h файлы TestU01. То же самое можно сделать и директории ~/usr/lib для файлов библиотек.

4.1.2. Установка gjrand. Скачиваем архив с исходными кодами с официального сайта [19]

```
wget https://datapacket.dl.sourceforge.net/project/gjrand/gjrand/
  ↪ gjrand.4.2.1/gjrand.4.2.1.tar.bz2
tar -xvjf gjrand.4.2.1.tar.bz2
cd gjrand.4.2.1
```

Для компиляции gjrand используются bash-скрипты compile. Исходные файлы для получения библиотеки расположены в директории src.

```
cd src && ./compile
```

на выходе получаем скомпилированный файлы динамической gjrand.so и статической gjrand.a библиотек, которые вручную переместим в директорию ~/usr/lib

```
cp -t ~/usr/lib/gjrand gjrand.a gjrand.so
cp -t ~/usr/include/gjrand gjrand.h
```

Вернемся в корневую директорию командой cd␣.. и пройдемся таким же образом по всем поддиректориям. В каждой из них присутствует скрипт compile, который необходимо выполнить.

```
cd testother/src && ./compile
```

На выходе получаем исполняемые файлы в testother/bin. Вновь вернемся на уровень выше cd .. и перейдем в следующую директорию:

```
cd testmisc/ &&./compile
```

получаем исполняемый файл kat, который можно использовать для проверки корректности работы библиотеки. При запуске он выполнит ряд тестов, чтобы проверить, работает ли программа так, как рассчитывал автор.

Перейдя в следующий каталог

```
cd testunif/src && ./compile
```

на выходе получим исполняемые файлы для каждого статистического теста. Они будут располагаться в директории testunif/bin. Непосредственно в testunif появятся два файла mcp и pmcp – командные утилиты для тестирования генерируемых битовых последовательностей. Эти утилиты используют исполняемые файлы из директории testunif/bin поэтому переместить их в другую директорию нельзя.

Следующий каталог

```
cd testfunif/src && ./compile
```

полностью аналогичен предыдущему, только тесты предназначены для чисел типа double из интервала $[0,1)$.

В каталоге testother

```
cd testother/src && ./compile
```

находятся тесты для неравномерных распределений. Исполняемые файлы также будут помещены в testother/bin.

В результате установки gjrand мы получили два файла библиотеки в каталоге ~/usr/lib/gjrand и заголовочный файл в каталоге ~/usr/include/gjrand. Остальные утилиты останутся в соответствующих каталогах.

Следует отметить, что процесс установки прошел без ошибок. Автор пакета указал опцию `-Wall` для компилятора и в процессе сборки были видны лишь незначительные предупреждения об использовании условия `if` без ограничивающих скобок.

4.1.3. Установка DieHarder. Для сборки программы необходимо наличие библиотеки GSL (GNU Scientific Library) [21]. В Ubuntu ее можно установить, выполнив команду

```
apt install libgsl-dev
```

Скачиваем архив с исходным кодом с официального сайта [20] и распаковываем:

```
wget https://webhome.phy.duke.edu/~rgb/General/
  ↪ dieharder/dieharder -3.31.1. tgz
uz dieharder -3.31.1. tgz
cd dieharder -3.31.1
```

Для установки DieHarder, как и в случае

```
./configure --prefix=$HOME/usr
make
make install
```

Дополнительно при выполнении `./configure` можно указать опцию `--disable-shared`, что приведет к статической компиляции утилиты командной строки и динамическая библиотека не будет создана. Это удобно, если вам нужна только утилита командной строки.

В процессе компиляции в нашем случае произошла ошибка: компилятор не смог найти определение типа `intptr_t`. Это можно исправить, включив в файл `./include/dieharder/libdieharde.h` заголовочный файл `stdint.h`. Дальнейшая установка прошла без ошибок. В процессе выполнения make install в директорию `~/usr/lib` были перенесены библиотечные файлы, в `~/usr/include` была автоматически создана поддиректория `dieharder` и в нее помещены все заголовочные файлы. В `~/usr/bin` оказалась утилита командной строки `dieharder`.

4.1.4. Установка PractRand. Скачиваем архив с исходным кодом с официального сайта [18] и распаковываем:

```
wget https://netcologne.dl.sourceforge.net/project/
  ↪ pracrand/PractRand_0.94.zip
uz PractRand_0.94.zip
```

В архиве поставляются уже собранные файлы динамической и статической библиотек для ОС Windows. Также присутствует файл проекта для Visual Studio. Для установки же под Unix перейдем в директорию `unix`, а сборку запустим командой `make`

```
cd PractRand_0.94/unix
make
```

Для соборки необходим комплятор для языка `C++`, поддерживающий стандарт `C++11`.

При сборке возникали ошибки, связанные с регистром букв в названии заголовочных файлов. Так, например, файл `Coup16.h` был указан в `test.cpp` в нижнем регистре, хотя само имя файла начинается с заглавной буквы. Еще несколько таких ошибок были вызваны той же ошибкой в именах файлов `NearSeq.h`, `birthday.h` и каталога `Tests`. Такая путаница вызвана скорее всего тем, что автор разрабатывал программу под ОС Windows, где регистр значения не имеет.

После устранения ошибок получаем 4 исполняемых файла и статическую библиотеку `libpracrand.a`.

Описав установку всех четырех рассматриваемых программных пакетов, перейдем к описанию утилит для тестирования последовательностей псевдослучайных чисел, которые эти пакеты предоставляют.

4.2. Утилиты командной строки

Пакеты `PractRand`, `DieHarder` и `gjrand` имеют в своем составе утилиты командной строки. Данные утилиты позволяют проводить статистические тесты над последовательностью случайных чисел, считывая ее из стандартного потока ввода или из файла. Наиболее функциональная утилита входит в состав `DieHarder`.

4.2.1. Утилиты PractRand. После компиляции и сборки PractRand мы получим в свое распоряжение четыре исполняемых файла.

• `RNG_output` позволяет запустить один из генераторов, входящих в состав пакета.

• `RNG_test` утилита, предназначенная для проведения статистических тестов над встроенными генераторами или над потоком данных, подаваемом на стандартный ввод.

• `RNG_benchmark` измерения скорости работы встроенных или добавленных пользователем генераторов.

• `Test_calibration` для тестирования и настройки наборов тестов.

Для целей тестирования используется `RNG_test`. Для примера при запуске теста для встроенного генератора `jsf32` достаточно выполнить команду

```
RNG_test jsf32
```

Для теста внешней программы нужно подать генерируемый бинарный поток беззнаковых целых чисел на стандартный вход `RNG_test`, например так:

```
./random -G 4 -N 1000000000000000 | RNG_test stdin32
```

Опция `stdin32` заставляет `RNG_test` интерпретировать поток бинарных данных как набор 32-битных чисел. Если указана опция `stdin64`, то числа будут восприниматься как 64 битные, а при указании `stdin` программа сама решит, какую разрядность использовать. Для тестирования данных из файла можно поступит, например так

```
cat file.data | RNG_test stdin64
```

Данные в файле также должны быть в бинарном формате. Данные в текстовом формате не поддерживаются.

Тесты проводятся довольно быстро и на выход распечатывается отчет, где перечисляются проваленные и подозрительные тесты. Указав опцию `-p1` можно заставить программу печатать все результаты, а не только проваленные.

Тесты проводятся довольно быстро и на выход распечатывается отчет, где перечисляются проваленные и подозрительные тесты. Указав опцию `-p1` можно заставить программу печатать все результаты, а не только проваленные.

4.2.2. Утилиты gjrand. После сборки появляется множество разных исполняемых файлов. Для тестирования внешних генераторов предназначены два из них. Один расположен в директории `testfunif` и называется `mcp` (master computer program), а второй в директории `testother` и называется `fmcp`. Утилита |mcp| предназначена для тестирования последовательностей целых беззнаковых псевдослучайных чисел, а `fmcp` для чисел из интервала $[0,1)$. В остальном между ними отличий нет.

Программа `mcp` принимает на стандартный вход только бинарный поток. Делается это с помощью конвейера

```
./random -G 4 -N 1000000000000000 | mcp --big
```

К переданному потоку применяется весь набор тестов. Результаты распечатываются по мере появления. Других опций, кроме настройки объема проверяемых данных (см. табл. 2), у программы нет.

Таблица 2. Опции mcp пакета gjrand

Опция	Описание
`-tiny`	(10 MB)
`-small`	(100 MB)
`-standard`	(1 GB) (default)
`-big`	(10 GB)
`-huge`	(100 GB)
`-tera`	(1 TB)
`-ten-tera`	(10 TB)
`number`	количество байтов
`-no-rewind`	не повторяться

4.2.3. Утилита `dieharder`. После установки пакета `DieHarder` в консоли станет доступна команда `dieharder`. Данная команда позволяет запускать и тестировать встроенные в `DieHarder` и библиотеку `GSL` генераторы псевдослучайных чисел. Кроме того, она может тестировать поток случайных чисел из файлов или из стандартного ввода. Подробную справку по всем возможным опциям можно получить, указав опцию `-h`. Перечислим кратко наиболее важные из них.

- `-l` – показать список доступных тестов.
- `-dn` – выбрать для применения тест `n`.
- `-a` – применить все тесты.
- `-g-1` – показать список доступных генераторов псевдослучайных чисел. Выбранный генератор можно протестировать, указав опцию `-gn`, где `n` – номер генератора из списка.

В списке генераторов присутствуют, в частности, следующие опции.

- Опция `200` позволяет считывать поток бинарных данных, подаваемый на стандартный ввод `stdin`.
- Опции `201` и `202` включают считывания данных из файла в бинарном формате и текстовом формате соответственно.
- Опции `500` и `501` позволяют считывать поток байтов с псевдоустройств `/dev/random` и `/dev/urandom`.

Для тестирования внешних генераторов наиболее предпочтительным способом будет передача непрерывного потока двоичных данных через конвейер (pipe |). Сделать это можно, например, следующим способом:

```
./random -G 4 -N 1000000000000000 | dieharder -g 200 -a
```

Программа `random` будет использовать генератор под номером 4 для генерации 10^{15} чисел. Заметим, что это не приведет к зависанию процесса, так как `dieharder` сам закроет канал и завершит процесс, когда проведет все тесты.

В случае считывания псевдослучайных чисел из текстового файла, данный файл должен иметь определенную структуру. Каждое псевдослучайное число должно располагаться на новой строке, а в первых строках файла необходимо указать следующие данные: тип чисел (`d` – целые числа двойной точности), количество чисел в файле и разрядность чисел (32 или 64 бита). Приведем пример начала такого файла:

```
type: d
count: 5
numbit: 64
1343742658553450546
16329942027498366702
3111285719358198731
2966160837142136004
17179712607770735227
```

Когда такой файл создан, можно передать его `dieharder` для проведения тестирования

```
dieharder -a -g 202 -f file.in > file.out
```

флаг `-f` задает входной файл с числами для анализа. Результаты тестирования будут сохранены в `file.out`. Для полноценного теста более 10^9 чисел, поэтому размер файла может превысить десятки гигабайт. В случае меньшего количества чисел `dieharder` может начать считывать файл сначала, что приведет к ухудшению результатов теста.

5. ПРИМЕР ТЕСТИРОВАНИЯ ГЕНЕРАТОРОВ SymPy И Maxima

Рассмотрим, как можно организовать тестирование любого генератора псевдослучайных чисел на примере CAS Maxima и библиотеки SymPy для Python. Будем использовать CAS Maxima версии 5.42.1 и Python 3.6.8, дистрибутив Miniconda. Нашей целью не является сравнение реализаций генераторов, а лишь предоставление примера, который может быть использован для тестирования генераторов в других библиотеках, математических пакетах и системах компьютерной алгебры.

Так как SymPy является python-модулем, то для генерации псевдослучайных чисел используется стандартный модуль `random`. Следующий фрагмент кода позволяет организовать вывод беззнаковых 64-битных целых чисел в стандартный поток вывода.

```
import random
import sys
while True:
    r = random.randint(0,2**64-1)
    r = r.to_bytes(8, byteorder="little", signed=False)
    sys.stdout.buffer.write(r)
```

В данном фрагменте генерируется целое число из полуинтервала $[0, 2^{64})$, затем оно преобразуется в двоичный вид и выводится в стандартный поток вывода. Записав данный скрипт в файл `rand_test.py`, мы можем организовать тестирование, выполнив в консоли следующую команду:

```
python rand_test.py | dieharder -g 200 –a
```

или

```
python rand_test.py | RNG_test stdin64
```

В случае Maxima сгенерировать последовательность случайных целых чисел можно с помощью следующего фрагмента кода:

```
for i: 1 thru 10 step 1 do print(random(2^64-1))$
```

Однако на выходе получим столбик целых чисел в текстовом виде, поэтому для тестирования данного генератора воспользуемся возможностью утилиты `dieharder` считывать данные в текстовом виде из заранее подготовленных файлов (см. п. 4.2.3).

Так как обе системы компьютерной алгебры используют алгоритм вихрь Мерсенна, то полученные результаты тестирования практически одинаковы. Утилита `RNG_test` для SymPy выдает результат `no anomalies in 133 test result(s)`, а утилита отображает в отчете 5 слабо пройденных тестов из более чем тридцати. Результаты dieharder для Maxima практически идентичны.

6. ЗАКЛЮЧЕНИЕ

Авторы постарались дать исчерпывающее описание практики построения стенда для исследования программной реализации генераторов псевдослучайных чисел. Описанное программное обеспечение дает исследователю выбор из трех утилит командной строки. Эти утилиты снимают необходимость низкоуровневого программирования. Кроме того, их удобно использовать вместе с каким-либо промежуточным связующим языком программирования.

Для использования этих утилит необходимо обеспечить вывод в стандартный поток результатов генерации чисел в бинарном виде. Так как большинство систем компьютерной алгебры ориентированы на высокоуровневое программирование, то обеспечить бинарный вывод может быть затруднительно. В этом случае можно использовать только утилиту `dieharder`, так как она может обрабатывать данные в текстовом виде.

В конце работы демонстрируются приемы исследования программных реализаций генераторов случайных чисел на примере двух свободных систем компьютерной алгебры.

СПИСОК ЛИТЕРАТУРЫ

1. *Дроздова И.И., Жилин В.В.* Генераторы случайных и псевдослучайных чисел // Технические науки в России и за рубежом: материалы VII Междунар. науч. конф. М.: Буки-Веди, 2017. С. 13–15. URL: https://moluch.ru/conf/tech/archive/286/13233.
2. *Колчин В.Ф., Севастьянов Б.А., Чистяков В.П.* Случайные размещения. М.: Наука, 1976. 224 с.
3. *Тихомирова М.И., Чистяков В.П.* Статистические критерии, предназначенные для выявления некоторых видов зависимостей между случайными последовательностями // Тр. по дискр. матем. 2006. Т. 3. С. 357–376. URL: http://mi.mathnet.ru/tdm153.
4. *Тихомирова М.И., Чистяков В.П.* Статистический критерий типа критерия пустых ящиков // Математические вопросы криптографии. 2010. Т. 1. № 1. С. 101–108. URL: http://mi.mathnet.ru/mvk6.

5. *Кириченко Л.О., Цехмистро Р.И., Круг О.Я., Стороженко А.В.* Сравнительный анализ генерации псевдослучайных чисел в современных технологиях беспроводной передачи данных // Системи обробки інформації. 2009. Т. 4 (78). С. 70–74. URL: http://www.hups. mil.gov.ua/periodic-app/article/6508.
6. *Тюрин Ю.Н., Макаров А.А.* Статистический анализ данных на компьютере / Под ред. В.Э. Фигурнова. М.: ИНФРА, 1998. 528 с. ISBN: 5-86225-662-8.
7. *Кнут Д.Э.* Искусство программирования. 3 изд. М.: Вильямс, 2001. Т. 2. 832 с. ISBN: 5-8459-0081-6.
8. Maple home site. 2019. URL: https://www.maplesoft.com/ products/maple/.
9. Mathematica home site. 2018. URL: https://www.wolfram.com/ mathematica/.
10. SymPy home site. 2019. URL: http://www.sympy.org/ru/index. html.
11. *Matsumoto M., Nishimura T.* Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator // ACM Trans. Model. Comput. Simul. 1998. V. 8. № 1. P. 3–30. URL: http://www.math.sci.hiroshima-u.ac.jp/$\sim$m-mat/MT/ ARTICLES/mt.pdf.
12. *Panneton F., L'Ecuyer P.* On the Xorshift Random Number Generators // ACM Trans. Model. Comput. Simul. 2005. V. 15. № 4. P. 346 361. URL: http://doi.acm.org/10.1145/1113316.1113319.
13. PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation: Rep.: HMC-CS-2014-0905 / Harvey Mudd College; Executor: Melissa E O'Neill. Claremont, CA: 2014. URL: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf.
14. *Boldi P., Vigna S.* On the Lattice of Antichains of Finite Intervals // Order. 2018. V. 35. № 1. P. 57–81. URL: https://doi.org/10.1007/s11083-016-9418-8
15. *Marsaglia G.* The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. 1995. URL: https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/.
16. *L'Ecuyer P., Simard R.* TestU01 – Empirical Testing of Random Number Generators. 2009. URL: http://simul.iro.umontreal.ca/testu01/tu01.html.
17. *L'Ecuyer P., Simard R.* TestU01: A C library for empirical testing of random number generators // ACM Transactions on Mathematical Software (TOMS). 2007. V. 33. № 4. P. 22. URL: http://www.iro.umontreal.ca/$\sim$lecuyer/myftp/papers/testu01.pdf.
18. *Doty-Humphrey C.* PractRand official site. 4-2018. URL: http:// pracrand.sourceforge.net/.
19. Gjrand random numbers official site. 2014. URL: http://gjrand. sourceforge.net/.
20. *Brown R.G., Eddelbuettel D., Bauer D.* Dieharder: A Random Number Test Suite. 2017.URL: http://www.phy.duke.edu/$\sim$rgb/ General/rand_rate.php.
21. *Galassi M., Gough B., Jungman G. et al.* GSL GNU Scientific Library. 2019. URL: https://www.gnu.org/software/gsl/.