V.A. Trapeznikov Institute of Control Sciences
Academy of Sciences
RUDN University
Tomsk State University
Institute of Information and Communication
Technologies Bulgarian Academy of Sciences
Research and development company
"Information and networking technologies"

DISTRIBUTED COMPUTER AND COMMUNICATION NETWORKS: CONTROL, COMPUTATION, COMMUNICATIONS (DCCN-2016)

Volume 2
Mathematical Modeling, Simulation and Control Problems

Proceedings
of the Nineteenth International Scientific Conference

Russia, Moscow, 21–25 November 2016

Under the general editorship
of D.Sc. V. M. Vishnevskiy and D.Sc. K. E. Samouylov

Moscow
2016

Федеральное государственное бюджетное учреждение науки
Институт проблем управления им. В. А. Трапезникова
РОССИЙСКОЙ АКАДЕМИИ НАУК
Федеральное государственное автономное
образовательное учреждение высшего образования
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»
Национальный исследовательский
Томский государственный университет
Институт информационных и телекоммуникационных технологий
БОЛГАРСКОЙ АКАДЕМИИ НАУК
Научно-производственное объединение
«Информационные и сетевые технологии»

РАСПРЕДЕЛЕННЫЕ КОМПЬЮТЕРНЫЕ И ТЕЛЕКОММУНИКАЦИОННЫЕ СЕТИ: УПРАВЛЕНИЕ, ВЫЧИСЛЕНИЕ, СВЯЗЬ (DCCN-2016)

В трех томах

Том 2
Математическое моделирование и задачи управления

Материалы
Девятнадцатой международной научной конференции

Россия, Москва, 21–25 ноября 2016 г.

Под общей редакцией
д.т.н. В.М. Вишневского и д.т.н. К.Е. Самуйлова

Москва
2016

УДК 004.7:004.4.001:621.391:007(063)
ББК 32.973.202:32.968
Р 24

Р 24 **Распределенные компьютерные и телекоммуникационные сети: управление, вычисление, связь (DCCN-2016) = Distributed computer and communication networks: control, computation, communications (DCCN-2016)** : материалы Девятнадцатой международной научной конференции, 21–25 нояб. 2016 г. : в 3 т.; под общ. ред. В. М. Вишневского и К. Е. Самуйлова — М.: РУДН, 2016.
ISBN 978-5-209-07666-7
Т. 2: Математическое моделирование и задачи управления = Mathematical modeling, simulation and control problems. — 300 с. : ил.
ISBN 978-5-209-07668-1 (т. 2)

В научном издании представлены материалы Девятнадцатой международной научной конференции «Распределенные компьютерные и телекоммуникационные сети: управление, вычисление, связь» по следующим направлениям:

— Оптимизация архитектуры компьютерных и телекоммуникационных сетей;
— Управление в компьютерных и телекоммуникационных сетях;
— Оценка производительности и качества обслуживания в беспроводных сетях;
— Аналитическое и имитационное моделирование коммуникационных систем последующих поколений;
— Беспроводные сети 4G/5G и технологии сантиметрового и миллиметрового диапазона радиоволн;
— RFID-технологии и их применение в интеллектуальных транспортных сетях;
— Интернет вещей, носимые устройства, приложения распределенных информационных систем;
— Распределенные системы и системы облачного вычисления, анализ больших данных;
— Вероятностные и статистические модели в информационных системах;
— Теория очередей, теория надежности и их приложения;
— Математическое моделирование высокотехнологичных систем;
— Математическое моделирование и задачи управления.

Сборник материалов конференции предназначен для научных работников и специалистов в области теории и практики построения компьютерных и телекоммуникационных сетей.

Текст воспроизводится в том виде, в котором представлен авторами

Утверждено к печати Программным комитетом конференции

ISBN 978-5-209-07668-1 (т. 2)

ISBN 978-5-209-07666-7

Contents

Plenary Speakers

Sessional Speakers

УДК 004.94+519.62

Обзор и тестирование генераторов псевдослучайных чисел и их реализация OpenModelica

М. Гнатич[‡§], **М. Н. Геворкян**[*], **А. В. Демидова**[*], **А. В. Королькова**[*], **Д. С. Кулябов**[*†], **Л. А. Севастьянов**[*‡]

[*] *Кафедра прикладной информатики и теории вероятностей, Российский университет дружбы народов, ул. Миклухо-Маклая, д.6, Москва, Россия, 117198*

[†] *Лаборатория информационных технологий, Объединённый институт ядерных исследований, ул. Жолио-Кюри 6, Дубна, Московская область, Россия, 141980*

[‡] *Лаборатория теоретической физики, Объединённый институт ядерных исследований, ул. Жолио-Кюри 6, Дубна, Московская область, Россия, 141980*

[§] *Университет Павла Йозефа Шафарика, Шробарова 2, 041 80 Кошице, Словакия*

Аннотация. Язык компонентно-ориентированного моделирования Modelica применяется для моделирования сложных процессов, задаваемых системой ОДУ. В стандартной библиотеке OpenModelica нет средств для генерации псевдослучайных чисел, что делает невозможным моделирование стохастических процессов. Целью данной статьи является краткий обзор некоторого числа алгоритмов генерации последовательности равномерно распределенных случайных чисел и оценка качества даваемых ими последовательностей, а также описание способов реализации некоторых из этих алгоритмов в системе OpenModelica.

Ключевые слова: псевдослучайные числа, Modelica.

1. Введение

В данной статья рассматривается вопрос генерации равномерных псевдослучайных чисел в среде OpenModelica. OpenModelica является открытой реализацией языка Modelica. Данный язык предназначен для моделирования различных систем и процессов, которые можно представить в виде системы алгебраических или дифференциальных уравнений. На данный момент, однако, в стандартной библиотеке OpenModelica нет средств даже для генерации равномерно распределенных случайных чисел (существует библиотека Noise, претендующая на включение в стандартный набор библиотек [1]).

В статьи дается обзор некоторых алгоритмов генерации псевдослучайных чисел. Все алгоритмы реализованы на языке `C`, а также часть из них непосредственно на языке `OpenModelica`. С помощью утилиты `dieharder` проводится тестирование рассмотренных алгоритмов. На основе результатов тестирования выбираются лучшие алгоритмы для

использования. В заключении описываются детали реализации генератора псевдослучайных чисел в среде OpenModelica и даются некоторые рекомендации по использованию псевдослучайных генераторов на практики.

Статья может служить кратким введением в теорию и практику генераторов равномерно распределенных случайных чисел.

2. Обзор алгоритмов генерации последовательности равномерно распределенных псевдослучайных чисел

2.1. Линейный конгруэнтный метод

Линейный конгруэнтный метод был впервые предложен в 1949 году Д. Г. Лехмером (D. H. Lehmer) [2,3]. Алгоритм задается одной формулой:

$$x_{n+1} = (ax_n + c) \mod m, \quad n \geqslant 0,$$

где m — *модуль* (mask) $m > 1$, a — *множитель* (multiplier) ($0 \leqslant a < m$), c — *приращение* ($0 \leqslant c < m$), x_0 — начальное значение, *зерно* (seed). Результатом многократного применения данной рекуррентный формулы является *линейная конгруэнтная последовательность* $x_1, \ldots, x_n$. Особый случай $c = 0$ называется *мультипликативным* конгруэнтным методом. Для краткого обозначения данного метода будем использовать аббревиатуру `LCG` (**l**inear **c**ongruential **g**enerator).

Числа m, a, c называют «волшебными» или «магическими» так как их значения задаются в коде программы и выбираются исходя из опыта применения генератора. Качество генерируемой последовательности существенно зависит от правильного выбора данных параметров. Последовательность $\{x\}_1^n$ периодична и ее период зависит от числа m, которое поэтому должно быть большим. На практике выбирают m равным машинному слову (для 32-х битной архитектуры — 2^{32} и для 64-х битной — 2^{64}). В [2, 3] рекомендуется выбрать $a = 6364136223846793005$, $c = 1442695040888963407$, $m = 2^{64}$. В статье [4] можно найти объемные таблицы с оптимальными значениями a, b и m.

Квадратичный конгруэнтный метод $x_n = (ax_{n-1}^2 + bx_{n-1} + d) \mod m$ кубический конгруэнтный метод $x_n = (ax_{n-1}^3 + bx_{n-1}^2 + cx_{n-1} + d) \mod 2^e$.

В настоящее время линейный конгруэнтный метод представляет по большей части лишь исторический интерес, так как он генерирует сравнительно некачественную псевдослучайную последовательность по сравнению с другими, не менее простыми генераторами.

Авторы реализовали линейный конгруэнтный метод на языке `C` и сгенерировали с помощью него последовательность из 10^9 чисел. Данная последовательность была протестирована с помощью открытого

набора тестов DieHarder [5]. В результате генератор LCG провалил около половины тестов.

2.2. Метод Фибоначчи с запаздываниями

Развитием LCG генератора можно считать идею использовать для генерации i-го элемента псевдослучайной последовательности не один, а несколько предыдущих элементов. Согласно [2, 3] первый такой генератор был предложен в начале 50-х годов и основывался на формуле:

$$x_{n+1} = (x_n + x_{n-1}) \mod m.$$

Однако на практике он показал себя не лучшим образом. В 1958 году Дж. Ж. Митчелом (G. J. Mitchell) и Д. Ф. Муром (D. Ph. Moore) был изобретен намного лучший генератор

$$x_n = (x_{n-n_a} + x_{n-n_b}) \mod m,\ n \geqslant \max(n_a, n_b).$$

Данный генератор получил название генератора Фибоначчи с запаздыванием (LFG, **l**agged **F**ibonacci **G**enerator).

Как и в случае LCG генератора, выбор «магических чисел» n_a и n_b сильно влияет на качество генерируемой последовательности. Авторы предложили использовать следующие магические числа n_a и n_b:

$$n_a = 24, n_b = 55.$$

Д. Кнут [2,3] приводит ряд других значений, начиная от $(37, 100)$ и заканчивая $(9739, 23209)$ Длина периода данного генератора в точности равна $2^{e-1}(2^{55} - 1)$ при выборе $m = 2^e$.

Как видно из алгоритма, для инициализации данного генератора необходимо использовать не одно начальное значение, а последовательность из $\max(n_a, n_b)$ случайных чисел.

В открытой библиотеке GNU Scientific Library (GSL) [6] используется *составной мульти-рекурсивный* генератор, предложенный в статье [7]. Данный генератор является разновидностью LFG.

Еще одни метод, предложный в статье [8] также является разновидностью метода Фибоначчи и определяется формулой:

$$x_n = (a_1 x_{n-1} + a_5 x_{n-5}) \mod 5,$$

В GSL использованы следующие параметры: $a_1 = 107374182$, $a_2 = 0$, $a_3 = 0$, $a_4 = 0$, $a_5 = 104480$, $m = 2^{31} - 1 = 2147483647$. Период этого генератора равен 10^{46}.

2.3. Инверсный конгруэнтный генератор

Инверсный конгруэнтный метод основан на использовании обратного по модулю числа.

$$x_{i+1} = (ax_i^{-1} + b) \mod m$$

где a — *множитель* ($0 \leqslant a < n$), b — *приращение* ($0 \leqslant b < n$), x_0 — начальное значение (seed). Кроме того НОД$(x_0, m) = 1$ и НОД$(a, m) = 1$.

Данный генератор превосходит обычный линейный метод, однако сложнее алгоритмически, так как необходимо искать обратные по модулю целые числа, что приводит к медленной скорости генерации чисел. Для вычисления обратного числа обычно применяется расширенный алгоритм Евклида [2, 3, §4.3.2].

2.4. Генераторы с использованием побитовых операций

Большинство генераторов, дающих наиболее качественные псевдослучайные последовательности используют в своих алгоритмах побитовые операции конъюнкции, дизъюнкции, отрицания, исключающей дизъюнкции (`xor`) и побитовые вправо/влево.

Вихрь Мерсенна считается одним из лучших псевдослучайных генераторов. Разработан в 1997 году Мацумото и Нишимура [9]. Существуют 32-,64-,128-разрядные версии вихря Мерсенна. Свое название алгоритм получил из-за использования простого числа Мерсенна $2^{19937} - 1$. В зависимости от реализации обеспечивается период вплоть до $2^{216091} - 1$. Вихрь Мерсенна используется во многих стандартных библиотеках, например в модуле `random` языка `Python 3` [10].

Генераторы `XorShift` дающие качественную псевдослучайную последовательность были разработаны в 2003 году Дж. Марсальей (G. Marsaglia) [11, 12].

Еще одно семейство генераторов — генераторы KISS (Keep It Simple Stupid) — дает не менее качественную последовательность псевдослучайных чисел [13]. Генератор `KISS` используется в процедуре `random_number()` языка `Frotran` (компилятор `gfortran` [14])

2.5. Устройства `/dev/random` и `/dev/urandom`

Для создания истинно-случайной последовательности чисел с помощью компьютера, в некоторых Unix системах (в частности GNU/Linux) используется сбор «фонового шума» окружения операционной системы и аппаратного обеспечения. Источником такого случайного шума являются моменты времени между нажатия клавиш пользователем (inter-keyboard timings), различные системные прерывания и другие события, которые удовлетворяют двум требованиям: не

детерминированности и сложной доступности для измерения внешним наблюдателем.

Неопределенность из таких источников собирается драйвером ядра и помещается в «энтропийный пул», который дополнительно перемешивается с помощью алгоритма, похожего на алгоритмы вычисления контрольных сум. Когда случайные байты запрашиваются системным вызовом, они извлекаются из энтропийного пула путем взятия SHA хеша от содержимого пула. Взятие хеша позволяет не показывать внутреннее состояние пула, так как восстановление содержимого по хешу считается вычислительно невыполнимой задачей. Дополнительно извлекающая процедура занижает размер содержимого пула для того, чтобы предотвратить выдачу хеша по всему содержимому и минимизировать теоретическую возможность определения его содержимого. Во вне энтропийный пул доступен в виде символьного псевдоустройства `/dev/random`.

3. Тестирование алгоритмов

Обзор большого числа критериев оценки качества массива псевдослучайных чисел можно найти в третей главе книги Д. Кнута [2, 3], а также статье [15] одного из ведущих специалистов по генераторам псевдослучайных чисел. Все описанные в нашей статье алгоритмы были реализованы на языке `C` и протестированы с помощью набора тестов `dieharder`, доступного на официальном сайте автора [5]. Также этот пакет тестов в ходит в состав официальных репозиториев многих дистрибутивов GNU/Linux.

3.1. Описание `dieharder`

Набор тестов `dieharder` реализован в виде утилиты командной строки, которая позволяет тестировать последовательности равномерно распределенных псевдослучайных чисел. Также `dieharder` может использовать любой генератор из библиотеки GSL [6] для генерирования чисел или непосредственного тестирования. `dieharder -l` — показать список доступных тестов, `dieharder -g -1` — показать список доступных генераторов псевдослучайных чисел, каждому генератору присвоено порядковое число, которое надо указать после флага `-g` для включения нужного генератора. `200 stdin_input_raw` — считывать стандартный входной бинарный поток, `201 file_input_raw` — считывать файл в бинарном формате, `202 file_input` — считывать файл в текстовом формате, `500 /dev/random` — использовать псевдоустройство `/dev/random`, `501 /dev/urandom` — использовать псевдоустройство `/dev/urandom`. Каждое псевдослучайное число должно располагаться на новой строке, а в первых строках файла необходимо указать: тип чисел (`d` — целые числа двойной точности), количество

чисел в файле и разрядность чисел (32 или 64 бита). Когда такой файл создан можно передать его в `dieharder`

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

где флаг `-a` включает все встроенные тесты, а флаг `-f` задает файл для анализа. Результаты тестирования будут сохранены в `file.out`.

3.2. Результаты тестов

Генератор	Провалено	Слабо	Пройдено
LCG	52	6	55
LCG2	51	8	54
LFG	0	2	111
ICG	0	6	107
KISS	0	3	110
jKISS	0	4	109
XorShift	0	4	109
XorShift+	0	2	111
XorShift*	0	2	111
MT	0	2	111
dev/urandom	0	2	111

3.3. Реализация генераторов псевдослучайных чисел.

Одним и ограничений при попытке реализовать генерацию псевдослучайных чисел используя лишь язык Modelica является отсутствие в OpenModelica побитовых логических операторов и операторов сдвига. В связи с этим, те алгоритмы, которые эти операции используют необходимо реализовать на языке `C`.

На языке Modelica нами были реализованы три генератора: линейный конгруэнтный генератор (LCG), метод Фибоначчи с запаздыванием (LFG) и инверсный конгруэнтный генератор (ICG).

Как показало тестирование с помощью `DieHarder`, из генераторов, не использующих побитовые операции, наиболее качественную последовательность псевдослучайных чисел дают метод Фибоначчи с запаздыванием и инверсный конгруэнтный генератор. Недостатком LFG генератора является необходимость в минимум 55 начальных значений. Эти начальные значения можно сгенерировать с помощью LCG генератора. ICG генератор требует только одного начального значения, однако его алгоритм сложнее по сравнению с LFG.

Из генераторов, использующих побитовые операции выделяются генераторы `xorshift*`, `xorshift+` и вихрь Мерсенна. Все они дают одинаково качественную последовательность. Алгоритм вихря Мерсенна, однако, намного более громоздок, чем `xorshift*` или

`xorshift+`, поэтому для генерирования больших последовательностей предпочтительней использовать `xorshift*` или `xorshift+`.

4. Заключение

В заключении кратко перечислим основные результаты работы: дан обзор наиболее известных и эффективных генераторов равномерно распределенных псевдослучайных чисел; описана методика тестирования псевдослучайных последовательностей с помощью набора тестов `DieHarder`; все описываемые алгоритмы реализованы в виде C функций или непосредственно на OpenModelica и протестированы с помощью `DieHarder`; описаны основные особенности реализации псевдослучайных генераторов на языке Modelica; приводится ряд практических рекомендаций по использованию псевдослучайных генераторов.

Благодарности

Работа частично поддержана грантами РФФИ № 14-01-00628, 15-07-08795 и 16-07-00556.

Литература

1. Klöckner A., van der Linden F. L. J., Zimmer D. Noise generation for continuous system simulation // Proceedings of the 10th International Modelica Conference. — Lund; Sweden, 2014. — P. 837–846.
2. Кнут, Дональд Эрвин. Искусство программирования. — 3 изд. — Москва : Вильямс, 2004. — Т. 2.
3. Knuth D. E. The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. — Vol. 2.
4. L'Ecuyer P. Tables of linear congruential generators of different sizes and good lattice structure // Mathematics of Computation. — 1999. — Vol. 68, no. 225. — P. 249–260.
5. Brown R. G., Eddelbuettel D., Bauer D. Dieharder: A Random Number Test Suite. — 2013. — URL: `http://www.phy.duke.edu/~rgb/General/rand_rate.php`.
6. Galassi M., Gough B., Jungman G. et al. — The GNU Scientific Library Reference Manual, 2015. — URL: `https://www.gnu.org/software/gsl/manual/gsl-ref.pdf`.
7. L'Ecuyer P. Combined multiple recursive random number generators // Operations Research. — 1996. — Vol. 44, no. 5. — P. 816–822.
8. L'Ecuyer P., Blouin F., Couture R. A search for good multiple recursive random number generators // ACM Transactions on Modeling and Computer Simulation (TOMACS). — 1993. — Vol. 3, no. 2. — P. 87–98.
9. Matsumoto M., Nishimura T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator // ACM

Trans. Model. Comput. Simul. — 1998. — January. — Vol. 8, no. 1. — P. 3–30.
10. gfortran team T. — Python 3.5.1 documentation, 2016. — Mar. — URL: https://docs.python.org/3/.
11. Marsaglia G. Xorshift rngs // Journal of Statistical Software. — 2003. — Vol. 8, no. 1. — P. 1–6.
12. Panneton F., L'Ecuyer P. On the xorshift random number generators // ACM Trans. Model. Comput. Simul. — 2005. — October. — Vol. 15, no. 4. — P. 346–361.
13. Rose G. KISS: A Bit Too Simple. — 2011. — URL: https://eprint.iacr.org/2011/007.pdf.
14. gfortran team T. — Using GNU Fortran, 2015. — URL: https://gcc.gnu.org/onlinedocs/.
15. L'Ecuyer P., Simard R. Testu01: A c library for empirical testing of random number generators // ACM Transactions on Mathematical Software (TOMS). — 2007. — Vol. 33, no. 4. — P. 22.

UDC 004.94+519.62

Review and testing of pseudo-random number generators and their implementation in OpenModelica

M. Hnatich[‡§], M. N. Gevorkyan[*], A. V. Demidova[*], A. V. Korolkova[*], D. S. Kulyabov[*†], L. A. Sevastianov[*‡]

[*] *Department of Applied Probability and Informatics*
RUDN University
Miklukho-Maklaya str. 6, Moscow, 117198, Russia
[†] *Laboratory of Information Technologies*
Joint Institute for Nuclear Research
Joliot-Curie 6, Dubna, Moscow region, 141980, Russia
[‡] *Bogoliubov Laboratory of Theoretical Physics*
Joint Institute for Nuclear Research
Joliot-Curie 6, Dubna, Moscow region, 141980, Russia
[§] *Pavol Jozef Šafárik University in Košice (UPJŠ),*
Šrobárova 2, 041 80 Košice, Slovak Republic

Component-based modeling language Modelica (OpenModelica is open source implementation) is used for the numerical simulation of complex processes of different nature represented by ODE system. However, in OpenModelica standard library there is no routines for pseudo-random numbers generation, which makes it impossible to use for stochastic modeling processes. The goal of this article is a brief overview of a number of algorithms for generation a sequence of uniformly distributed pseudo random numbers and quality assessment of the sequence given by them, as well as the ways to implement some of these algorithms in OpenModelica system.

Keywords: pseudo-random numbers generation, Modelica.