Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

по дисциплине «Программирование на языках высокого уровня»

«Запаздывающие генераторы Фибоначчи»

Пояснительная записка

Выполнил:
студент гр. 3331506/70401 Паньков И.С.

Проверил:
Преподаватель Ананьевский М.С.

« »_____2020 г.

Санкт-Петербург
2020

Содержание

Введение

В современной информатике широкое применение находят последовательности случайных чисел: начиная от методов Монте-Карло (группа численных методов на основе использования случайных процессов) и заканчивая криптографическими методами шифрования данных.

Физические источники случайных чисел (на основе различных шумов) обладают рядом недостатков: большие временные и трудовые затраты при установке и настройке, низкая скорость генерации случайных чисел, высокая дороговизна и отсутствие воспроизводимости ранее полученных результатов. По этой причине широкое применение получили генераторы псевдослучайных чисел (ГСПЧ).

1. Генераторы псевдослучайных чисел

Строго говоря, последовательности случайных чисел, которые производят эти генераторы вовсе не являются случайными — они лишь аппроксимацией некоторые свойства случайных последовательностей (отсюда и название «псевдослучайных чисел»). Кроме того, вследствие особенностей алгоритмов ГСПЧ, последовательности псевдослучайных чисел являются периодичными, то есть начиная с некоторого члена последовательности случайные числа начинают повторяться. Также они обладают и другими недостатками:

- Последовательные значения не являются независимыми;
- При одинаковых начальных значениях получается одинаковые последовательности случайных чисел;
- Неравномерное распределение вероятностей появления чисел в последовательности;
- Неравномерное распределение вероятностей по битам в числе;
- Обратимость.

Тем не менее, для различных алгоритмов ГСПЧ те или иные недостатки проявляются в большей или меньшей степени.

Так в 1949 году Д.Г. Лехмером был предложен линейный конгруэнтный алгоритм. Согласно нему члены последовательности x_n вычисляются по формуле

$$x_k = \left(ax_{k-1} + c\right) \bmod m = \left(a^k x_0 + \frac{c\left(a^k - 1\right)}{a-1}\right) \bmod m,\ k > 1,$$

где x_k, x_{k-1} и x_0 — k-ый, $(k-1)$-ый и начальный члены последовательности x_n;

$m = z^e > 0$ — модуль;

$a \in [0; m]$ — множитель;

$c \in [0; m]$ — приращение;

z — разрядность системы счисления ЭВМ;

e — число битов в машинном слове.

Правильный подбор параметров улучшает свойства псевдослучайной последовательности (приближая её к действительно случайной). Например, в компиляторе GCC для языка *C* используется следующая реализация этого алгоритма:

```
#define RAND_MAX 32767

static unsigned long int next = 1;

int rand(void)
{
    next = next * 1103515245 + 12345;
    return (unsigned int)(next / 65536) % (RAND_MAX + 1);
}

void srand(unsigned int seed)
{
    next = seed;
}
```

Тем не менее в некоторых алгоритмах к качеству последовательностей псевдослучайных чисел предъявляются более высокие требования. Для их достижения применяются аддитивные генераторы псевдослучайных чисел.

2. Аддитивные ГПСЧ

2.1. Генератор Фибоначчи

Для улучшения свойств последовательности псевдослучайных чисел можно для вычисления значения члена последовательности x_k использовать два предыдущих значения — x_{k-1} и x_{k-2}. В этом случае возрастёт период последовательности псевдослучайных чисел (в лучшем случае до m^2). Простейшая последовательность задаётся формулой

$$x_k = \left(x_{k-1} + x_{k-2}\right) \bmod m$$

и называется последовательностью Фибоначчи.

Эта последовательность была предложена в начале 1950-х годов для использования в генераторах последовательностей псевдослучайных чисел. Она действительно даёт период больший, нежели линейный конгруэнтный алгоритм, но, к сожалению, числа, которые она выдаёт, являются недостаточно случайными (то есть не соблюдаются некоторые свойства последовательностей случайных чисел).

Дальнейшие исследования этой последовательности с целью её модификации привели к созданию последовательности Фибоначчи с запаздыванием и генераторов Фибоначчи с запаздыванием на её основе.

2.2. Генераторы Фибоначчи с запаздыванием

Последовательность x_n, члены которой вычисляются по формуле

$$x_k = \left(x_{k-a} + x_{k-b}\right) \bmod m,\ a, b > 0,\ a \neq b,\ k > \max\{a, b\},$$

где x_k, x_{k-a} и x_{k-b} — k-ый, $(k-a)$-ый и $(k-b)$-ый члены последовательности x_n;

$m = z^e > 0$ — модуль;

a и b — запаздывания;

$x_0, \ldots, x_{\max\{a,b\}}$ — произвольные целые числа;

z — разрядность системы счисления ЭВМ;

e — число битов в машинном слове;

называется последовательностью Фибоначчи с запаздыванием.

В 1958 году Дж.Ж. Митчел и Д.Ф. Мур провели исследования и предложили последовательность Фибоначчи с запаздываниями $a = 24$ и $b = 55$:

$$x_k = \left(x_{k-24} + x_{k-55} \right) \bmod m,\ k > 55.$$

Такая последовательность выдаёт более качественные псевдослучайные числа по сравнению с другими запаздываниями и имеет период $2^{32}(2^{55} - 1)$ при $m = 2^{32}$, что позволяет генерировать достаточно длинные последовательности псевдослучайных чисел. Позднее были найдены и другие качественные пары запаздываний (a, b); вот некоторые из них: (9, 49), (19, 58), (18, 65), (25, 73), (38, 89), (2, 93), (21, 94), (11, 95), (37, 100), (33, 118), (10, 111), (37, 124), (29, 132), (52, 145), (57, 134), (83, 258) (107, 378), (273, 607), (1029, 2281), (576, 3217), (4187, 9689), (7083, 19937), (9739, 23209). Стоит заметить, что дальнейшее увеличение значений запаздываний является неэффективным, так как требует больший объём памяти для хранения инициализирующего вектора $\left\{ x_0, \ldots, x_{\max\{a, b\}} \right\}$.

Генераторы Фибоначчи с запаздыванием показали неплохие результаты при генерации последовательностей псевдослучайных чисел, и на их основе были созданы несколько алгоритмов шифрования — FISH, PIKE и MUSH.

2.3. Алгоритм шифрования FISH

FISH — сокращение от Fibonacci Shrinking Generator — прореживаемый генератор Фибоначчи. Он является композицией аддитивного ГПСЧ и идеи прореживания последовательности чисел.

Алгоритм состоит из следующих шагов:

1. Задаются начальные состояния двух генераторов $\left\{ a_0, \ldots, a_{55} \right\}$ и $\left\{ b_0, \ldots, b_{52} \right\}$;

2. На *k*-ом шаге вычисляются *k*-ые значения членов последовательностей a_n и b_n по следующим формулам:

$$a_k = \left(a_{k-55} + a_{k-24} \right) \bmod 2^{32},$$
$$b_k = \left(b_{k-52} + b_{k-19} \right) \bmod 2^{32};$$

3. Затем к этим последовательностям чисел применяется процедура прореживания в зависимости от младшего значащего бита *k*-ого члена последовательности b_n: если значение бита равно 1, то пара используется, если 0 — игнорируется. В результате прореживания пары используемых чисел a_k и b_k образуют пару новых последовательностей c_n и d_n;

4. Результатом работы генератора на *l*-м шаге является 32-битное слово, вычисленное по следующим формулам:

$$e_{2l} = c_{2l} \oplus (d_{2l} \cap d_{2l+1}),$$
$$f_{2l} = d_{2l+1} \oplus (e_{2l} \cap c_{2l+1}),$$
$$k_{2l} = e_{2l} \oplus f_{2l},$$
$$k_{2l+1} = c_{2l+1} \oplus f_{2l},$$

где k_{2l} и k_{2l+1} — два 32-битовых слова, получаемых вследствие работы генератора.

2.4. Алгоритм шифрования PIKE

PIKE (англ. pike — «щука») — более быстрая модификация алгоритма FISH, предложенная Р. Андерсоном, которому удалось взломать алгоритм FISH. Помимо основной идеи использования аддитивного ГПСЧ, PIKE использует идею порогового механизма управления движением регистров. В алгоритме используется три аддитивных генератора:

$$a_k = (a_{k-55} + a_{k-24}) \bmod 2^{32},$$
$$b_k = (b_{k-57} + b_{k-7}) \bmod 2^{32},$$
$$c_k = (c_{k-58} + c_{k-19}) \bmod 2^{32}.$$

Для генерации 32-битного слова последовательности ключей на каждой итерации рассматриваются биты переноса при сложении. Если все три одинаковы, то тактируются все три генератора. Если нет, то тактируются только два совпадающих генератора. Биты переноса сохраняются для следующей итерации. Окончательным выходом является побитовая сумма по модулю 2 выходов всех трех генераторов.

2.5. Алгоритм шифрования MUSH

MUSH также основан на аддитивном генераторе, но использует процедуру прореживания. Алгоритм состоит из следующих шагов:

1. Вычисляются значений аддитивных генераторов по следующим формулам:

$$a_k = \left(a_{k-55} + a_{k-24}\right) \bmod 2^{32},$$
$$b_k = \left(b_{k-52} + b_{k-19}\right) \bmod 2^{32};$$

2. Если бит переноса a_k установлен, тактируется b_k. Если бит переноса b_k установлен, тактируется a_k;

3. Тактируем a_k и при переполнении устанавливаем бит переноса. Тактируем b_k и при переполнении устанавливаем бит переноса;

4. Выходом является побитовая сумма по модулю 2 выходов a_n и b_n.

3. Реализация алгоритма

Реализовывать алгоритм будем на языке программирования *C*++14 (*International Standard ISO/IEC* 14882:2014(*E*) *Programming Language C++*). В качестве запаздывающего генератора Фибоначчи будем использовать следующую последовательность:

$$x_k = \left(x_{k-a} + x_{k-b} \right) \bmod 2^{15}.$$

Инициализирующий вектор запаздывающего генератора Фибоначчи $\left\{ x_0, \ldots, x_{\max\{a,b\}} \right\}$ заполним псевдослучайными числами с помощью стандартной библиотечной функции `rand`. Именно по этой причине был взят модуль $m = 2^{15}$: функция `rand` выдаёт числа от 0 до `RAND_MAX`. Стандартная библиотека `stdlib.h` определяет `RAND_MAX` как $7\text{FFF}_{16} = 2^{15} - 1$.

Чтобы повысить степень случайности чисел в инициализирующем векторе, в качестве начального значения для функции `rand` зададим системное время функцией `time` стандартной библиотеки `time.h`.

Заполнив инициализирующий вектор, начнём генерировать последовательность псевдослучайных чисел с помощью функции `laggedFibGen`.

Получившуюся последовательность псевдослучайных чисел будем выводить в текстовый файл для дальнейшей обработки получившихся результатов. Делать это будем, создав объект класса `fstream` библиотеки `fstream`.

Ниже представлен листинг программы для генерации последовательности из 10 тысяч псевдослучайных чисел:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <fstream>

using namespace std;

#define SIZE       10000
#define MAX_NUMBER 500

static const unsigned int a = 24;
static const unsigned int b = 55;

static unsigned int laggedFibGen(const unsigned int *x, const unsigned
int index)
{
    return (unsigned int)(x[index - a] + x[index - b]) % (RAND_MAX +
1);
}

int main()
{
    // Задаём начальное значение для функции rand
    srand(time(NULL));
    // Создаём массив для последовательности псевдослучайных чисел
    unsigned int x[__max(a, b) + SIZE + 1] = {0};
    // Заполняем инициализирующий вектор псевдослучайными числами
    for (unsigned int index = 0; index <= __max(a, b); index++)
    {
        x[index] = rand() % (MAX_NUMBER + 1);
    }
    // Генерируем последовательность псевдослучайных  чисел
    for (unsigned int index = __max(a, b) + 1; index <= __max(a, b) +
SIZE; index++)
    {
        x[index] = laggedFibGen(x, index) % (MAX_NUMBER + 1);
    }
    // создаём поток для вывода последовательности в текстовый файл
    fstream result;
    result.open("result.txt", fstream::in | fstream::out);
    // Выводим последовательность в текстовый файл
    for (unsigned int index = __max(a, b) + 1; index <= __max(a, b) +
SIZE; index++)
    {
        result << x[index] << endl;
    }
    result.close();
}
```

4. Анализ алгоритма

Алгоритм практически не представляет интереса с точки зрения анализа времени выполнения или затрачиваемой памяти. Очевидно, что он обладает линейной сложностью $O(n)$ в силу линейности операций генерации каждого следующего числа. Также он требует затрат памяти порядка $O(n)$ в силу необходимости создания массива и хранения последовательности чисел в нём.

Несколько больший интерес представляет качество генерируемых последовательностей псевдослучайных чисел, то есть степень схожести получаемых чисел с выборкой случайных чисел, распределённых по закону равномерного распределения.

Начнём обработку получаемых последовательностей с того, что представим их в графическом виде. На рисунках 1, 2 и 3 представлены последовательности из 100, 1000 и 10000 чисел в диапазоне от 0 до 500.

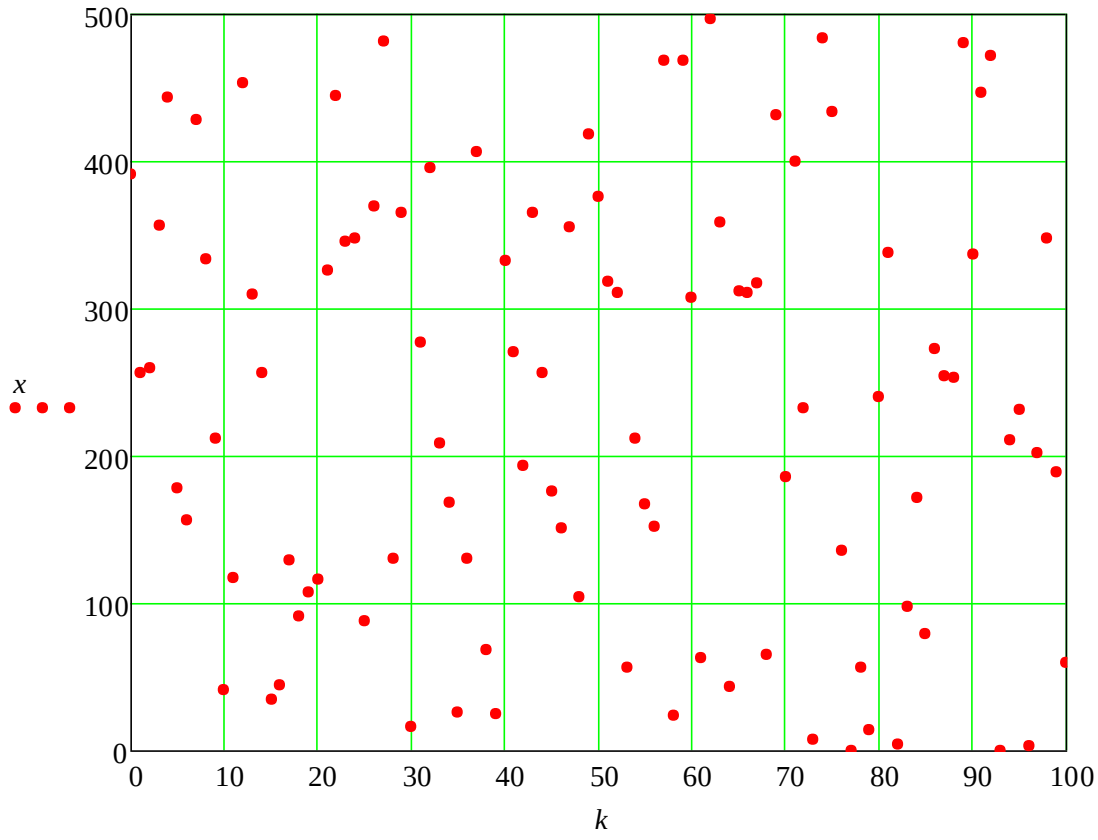

Рисунок 1 — Последовательность из 100 псевдослучайных чисел

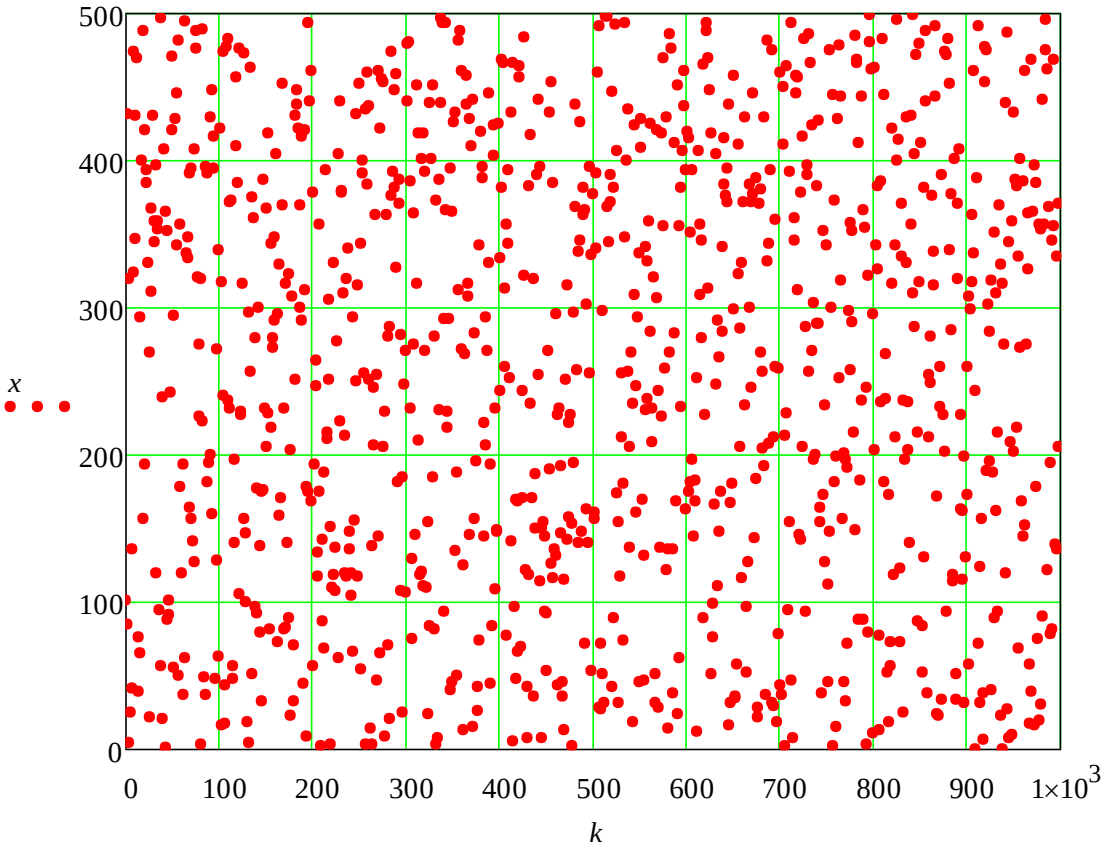

Рисунок 2 — Последовательность из 1000 псевдослучайных чисел

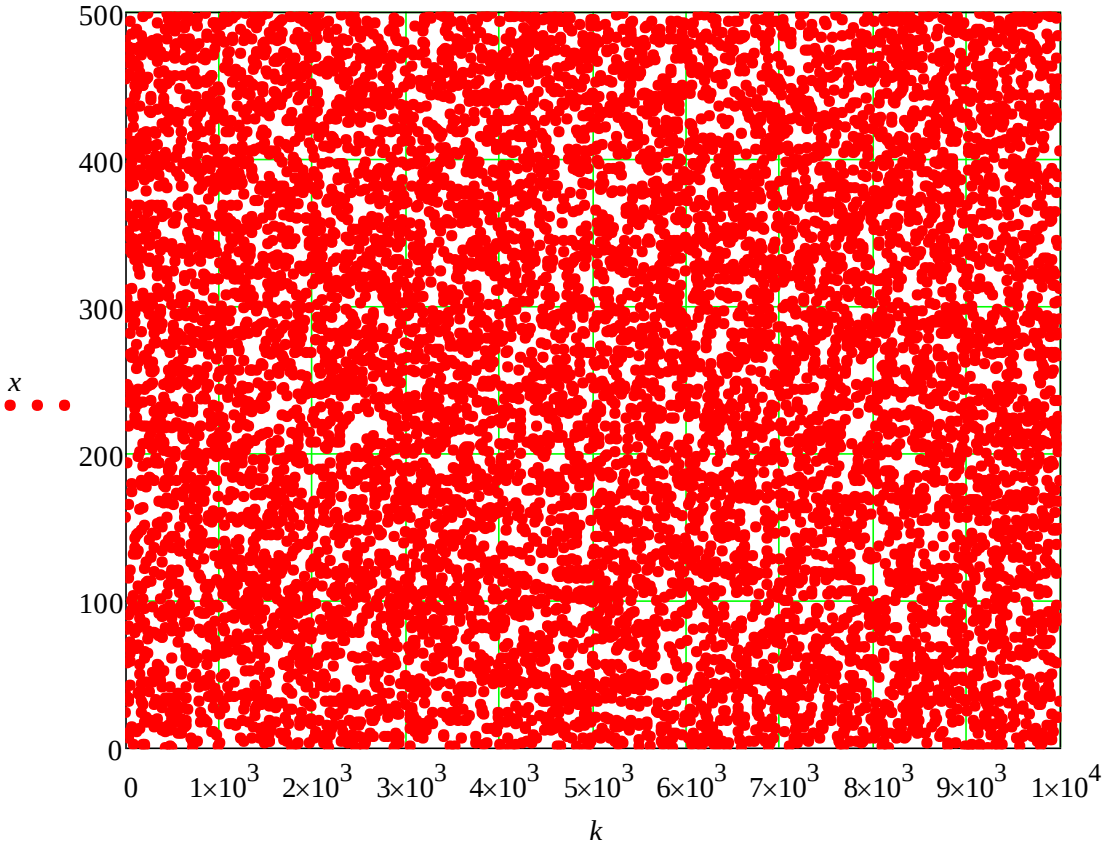

Рисунок 3 — Последовательность из 10000 псевдослучайных чисел

Рассмотрим последовательность из 100 псевдослучайных чисел. Гистограмма и график эмпирической функции распределения последовательности представлены на рисунках 4 и 5.

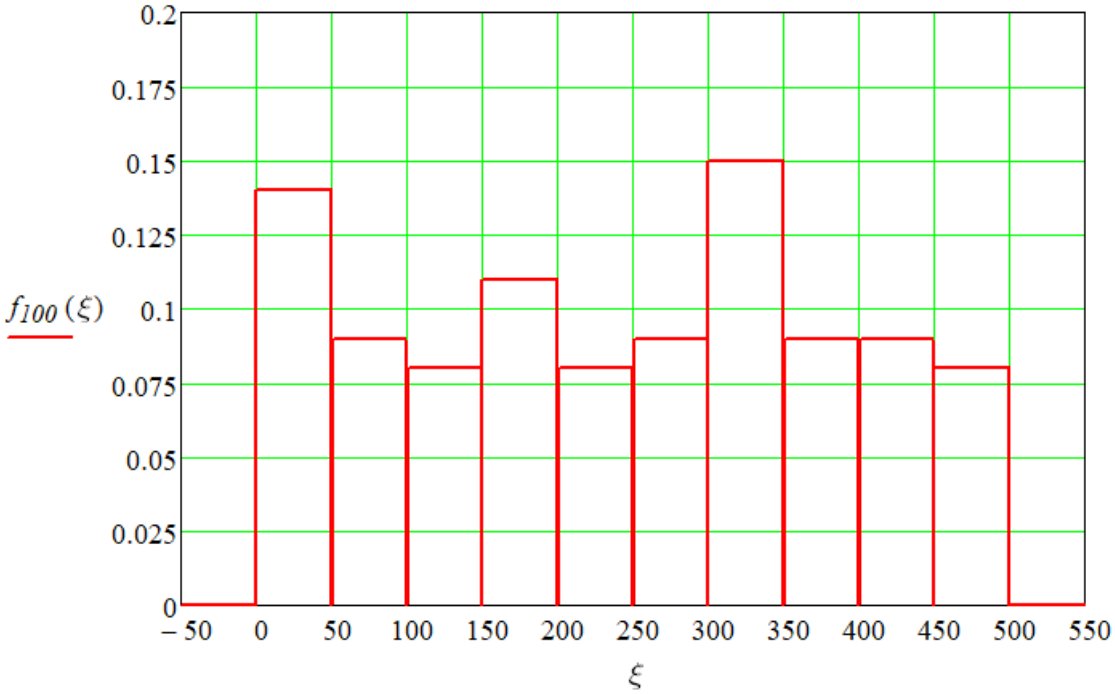

Рисунок 4 — Гистограмма последовательности из 100 псевдослучайных чисел

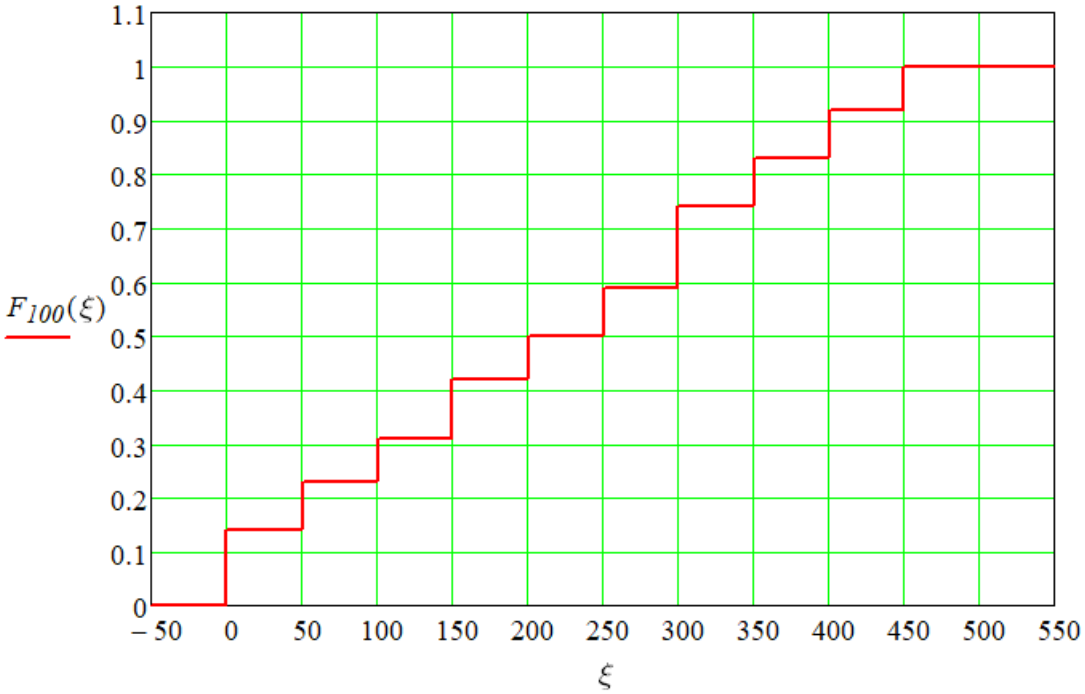

Рисунок 5 — График эмпирической функции распределения последовательности из 100 псевдослучайных чисел

Рассчитаем в *MathCAD* значения выборочного среднего $x_{ср.}$, несмещённой оценки дисперсии $\tilde{S}^2$ и среднеквадратичного отклонения $\tilde{S}$ для последовательности из 100 псевдослучайных чисел и сравним их со значениями математического ожидания $M[x]$, дисперсии $D[x]$ и среднеквадратичного отклонения σ_x для случайной величины, равномерно распределённой на интервале $[a;b]$ $(a=0,\ b=500)$. Результаты расчёта представлены на рисунке 6.

$$x_{cp.} = \frac{1}{size}\cdot\sum_{k=0}^{size-1} x_k = 238.09 \qquad Mx = \frac{b-a}{2} = 250$$

$$\left[S'^2\right] = \frac{1}{size-1}\cdot\sum_{k=0}^{size-1}\left(x_{cp.}-x_k\right)^2 = 2.164\times 10^4 \qquad Dx = \frac{b^2-a^2}{12} = 2.083\times 10^4$$

$$S' = \sqrt{\left[S'^2\right]} = 147.091 \qquad \sigma_x = \sqrt{Dx} = 144.338$$

Рисунок 6 — Фрагмент листа *MathCAD* с результатами расчёта

Рассмотрим последовательность из 1000 псевдослучайных чисел. Гистограмма и график эмпирической функции распределения последовательности представлены на рисунках 7 и 8.

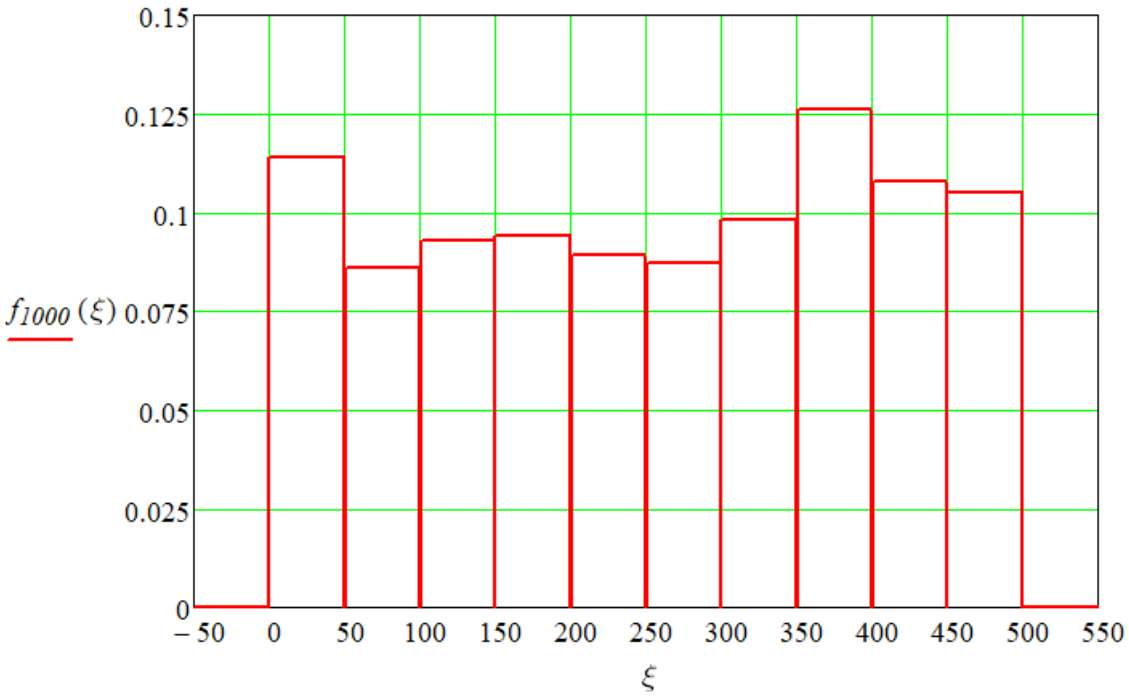

Рисунок 7 — Гистограмма последовательности из 1000 псевдослучайных чисел

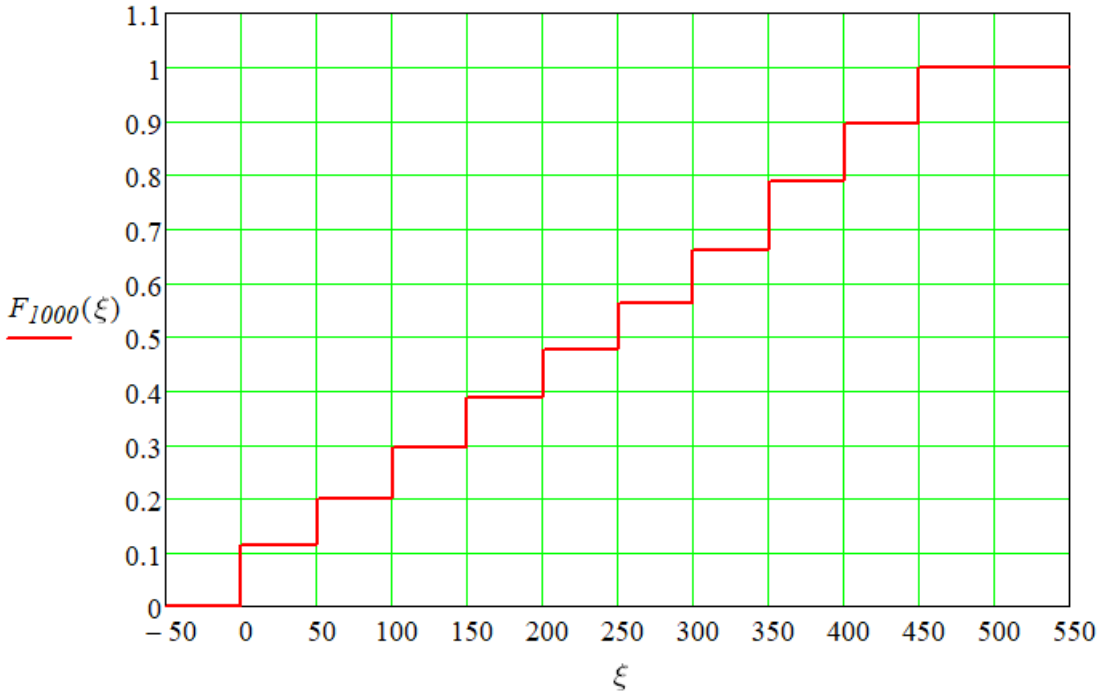

Рисунок 8 — График эмпирической функции распределения последовательности из 1000 псевдослучайных чисел

Рассчитаем в *MathCAD* значения выборочного среднего $x_{ср.}$, несмещённой оценки дисперсии $\tilde{S}^2$ и среднеквадратичного отклонения $\tilde{S}$ для последовательности из 1000 псевдослучайных чисел и сравним их со значениями математического ожидания $M[x]$, дисперсии $D[x]$ и среднеквадратичного отклонения σ_x для случайной величины, равномерно распределённой на интервале $[a;b]$ $(a=0,\ b=500)$. Результаты расчёта представлены на рисунке 9.

$$x_{cp.} = \frac{1}{size}\cdot\sum_{k=0}^{size-1} x_k = 256.384 \qquad Mx = \frac{b-a}{2} = 250$$

$$\left[S'^2\right] = \frac{1}{size-1}\cdot\sum_{k=0}^{size-1}\left(x_{cp.}-x_k\right)^2 = 2.177\times10^4 \qquad Dx = \frac{b^2-a^2}{12} = 2.083\times10^4$$

$$S' = \sqrt{\left[S'^2\right]} = 147.557 \qquad \sigma_x = \sqrt{Dx} = 144.338$$

Рисунок 9 — Фрагмент листа *MathCAD* с результатами расчёта

Рассмотрим последовательность из 10000 псевдослучайных чисел. Гистограмма и график эмпирической функции распределения последовательности представлены на рисунках 10 и 11.

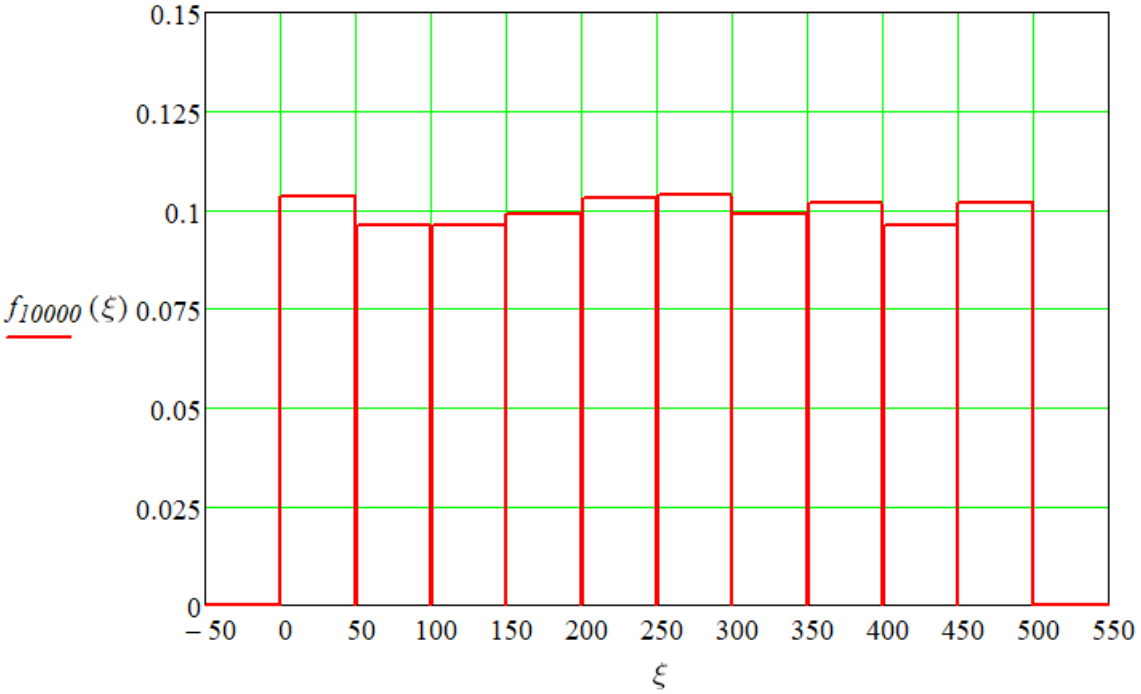

Рисунок 10 — Гистограмма последовательности из 10000 псевдослучайных чисел

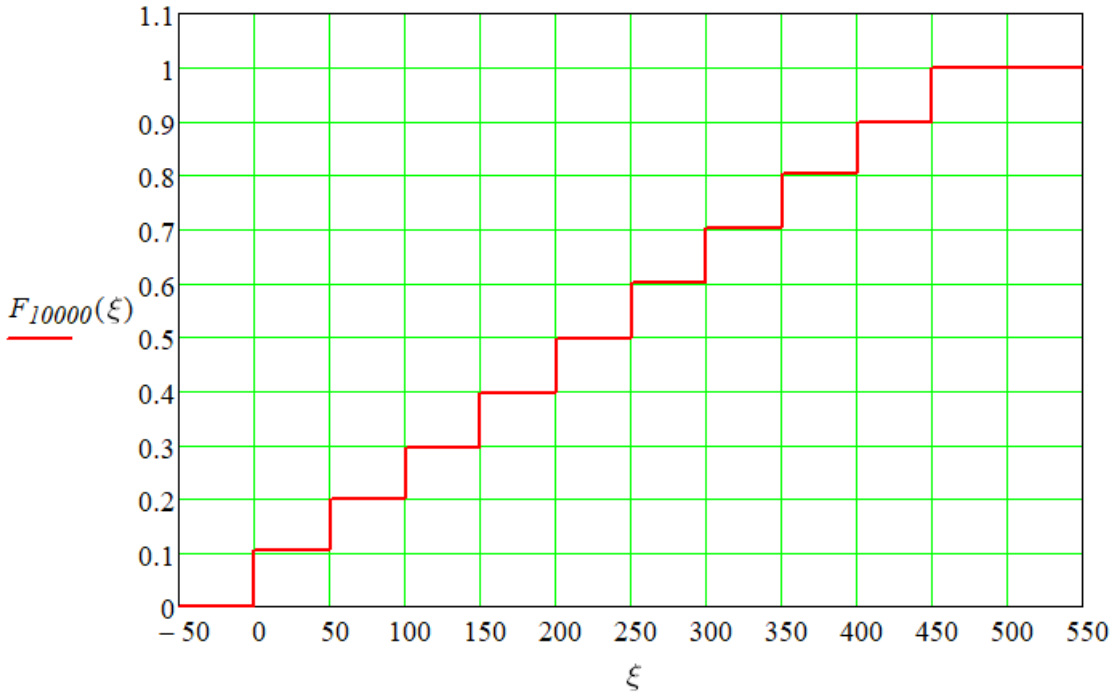

Рисунок 11 — График эмпирической функции распределения последовательности из 10000 псевдослучайных чисел

Рассчитаем в *MathCAD* значения выборочного среднего $x_{ср.}$, несмещённой оценки дисперсии $\tilde{S}^2$ и среднеквадратичного отклонения $\tilde{S}$ для последовательности из 10000 псевдослучайных чисел и сравним их со значениями математического ожидания $M[x]$, дисперсии $D[x]$ и среднеквадратичного отклонения σ_x для случайной величины, равномерно распределённой на интервале $[a;b]$ $(a=0,\ b=500)$. Результаты расчёта представлены на рисунке 12.

$$x_{cp.} = \frac{1}{size}\cdot\sum_{k=0}^{size-1} x_k = 250.041 \qquad Mx = \frac{b-a}{2} = 250$$

$$\left[S'^2\right] = \frac{1}{size-1}\cdot\sum_{k=0}^{size-1}\left(x_{cp.}-x_k\right)^2 = 2.087\times 10^4 \qquad Dx = \frac{b^2-a^2}{12} = 2.083\times 10^4$$

$$S' = \sqrt{\left[S'^2\right]} = 144.476 \qquad \sigma_x = \sqrt{Dx} = 144.338$$

Рисунок 12 — Фрагмент листа *MathCAD* с результатами расчёта

Как видно из представленных выше результатов, с увеличением размера выборки параметры распределения последовательности псевдослучайных чисел приближаются к параметрам равномерного распределения случайной величины, что говорит о высоком качестве получаемых запаздывающим генератором Фибоначчи последовательностей псевдослучайных чисел.

Заключение

Аддитивные генераторы псевдослучайных чисел, или запаздывающие генераторы Фибоначчи, позволяют генерировать качественные последовательности псевдослучайных чисел с параметрами распределения, близкими к параметрам распределения соответствующих случайных величин, что и было показано в этой работе.

Список литературы

1. Слеповичев И.И. Генераторы псевдослучайных чисел : учеб. пособие / Слеповичев И.И. — Самара : LAP LAMBERT Academic Publishing, 2017 – 118 с.

2. Кнут Д. Исскусство программирования. Том 2. Получисленные алгоритмы : учеб. пособие / 3-е изд. — М. : Вильямс, 2000 – 832 с.