Распродажа на Wildberries.ru
wildberries.ru
Одежда и обувь со скидками до 70%. Постоянным клиентам скидка до 20%!

ГОСТ Р ИСО 28640-2012

Статистические методы. Генерация случайных чисел

Обозначение:	ГОСТ Р ИСО 28640-2012
Статус:	**действующий**
Название рус.:	Статистические методы. Генерация случайных чисел
Дата актуализации текста:	01.07.2014
Дата актуализации описания:	01.07.2014
Дата введения:	01.12.2013
Дата последнего изменения:	21.04.2014
Область применения:	В настоящем стандарте установлены методы генерации случайных чисел, подчиняющихся равномерному и другим законам распределения, используемых при применении метода Монте-Карло. В настоящий стандарт не включены криптографические методы генерации случайных чисел. Настоящий стандарт будет полезен в первую очередь: - научным работникам, технологам и специалистам в области систем управления, использующим статистическое моделирование; - специалистам в области математической статистики, использующим рандомизацию при разработке методов статистического контроля качества продукции и процессов, планирования экспериментов и обработки данных; - математикам, разрабатывающим сложные процедуры оптимизации с использованием метода Монте-Карло, разработчикам программного обеспечения при создании алгоритмов генерации псевдослучайных чисел
Расположен в:	**База ГОСТов** Общероссийский классификатор стандартов Услуги. Организация фирм, управление и качество. Администрация. Транспорт. Социология. Качество Применение статистических методов Классификатор государственных стандартов Общетехнические и организационно-методические стандарты Система документации Общие методы и средства контроля и испытания продукции. Методы статистического контроля качества, надежности, долговечности

ФЕДЕРАЛЬНОЕ АГЕНТСТВО
ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО 28640 – 2012

Статистические методы

ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ЧИСЕЛ

ISO 28640:2010
Random variate generation methods
(IDT)

Издание официальное

Москва
Стандартинформ
2014

Предисловие

Цели и принципы стандартизации в Российской Федерации установлены Федеральным законом от 27 декабря 2002 г. № 184-ФЗ «О техническом регулировании», а правила применения национальных стандартов Российской Федерации – ГОСТ Р 1.0 – 2004 «Стандартизация в Российской Федерации. Основные положения»

Сведения о стандарте

1 ПОДГОТОВЛЕН Автономной некоммерческой организацией «Научно-исследовательский центр контроля и диагностики технических систем» (АНО «НИЦ КД») на основе собственного аутентичного перевода стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 125 «Статистические методы в управлении качеством продукции»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от ____________2012 г. № ________

4 Настоящий стандарт идентичен международному стандарту ИСО 28640:2010 «Методы генерации случайных чисел» (ISO 28640:2010 «Random variate generation methods», IDT).

Наименование настоящего стандарта изменено относительно наименования указанного международного стандарта для приведения в соответствие с ГОСТ Р 1.5 - 2004 (подраздел 3.5).

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им национальные стандарты Российской Федерации, и межгосударственные стандарты, сведения о которых приведены в дополнительном приложении ДА

5 ВВЕДЕН ВПЕРВЫЕ

Информация об изменениях к настоящему стандарту публикуется в ежегодно издаваемом информационном указателе «Национальные стандарты», а текст изменений и поправок – в ежемесячно издаваемых информационных указателях «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ежемесячно издаваемом информационном указателе «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования - на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет

© Стандартинформ, 201

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

Введение

В настоящем стандарте установлены типовые алгоритмы, позволяющие генерировать последовательности псевдослучайных чисел, используемых при моделировании реализации случайной величины.

В настоящее время большое количество специалистов, работающих в области математической статистики, используют компьютерное моделирование. Поэтому очень важно, чтобы при этом были использованы псевдослучайные числа, хорошо согласующиеся с выбранным распределением. Настоящий стандарт также позволяет установить правильность рандомизации.

Существует шесть основных направлений использования рандомизации:

– отбор случайной выборки;

– анализ выборочных данных;

– разработка стандартов;

– проверка теоретических результатов;

– проверка того, что предложенная процедура соответствует заявленным свойствам;

– принятие решений в условиях неопределенности.

Приведенные в стандарте методы и алгоритмы обладают большим периодом повторения и хорошо согласуются с генерируемым законом распределения. При необходимости использования других алгоритмов генерации псевдослучайных чисел до их применения следует убедиться, что период последовательности псевдослучайных чисел является достаточным для решения задачи, а генерируемые псевдослучайные числа хорошо согласуются с моделируемым распределением.

Применяемый в настоящем стандарте международный стандарт разработан техническим комитетом ИСО/ТС 69 «Применение статистических методов».

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Статистические методы
ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ЧИСЕЛ

Statistical methods. Random variate generation

Дата введения – 2013 – 12 – 01

1 Область определения

В настоящем стандарте установлены методы генерации случайных чисел, подчиняющихся равномерному и другим законам распределения, используемых при применении метода Монте Карло. В настоящий стандарт не включены криптографические методы генерации случайных чисел. Настоящий стандарт будет полезен в первую очередь:

–научным работникам, технологам и специалистам в области систем управления, использующим статистическое моделирование;

–специалистам в области математической статистики, использующим рандомизацию при разработке методов статистического контроля качества продукции и процессов, планирования экспериментов и обработки данных;

–математикам, разрабатывающим сложные процедуры оптимизации с использованием метода Монте-Карло,

–разработчикам программного обеспечения при создании алгоритмов генерации псевдослучайных чисел.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ИСО/МЭК 2382-1 Информационная технология. Словарь. Часть 1. Основные термины (ISO/IEC 2382-1, Information technology – Vocabulary – Part 1: Fundamental terms)

ИСО 3534-1:2006 Статистика. Словарь и условные обозначения. Часть 1. Общие статистические термины и термины, используемые в вероятностных задачах (ISO 3534-1:2006 Statistics – Vocabulary and symbols – Part 1: Probability and general statistical terms)

ИСО 3534-2:2006 Статистика. Словарь и условные обозначения. Часть 2. Прикладная статистика (ISO 3534-2:2006 Statistics – Vocabulary and symbols – Part 2: Applied statistics)

3 Термины и определения

В настоящем стандарте применены термины по ИСО/МЭК 2382-1, ИСО 3534-1, ИСО 3534-2, а также следующие термины с соответствующими определениями:

3.1 **случайное число** (random variate, random number): Число, представляющее собой реализацию случайной величины.

Примечание 1 – Термин «случайное число» часто используют по отношению к равномерно распределенной случайной величине.

Примечание 2 – Случайные числа, представленные в виде последовательности, называют последовательностью случайных чисел.

3.2 **псевдослучайное число** (pseudo-random number): Число, полученное в соответствии с заданным алгоритмом, используемое в качестве случайного числа.

Примечание – В ситуациях, когда из контекста ясно, что речь идет о псевдослучайных числах, псевдослучайное число иногда называют «случайным числом».

3.3 **физическое случайное число** (physical random number): Случайное число (3.1), полученное на основе некоторого физического явления.

3.4 **двоичная последовательность случайных чисел** (binary random number sequence): Последовательность случайных чисел (3.1), состоящая из нулей и единиц.

3.5 **начальное число** (seed): Исходное число, необходимое для начала генерации псевдослучайных чисел.

4 Условные обозначения и математические операции над двоичными числами

4.1 Условные обозначения

В настоящем стандарте применены обозначения по ИСО/МЭК 2382-1, ИСО 3534-1, ИСО 3534-2, а также следующие условные обозначения и сокращения:

X – целое равномерно распределенное случайное число (целое случайное число, подчиняющееся равномерному распределению);

U – стандартное (из интервала [0, 1]) равномерно распределенное случайное число (случайное число из интервала [0, 1], подчиняющееся стандартному равномерному распределению);

Z – нормальная случайная величина (случайная величина, подчиняющаяся нормальному распределению);

n – индекс последовательности случайных чисел.

4.2 Математические операции над двоичными числами

В настоящем стандарте использованы следующие математические операции над двоичными числами:

mod (m; k) – остаток от деления целого числа m на целое число k;

$m \oplus k$ – побитовая логическая операция над двоичными целыми числами m и k «исключающее ИЛИ».

Пример 1 – Правила побитовой логической операции $\oplus$

$1 \oplus 1 = 0$,

$0 \oplus 1 = 1$,

$1 \oplus 0 = 1$,

$0 \oplus 0 = 0$.

Пример побитовой логической операции $\oplus$: $1010 \oplus 1100 = 0110$.

$m \wedge k$ – побитовая логическая операция «И» над двоичными целыми числами m и k.

Пример 2 – Правила побитовой логической операции $m \wedge k$

$1 \wedge 1 = 1$,

$0 \wedge 1 = 0$,

$1 \wedge 0 = 0$,

$0 \wedge 0 = 0$.

Пример побитовой логической операции $\wedge$: $1010 \wedge 1100 = 1000$.

$m := k$ – замена значения m на значение k;

$m >> k$ – сдвиг вправо двоичного целого числа m на k битов;

$m << k$ – сдвиг влево двоичного целого числа m на k битов.

5 Псевдослучайные числа (равномерное распределение)

5.1 Общие положения

В данном подразделе приведены алгоритмы генерации псевдослучайных чисел, соответствующих равномерному распределению, основанные на методах М-последовательности (см. 5.2).

В приложении А для информации приведен принцип генерации физических случайных чисел.

В приложении В приведены тексты компьютерных программ для всех рекомендуемых алгоритмов. Несмотря на то, что линейный конгруэнтный метод не рекомендован для решения сложных задач моделирования методом Монте-Карло, он также включен в приложение В для информации.

5.2 Метод М-последовательности

а) Для натурального числа p и чисел c_1, c_2, ..., c_{p-1}, принимающих значения 0 или 1, определяют рекуррентную формулу

$$x_{n+p} = c_{p-1}\, x_{n+p-1} + c_{p-2}\, x_{n+p-2} + \ldots + c_1\, x_{n+1} + x_n \pmod 2 \quad (n = 1, 2, 3, \ldots).$$

b) Наименьшее положительное целое N, такое, что $x_{n+N} = x_n$ для всех значений n называют периодом последовательности. Эту последовательность называют М-последовательностью, период которой составляет $(2^p - 1)$.

c) Полином

$$t^p + c_{p-1} t^{p-1} + ... + c_1 t + 1$$

является характеристическим полиномом для приведенной выше рекуррентной формулы.

Примечание 1 – Необходимым и достаточным условием того, что приведенная в перечислении а) формула может быть использована для генерации М-последовательности является то, что хотя бы одно из начальных чисел $x_1, x_2, \ldots, x_p$ отлично от нуля.

Примечание 2 – Буква М в обозначении М-последовательности является первой буквой английского слова «maximum» (наибольший). Период любой последовательности, сгенерированной по приведенной в перечислении а) рекуррентной формуле, не может быть больше $(2^p - 1)$. Поэтому, если есть ряд с периодом $(2^p - 1)$, это ряд с наибольшим периодом.

Примечание 3 – При использовании данного метода в качестве характеристического полинома применяют или один из полиномов, приведенных в таблице 1, или другой, более простой полином из справочной литературы, а его коэффициенты используют в формуле перечисления а).

5.3 Пятипараметрический метод

Данный метод использует характеристический полином из 5 членов и позволяет генерировать последовательности w-битовых двоичных целых чисел в соответствии со следующей рекуррентной формулой. Такой алгоритм называют GFSR[1)] или генератором случайных чисел «сдвиговый регистр с обратной связью».

$$X_{n+p} = X_{n+q_1} \oplus X_{n+q_2} \oplus X_{n+q_3} \oplus X_n \quad (n = 1, 2, 3...).$$

Параметры (p, q_1, q_2, q_3, w), и X_1, ..., X_p первоначально задают как начальные числа. Примеры параметров p, q_1, q_2, q_3 с наибольшим периодом $(2^p - 1)$ приведены в таблице 1.

[1)] GFSR – Generalized Feedback Shift Register.

Таблица 1 – Пятипараметрические характеристические полиномы

p	q_1	q_2	q_3
89	20	40	69
107	31	57	82
127	22	63	83
521	86	197	447
607	167	307	461
1279	339	630	988
2203	585	1197	1656
2281	577	1109	1709
3217	809	1621	2381
4253	1093	2254	3297
4423	1171	2273	3299
9689	2799	5463	7712

Примечание – q_1, q_2, q_3 являются показателями степени ненулевых членов характеристического полинома.

5.4 Комбинированный метод Таусворта

При генерации случайных чисел методом Таусворта используют рекуррентную формулу

$$x_{n+p} = (x_{n+q} + x_n) \ (\mathrm{mod}\ 2), \quad (n = 0, 1, 2, \ldots),$$

где $x_0, x_1, x_2, \ldots$ – соответствующая М-последовательность.

При использовании такой М-последовательности последовательность w-битовых целых чисел, называемую простой последовательностью Таусворта с параметрами (p, q, t), получают по формуле

$$X_n = x_{nt}\, x_{nt+1} \ldots x_{nt+w-1}, \quad (n = 0, 1, 2, \ldots),$$

где t – натуральное число взаимно простое с периодом $(2^p - 1)$ М-последовательности;

w – длина слова, не превышающая p бит.

Период этой последовательности составляет $(2^p - 1)$.

Примечание 1 – Два целых числа являются взаимно простыми или относительно простыми, если у них нет общих делителей кроме единицы.

Пример – Если в качестве исходного выбран многочлен $t^4 + t + 1$, установлены параметры $p=4$ и $q=1$ и в приведенной выше рекуррентной формуле заданы начальные числа $(x_0, x_1, x_2, x_3) = (1,1,1,1)$, то М-последовательность, полученная в соответствии с рекуррентной формулой, будет иметь вид: 1,1,1,1, 0,0,0,1, 0,0,1,1, 0,1,0,1, 1,1,1,0, …, с периодом $(2^4 - 1) = 15$. В случае $t = 4$ (4 является взаимно простым числом по отношению к 15) и $w = 4$ простая последовательность Таусворта $\{X_n\}$ с параметрами (4, 1, 4) имеет вид

$$X_0 = x_0\, x_1\, x_2\, x_3 \qquad = 1111\ (= 15),$$

$$X_1 = x_4\, x_5\, x_6\, x_7 \qquad = 0001\ (= 1),$$

$$X_2 = x_8\, x_9\, x_{10}\, x_{11} \qquad = 0011\ (= 3),$$

$$X_3 = x_{12}\, x_{13}\, x_{14}\, x_0 \qquad = 0101\ (= 5),$$

$$X_4 = x_1\, x_2\, x_3\, x_4 \qquad = 1110\ (= 14),$$

$X_5 = x_5\, x_6\, x_7\, x_8 \qquad = 0010\ (= 2).$

. . . .

Простая последовательность Таусворта, полученная таким образом в десятичных числах, имеет вид: 15, 1, 3, 5, 14, 2, 6, 11, 12, 4, 13, 7, 8, 9, 10, 15, 1, 3, …, с периодом $(2^4 - 1) = 15$.

Если имеется несколько, например, J простых последовательностей Таусворта $\{X_n^{(j)}\}$, j = 1, 2, ..., J с одной и той же длиной слова w, комбинированный метод Таусворта генерирует последовательность псевдослучайных чисел $\{X_n\}$ как результат побитовой операции «исключающее ИЛИ» при двоичном представлении чисел в этих J последовательностях.

$X_n = X^{(1)}{}_n \oplus X^{(2)}{}_n \oplus \ldots \oplus X^{(J)}{}_n\ (n = 0, 1, 2, \ldots).$

Параметры и начальные числа комбинированной последовательности Таусворта представляют собой комбинацию параметров и начальных чисел каждой простой последовательности Таусворта. Если периоды J простых последовательностей Таусворта являются взаимно простыми, то период комбинированной последовательности Таусворта равен произведению периодов J последовательностей.

Примечание 2 – Данный метод может генерировать последовательности чисел многомерного равномерного распределения. Алгоритм taus88_31(), приведенный в приложении А, позволяет генерировать последовательность 31-битовых целых чисел, комбинируя три простых генератора Таусворта с параметрами (p, q, t) равными (31, 13, 12), (29, 2, 4) и (28, 3, 17) соответственно. Длина периода комбинированной последовательности $(2^{31} - 1)\,(2^{29} - 1)\,(2^{28} - 1)$, т. е. приблизительно 2^{88}. Другие комбинации предложены в [7] и [8].

5.5 Метод Мерсенна Твистера

Метод Мерсенна Твистера позволяет генерировать последовательность двоичных псевдослучайных целых w-битовых чисел в соответствии со следующей рекуррентной формулой

$$X_{n+p} = X_{n+q} \oplus (X^{\mathrm{f}}{}_n | X^{\mathrm{l}}{}_{n+1})^{(r)}\, \mathbf{A},\ (n = 1, 2, 3, \ldots),$$

где p, q, r – целые константы;

a – двоичное w-битовое целое число (формирующее матрицу А);

X_n – w-битовое двоичное целое число;

$(X^{\mathrm{f}}{}_n | X^{\mathrm{l}}{}_{n+1})^{(r)}$ – двоичное целое число, полученное конкатенацией чисел $X^{\mathrm{f}}{}_n$ и $X^{\mathrm{l}}{}_{n+1}$, когда первые $(w - r)$ битов взяты из X_n, а последние r битов из (X_{n+1}) в том же порядке;

A – матрица размера $w \times w$, состоящая из нулей и единиц, определенная посредством a;

$X \cdot \mathbf{A}$ – произведение, при вычислении которого сначала выполняют операцию $X >> 1$, если последний бит X равен 0, а затем, когда последний бит $X = 1$, вычисляют $X\mathbf{A} = (X >> 1) \oplus a$.

(Здесь X также как и a представляет собой w-мерный вектор, состоящий из нулей и единиц).

Примечание – Необходимый объем памяти для этих вычислений – p слов, период – $(2^{pw-r} - 1)$, а эффективность метода Мерсена Твистера выше эффективности метода GFSR. Для улучшения рандомизации первых $(w-r)$ битов можно применить следующий ряд преобразований X_n

$y := X_n$,

$y := y \oplus (y >> u)$,

$y := y \oplus [(y << s) \wedge b]$,

$y := y \oplus [(y << t) \wedge c]$,

$y := y \oplus (y >> l)$,

где b, c – постоянные маски битов для улучшения рандомизации первых $(w-r)$ битов. Параметрами этого алгоритма являются $(p, q, r, w, a, u, s, t, l, b, c)$. Начальными числами являются $X_2, ..., X_{q+1}$ и первые $(w-r)$ битов числа X_1.

Заключительное значение y является псевдослучайным числом.

6 Генерация случайных чисел

6.1 Введение

В данном разделе приведено описание методов генерации случайных чисел Y, соответствующих различным распределениям, при использовании целых случайных чисел X, соответствующих равномерному распределению. При этом использованы следующие обозначения:

$F(y)$ – функция распределения;

$f(y)$ – функция плотности вероятности непрерывного распределения;

$p(y)$ – функция дискретного распределения.

6.2 Равномерное распределение

6.2.1 Стандартное равномерное распределение

6.2.1.1 Функция плотности вероятности

$$f(y) = \begin{cases} 1, & \text{если } 0 \le y \le 1 \\ 0, & \text{если } y \notin [0,1] \end{cases}$$

6.2.1.2 Метод генерации случайной величины

Если максимальное значение равномерного случайного целого числа X равно $(m-1)$, для генерации стандартных равномерных случайных чисел необходимо применять следующую формулу

$$U = \frac{X}{m}.$$

Пример – Для всех w-битовых последовательных целых чисел, генерированных методом, описанным в 5.2 с помощью 5.5, m = 2.

Примечание 1 – Поскольку X принимает дискретные значения, величина U также является дискретной.

Примечание 2 – Величина U никогда не принимает значения 1 и 0. Величина U принимает значение 0,0 только, если X = 0. В случае М-последовательности случайных чисел любой метод генерации может выявить эту особенность.

Примечание 3 – Случайные числа, соответствующие стандартному равномерному распределению, называют стандартными равномерными случайными числами и обозначают U_1, U_2, …. Эти числа считают независимыми по отношению друг к другу.

6.2.2 Общий случай равномерного распределения

6.2.2.1 Функция плотности вероятности

$$f(y) = \begin{cases} 1/b, & \text{если } a \le y \le a+b \\ 0, & \text{если } y \notin [a, a+b] \end{cases},$$

где $b > 0$.

6.2.2.2 Метод генерации случайной величины

Если стандартное равномерное случайное число U получено методом, установленным в 6.2.1.2, то равномерное случайное число должно быть получено в соответствии со следующей формулой

$$Y = bU + a.$$

6.3 Стандартное бета-распределение

6.3.1 Функция плотности вероятности

$$f(y) = \begin{cases} \dfrac{y^{c-1}(1-y)^{d-1}}{\mathrm{B}(c,d)}, & \text{если } 0 \le y \le 1 \\ 0, & \text{если } y \notin [0,1] \end{cases},$$

где $\mathrm{B}(c,d) = \int_0^1 x^{c-1}(1-x)^{d-1}\,\mathrm{d}x$ – бета-функция с параметрами c и d ($c > 0$, $d > 0$).

6.3.2 Метод Йонка

Если стандартные равномерные случайные числа U_1 и U_2 независимо генерированы методом, установленным в 6.2.1, то соответствующее стандартному бета-распределению случайное число Y получают в соответствии со следующими процедурами.

Если $\tilde{Y} = U_1^{1/c} + U_2^{1/d} \le 1$, то $Y = U_1^{1/c} / \tilde{Y}$; в противном случае генерируют два новых стандартных равномерных случайных числа до тех пор, пока неравенство не будет выполнено.

6.3.3 Метод Ченга

Если стандартные равномерные случайные числа U_1 и U_2 независимо получены методом, установленным в 6.2.1, то случайное число Y, соответствующее стандартному бета-распределению, получают в соответствии со следующей процедурой

a) $$q = \begin{cases} \min(c, d), & \text{если } \min(c,d)<1 \\ \sqrt{\dfrac{2cd-(c+d)}{c+d-2}}, & \text{в противном случае} \end{cases}.$$

b) Вычисляют

$$V = \frac{1}{q} \cdot \frac{U_1}{(1-U_1)}, \quad W = c\exp(V).$$

c) Если

$$(c+d)\ln\left(\frac{c+d}{d+W}\right) + (c+q)V - \ln 4 \ge \ln\left(U_1^2 U_2\right),$$

тогда

$$Y = \frac{W}{d+W} \quad \text{(выход)}.$$

d) В противном случае генерируют U_1, U_2 и переходят к b).

Если $\max(c, d) \le 1$, применяют метод Йонка, в противном случае применяют метод Ченга.

Примечание – Случайные числа, соответствующие бета-распределению, заданному на интервале $[a, a+b]$, получают линейным преобразованием аналогично описанному в 6.2.2.2.

6.4 Треугольное распределение

6.4.1 Функция плотности вероятности

$$f(y) = \begin{cases} \dfrac{b-|a-y|}{b^2}, & \text{если } a-b \le y \le a+b \\ 0, & \text{если } y \notin [a-b, a+b] \end{cases},$$

где $b > 0$.

6.4.2 Метод генерации случайной величины

Если стандартные равномерные случайные числа U_1 и U_2 независимо генерированы методом, установленным в 6.2.1, то случайное число Y, подчиняющееся треугольному распределению, определяют по формуле $Y = a + b(U_1 + U_2 - 1)$.

6.5 Общее экспоненциальное распределение с параметрами положения и масштаба

6.5.1 Функция плотности вероятности

$$f(y) = \begin{cases} \dfrac{1}{b}\exp\{-(y-a)/b\}, & y \ge a \\ 0, & y < a \end{cases},$$

где a и b – параметры положения и масштаба экспоненциального распределения соответственно.

6.5.2 Метод генерации случайной величины

Если стандартное равномерное случайное число U генерировано методом, установленным в 6.2.1, то случайное число, соответствующее экспоненциальному распределению, получают по формуле

$Y = -b\ln(U) + a$.

6.6 Нормальное распределение (распределение Гаусса)

6.6.1 Функция плотности вероятности

$$f(z) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(z-\mu)^2\right\}, \quad -\infty < z < \infty,$$

где μ и σ – среднее и стандартное отклонение нормального распределения соответственно.

Примечание – Обычно нормальную случайную величину обозначают Z.

6.6.2 Метод Бокса-Мюллера

Если стандартные равномерные случайные числа U_1 и U_2 независимо генерированы методом, установленным в 6.2.1, то два независимых нормальных случайных числа Z_1, Z_2 получают в соответствии со следующей процедурой

$$Z_1 = \mu + \sigma\sqrt{-2\ln(1-U_1)}\cos(2\pi U_2),$$

$$Z_2 = \mu + \sigma\sqrt{-2\ln(1-U_1)}\sin(2\pi U_2).$$

Примечание 1 – Поскольку U_1 – дискретная величина, то Z_1, Z_2 не подчиняются нормальному распределению в строгом смысле. Например, используя эту процедуру, верхней границей абсолютных значений псевдослучайных стандартных нормальных величин является $M = \sqrt{-2\ln(m^{-1})} = \sqrt{2\ln m}$. Таким образом, если $m = 2^{32}$, то $M \approx 6{,}6604$, а если $m = (2^{31} - 1)$, то $M \approx 6{,}5555$. Однако, так как вероятность того, что абсолютные значения случайных величин истинного стандартного нормального распределения, превышающих M, приблизительно равна 10^{-10}, это редко создает трудности на практике.

Примечание 2 – При получении U_1, U_2 линейным конгруэнтным методом последовательно, U_1 и U_2 являются зависимыми, таким образом хвост распределений, полученных Z_1 и Z_2, может существенно отличаться от истинного нормального распределения.

6.7 Гамма-распределение

6.7.1 Функция плотности вероятности

$$f(y) = \begin{cases} \dfrac{1}{b\Gamma(c)}\{(y-a)/b\}^{c-1}\exp\{-(y-a)/b\}, & \text{если } y \ge a \\ 0, & \text{если } y < a \end{cases},$$

где a, b, c – параметры положения, масштаба и формы соответственно.

6.7.2 Методы генерации случайной величины

6.7.2.1 Общие положения

Алгоритмы приведены для трех ситуаций в зависимости от значения параметра формы c.

6.7.2.2 Алгоритм для $c = k$ (k – целое число)

Используя независимые равномерные случайные числа U_1, U_2, ..., U_k, применяют формулу

$$Y = a - b\ln\{(1 - U_1)(1-U_2)\ \ldots\ (1-U_k)\}.$$

Примечание – Этим методом для $a = 0$ и $b = 2$ может быть получено распределение χ^2 с четным числом степеней свободы.

6.7.2.3 Алгоритм для $c = k + 1/2$ (k – целое число)

Используя стандартное нормальное случайное число Z_0 и равномерные случайные числа U_1, U_2, ..., U_k, применяют формулу

$$Y = a + b\left[Z_0^2/2 - \ln\left\{(1-U_1)(1-U_2)\ldots(1-U_k)\right\}\right].$$

В случае, когда $k = 0$, член с логарифмом исчезает.

Примечание – Тем же методом при $a = 0$ и $b = 2$ может быть получено распределение χ^2 с нечетным числом степеней свободы.

6.7.2.4 Приближенный метод для $c > 1/3$

a) Задают $r = c - 1/3$, $s = \sqrt[3]{r}$, $t = r - r\ln(r)$, $p = 1/(3\sqrt{s})$ и $q = -3\sqrt{r}$.

b) Генерируют стандартное нормальное случайное число Z.

c) Если $Z < q$, то переходят к b).

d) Вычисляют $Y = (pZ + s)^3$, $V = Z^2/2$ и генерируют U.

e) Если $(Y - r)^2/Y - V \le U$, выполняют $Y := a + bY$ (конец).

f) Вычисляют $W = Y - r\ln(Y) - t - V$.

g) Если $W \le U$, то выполняют $Y := a + bY$ (конец).

h) Если $W > -\ln(1{,}0 - U)$, то переходят к b).

Примечание – Метод основан на преобразовании Уилсона-Хилферти, приводящем χ^2-распределение к приближенному стандартному нормальному распределению. Точность такого приближения зависит от значения параметра c. Идея преобразования состоит в следующем: абсолютная разность между процентной точкой приближенного и точного распределений всегда меньше 0,2.

6.7.2.5 Точный метод генерации Ченга для $c > 1/2$

a) Задают $q = c - \ln 4$ и $r = c + \sqrt{2c-1}$.

b) Генерируют стандартные равномерные случайные числа U_1 и U_2.

c) Вычисляют $V = c\ln\left(\dfrac{U_1}{1-U_1}\right)$, $W = c\cdot\exp(U_1)$, $Z = U_1^2U_2$, $R = q + rV - W$.

d) Если $R \ge 4{,}5Z - (1 + \ln 4{,}5)$, то вычисляют $Y = a + bW$ (выход).

e) Если $R \ge \ln Z$, то вычисляют $Y = a + bW$ (выход).

f) Генерируют стандартные равномерные случайные числа U_1 и U_2 и вычисляют

$p = 1/\sqrt{2c-1}$, $q = c - \ln 4$, $r = c + \sqrt{2c-1}$.

Если $q + pr\ln\left(\dfrac{U_1}{1-U_1}\right) - c\left(\dfrac{U_1}{1-U_1}\right)^p \geq 4,5(U_1^2 U_2) - (1+\ln 4,5)$,

то $Y = a + bc\left(\dfrac{U_1}{1-U_1}\right)^p$ (выход)

6.8 Распределение Вейбулла

6.8.1 Функция распределения вероятности

$$F(y) = \begin{cases} 1-\exp\left\{-\left(\dfrac{y-a}{b}\right)^c\right\}, & \text{если } y \geq a \\ 0, & \text{если } y < a \end{cases},$$

где a, b и c – параметры положения, масштаба и формы соответственно.

6.8.2 Метод генерации случайной величины

Если стандартные равномерные случайные числа U генерированы методом, установленным в 6.2.1, то случайные числа, соответствующие распределению Вейбулла, получают по формуле

$Y = a - b\{\ln(1 - U)\}^{1/c}$.

6.9 Логнормальное распределение

6.9.1 Функция плотности вероятности

$$f(y) = \begin{cases} \dfrac{1}{\sqrt{2\pi}\{(y-a)/b\}}\exp\left\{-\dfrac{1}{2}\left(\dfrac{y-a}{b}\right)^2\right\}, & \text{если } y \geq a \\ 0, & \text{если } y < a \end{cases},$$

где a и b – параметры положения и масштаба, соответствующего нормального распределения.

6.9.2 Метод генерации случайной величины

Используя стандартные нормальные случайные числа Z, применяют формулу

$Y = a + \exp(b \cdot Z)$

для получения случайных чисел, соответствующих логнормальному распределению.

6.10 Логистическое распределение

6.10.1 Функция вероятности

$$F(y) = \frac{1}{1+\exp\{-(y-a)/b\}}, \qquad -\infty < y < \infty,$$

где a и b – параметры положения и масштаба соответственно.

6.10.2 Метод генерации случайной величины

Если стандартные равномерные случайные числа U генерированы методом, установленным в 6.2.1, то случайные числа, соответствующие логистическому распределению, получают по формуле

$$Y = a + b\ln\left(\frac{U}{1-U}\right).$$

6.11 Многомерное нормальное распределение

Случайные числа $Y_1, Y_2, \ldots, Y_n$, соответствующие n-мерному нормальному распределению со средними $\mu_1, \mu_2, \ldots, \mu_n$, дисперсиями и ковариациями σ_{ij} $(1 \le i, j \le n)$, получают, используя взаимно независимые стандартные нормальные случайные числа $Z_1, \ldots, Z_n$

$$Y_1 = \mu_1 + a_{11} Z_1,$$

$$Y_2 = \mu_2 + a_{21}Z_1 + a_{22}Z_2,$$

$$\ldots$$

$$Y_n = \mu_n + a_{n1}Z_1 + a_{n2}Z_2 + \ldots + a_{nn}Z_n,$$

где $a_{11}, \ldots, a_{nn}$ – константы, вычисляемые до начала генерации в соответствии с процедурой факторизации Холецкого.

Примечание – σ_{ij} $(1 \le i, j \le n)$ ковариации, σ_{ii} – дисперсии Y_i.

a) Для $j = 1$ $\quad a_{11} = \sqrt{\sigma_{11}}, \quad a_{i1} = \sigma_{i1}/a_{11}$ $(2 \le i \le n)$.

b) Для $j = 2, \ldots, n$

$$a_{jj} = \left(\sigma_{jj} - \sum_{k=1}^{j-1} a_{jk}^2\right)^{\frac{1}{2}},$$

$$a_{ij} = \left(\sigma_{ij} - \sum_{k=1}^{j-1} a_{ik} a_{jk}\right) \Big/ a_{jj} \ (j+1 \le i \le n).$$

6.12 Биномиальное распределение

6.12.1 Функция распределения

Если вероятность появления события при каждом испытании равна p, то вероятность того, что это событие произойдет y раз за n испытаний, определяют по формуле

$$p(y) = \binom{n}{y} p^y (1-p)^{n-y}, \ y = 0, 1, \ldots, n,$$

где $0 < p < 1$.

6.12.2 Методы генерации случайной величины

6.12.2.1 Общие положения

Рассматриваемые в данном разделе методы позволяют получить случайные числа Y, соответствующие биномиальному распределению.

6.12.2.2 Прямой метод

Генерируют n стандартных равномерных случайных чисел U. Искомое число Y равно количеству чисел U менее p из n полученных чисел U.

6.12.2.3 Метод обратной функции

Вычисляют функцию распределения

$$F(y) = \sum_{k=0}^{y} \binom{n}{k} p^k (1-p)^{n-k}, \qquad y = 0, 1, ..., n.$$

Для получения случайного числа Y генерируют стандартное равномерное случайное число U. Случайное число Y является наименьшим значением y, для которого $U \le F(y)$.

6.12.2.4 Метод положения

Вычисляют $(n + 1)$ параметров $v_0, v_1, ..., v_n$ и $(n + 1)$ параметров $a_0, a_1, ..., a_n$

a) $v_y = (n + 1)\, p(y),\ y = 0, 1, \ldots, n$.

b) Составляют набор индексов G таких, для которых соответствующий параметр v удовлетворяет условию $v_y \ge 1$, и набор индексов S, для которых соответствующий параметр v удовлетворяет условию $v_y < 1$.

c) Для не пустого набора S выполняют операции 1) – 4).

1) Выбирают любой элемент i из G и любой элемент j от S.

2) Устанавливают $a_j = i$ и $v_i = v_i - (1 - v_j)$.

3) Если $v_i < 1$, удаляют элемент i из G и перемещают его в S.

4) Удаляют элемент j из S.

Если приведенные выше подготовительные действия выполнены, то двухмерное случайное число Y получают, выполняя операции d) – f).

d) Генерируют стандартное равномерное случайное число U и вычисляют $V = (n+1)U$.

e) Вычисляют $k = \lfloor V \rfloor$ и $u = V - k$, где $\lfloor V \rfloor$ – целая часть числа V.

f) Если $u \le v_k$, то $Y = k$; в противном случае $Y = a_k$.

6.13 Распределение Пуассона

6.13.1 Функция распределения

Функция распределения Пуассона со средним μ имеет вид

$$p(y) = \frac{\mu^y}{y!} \exp(-\mu),\ y = 0, 1, 2, \ldots,$$

где $\mu > 0$.

6.13.2 Метод, использующий связь с экспоненциальным распределением

Генерируют стандартные равномерные случайные числа $U_1, U_2, \ldots$. В качестве Y используют максимальное значение n, удовлетворяющее следующему неравенству

$-\ln\{(1-U_1)(1-U_2) \ldots (1-U_n)\} < \mu$.

6.13.3 Метод наложения

Сначала выбирают постоянную n, для которой вероятность того, что $Y > n$ пренебрежимо мала, например, целая часть числа ($\mu + 6\sqrt{\mu}$) может быть установлена равной n. Затем применяют процедуру 6.12.2.4, приведенную для биномиального распределения. Однако на сей раз для $p(y)$ должна быть использована функция распределения Пуассона.

Примечание – Этот метод эффективен, когда μ имеет значение от 10 до 100.

6.14 Дискретное равномерное распределение

Для генерации дискретных равномерных случайных чисел от M до N r-битовое двоичное случайное число, полученное в соответствии с рекомендациями 5.1, преобразуют в соответствии со следующими процедурами, где $(N - M + 1)$ не превышает 2^r.

a) Определяют натуральное число k, удовлетворяющее следующему неравенству:

$2^{k-1} + 1 \leq N - M + 1 \leq 2^k$.

Примечание 1 – Величина k – наименьшее натуральное число, удовлетворяющее неравенству $k \geq \log_2(N - M + 1)$.

Пример 1 – Если $(N - M + 1) = 100$, то $k = 7$, поскольку$(2^6 + 1) = 65 \leq 100 \leq 2^7 = 128$.

b) Добавляют 1 к двоичному целому числу, которое сформировано из первых k битов случайного числа, и конвертируют его в десятичное число.

Примечание 2 – k-битовое двоичное число $Z_1Z_2Z_3Z_4\ldots Z_k$ соответствует десятичному числу $2^{k-1}Z_1 + 2^{k-2}Z_2 + 2^{k-3}Z_3 + 2^{k-4}Z_4 + \ldots + Z_k$.

Пример 2 – Если первые 7 битов числа 1011001, то соответствующим десятичным числом является 89 (64 + 16 + 8 + 1 = 89).

c) Искомое десятичное случайное число – это соответствующее десятичное число плюс $(M - 1)$ при отбрасывании чисел более N.

Примечание 3 – Если $(N - M + 1) > 2^r$, то искомое десятичное случайное число может быть получено конкатенацией двух или большего количества двоичных случайных чисел в одно двоичное случайное число.

Примечание 4 – При использовании линейного конгруэнтного метода для генерации псевдослучайных чисел k не должно быть равным r.

Далее, если $(N - M + 1)$ является десятичным k-значным натуральным числом и k не является слишком большим, например k меньше 20, может быть использован метод, установленный в 5.2. При этом выполняют процедуру в соответствии с d) и e).

d) Генерируют последовательность десятичных случайных чисел из k цифр, используя процедуру 5.2.

e) Из последовательности случайных чисел, полученной в соответствии с d), удаляют числа более N. Полученная таким образом последовательность является искомой последовательностью десятичных случайных чисел.

Приложение А
(справочное)
Таблица физических случайных чисел

А.1 Таблица случайных чисел

В отличие от псевдослучайных чисел у физических случайных чисел отсутствуют функциональная связь и периодичность. В таблице А.1 приведена последовательность случайных чисел, полученных в результате измерений характеристики физической системы со случайными свойствами.

Таблица А.1 – Таблица физических случайных чисел

1	93	90	60	02	17	25	89	42	27	41	64	45	08	02	70	42	49	41	55	98
2	34	19	39	65	54	32	14	02	06	84	43	65	97	97	65	05	40	55	65	06
3	27	88	28	07	16	05	18	96	81	69	53	34	79	84	83	44	07	12	00	38
4	95	16	61	89	77	47	14	14	40	87	12	40	15	18	54	89	72	88	59	67
5	50	45	95	10	48	25	29	74	63	48	44	06	18	67	19	90	52	44	05	85
6	11	72	79	70	41	08	85	77	03	32	46	28	83	22	48	61	93	19	98	60
7	19	31	85	29	48	89	59	53	99	46	72	29	49	06	58	65	69	06	87	09
8	14	58	90	27	73	67	17	08	43	78	71	32	21	97	02	25	27	22	81	74
9	28	04	62	77	82	73	00	73	83	17	27	79	37	13	76	29	90	07	36	47
10	37	43	04	36	86	72	63	43	21	06	10	35	13	61	01	98	23	67	45	21
11	74	47	22	71	36	15	67	41	77	67	40	00	67	24	00	08	98	27	98	56
12	48	85	81	89	45	27	98	41	77	78	24	26	98	03	14	25	73	84	48	28
13	55	81	09	70	17	78	18	54	62	06	50	64	90	30	15	78	60	63	54	56
14	22	18	73	19	32	54	05	18	36	45	87	23	42	43	91	63	50	95	69	09
15	78	29	64	22	97	95	94	54	64	28	34	34	88	98	14	21	38	45	37	87
16	97	51	38	62	95	83	45	12	72	28	70	23	67	04	28	55	20	20	96	57
17	42	91	81	16	52	44	71	99	68	55	16	32	83	27	03	44	93	81	69	58
18	07	84	27	76	18	24	95	78	67	33	45	68	38	56	64	51	10	79	15	46
19	60	31	55	42	68	53	27	82	67	68	73	09	98	45	72	02	87	79	32	84
20	47	10	36	20	10	48	09	72	35	94	12	94	78	29	14	80	77	27	05	67
21	73	63	78	70	96	12	40	36	80	49	23	29	26	69	01	13	39	71	33	17
22	70	65	19	86	11	30	16	23	21	55	04	72	30	01	22	53	24	13	40	63
23	86	37	79	75	97	29	19	00	30	01	22	89	11	84	55	08	40	91	26	61
24	28	00	93	29	59	54	71	77	75	24	10	65	69	15	66	90	47	90	48	80
25	40	74	69	14	01	78	36	13	06	30	79	04	03	28	87	59	85	93	25	73

А.2 Метод генерации физических случайных чисел

Для генерации случайных чисел, приведенных в таблице А.1, использован электрический шум диода. В диоде шумовой сигнал достаточно велик, вследствие эффекта лавинного нарастания заряда. Поэтому диод часто используют как источник шума. Был использован лавинно-пролетный диод NC2401[1]. У этого элемента есть источник шума и встроенный усилитель, ширина полосы частот которого составляет 1 ГГц, а амплитуда – 160 мВ.

Методы преобразования шумового сигнала в цифровую форму:

a) аналогово-цифровое преобразование;

b) наблюдение последовательности импульсов с определением количества импульсов в единицу времени;

c) наблюдение последовательности импульсов с определением интервала времени между последовательными импульсами.

Например, для аналогово-цифрового преобразования может быть использован конвертер DAS-4102[2]. У этого оборудования разрешающая способность составляет 8 битов с максимальным периодом отбора данных 64 МГц. Данные для приведенной таблицы были отобраны за 1 МГц. Измерение было выполнено с разрешением 3,91 мВ на цифру, и только низший бит был использован для получения случайного числа.

Поскольку у аналогово-цифрового конверсионного оборудования могут появляться ошибочные значения, гистограммы значений после преобразования не показывают равномерного распределения. Для получения большей равномерности распределения 2 бита были получены из одного и того же источника случайных чисел

(0,1) → Случайное число (Rn) = 0,

(1,0) → Случайное число (Rn) = 1,

(0,0), (1,1) → не использованы.

Случайные числа, приведенные в таблице 1, получены в соответствии с вышеупомянутым методом. Если вероятности появления чисел (0,1) и (1,0) равны друг другу, случайное число распределено равномерно. Поскольку интервал между последовательными измерениями равен 1 мс, характеристики оборудования можно считать постоянными. Поэтому (0,1) и (1,0) считают соответствующими одному и тому же распределению вероятностей. Альтернативный метод корректировки сводится к определению распределения вероятностей характеристик, но поскольку распределение зависит от особенностей оборудования, этот метод в данном случае не был использован. Далее, для безопасности 32 бита были собраны в одну единственную группу, или были сформированы из псевдослучайных чисел с использованием метода Мерсенна Твистера (обычно называемого genrand), установленного в 5.5. Метод Мерсенна Твистера обычно инициируют функцией init_genrand (s), где s = 19660809. Если необходимы оригинальные последовательности случайных чисел, они могут быть

[1] NC2401 – торговая марка продукта, поставляемого Noisecom (информация дана только для удобства пользователей настоящего стандарта).

[2] DAS-4102 – торговая марка продукта, поставляемого Keithley Instruments, Inc. (информация дана только для удобства пользователей настоящего стандарта).

восстановлены с помощью единственного или повторного использования метода Мерсенна Твистера.

В таблице А.1 приведены десятичные случайные числа, полученные вышеупомянутым методом, путем отбора высших четырех бит 32-битового представления случайного числа. Если значение этого числа не меньше нуля и не больше девяти, значение используют в качестве случайного числа. Однако, если значение этого числа 10 или больше, его отбрасывают и генерируют следующее случайное число.

Приложение В

(справочное)

Алгоритмы генерации псевдослучайных чисел

В.1 Текст программы трехпараметрического метода GFSR

Ниже приведен текст программы на языке Си, которая в соответствии с ИСО/МЭК 9899 является примером метода GFSR с параметрами (p, q, w) = (1279, 418, 32) и периодом ($2^{1279}-1$). При обращении к функции gfsr () происходит генерация целого числа из интервала от 0 до ($2^{32}-1$) включительно. При обращении к функции gfsr_31 () происходит генерация целого числа из интервала от 0 до ($2^{31}-1$) включительно. Для обращения к функциям gfsr () и gfsr_31 () необходима единственная инициализация init_gfsr (s). Функция init_gfsr (s) выполняет инициализацию при условии, что в качестве начального числа используется 32-битовое целое число без знака [целое число из интервала от 0 до ($2^{32}-1$)]. Полученная последовательность обеспечивает 39 независимых серий псевдослучайных чисел, каждая из которых обладает незначительной автокорреляцией, имеет 39-мерное распределение (однородно распределена в 39-мерном гиперкубе) с 32-битовой точностью, и ее функция автокорреляции такова, что значения близкие к нулю появляются через 2^{1274} чисел.

Чтобы получить другую последовательность псевдослучайных чисел, необходимо изменить начальное число s в функции init_gfsr (s). В программе могут быть изменены только константы p, q, w. Значение w должно быть равно 2 в целой степени в соответствии с длиной слова машины. Значение w, в общем случае, равно 32 или 64 в зависимости от возможностей машины. Например, если длина слова машины 64, постоянную w в программе устанавливают равной 64, а функция gfsr ( ) генерирует целые числа из интервала от 0 до ($2^{64}-1$) включительно, в то время как функция gfsr_31 ( ) генерирует целые числа из интервала от 0 до ($2^{63}-1$) включительно.

В данной программе предполагается, что длина «беззнакового длинного целого» составляет не меньше 32 бит.

```
/************************************************
Текст программы трехпараметрического GFSR на языке Си
************************************************/
#define P 1279
#define Q 418
#define W 32 /* значения W должны быть степенью 2 */

static unsigned long state [P] ;
static int state_i ;
```

```
void init_gfsr (unsigned long s)
{
int i, j, k;
static unsigned long x [P] ;

s &= 0xffffffffUL;

  for (i=0 ; i<P ; i++) {
    x [i] = s>>31 ;
    s = 1664525UL * s + 1UL ;
    s &= 0xffffffffUL ;
  }
  for (k=0, i=0 ; i<P ; i++)    {
    state [i] = 0UL ;
    for (j=0 ; j<W ; j++)   {
      state [i] <<= 1 ;
      state [i] |= x [k] ;
      x [k] ^= x [ (k+Q) %P] ;
      k++;
      if (k==P) k = 0 ;
    }
  }
  state_i = 0 ;
}

unsigned long gfsr (void)
{
int i ;
unsigned long *p0, *p1 ;

    if (state_i >= P) {
      state_i = 0 ;
      p0 = state ;
      p1 = state + Q ;
      for (i=0 ; i<(P-Q) ; i++)
        *p0++ ^= *p1++ ;
      p1 = state ;
      for ( ; i<P ; i++)
        *p0++ ^= *p1++ ;
    }
return state [state_i++] ;
}
/* (W-1)-битовое целое */
long gfsr_31 (void)
{
  return (long) (gfsr( ) >>1) ;
}
```

Примечание – Соответствующий текст программы трехпараметрического GFSR на языке BASIC приведен для информации

```
REM /**********************************************
REM Текст программы трехпараметрического GFSR на языке BASIC
REM **********************************************/

OPTION BASE 0

REM
/*********************************************************************************/
DECLARE NUMERIC P
LET P=1279                                   !#define P 1279
DECLARE NUMERIC Q
LET Q=418                                    !#define Q 418
DECLARE NUMERIC W
                                             /* значения W должны быть степенью 2 */
LET W=32                                     !#define W 32

DIM state(P)                                 !static unsigned long state[P];
DECLARE NUMERIC state_i                      !static INT state_i;


REM
/*********************************************************************************/
FUNCTION init_gfsr(s)        !               ! void init_gfsr(unsigned long s){
    DECLARE NUMERIC i,j,k                    !     int i, j, k;
    DIM x(P)                                 !     static unsigned long x[P];
    LET s = And32(s , MskF_f)                !     s &= 0xffffffffUL;
    FOR i = 0 TO P -1                        !     for (i=0; i<P; i++) {
        LET x(i) = SR32U(s , 31)             !         x[i] = s>>31;
        LET s = Mul32U(1664525 , s) + 1      !         s = 1664525UL * s + 1UL;
        LET s = And32(s, MskF_f)             !         s &= 0xffffffffUL;
    NEXT I                                   !     }
    LET k=0
    FOR i = 0 TO P -1                        !     for (k=0, i=0; i<P; i++) {
        LET state(i) = 0                     !         state[i] = 0UL;
        FOR j=0 TO W-1                       !         for (j=0; j<W; j++) {
        LET state(i) = SL32U(state(i) , 1)   !         state[i]  <<= 1;
        LET state(i) = Or32(state(i) , x(k)) !         state[i]  |= x[k] ;
        LET x(k) = Xor32(x(k) , x            !         x[k] ^= x[(k+Q)%P] ;
        (REMAINDER(k + Q , P)))
        LET k = k + 1                        !         k++;
        IF k = P THEN LET k = 0              !         if (k==P) k = 0;
    NEXT j                                   !         }
NEXT I                                       !     }
LET state_i = 0                              !     state_i = 0;
END FUNCTION                                 !}
```

```
REM
/*****************************************************************************/
FUNCTION gfsr                                          !unsigned long gfsr(void){
      DECLARE  NUMERIC I                               !      int i;
      DECLARE  NUMERIC p0, p1                          !      unsigned  long *p0, *p1;
      IF state_i >= P THEN                             !      if (state_i >= P) {
        LET state_i = 0                                !        state_i = 0;
        LET p0 = 0                                     !        p0 = state;
        LET p1 = Q1                                    !        p1 = state + Q1;
        FOR i=0 TO P-Q-1                               !        for (i=0; i<(P-Q); i++)
           LET state(p0) = Xor32(state(p0) , state(p1))
           LET p0 = p0 + 1
           LET p1 = P1 + 1                             !      *p0++ ^= *p1++;
        NEXT i
      LET p1 = 0                                       !      p1 = state;
      FOR i=i TO P-1                                   !      for (; i<P; i++)
          LET state(p0) = Xor32(state(p0) , state(p1))
          LET p0 = p0 + 1
          LET p1 = P1 + 1                              !      *p0++ ^= *p1++;
      NEXT i
   END IF                                              !      }
   LET gfsr = state(state_i)
   LET state_i = state_i + 1                           !      return state[state_i++];
END FUNCTION                                           !      }
REM
/*****************************************************************************/
REM /* (W-1)-битовое целое */
FUNCTION gfsr_31                                       !long gfsr_31(void){
      LET gfsr_31 = SR32U(gfsr , 1)                    !      return (long)(gfsr()>>1);
END FUNCTION                                           !}
```

В.2 Текст программы пятипараметрического метода GFSR

Параметры и период данной программы составляют (521, 86, 197, 447, 32) и ($2^{521}-1$). При обращении соответственно к функции gfsr5() происходит генерация целого числа из интервала от 0 до ($2^{32}-1$) включительно. При обращении к функции gfsr5_31() происходит генерация целого числа из интервала от 0 до ($2^{31}-1$) включительно. Функция init_gfsr5 (s) выполняет инициализацию при условии, что начальное число является 32-битовым целым числом без знака [целое число из интервала от 0 до ($2^{32}-1$)]. До обращения к функции gfsr5 () и gfsr5_31 () необходимо выполнить первоначальную инициацию init_gfsr5 (s). Полученная последовательность имеет 16-мерное распределение (равномерное распределение в 16-мерном гиперкубе) с 32-битовой точностью, а ее функция автокорреляции такова, что период появления близкого к нулю значения составляет 2^{516}.

При необходимости многократного получения наборов случайных чисел для моделирования инициализацию init_gfsr5 (s) необходимо выполнить один раз перед началом моделирования. После каждого повторения содержание таблицы x[P] размера P и переменную state_i необходимо сохранять и использовать их в качестве начальных значений при следующем повторении.

Если необходима другая последовательность с другим периодом, то значения p, q_1, q_2 и q_3 необходимо выбирать из таблицы 1.

```
/*****************************************************************************/
Текст программы пятипараметрического GFSR на языке Си
/*****************************************************************************/
#define P 521
                                                  /* Q1 < Q2 < Q3 */
#define Q1 86
#define Q2 197
#define Q3 447
#define W 32                                      /* W должно быть степенью 2 */

Static unsigned long state [P] ;\
Static int state_i ;
void init_gfsr5 (unsigned long s)
{
   int i, j, k ;
   static unsigned long x [P] ;
   s &= 0xffffffffUL ;
   for (i=0 ; i<P ; i++) {
     x [i] = s>>31 ;
     s = 1664525UL * s + 1UL ;
     s &= 0xffffffffUL ;
   }
   for (k=0, i=0 ; i<P ; i++) {
   state [i] = 0UL ;
   for (j=0 ; j<W ; j++) {
     state [i] <<= 1 ;
     state [i] |= x [k] ;
     x [k] ^=x [ (k+Q1) %P] ^x [ (k+Q2) %P] ^x [ (k+Q3) %P] ;
     k++;
     if (k==P) k = 0 ;
   }
}

state_i = 0 ;
}
unsigned long gfsr5 (void)
{
```

```
 int i ;

unsigned long *p0, *p1, *p2, *p3 ;

    if (state_i >= P) {
    state_i = 0 ;
    p0 = state ;
    p1 = state + Q1 ;
    p2 = state + Q2 ;
    p3 = state + Q3 ;

   for (i=0 ; i<(P-Q3) ; i++)
     *p0++ ^= *p1++ ^ *p2++ ^ *p3++;
   p3 = state ;
   for ( ; i<(P-Q2) ; i++)
     *p0++ ^= *p1++ ^ *p2++ ^*p3++;
   p2 = state;

   for ( ; i<(P-Q1) ; i++)
     *p0++ ^= *p1++ ^ *p2++ ^*p3++;
   p1 = state;
   for ( ; i<P ; i++)
     *p0++ ^= *p1++ ^*p2++ ^*p3++;
   }

return state [state_i++] ;
}

/* (W-1)-битовое целое */
long  gfsr5_31 (void)
{
return  (long)  (gfsr5( ) >>1);
}
```

Примечание – Соответствующий текст программы пятипараметрического GFSR на языке BASIC приведен для информации.

```
REM
/************************************************************************************/
Текст программы пятипараметрического GFSR на языке BASIC
/************************************************************************************/
OPTION BASE 0
REM
/************************************************************************************/
DECLARE NUMERIC P
LET P = 521                                          !#define P 512
REM /* Q1 < Q2 < Q3 */
DECLARE NUMERIC Q1
```

```
LET Q1 = 86                                   !#define Q1 86
DECLARE NUMERIC Q2
LET Q2 = 197                                  !#define Q2 197
DECLARE NUMERIC Q3
LET Q3 = 447                                  !#define Q3 447
DECLARE NUMERIC W
                                              /* W должно быть степенью 2 */
LET W = 32                                    !#define W 32

DIM state(P)                                  !static unsigned long state[P];
DECLARE NUMERIC state_i                       !static int state_i;

REM
/*******************************************************************************/
FUNCTION init_gfsr5(s)                        !void init_gfsr5(unsigned long s) {
    DECLARE NUMERIC i, j, k                   !     int i, j, k;
    DIM x(P)                                  !     static unsigned long x[P];

    LET s = And32(s , MskF_f)                 !      s &= 0xffffffffUL;
    FOR i=0 TO P-1                            !     for (i=0; i<P; i++) {
        LET x(i) = SR32U(s , 31)              !         x[i] = s>>31;
        LET s = Mul32U(1664525 , s) + 1       !         s = 1664525UL * s + 1UL;
        LET s = And32(s , MskF_f)             !         s &= 0xffffffffUL;
    NEXT I                                    !     }
LET k=0
FOR i=0 TO P-1                                !     for (k=0,i=0; i<P; i++) {
    LET state(i) = 0                          !       state[i] = 0UL;
    FOR j=0 TO W-1                            !       for (j=0; j<W; j++) {
      LET state(i) = SL32U(state(i) , 1)      !       state[i] <<= 1;
      LET state(i) = Or32(state(i) , x(k))    !       state[i] |= x[k];
      LET x(k) = Xor32(Xor32(Xor32(x(k) ,     !     x[k] ^= x[(k+Q1)%P] ^ x[(k+Q2)%P] ^
      x(REMAINDER(k + Q1 , P))) ,                   x[(k+Q3)%P];
      x(REMAINDER(k + Q2 , P))) ,
      x(REMAINDER(k + Q3 , P)))

      LET k = k + 1                           !     k++;
      IF k = P THEN LET K = 0                 !     if (k==P) k = 0;
    NEXT j                                    !     }
  NEXT I                                      ! }

  LET state_i = 0                             ! state_i = 0;
END FUNCTION                                  !}

      REM
/*******************************************************************************/
FUNCTION gfsr5                                !unsigned long gfsr5(void)    {
  DECLARE NUMERIC I                           !     int i;
```

```
DECLARE NUMERIC p0, p1, p2, p3                 !    unsigned long *p0, *p1, *p2, *p3;

IF state_i >= P THEN                           !    if (state_i >= P) {
  LET state_i = 0                              !    state_i = 0;
  LET p0 = 0                                   !    p0 = state;
  LET p1 = Q1                                  !    p1 = state + Q1;
  LET p2 = Q2                                  !    p2 = state + Q2;
  LET p3 = Q3                                  !    p3 = state + Q3;
FOR i=0 TO P-Q3-1                              !  FOR (i=0; i<(P-Q3); i++)

  LET state(p0) = Xor32(Xor32(Xor32(state(p0) ,
  state(p1)) , state(p2)) , state(p3))
  LET p0 = p0 + 1
  LET p1 = p1 + 1
  LET p2 = p2 + 1
  LET p3 = p3 + 1                              !    *p0++ ^= *p1++ ^ *p2++ ^*p3++;
NEXT i
LET p3 = 0                                     !    p3 = state;
FOR i=i TO P-Q2-1                              !    for (; i<(P-Q2); i++)

  LET state(p0) = Xor32(Xor32(Xor32(state(p0) ,
  state(p1)) , state(p2)) , state(p3))
  LET p0 = p0 + 1
  LET p1 = p1 + 1
  LET p2 = p2 + 1
  LET p3 = p3 + 1                              !    *p0++ ^= *p1++ ^ *p2++ ^*p3++;
NEXT i
LET p2 = 0                                     !    p2 = state;
FOR i=i TO P-Q1-1                              !    for (; i<(P-Q1); i++)

   LET state(p0) = Xor32(Xor32(Xor32(state(p0) ,
   state(p1)) , state(p2)) , state(p3))
   LET p0 = p0 + 1
   LET p1 = p1 + 1
   LET p2 = p2 + 1
   LET p3 = p3 + 1                             !    *p0++ ^= *p1++ ^ *p2++ ^*p3++;
NEXT i
LET p1 = 0                                     !    p1 = state;
FOR i=i TO P-1                                 !    for (; i<P; i++)

    LET state(p0) = Xor32(Xor32(Xor32(state(p0) ,
    state(p1)) , state(p2)) , state(p3))
    LET p0 = p0 + 1
    LET p1 = p1 + 1
    LET p2 = p2 + 1
    LET p3 = p3 + 1                            !  *p0++ ^= *p1++ ^ *p2++ ^*p3++;
   NEXT i
END IF                                         !  }
```

```
   LET gfsr5 = state(state_i)
   LET state_i = state_i + 1                     !     return state[state_i++];
END FUNCTION                                     !}
REM
/*****************************************************************************/
REM /* (W-1)-битовое целое */
FUNCTION gfsr5_31                                !long gfsr5_31(void) {
     LET gfsr5_31 = SR32U(gfsr5 , 1)             !       return (long)(gfsr5()>>1);
END FUNCTION                                     !}
```

B.3 Текст программы комбинированного метода Таусворта

Ниже приведен текст программы комбинированного метода Таусворта на языке Си, генерирующего целые числа из интервала от 0 до ($2^{31} - 1$) включительно, на основе комбинации трех последовательностей параметров Таусворта (31, 13, 12), (29, 2, 4) и (28, 3, 17).

Инициализация init_taus88 (s) происходит при условии, что начальное число s является 32-битовым целым числом без знака [целое число из интервала от 0 до ($2^{32} - 1$) включительно]. Для получения другой последовательности псевдослучайных чисел необходимо изменить начальное число s. Перед обращением к taus88_31 ( ) необходимо один раз выполнить инициализацию init_taus88 (s). Инициализация может быть выполнена без использования функции init_taus88 (s) с помощью выбора подходящих значений s_1, s_2 и s_3. Однако для инициализации должны быть выполнены следующие три условия:

– хотя бы один разряд из высших 31 разряда s_1 равен единице;

– хотя бы один разряд из высших 29 разрядов s_2 равен единице;

– хотя бы один разряд из высших 28 разрядов s_3 равен единице.

Поскольку самый низший разряд s_1, три низших разряда s_2 и четыре низших разряда s_3 проигнорированы, полученная последовательность случайных чисел не зависит от изменений в этих разрядах. В данной программе предполагается, что длина «беззнакового длинного целого числа» составляет 32 бита (разряда).

```
/*****************************************************************************/
Текст программы комбинированного метода Таусворта на языке Си
/*****************************************************************************/

static unsigned long s1, s2, s3, b ;

void init_taus88 (unsigned long s)
{
   int i ;
```

```
   unsigned long x [3] ;

   i=0 ;
   while (i<3) {
    if (s & 0xfffffff0UL) {
    x [i] = s ;
    i++;
   }
   s = 1664525UL * s + 1UL;
}

s1 = x [0] ; s2 = x [1] ; s3 = x [2] ;
}
/* 31-битовое целое */

long taus88_31 (void)
{
  b = (((s1 << 13) ^ s1) >> 19) ;
  s1 = (((s1 & 4294967294UL) << 12) ^b) ;
  b = (((s2 << 2) ^ s2) >> 25);
  s2 = (((s2 & 4294967288UL) << 4) ^b) ;
  b = (((s3 << 3) ^ s3) >> 11) ;
  s3 = (((s3 & 4294967280UL) <<17) ^b) ;

  return (long) ((s1 ^ s2 ^ s3) >>1) ;
}
```

В этой программе операторы

```
  b = (((s1 << 13) ^ s1) >> 19),
  s1 = (((s1 & 4294967294UL) << 12) ^b)
```

генерируют числа в последовательности Таусворта с параметрами (31, 13, 12) в s_1, а операторы

```
  b = (((s2 << 2) ^ s2) >> 25),
  s2 = (((s2 & 4294967288UL) << 4) ^b)
```

и

```
  b = (((s3 << 3) ^s3) >> 11);
  s3 = (((s3 & 4294967280UL) << 17) ^b)
```

генерируют числа последовательности Таусворта с параметрами (29, 2, 4) и (28, 3, 17), s_2 и s_3 соответственно. Три двоичных целых числа (p, q, t) объединяют с помощью побитовой операции «исключающее ИЛИ» и 31-битовая последовательность псевдослучайного числа получена.

Выбор трех параметров (p, q, t) позволяет получить лучшее многомерное равномерное распределение последовательности псевдослучайных чисел после операции объединения. Значения параметров необходимо сохранять. Чтобы получить другую последовательность псевдослучайных чисел, необходимо изменить начальное число.

Если необходимо получить независимый набор случайных чисел для каждого повторения моделирования, функцию инициализации init_taus88 (s) необходимо вызвать один раз в начале моделирования. После каждого повторения значения s_1, s_2 и s_3 необходимо сохранять и присваивать переменным s_1, s_2 и s_3 соответственно, как начальные значения при следующем повторении.

Примечание – Соответствующий текст программы метода Таусворта на языке BASIC приведен для информации.

```
REM ***************************************************************************/
REM Текст программы для метода Таусворта на языке BASIC

REM ***************************************************************************/

OPTION BASE 0

REM
/*****************************************************************************/
FUNCTION init_taus88(s)                       !void init_taus88(unsigned long s) {
   DECLARE NUMERIC I                          !     int i;
   DIM x(3)                                   !     unsigned long x[3];
   FOR i = 0 TO 2                             !     i=0; while (i<3) {
      IF And32(s , MskF_0) <> 0 THEN          !        if (s & 0xfffffff0UL)  {
         LET x(i) = s                         !        x[i] = s; i++;
      END IF                                  !     }
      LET s = Mul32U(1664525, s) + 1          !        s = 1664525UL * s +1UL;
  NEXT I                                      !     }
  LET s1 = x(0)
  LET s2 = x(1)
  LET s3 = x(2)                               !     s1 = x[0]; s2 = x[1]; s3 = x[2];
END FUNCTION                                  !}
REM
/*****************************************************************************/
FUNCTION taus88_31                            !long taus88_int(void)
                                              {
  REM /***** 31-битовое целое *****/
  LET b = SR32U(Xor32(SL32U(s1, 13), s1), 19  !     b = (((s1 << 13) ^ s1) >> 19);
  LET s1 = Xor32(SL32U(And32(s1 , MskF_e),    !     s1 = (((s1 &4294967294) << 12) ^ b);
  12), b)
  LET b = SR32U(Xor32(SL32U(s2, 2), s2), 25)  !     b = (((s2 << 2) ^ s2) >> 25);
  LET s2 = Xor32(SL32U(And32(s2 , MskF_8),    !     s2 = (((s2 &4294967288) << 4) ^ b);
  4), b)
```

```
   LET b = SR32U(Xor32(SL32U(s3, 3), s3), 11            !     b = (((s3 << 3) ^ s3) >> 11);
   LET s3 = Xor32(SL32U(And32(s3 , MskF_0),             !     s3 = (((s3 &4294967280) << 17) ^ b);
   17), b)
   LET taus88_31 = SR32U(Xor32(Xor32(s1, s2),           !     return (long)((s1 ^ s2 ^ s3) >> 1);
   s3), 1)
END FUNCTION                                            ! }
```

В.4 Текст программы для метода Мерсенна Твистера

Ниже приведен текст программы для метода Мерсенна Твистера на языке Си. Функция genrand () генерирует псевдослучайные числа в виде целого 32-битового числа без знака из интервала от 0 до ($2^{32} - 1$) включительно. Функция genrand_31 () генерирует псевдослучайные числа в виде целого 31-битового числа без знака из интервала от 0 до ($2^{31} - 1$) включительно. Функция init_genrand (s) инициализирует начальное число в виде 32-битового целого числа без знака [целое число из интервала от 0 до ($2^{32} - 1$) включительно]. Перед обращением к genrand () или genrand_31 () необходимо один раз выполнить инициализацию init_genrand (s). Различные начальные числа *s* приводят к генерации разных последовательностей случайных чисел. Параметры в этой программе необходимо сохранять.

Если необходимо получить независимый набор случайных чисел в каждом из многократных повторений при моделировании, к функции инициализации init_genrand (*s*) следует обращаться только один раз перед началом моделирования. После каждого повторения содержимое таблицы mt[*P*] размера *P* и значение переменной mti должны быть сохранены и использованы как начальные значения при следующем повторении.

Пример – В данном примере использовано p = 624 слова с параметрами (624, 397, 31, 32, 0x9908b0df, 11, 7, 15, 18, 0x9d2c5680, 0xefc60000). Здесь числа из 10 знаков, начинающиеся с 0x, являются 32-битовыми целыми без знака, представленными в шестнадцатеричном коде. Период равен ($2^{19937} - 1$), а числа распределены равномерно в 623 размерном гиперкубе с 32-битовой точностью. Кроме того последовательность соответствует равномерному распределению в 3 115-мерном пространстве с 6-битовой точностью.

В данной программе длина «беззнакового длинного целого» составляет не менее 32 бит.

```
*************************************************
Текст программы метода Мерсенна Твистера на языке Си
*************************************************/
/* Параметры периода*/

 #define P 624
 #define Q 397
 #define MATRIX_A 0x9908b0dfUL                 /* постоянный вектор а */
 #define UPPER_MASK 0x80000000UL               /* наиболее значимые (w-r) бит */
 #define LOWER_MASK 0x7fffffffUL               /* последние значимые r бит */
```

```
static unsigned long mt [P] ;                    /* массив состояния вектора*/
static int mti=P+1 ;                             /* mti==P+1 означает, что mt [P] не
                                                 инициализирован */
/* инициализация mt [P] с начальным значением a */
void init_genrand (unsigned long s)
 {
    mt [0] = s & 0xffffffffUL ;
    for (mti=1 ; mti<P ; mti++)    {
    mt [mti] = (1664525UL * mt [mti-1] + 1UL) ;
    mt [mti] &= 0xffffffffUL ;
 }
}
/* генерация случайного числа из интервала[0,   0xffffffff] */
unsigned long genrand (void)
{
unsigned long y ;
static  unsigned  long  mag01 [2] = {0x0UL, MATRIX_A} ;
                                                 /* mag01 [x] = x * MATRIX_A для x=0, i */

if (mti >=P) {                                   /* генерация P слов одновременно */
  int kk ;

if (mti == P+1)                                  /* если init _genrand ( ) не вызвано */
  init_genrand (5489UL) ;                        /* использовано не подходящее начальное
                                                 значение*/

for (kk=0 ; kk<P-Q ; kk++) {
y = (mt [kk] &UPPER_MASK) | (mt [kk+1] &LOWER_MASK) ;
mt [kk] = mt [kk+Q] ^ (y >> 1) ^ mag01 [y & 0x1UL] ;
}
for ( ; kk<P-1 ; kk++) {

y = (mt [kk] &UPPER_MASK) | (mt [kk+1] &LOWER_MASK) ;
mt [kk] = mt [kk+ (Q-P) ] ^ (y >> 1) ^ mag01 [y & 0x1UL] ;
}

y = (mt [P-1] &UPPER_MASK) | (mt [0] &LOWER_MASK) ;
mt [P-1] = mt [Q-1] ^ (y >> 1) ^ mag01 [y & 0x1UL] ;

mti = 0 ;
}

y = mt [mti++] ;
                                                    /* Темперирование */

y ^= (y >> 11) ;
y ^= (y << 7) & 0x9d2c5680UL ;
```

```
y ^= (y << 15) & 0xefc60000UL ;
y ^= (y >> 18) ;

return y ;
}

long genrand_31 (void)                        /* генерация случайного числа из
                                              интервала [0, 0x7fffffff] */
{
return (long) (genrand( ) >>1) ;
}
```

Примечание – Соответствующий текст программы метода Мерсенна Твистера на языке BASIC приведен для информации

```
REM /****************************************************/
REM Текст программы для метода Мерсенна Твистера на языке BASIC

OPTION BASE 0

REM
/*****************************************************************************/
REM /* Параметры периода */
DECLARE NUMERIC P
LET P = 624                                   !#define P 624
DECLARE NUMERIC Q
LET Q = 397                                   !#define Q 397
DECLARE NUMERIC MATRIX_A
LET MATRIX_A = BVAL("9908b0df" , 16)          !#define MATRIX_A 0x9908b0dfUL
                                              /*   постоянный вектор a     */

DECLARE NUMERIC UPPER_MASK
LET UPPER_MASK = BVAL("80000000" , 16)        !#define UPPER_MASK 0x80000000UL
                                              /*   наиболее значимые w-r бит     */

DECLARE NUMERIC LOWER_MASK
LET LOWER_MASK = BVAL("7fffffff" , 16)        !#define LOWER_MASK 0x7fffffffUL
                                              /*   последние значимые r бит     */

DIM mt(P)                                     !static unsigned long mt[P];
                                              /*    массив состояния вектора      */
DECLARE NUMERIC mti
LET mti = P + 1                               !static int mti=P+1;
                                              /* mti==P+1 означает, что mt[P] не
                                              инициализирован */
REM
/*****************************************************************************/
```

```
REM /* инициализация mt[P] с начальным значением */
FUNCTION init_genrand(s)                          !void init_genrand(unsigned long s) {
   LET mt(0) = And32(s , MskF_f)                  !     mt[0]= s & 0xffffffffUL;
   FOR mti = 1 TO P - 1                           !     for (mti=1; mti<P; mti++) {

    LET mt(mti) = Mul32U(1664525 , mt(mti-1)) + 1
                                                  ! mt[mti] = (1664525UL * mt[mti-1] + 1UL);
    LET mt(mti) = And32(mt(mti) , MskF_f)         ! mt[mti] &= 0xffffffffUL;
NEXT mti                                          !    }
END FUNCTION                                      !}

REM
/**************************************************************************/
REM /* генерация случайного числа из интервала [0,0xffffffff] */
FUNCTION genrand                                  ! unsigned long genrand(void) {
DECLARE NUMERIC y                                 ! unsigned long y;
DIM mag01(2)
LET mag01(0) = 0
LET mag01(1) = MATRIX_A                           !static unsigned long mag01[2]={0x0UL,
                                                   MATRIX_A};
                                                  /* mag01[x] = x * MATRIX_A for x=0,1 */
  IF mti >= P THEN                                ! if (mti >= P) {
                                                  /* генерация P слов одновременно */
  DECLARE NUMERIC kk                              ! int kk;
  IF mti = P + 1 THEN                             ! if (mti == P+1)
                                                  /* если init_genrand() не вызван */
  LET y = init_genrand(5489)                      ! init_genrand(5489UL);
/* использована ошибочная инициализация S*/
  END IF
  FOR kk=0 TO P-Q-1                               !   for (kk=0;kk<P-Q;kk++) {

 LET y = Xor32(And32(mt(kk) , UPPER_MASK) , !  y = (mt [kk] &UPPER_MASK) | (mt
 And32(mt(kk+1) , LOWER_MASK))                        [kk+1] &LOWER_MASK);

 LET mt(kk) = Xor32(Xor32(mt(kk+Q) , SR32U(y , 1)) , !  mt[kk] = mt[kk+Q] ^ (y>> 1) ^ mag01[y
 mag01(And32(y , 1)))                                 & 0x1UL];
 NEXT kk                                          ! }

 FOR kk=kk TO P-2                                 !      for (;kk<P-1;kk++) {
 LET y = Xor32(And32(mt(kk) , UPPER_MASK) ,       ! y = (mt [kk] & UPPER_MASK) | (mt[kk+1]
 And32(mt(kk+1) , LOWER_MASK))                     &LOWER_MASK);
 LET    mt(kk)    =    Xor32(Xor32(mt(kk+Q-P)    , ! mt[kk] = mt[kk+(Q-P)] ^(y >> 1) ^
 SR32U(y , 1)) , mag01(And32(y , 1)))              mag01[y & 0x1UL];
 NEXT kk                                          ! }
```

```
 LET y = Xor32(And32(mt(P-1) , UPPER_MASK) ,      ! y = (mt[P-1] & UPPER_MASK)|
 And32(mt(0) , LOWER_MASK))                          (mt[0]&LOWER_MASK);
 LET mt(P-1) = Xor32(Xor32(mt(Q-1),                ! mt[P-1] = mt[Q-1] ^ (y >> 1) ^ mag01
 SR32U(y ,1)) , mag01(And32(y , 1)))                 [y & 0x1UL];
 LET mti = 0                                       ! mti = 0;
END IF                                             ! }
LET y = mt(mti)
LET mti = mti + 1                                  ! y = mt[mti++];
REM /* Темперирование */
   LET y = Xor32(y , SR32U(y , 11))                ! y ^= (y >> 11);
   LET y = Xor32(y , And32(SL32U(y , 7) ,          ! y ^= (y << 7) & 0x9d2c5680UL;
   BVAL("9d2c5680" , 16)))
   LET y = Xor32(y , And32(SL32U(y ,15) ,          ! y ^= (y << 15) & 0xefc60000UL;
   BVAL("efc60000" , 16)))
   LET y = Xor32(y , SR32U(y , 18))                ! y ^= (y >> 18);
LET genrand = y                                    ! return y;
END FUNCTION                                       !}
REM
/*******************************************************************************/
REM /* генерация случайного числа из интервала [0,0x7fffffff] */
FUNCTION genrand_31                                !long genrand_31(void) {
   LET genrand_31 = SR32U(genrand , 1)             !      return (long)(genrand()>>1);
END FUNCTION                                       !}
```

В.5 Линейный конгруэнтный метод

В.5.1 Общие положения

В.5.1.1 Применение

Линейные конгруэнтные методы широко применяют в программном обеспечении, так как они сочетают в себе экономное использование памяти компьютера с высокой скоростью исполнения. Однако эти методы имеют относительно короткий период и, следовательно, не всегда обеспечивают достаточную случайность генерируемых чисел, особенно при необходимости получения случайных многомерных последовательностей.

В.5.1.2 Определение

Большая часть линейных конгруэнтных методов генерирует последовательности псевдослучайных чисел X_1, X_2, ..., используя следующее рекуррентное соотношение

$$X_n = \mathrm{mod}\,(aX_{n-1} + c;\ m)\ n = 1, 2, \ldots ,$$

где a и m – положительные целые числа, c – неотрицательное целое число.

Линейный конгруэнтный метод определен, как только заданы значения параметров a, m и c. Кроме того, если задано начальное число X_0, полученная последовательность чисел однозначно определена.

Примечание 1 – Алгоритм рекуррентных соотношений состоит в следующем. Вычисляют$(aX_0 + c)$, используя начальное число X_0, делят результат на m и определяют остаток от деления X_1. Затем вычисляют $(aX_1 + c)$, делят результат на m и определяют остаток от деления X_2. Эту процедуру повторяют столько раз, сколько необходимо.

Примечание 2 – Значение n, для которого $X_n = X_0$ в первый раз называют периодом последовательности.

В.5.1.3 Метод выбора значений параметров

Значения a, m и c не могут быть определены произвольно. Они должны быть выбраны следующим образом.

Поскольку m является верхним пределом периода последовательности чисел, полученной линейным конгруэнтным методом, значение m должно быть установлено как можно больше. Следовательно, при использовании, например 32-битовых компьютеров, рекомендуется устанавливать $m = 2^{32}$ или $m = 2^{31} - 1$.

Для выбора c нет строгого критерия. Однако период полученной последовательности зависит от того равно c нулю или положительному целому числу.

Для множителя a должно быть установлено значение, которое обеспечивает хорошие результаты в комбинации с выбранными значениями m и c (см. таблицу В.1).

Примечание – В случае, когда m равно 2 в целой степени и c равно 0, период не превышает $m/4$. Если c – нечетное число, период равен m.

В.5.1.4 Пример параметров

Для 32-битовых компьютеров следует использовать один из наборов параметров, приведенных в таблице В.1.

Таблица B.1 – Наборы параметров для использования линейного конгруэнтного метода

Номер строки	A	c	M
1	1 664 525	*	2^{32}
2	1 566 083 941	0	2^{32}
3	48 828 125	0	2^{32}
4	2 100 005 341	0	$2^{31} - 1$
5	397 204 094	0	$2^{31} - 1$
6	314 159 369	0	$2^{31} - 1$

Примечание 1 – Символ * указывает, что может быть использовано любое нечетное число.

Примечание 2 – Использование параметров, приведенных в первой строке, обеспечивает генерацию целых чисел из интервала от 0 до $(2^{32} - 1)$.

Примечание 3 – Использование параметров, приведенных во второй или третьей строках, обеспечивает генерацию чисел вида $(4i + 1)$ для $i = 0, 1, ..., (2^{30} - 1)$, или $(4i + 3)$ для $i = 0, 1, ..., (2^{30} - 1)$ в зависимости от начального числа X_0.

Примечание 4 – Использование параметров, приведенных в строках 4, 5 и 6, обеспечивает генерацию положительных целых чисел из интервала от 1 до $(2^{31} - 2)$.

Примечание 5 – Следует использовать более высокие разряды генерированных случайных чисел, а низкие разряды не следует использовать.

B.5.2 Текст программы для линейного конгруэнтного метода

B.5.2.1 Общие положения

Ниже приведен текст программы для линейного конгруэнтного метода на языке Си. Программа соответствует ИСО/МЭК 9899. Приведено два варианта программы, один – для случая, когда $m = 2^{32}$, другой - для случая, когда $m = (2^{31} - 1)$. Эти варианты соответствуют рекомендациям B.5.1.

B.5.2.2 Вариант $m = 2^{32}$

При каждом обращении функция lcong32 () формирует случайное целое число от нуля до $(2^{32} - 1)$ включительно. Результат имеет тип «длинного целого числа без знака». При каждом обращении функция lcong32_31() формирует случайное целое число от нуля до $(2^{31} - 1)$ включительно. Результат имеет тип «длинного целого числа». Функция инициализации init_lcong32 (использующая длинное начальное целое без знака) выполняет инициализацию так, чтобы неотрицательное целое число типа «длинное целое без знака», было установлено как начальное число. Если второе слагаемое $c = 0$ и исходное начальное число X_0^* является нечетным, то X_0 может быть установлено как $X_0 = X_0^*$. Однако, если задано четное исходное начальное число X_0*, в случае $c = 0$ для получения начального числа X_0 к исходному начальному числу прибавляют единицу $X_0 = X_0^* + 1$. В других случаях используют в качестве X_0 начальное число mod(X_0*; m).

Множитель и второе слагаемое изменяются при изменении значений MULTIPLIER и INCREMENT в каждой программе. Последовательность случайных чисел перезапускается при использовании выхода lcong32 () в качестве аргумента функции инициализации init_lcong32 (начальное число).

В.5.2.3 Вариант $m = 2^{31} - 1$

Каждый раз при обращении к функции lcong31 () она формирует случайное целое число из интервала от 1 до (2^{31} – 2) включительно. Результат имеет тип «длинного целого числа». Функция инициализации init_lcong31 (использующая длинное целое начальное число без знака) устанавливает в качестве начального числа неотрицательное целое число. Как и в таблице В.1, второе слагаемое c всегда равно 0. Поэтому начальное число X_0 не должно быть равно 0. Однако, если $X_0^* = 0$, в качестве X_0 используют специальное число (19 660 809). В других случаях в качестве X_0 используют mod(X_0*; m).

Для изменений значений множителя необходимо его указать в операторе присвоения константе MULTIPLIER в тексте программы.

```
*********************************************************/
Текст программы линейного конгруэнтного метода на языке Си
*********************************************************/

*********************************************************/
Часть 1. Modulus = 2^32
*********************************************************/


#define MULTIPLIER 1664525UL
#define INCREMENT 1UL

Static  unsigned  long  state32 ;
Unsigned  long  lcong32( void )
{
   state32 = ( state32 * MULTIPLIER + INCREMENT ) & 0xFFFFFFFFUL;
   return state32;
}

long lcong32_31( void )
{
   state32 = ( state32 * MULTIPLIER + INCREMENT ) & 0xFFFFFFFFUL;
   return state32>>1;
}
void init_lcong32(unsigned long s)
{
/* начальное число должно быть нечетным, если слагаемое INCREMENT равно 0 */
     if ( (INCREMENT==0) && (s%2 == 0) ){
      s++ ;
     }
     state32 = s ;
}
```

```
/*********************************************************
Часть 2. Modulus = 2^31-1 = 2147483647
*********************************************************/

#undef MULTIPLIER
#undef INCREMENT
#undef NBIT
#define NBIT 15
#define MASK ( (1<<NBIT)-1)
#define MASK2 ( (1<<2*NBIT) -1)
#define MULTIPLIER 2100005341UL
#define MULTIPLIER_LO (MULTIPLIER & MASK)
#define MULTIPLIER_HI (MULTIPLIER >> NBIT)
Static unsigned long state31 ;

long lcong31 ( void )
{
   unsigned long xlo, xhi ;
   unsigned long z0, z1, z2 ;

   xlo = state31 & MASK ;
   xhi = state31 >> NBIT ;
   z0 = xlo * MULTIPLIER_LO ;                      /* 15bit * 15bit => 30bit */
   z1 = xlo * MULTIPLIER_HI+ xhi * MULTIPLIER_LO ;
                                                   /* 15bit * 16bit * 2 => 32bit */
   z2 = xhi * MULTIPLIER_HI ;                      /* 16bit * 16bit => 32bit */
   z0 += (z1 & MASK) << NBIT ;
   z2 += (z1 >> NBIT) + (z0 >> (2*NBIT)) ;
   z0 = (z0 & MASK2) | ((z2&1) << (2*NBIT)) ;
   z2 >>=1 ;
   state31 = z0 + z2 ;
   if (state31>=0x7fffffffUL) state31 -= 0x7fffffffUL ;
   /* Это число не должно превышать 2*0x7fffffffUL */
   return (long) state31 ;
}
void init_lcong31 (unsigned long s)
{
if ( s == 0UL ) s=19660809UL ;                     /* начальное число не должно быть 0 */
s = s % 0x7fffffffUL ;
state31 = s ;
}
```

Примечание 1 – Текст программы для линейного конгруэнтного метода на языке Basic приведен для информации.

```
REM /*******************************************************
REM Программные коды линейного конгруэнтного метода на языке BASIC
REM *******************************************************/
REM
REM /*******************************************************
REM Часть 1. Modulus = 2^32
REM *******************************************************/
OPTION BASE 0
REM
/****************************************************************************/
DECLARE NUMERIC MULTIPLIER
LET MULTIPLIER = 1664525                          !#define MULTIPLIER 1664525UL
DECLARE NUMERIC INCREMENT
LET INCREMENT = 1                                 !#define INCREMENT 1UL

DECLARE NUMERIC state32                           !static unsigned long state32;
REM
/****************************************************************************/
FUNCTION lcong32                                  !unsigned long lcong32u( void )     {
   LET state32 = And32((state32 * MULTIPLIER) +   ! state32 = ( state32 * MULTIPLIER +
   INCREMENT , MskF_f)                              INCREMENT ) & 0xFFFFFFFFUL;
   LET lcong32 = state32                          !      return state32;
END FUNCTION                                      !}
REM
/****************************************************************************/
FUNCTION lcong32_31                               !long lcong32( void )     {
                                                  !  state32 = ( state32 * MULTIPLIER
                                                     +INCREMENT ) & 0xFFFFFFFFUL;
   LET lcong32_31 = SR32U(lcong32 , 1)            ! return state32>>1;
END FUNCTION                                      !}
REM
/****************************************************************************/
FUNCTION init_lcong32(s)                          !void init_lcong32(unsigned long s) {
REM /* начальное число  должно быть нечетным, если слагаемое INCREMENT равно 0 */
 IF (INCREMENT = 0) AND (REMAINDER(s , 2) = 0) !    if ( (INCREMENT==0) && (s%2 == 0) ) {
THEN
 LET s = s + 1                                    !   s++;
   END IF                                         !   }
   LET state32 = s                                !   state32 = s;
END FUNCTION                                      !}
```

```
REM /*********************************************************
REM Текст программы линейного конгруэнтного метода на языке BASIC
REM *********************************************************/

REM
REM /*********************************************************
REM Часть 2. Modulus = 2^31-1 = 2147483647
REM *********************************************************/
OPTION BASE 0

DECLARE NUMERIC NBIT
LET NBIT = 15                                  !#define NBIT 15
DECLARE NUMERIC MASK
LET MASK = SL32U(1 , NBIT) - 1                 ! #define MASK ((1<<NBIT)-1)
DECLARE NUMERIC MASK2
LET MASK2 = SL32U(1 , 2*NBIT) - 1              ! #define MASK2 ((1<<(2*NBIT))-1)

DECLARE NUMERIC MULTIPLIER
LET MULTIPLIER = 2100005341                    ! #define MULTIPLIER 2100005341UL

DECLARE NUMERIC MULTIPLIER_LO
LET MULTIPLIER_LO = And32 (MULTIPLIER,         ! #define MULTIPLIER_LO (MULTIPLIER&
MASK)                                            MASK )
DECLARE NUMERIC MULTIPLIER_HI
LET   MULTIPLIER_HI   =   SR32U(MULTIPLIER   , ! #define MULTIPLIER_HI (MULTIPLIER>>
NBIT)                                            NBIT )

DECLARE NUMERIC state31                        !static unsigned long state31;
REM /*******************************************************************************/
FUNCTION lcong31                               !long lcong31( void ) {
   DECLARE NUMERIC xlo, xhi                    ! unsigned long xlo, xhi;
   DECLARE NUMERIC z0, z1, z2                  ! unsigned long z0, z1, z2;
   LET xlo = And32(state31 , MASK)             ! xlo = state31 & MASK;     //1st val:9
   LET xhi = SR32U(state31 , NBIT)             ! xhi = state31 >> NBIT;     //1st val:600
   LET z0 = xlo * MULTIPLIER_LO                ! z0 = xlo * MULTIPLIER_LO;
                                               /*15bit * 15bit => 30bit */
   LET z1 = xlo * MULTIPLIER_HI                ! z1 = xlo * MULTIPLIER_HI + xhi *
                                                       MULTIPLIER_LO;

   LET z1 = z1 + xhi * MULTIPLIER_LO           ! /*15bit * 16bit * 2 => 32bit */
   LET z2 = xhi * MULTIPLIER_HI                ! z2 = xhi * MULTIPLIER_HI;
                                               /* 16bit* 16bit => 32bit */
```

```
   LET z0 = z0 + SL32U(And32(z1 , MASK) , NBIT)   ! z0 += (z1 & MASK) << NBIT;
                                                  //1st val:897833157
   LET z2 = z2 + SR32U(z1 , NBIT) + SR32U(z0, 2 * ! z2 += (z1 >> NBIT) + (z0 >>(2*NBIT));
   NBIT)
   LET   z0   =   Or32(And32(z0   ,   MASK2)   ,  ! z0 = (z0 & MASK2) | ( (z2&1)
   SL32U(And32(z2 , 1) , 2 * NBIT))                     <<(2*NBIT));
   LET z2 = SR32U(Z2 , 1)                         ! z2 >>=1;


   LET state31 = z0 + z2                          ! state31 = z0 + z2;
REM /* Это число не должно быть более 2*0x7fffffffUL */
   IF  state31  >=  2147483647  THEN  LET  state31  = ! IF (state31>=0x7fffffffUL) state31-=
   state31 - 2147483647                                 0x7fffffffUL;
   LET lcong31 = state31                          ! return (long) state31;
END FUNCTION                                      !}
REM
/********************************************************************************/
FUNCTION init_lcong31(s)                          !void init_lcong31(unsigned long s) {
REM ! /* начальное число не должно быть 0 */
   IF s = 0 THEN LET s = 19660809                 !  if ( s == 0UL ) s=19660809UL;
   LET s = REMAINDER(s , 2147483647)              !  s = s % 0x7fffffffUL;
   LET state31 = s                                !  state31 = s;
END FUNCTION                                      !}
```

Примечание 2 – Приведенный ниже текст программы на языке BASIC необходим для преобразования программных кодов языка Си в программные коды языка BASIC, преобразования случайных чисел в соответствии с приложением А и выполнения побитовых операций на языке BASIC.

```
REM /*********************************************
REM Программные коды функций на языке BASIC.
REM *********************************************/

OPTION BASE 0

REM /*****************************************************/

DECLARE NUMERIC MskF_f
DECLARE NUMERIC MskF_e
DECLARE NUMERIC MskF_8
DECLARE NUMERIC MskF_0

LET MskF_f = BVAL("ffffffff" , 16)
LET MskF_e = BVAL("fffffffe" , 16)
LET MskF_8 = BVAL("fffffff8" , 16)
LET MskF_0 = BVAL("fffffff0" , 16)
```

```
REM /*******************************************/
FUNCTION Or32(xA,xB)

DECLARE NUMERIC Ori, OrC

   LET OrC = 0
   FOR Ori=0 TO 31
     LET xA = xA / 2
     LET xB = xB / 2
     IF (INT(xA) <> xA) OR (INT(xB) <> xB) THEN
       LET OrC = OrC + 2 ^ Ori
     END IF
     LET xA = INT(xA)
     LET xB = INT(xB)
     IF (xA = 0) AND (xB = 0) THEN EXIT FOR
NEXT Ori
LET Or32 = OrC
END FUNCTION

REM /*******************************************/

FUNCTION And32(xA,xB)
DECLARE NUMERIC Andi, AC
   LET AC = 0
   IF xA > MskF_f THEN
      LET xA = xA - INT(xA / 4294967296) * 4294967296
   END IF
   IF xB > MskF_f THEN
      LET xB = xB - INT(xB / 4294967296) * 4294967296
   END IF

   IF (xA = 0) OR (xB = 0) THEN
      LET And32 = 0
   ELSEIF xB = MskF_f THEN                          !&Hffffffff
      LET And32 = xA
   ELSEIF xA = MskF_f THEN                          !&Hffffffff
      LET And32 = xB
   ELSEIF xB = MskF_8 THEN                          !&Hfffffff8
      LET And32 = INT(xA / 8) * 8
   ELSEIF xA = MskF_8 THEN                          !&Hfffffff8
      LET And32 = INT(xB / 8) * 8
```

```
    ELSEIF xB = MskF_0 THEN                                   !&Hfffffff0
        LET And32 = INT(xA / 16) * 16
    ELSEIF xA = MskF_0 THEN                                   !&Hfffffff0
        LET And32 = INT(xB / 16) * 16
    ELSE
       FOR Andi=0 TO 31
       LET xA = xA / 2
       LET xB = xB / 2
       IF (INT(xA) <> xA) AND (INT(xB) <> xB) THEN
           LET AC = AC + 2 ^ Andi
       END IF
       LET xA = INT(xA)
       LET xB = INT(xB)
       IF (xA = 0) OR (xB = 0) THEN EXIT FOR
    NEXT Andi
    LET And32 = AC
    END IF

END FUNCTION

REM /******************************************/
FUNCTION Xor32(xA,xB)
    DECLARE NUMERIC Xori, XC

    LET XC = 0
    FOR Xori=0 TO 31
       LET xA = xA / 2
       LET xB = xB / 2
       IF ((INT(xA) = xA) AND (INT(xB) <> xB)) OR (INT(xA) <> xA) AND (INT(xB) = xB) THEN
           LET XC = XC + 2 ^ Xori
       END IF
       LET xA = INT(xA)
       LET xB = INT(xB)
       IF (xA = 0) OR (xB = 0) THEN EXIT FOR
       NEXT Xori
       LET Xori = Xori + 1
       IF (xA =0) AND (xB = 0) THEN
       elseIF xA = 0 THEN
          FOR Xori=Xori TO 31
             LET xB = xB / 2
             IF INT(xB) <> xB THEN
```

```
                LET XC = XC + 2 ^ Xori
            END IF
            LET xB = INT(xB)
            IF xB = 0 THEN EXIT FOR
    NEXT Xori
    ELSE
        FOR Xori=Xori TO 31
            LET xA = xA / 2
            IF INT(xA) <> xA THEN
                LET XC = XC + 2 ^ Xori
            END IF
            LET xA = INT(xA)
            IF xA = 0 THEN EXIT FOR
        NEXT Xori
    END IF
    LET Xor32 = XC
END FUNCTION

REM /*******************************************/
FUNCTION SL32U(xA,xL)

   REM 2006-05-31
   DECLARE NUMERIC slAH,slAL

   IF (xA = 0) OR (xL = 0) THEN
   LET SL32U = xA
   ELSEIF xL >= 16 THEN
      LET slAL = xA - INT(xA / 65536) * 65536
      LET slAL = slAL * 2 ^ (xL - 16)
      LET slAL = slAL - INT(slAL / 65536) * 65536
      LET SL32U = slAL * 65536
   ELSE
      LET slAL = xA - INT(xA / 65536) * 65536
      LET slAH = INT(xA / 65536)
      LET slAL = slAL * 2 ^ xL
      LET slAH = slAH * 2 ^ xL
      LET slAH = slAH - INT(slAH / 65536) * 65536
      LET SL32U = slAL + slAH * 65536
   END IF
END FUNCTION
```

```
REM /******************************************/
FUNCTION SR32U(xA,xL)
   REM 2006-06-03
   IF xL = 0 THEN
      LET SR32U = xA
   ELSEIF xA = 0 THEN
      LET SR32U = 0
   ELSE
      LET SR32U = INT(xA / 2 ^ xL)
   END IF
END FUNCTION

REM /******************************************/
            FUNCTION Mul32U(xA,xB)

   REM /***** A,B : unsigned long (32-bit) *****/
   REM 2006-06-02
   DECLARE NUMERIC MAH, MAL, MBH, MBL

   LET MAH = INT(xA / 65536)
   LET MBH = INT(xB / 65536)
   IF (xA = 0) OR (xB = 0) THEN
      LET Mul32U = 0
   ELSEIF (MAH = 0) AND (MBH = 0) THEN
      LET Mul32U = xA * xB
   ELSE
      LET MAL = xA - INT(xA / 65536) * 65536
      LET MBL = xB - INT(xB / 65536) * 65536
      LET MBH = MAH * MBL + MAL * MBH + INT((MAL * MBL) / 65536)
      LET MBH = MBH - INT(MBH / 65536) * 65536
      LET MAL = MAL * MBL
      LET MBL = MAL - INT(MAL / 65536) * 65536
      LET Mul32U = MBL + 65536 * MBH
   END IF
END FUNCTION
```

В.6 Примеры

В таблице В.2 приведены примеры последовательностей случайных чисел, полученных с использованием программ, приведенных в приложении В с установленными значениями параметров (для проверки). Первые 5 псевдослучайных чисел и 5 псевдослучайных чисел с промежутками в 1000 приведены для сравнения.

Таблица В.2 – Примеры случайных чисел

Метод генерации	Линейный конгруэнтный метод		Трехпараметрический метод GFSR	Пятипараметрический метод GFSR	Комбинированный метод Таусворта	Метод Мерсенна Твистера	Физическое случайное число
Обращение к функции. Параметры	lcong32_31 $M = 2^{32}$, A = 1 664 525, $C = 1$	lcong31 $m = 2^{31} - 1$, a = 2 100 005 341, $c = 0$	gfsr_31 P = 1 279, $q = 418$, $w = 32$	gfsr5_31 $p = 521$, $q_1 = 86$, $q_2 = 197$, $q_3 = 447$, $w = 32$	taus88_31 Указанные в настоящем стандарте	genrand_31 Указанные в настоящем стандарте	rndtable31 Файл prnd01. bin
Обращение к инициализации	init_lcong32	init_lcong31	Init_gfsr	init_gfsr5	init_taus88	init_genrand	init_rndtable
Начальное число для инициализации	19 660 809	19 660 809	19 660 809	19 660 809	19 660 809	19 660 809	19 660 809
Результаты							
Номер случайного числа	Случайное число						
1	1 276 136 251	1 990 801 112	716 530 710	716 530 710	116 464 117	652 430 828	57 316 494
2	865 096 703	549 424 302	1 004 066 893	1 004 066 893	1 350 114 716	769 118 065	905 630 297
3	1 405 063 418	2 128 986 934	1 271 815 862	1 271 815 862	14 524 262	902 643 984	1 460 801 524
4	1 021 835 442	637 203 998	955 533 625	955 533 625	565 035 872	1 576 219 271	751 624 663
5	1 313 685 521	965 379 446	626 736 785	626 736 785	1 079 577 460	859 869 705	1 289 292 436
1 000	1 292 340 048	294 652 208	1 588 358 191	1 935 299 389	1 404 867 807	1 194 038 620	2 001 042 935
2 000	517 257 756	407 927 492	2 027 766 761	43 898 710	2 022 781 177	563 296 554	1 638 049 143
3 000	1 420 573 800	216 557 927	1 495 802 935	1 516 572 896	2 098 228 799	1 515 829 663	41 578 219
4 000	1 195 033 140	919 639 774	1 360 928 075	1 923 029 091	1 089 352 213	1 803 857 212	87 938 653
5 000	971 701 120	639 093 944	1 950 421 053	2 129 964 021	262 361 229	1 203 434 155	1 851 047 367

Приложение ДА

(справочное)

Сведения о соответствии ссылочных международных стандартов ссылочным национальным стандартам Российской Федерации (и действующим в этом качестве межгосударственным стандартам)

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
ИСО/МЭК 2382-1:1993	—	*
ИСО 3534-1:2006	—	*
ИСО 3534-2:2006	—	*
* Соответствующий национальный стандарт отсутствует. До его утверждения рекомендуется использовать перевод на русский язык данного международного стандарта. Перевод данного международного стандарта находится в Федеральном информационном фонде технических регламентов и стандартов.		

Библиография

[1] ISO 80000-2 Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology[1)]

[2] ISO/IEC 9899 Programming languages — C

[3] FERRENBERG A.M., LANDAU D.P. and WONG Y.J. Monte Carlo Simulations: Hidden Errors from"Good" Random Number Generators. Physical Review Letters, 69(23), 1992, pp. 3382-3384

[4] GENTLE J.E. Random Number Generation and Monte Carlo Methods, Springer-Verlag, 2003

[5] HERINGA J.R., BLÖTE H.W.J. and COMPAGNER A. New Primitive Trinomials of Mersenne-Exponent Degrees for Random-Number Generation. International Journal of Modern Physics C, **3**(3), 1992, pp. 561-564

[6] ISHIDA M., SATO T., SUZUKI K., SHIMADA S. and KAWASE T. Random Number Generator Using a Diode Noise. The Institute of Statistical Mathematics Research Memorandum, Number 968, 2005

[7] JÖHNK M.D. Erzeugung Von Betavesteilten und Gammavesteilten Zufellszahlen. Metrica, 8(1), 1964, pp. 5-15

[8] KNUTH D.E. Seminumerical Algorithms (The Art of Computer Programming, Volume 2), 3rd. ed., Addison Wesley, 1998

[9] KURITA Y. and MATSUMOTO M. Primitive t-nomials (t = 3, 5) over GF (2) Whose Degree is a Mersenne Exponent u 44497. Mathematics of Computation, 56(194), 1991, pp. 817-821

[10] L'ECUYER P. Maximally Equidistributed Combined Tausworthe Generators. Mathematics of Computation, 65(213), 1996, pp. 203-213

[11] L'ECUYER P. Tables of Maximally-Equidistributed Combined LFSR Generators. Mathematics of Computation, 68(225), 1996, pp. 261-269

[12] LEWIS T.G. and PAYNE W.H. Generalized Feedback Shift Register Pseudorandom Number Generators. Journal of the Association for Computing Machinery, **20**(3), 1973, pp. 456-468

[13] MASSEY J.L. Shift-Register Synthesis and BCH Decoding. IEEE Trans. on Information Theory, IT-15 (1), 1969, pp. 122 127

[14] MATSUMOTO M. and NISHIMURA, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM. Trans. Model. Comput. Simul., 8(1), 1998, pp. 3-30

[15] VON NEUMANN J. Various Techniques Used in Connection with Random Digits, Monte Carlo Method, Applied Mathematics Series, No.12, U.S. National Bureau of Standards, Washington D.C., 1951, pp. 36-38

[16] NIKI N. Machine Generation of Randum Numbers. The Institute of Statistical Mathematics Research Memorandum, Number 969, 2005

[17] NIKI N. Physical Random Number Generator for Personal Computers. The Institute of Statistical Mathematics Research Memorandum, Number 970, 2005

[18] RUEPPEL R.A. Analysis and Design of Stream Ciphers, Springer-Verlag, 1986

[19] TAUSWORTHE R.C. Random Numbers Generated by Linear Recurrence Modulo Two. Mathematics of Computation, 19, 1965, pp. 201-209

[1)] Международному стандарту ISO 80000:2009 соответствует ГОСТ Р 54521 – 2011 Статистические методы. Математические символы и знаки для применения в стандартах.

[20] TEZUKA S. and L'ECUYER P. Efficient and Portable Combined Tausworthe Random Number Generators. ACM. Trans. Model. Comput. Simul., **1**, 1991, pp. 99-112

[21] VATTULAINEN I., ALA-NISSILÄ T. and KANKAALA, K. Physical Tests for Random Numbers in Simulations. Physical Review Letters, 73(19), 1994, pp. 2513 2516

[22] VATTULAINEN I., ALA-NISSILÄ T. and KANKAALA, K. Physical Models as Tests of Randomness. Physical Review, E52, 1995, pp. 3205-3214

[23] Random Number Generator, The Institute of Statistical Mathematics, http://random.ism.ac.jp/random_e/index.php

УДК 658.562.012.7:65.012.122:006.352 **ОКС 03.120.30** **Т59**

Ключевые слова: псевдослучайное число, физическое случайное число, начальное число, генератор случайных чисел, двоичные числа

Подписано в печать 30.04.2014. Формат 60x84 1/8.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

ФГУП «СТАНДАРТИНФОРМ»

123995 Москва, Гранатный пер., 4.
www.gostinfo.ru info@gostinfo.ru

сертификация автотранспорта