*Институт Транспорта и связи*

****

**Лабораторная работа №1**

По дисциплине

«Моделирование систем»

Тема: **Реализация модели системы массового обслуживания**

Студент: Эдуард Арефьев

Группа: 4402BD

Рига

2017 г.

Оглавление

[Задача 3](#_Toc478378265)

[Таблица последовательных шагов 3](#_Toc478378266)

[Коэффициент простоя сервера 4](#_Toc478378267)

[Среднее число заявок, стоявших в очереди за все время 4](#_Toc478378268)

[Генераторы 4](#_Toc478378269)

[Распределение Эрланга 4](#_Toc478378270)

[Распределение Пуассона 6](#_Toc478378271)

[Нормальное распределение 7](#_Toc478378272)

[Экспоненциальное распределение 8](#_Toc478378273)

[Генератор псевдослучайных чисел Лекура 10](#_Toc478378274)

[Генератор псевдослучайных чисел языка C++ 11](#_Toc478378275)

[Выводы 12](#_Toc478378276)

[Код реализуемой программы 13](#_Toc478378277)

# Задача

1. Используя любой язык программирования, реализовать модель системы массового обслуживания на основе дискретно-событийного принципа отсчета модельного времени. При реализации алгоритма имитации работы СМО самостоятельно программно реализовать генераторы случайных величин.

2. Результаты имитации отразить в таблице. В таблицу занести последовательно шаги имитационного процесса. Каждая строка соответствует одному событию. В нее заносятся разыгрываемые времена осуществления будущих событий и значения парамеров, которые были изменены при наступлении текущего. В качестве следующего выбирается событие с минимальным временем осуществления. Первая строка таблицы обязательно должна отображать начальное состояние системы, при котором не пришла ни одна заявка, но сделан розыгрыш времени наступления ближайших событий.

3. Используя пакет Statistica протестировать разработанные генераторы.

4. Найти коэффициент использования сервера.

5. Определить максимальное число заявок, стоявших в очереди за все время .

# Таблица последовательных шагов

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер события | Событие | TimeTact | L1 | L2 | H | S | Q | V |
| 1 | Начальное событие | 0 | 4.18089 | 3.30825 | 501 | 1 | 0 | - |
| 2 | L2 | 3.30825 | 4.18089 | 3.7938 | 3.37929 | 0 | 0 | - |
| 3 | H | 3.3729 | 4.18089 | 3.7938 | 501 | 1 | 0 | - |
| 4 | L2 | 3.7938 | 4.18089 | 3.83312 | 4.18492 | 0 | 0 | - |
| 5 | L2 | 3.83312 | 4.18089 | 4.16708 | 4.18492 | 0 | 1 | 2 |
| 6 | L2 | 4.16708 | 4.18089 | 6.8557 | 4.18492 | 0 | 2 | 2.2 |
| 7 | L1 | 4.18089 | 23.3757 | 6.8557 | 4.18492 | 0 | 3 | 2.2.1 |
| 8 | H | 4.18492 | 23.3757 | 6.8557 | 4.45098 | 0 | 2 | 2.1 |
| 9 | H | 4.45098 | 23.3757 | 6.8557 | 5.43528 | 0 | 1 | 1 |
| 10 | H | 5.43528 | 23.3757 | 6.8557 | 16.177 | 0 | 0 | - |
| 11 | L2 | 6.8557 | 23.3757 | 10.1528 | 16.177 | 0 | 1 | 2 |
| 12 | L2 | 10.1528 | 23.3757 | 10.9926 | 16.177 | 0 | 2 | 2,2 |
| 13 | L2 | 10.9926 | 23.3757 | 11.0327 | 16.177 | 0 | 3 | 2,2,2 |
| 14 | L2 | 11.0327 | 23.3757 | 12.6595 | 16.177 | 0 | 4 | 2,2,2,2 |
| 15 | L2 | 12.6595 | 23.3757 | 14.0314 | 16.177 | 0 | 5 | 2,2,2,2,2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 499 | L2 | 487.013 | 490.53 | 487.748 | 487.013 | 0 | 27 | 2.2.1.2.2.. |
| 500 | H | 487.019 | 490.53 | 487.748 | 487.019 | 0 | 26 | 2.1.2.2.2... |
| 501 | H | 487.748 | 490.53 | 487.748 | 488.079 | 0 | 25 | 1.2.2.2.2... |
| 502 | L2 | 488.079 | 490.53 | 489.304 | 488.079 | 0 | 26 | 2.2.2.2.2... |
| 503 | H | 489.304 | 490.53 | 489.304 | 497.302 | 0 | 25 | 2.2.2.2.2... |
| 504 | L2 | 489.885 | 490.53 | 489.885 | 497.302 | 0 | 26 | 2.2.2.2.2... |
| 505 | L2 | 490.53 | 490.53 | 492.272 | 497.302 | 0 | 27 | 2.2.2.2.2... |
| 506 | L1 | 492.272 | 503.35 | 492.272 | 497.302 | 0 | 28 | 2.2.2.2.2... |
| 507 | L2 | 497.302 | 503.35 | 498.067 | 497.302 | 0 | 29 | 2.2.2.2.2... |
| 508 | H | 497.642 | 503.35 | 498.067 | 497.642 | 0 | 28 | 2.2.2.2.2... |
| 509 | H | 497.302 | 503.35 | 498.067 | 498.794 | 0 | 27 | 2.2.2.2.1... |
| 510 | L2 | 498.067 | 503.35 | 498.937 | 498.794 | 0 | 28 | 2.2.2.2.1... |
| 511 | H | 498.794 | 503.35 | 498.937 | 500.957 | 0 | 27 | 2.2.2.1.2... |
| 512 | L2 | 498.937 | 503.35 | 500.734 | 500.957 | 0 | 28 | 2.2.2.1.2... |
| 513 | END | 500 | 503.35 | 500.734 | 500.957 | 0 | 28 | 2.2.2.1.2... |

В таблице использованы следующие обозначения для событий и переменных:

* TimeTact – модельное время
* L1 – заявка 1 типа
* L2 – заявка 2 типа
* H – окончание обслуживания текущей заявки
* S – статус сервера (1 – свободен, 0 - занят)
* Q – количество заявок в очереди
* V –очередь содержащая типы заявок

# Коэффициент простоя сервера

# Среднее число заявок, стоявших в очереди за все время

# Проверка генераторов

Для последующий распределений были использованны следующие параметры:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Эрланговское  распределение | Пуассоновское распределение | Нормальное распределение | Экспоненциальное распределение | Уровень значимости |
| l=3  λ =0.25 | λ =0.5 | μ = 12  σ = 2 | λ =2 | α=0.05 |

## Проверка распределение Эрланга

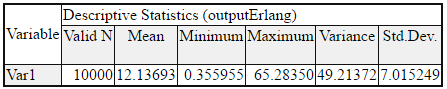


Рис 1. распределение Эрланга

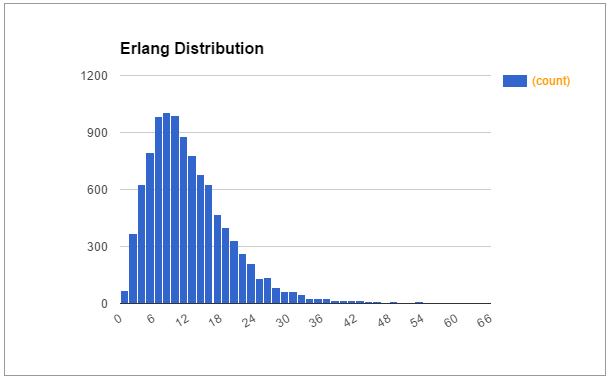


Рис 2. Гистограмма распределение Эрланга

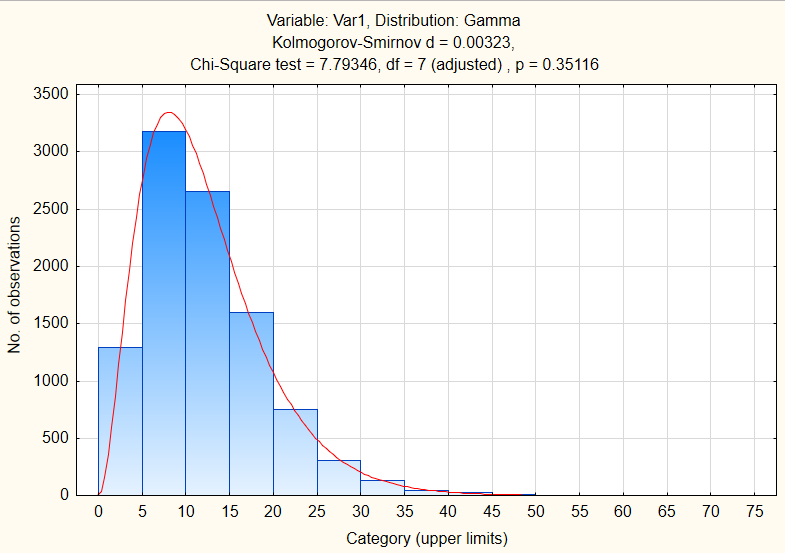


Рис 3. Гистограмма распределение Эрланга (statistica)

При уровне значимости 0.05:

* ОПГ для Хи-квадрат – [0; 14.06]
* ОПГ для Колмогора-Смирнова – [0; 1.36)

## Проверка распределение Пуассона

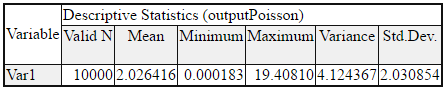


Рис 4. распределение Пуассона

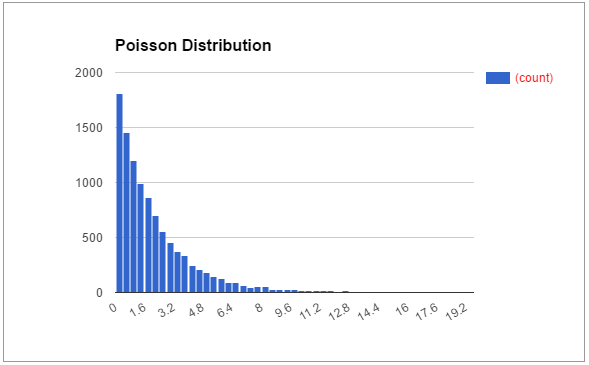


Рис 5. Гистограмма распределение Пуассона

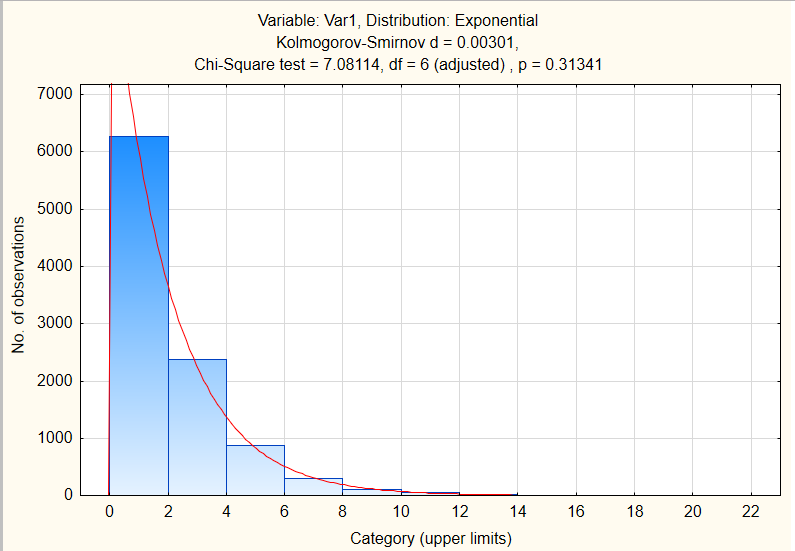


Рис 6. Гистограмма распределения Пуассона (statistica)

При уровне значимости 0.05:

* ОПГ для Хи-квадрат – [0; 12.59]
* ОПГ для Колмогора-Смирнова – [0; 1.36)

## Нормальное распределение

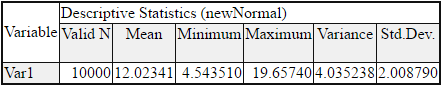


Рис 7. Нормальное распределение

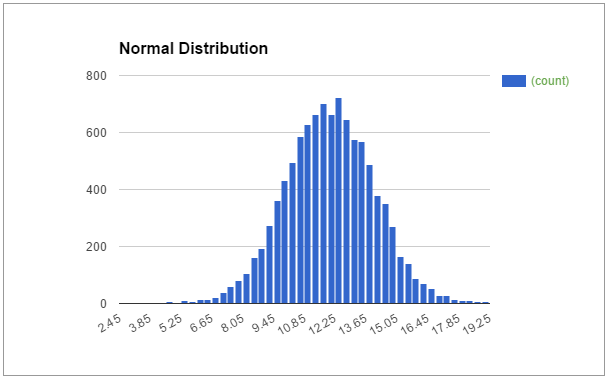


Рис 8. Гистограмма нормального распределение

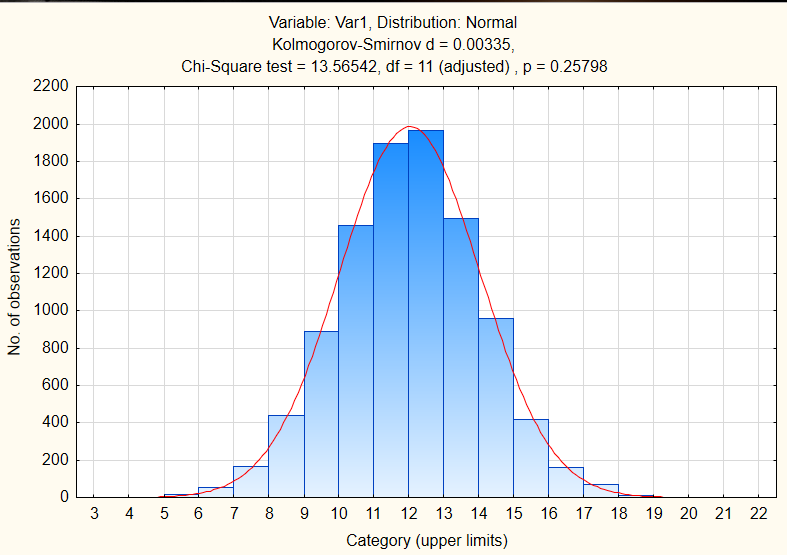


Рис 9. Гистограмма нормальное распределение (statistica)

При уровне значимости 0.05:

* ОПГ для Хи-квадрат – [0; 19,67]
* ОПГ для Колмогора-Смирнова – [0; 1.36)

## Экспоненциальное распределение

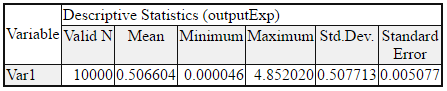


Рис 10. Экспоненциальное распределение

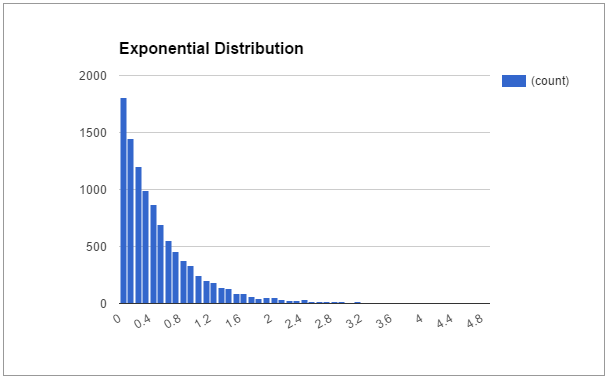


Рис 11. Гистограмма экспоненциального распределение

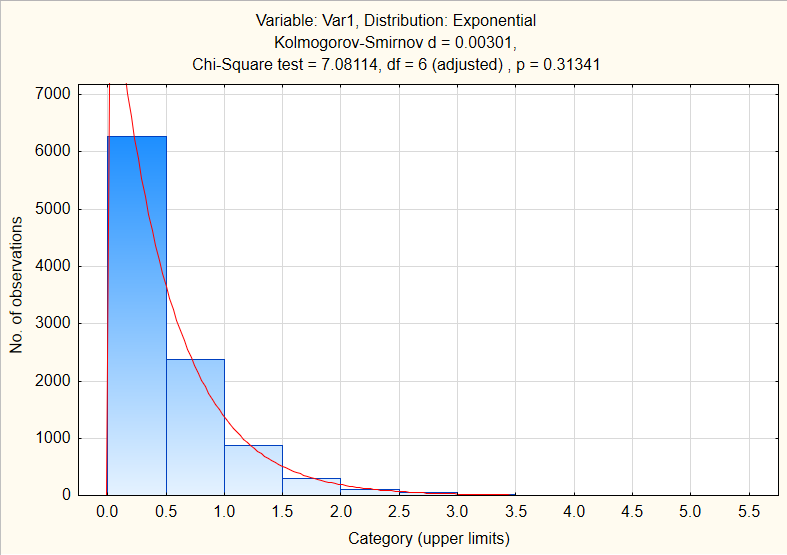


Рис 12. Гистограмма экспоненциального распределение (statistica)

При уровне значимости 0.05:

* ОПГ для Хи-квадрат – [0; 12.59]
* ОПГ для Колмогора-Смрнова – [0; 1.36)

## Выводы

Полученные результаты из выборок случайных величин для каждого закона распределения, все хорошо совпадает. Так как у нас p-value> 0.05. Так же, все распределения входят в ОПГ для Хи-квадрата и для Колмогорова-Смирнова. Реализованные генераторы последовательностей псевдослучайных чисел работают исправно.

# Генераторы псевдослучайных чисел

## Генератор псевдослучайных чисел Лекура

Был реализован алгоритм Лекура для генерации псевдослучайных чисел равномерного распределения. Так же он был сравнен с существующим псевдогенератором случайных чисел языка С++.

Использованны следующие константы для реализации генератора.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | m | a | c | биты |
| Геренатор 1 | 231 - 1 | 1103515245 | 12345 | 30..0 |
| Генератор 2 | 231 - 1 | 16807 | 0 | 30..0 |

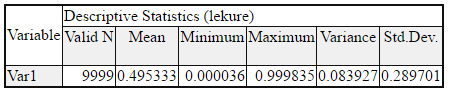
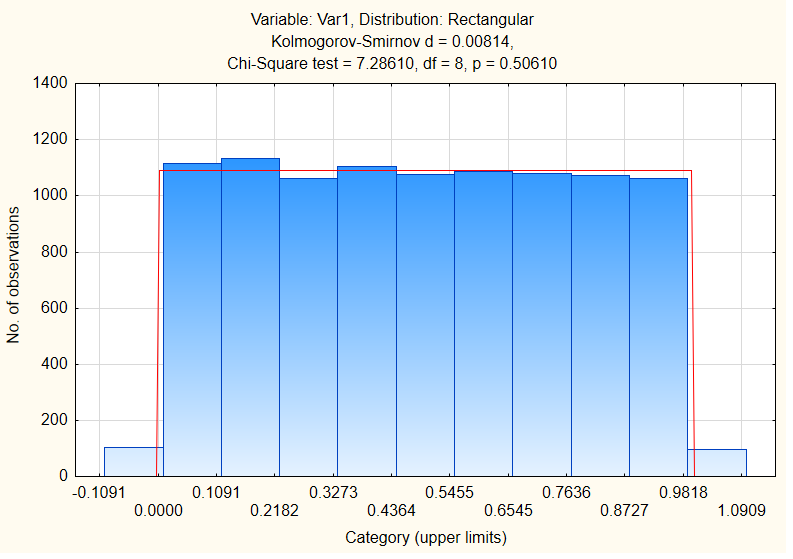


Рис 13. Генератор псевдослучайных чисел Лекура

 Рис 14. Гистограмма псевдослучайных чисел Лекура в интервале [0;1]

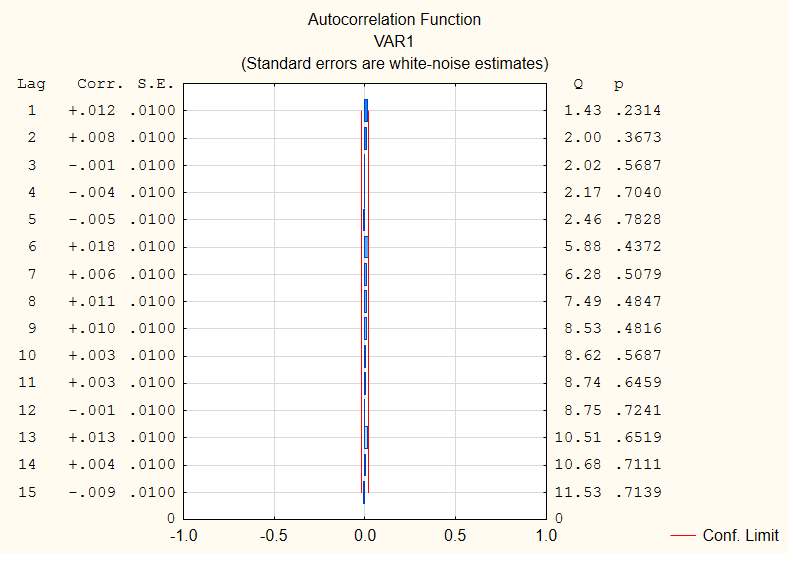


Рис 15. Автокорреляци псевдослучайных чисел Лекура

Гипотеза о равномерном распределении не может быть отвергнута, так как значения не выходят за границы ОПГ при уровне значимости 0.05:

* ОПГ для критерия Хи-квадрат – [0; 15.50]
* ОПГ для критерия Колмогорова-Смирнова – [0; 1.36]

Так же можно заметить что отсутсвует зависимость между значениями.

## Генератор псевдослучайных чисел языка C++

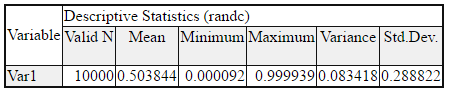
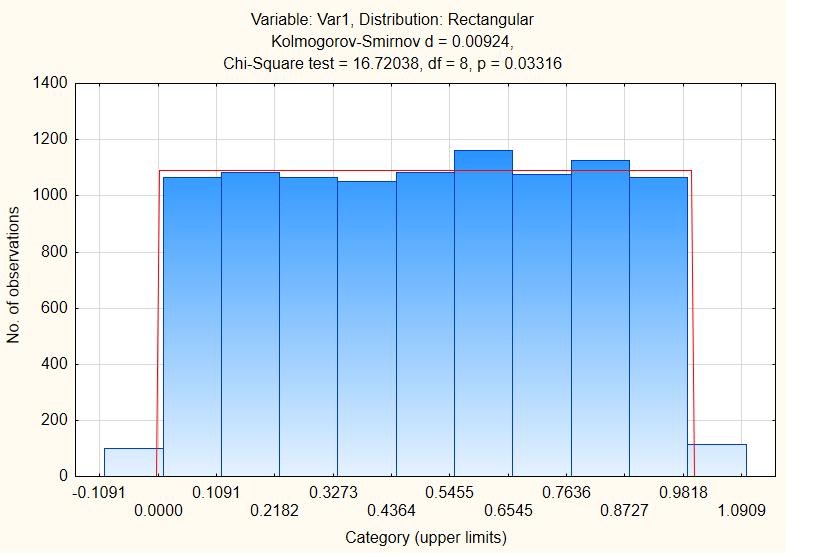


Рис 16. Генератор псевдослучайных чисел языка С++

 Рис 17. Гистограмма псевдослучайных чисел языка С++

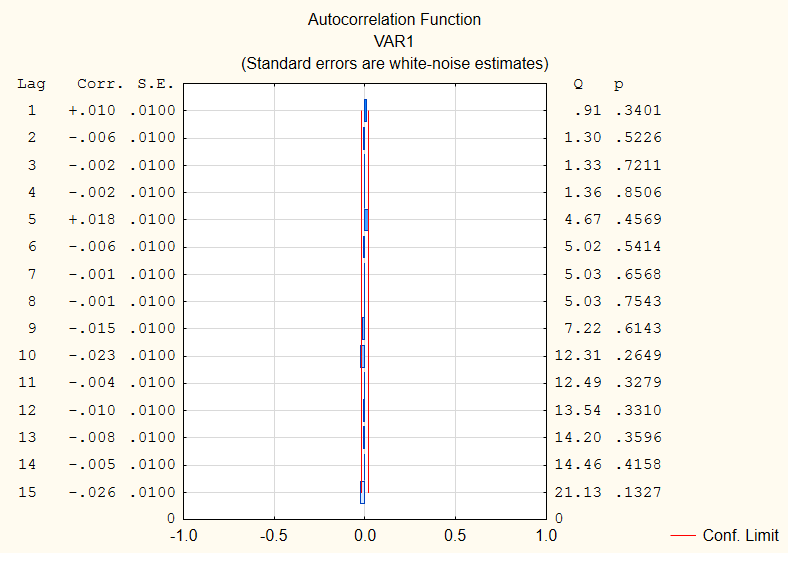


Рис 18. Автокореляция псевдослучайных чисел языка С++

Гипотеза о равномерном распределении не может быть отвергнута, так как значения не выходят за границы ОПГ при уровне значимости 0.05:

* ОПГ для критерия Хи-квадрат – [0; 15.50]
* ОПГ для критерия Колмогорова-Смирнова – [0; 1.36]

Так же можно заметить что отсутсвует зависимость между значениями.

# Выводы

В данной симуляции, очень редко встречаются заявки первого типа и когда они встречаются, начинается большая задержка на сервере в виде очереди. У сервера получился примерно 3% простоя, следовательно, он почти все время обрабатывал заявки. Чтобы улучшить данные показатели, нужно либо реже, либо быстрее обрабатывать заявки первого типа, либо комбинировать заявки первого и второго типа, чтобы не накапливались в большую очередь.

# Код реализуемой программы

#include <iostream>

#include <conio.h>

#include <algorithm>

#include <iomanip>

#include <ctime>

#define \_USE\_MATH\_DEFINES

#include <math.h>

using namespace std;

double erlangDist();

double poissonDist();

double normalDist();

double exponentialDist(double lambda);

double randomize();

void show();

unsigned int LecureRand();

double counter = 0;

double queCount = 0;

double queSum = 0;

double timeTact = 0;

double timeEnd = 500;

double l1 = erlangDist();

double l2 = poissonDist();

double h = timeEnd + 1;

bool server = true;

int que = 0;

double minVal;

double downTime = 0;

double t1 = 0, t2;

int \*queArr = new int[255];

unsigned int pRnd\_1 = 0;

unsigned int pRnd\_2 = 0;

unsigned int LECURE\_RAND\_MAX = pow(2, 31) - 1;

int main()

{

srand(time(NULL));

cout << "eve: S ";

while (timeEnd != 501)

{

counter++;

show();

//\_getch();

minVal = min({ l1, l2, h, timeEnd });

timeTact = minVal;

cout << "eve: ";

if (minVal == timeEnd)

{

cout << "END ";

downTime = downTime + timeTact - t1;

timeEnd++;

}

if (minVal == l1)

{

cout << "l1 ";

if (server)

{

server = false;

h = minVal + normalDist();

t1 = minVal;

}

else

{

queArr[que] = 1;

que++;

queCount++;

queSum += que;

}

l1 = minVal + erlangDist();

}

if (minVal == l2)

{

cout << "l2 ";

if (server)

{

server = false;

h = minVal + exponentialDist(2);

t1 = minVal;

}

else

{

queArr[que] = 2;

que++;

queCount++;

queSum += que;

}

l2 = minVal + poissonDist();

}

if (minVal == h)

{

cout <<"h ";

if (que == 0)

{

h = 501;

server = true;

}

else

{

que--;

if (queArr[0] == 1)

{

//FIFO

for (int i = 0; i < que; i++)

queArr[i] = queArr[i+1];

h = minVal + normalDist();

}

else

{

for (int i = 0; i < que; i++)

queArr[i] = queArr[i + 1];

h = minVal + exponentialDist(2);

}

}

}

if (server)

downTime = downTime + timeTact - t1;

}

show();

double avgQue = queSum / counter;

cout << "\nINFO\n"<< "Count: "<< counter << "\nqueCount: " << queCount << "\nqueElemSum: " << queSum<< "\navgQue: " << avgQue << "\ndownTime: "<< 500 - downTime << endl;

\_getch();

return 0;

}

double erlangDist()

{

double num = 0;

double lambda = 0.25;

int l = 3;

for (int i = 0; i < l; i++)

{

num += exponentialDist(lambda);

}

return num;

}

double poissonDist()

{

double lambda = 0.5;

return exponentialDist(lambda);

}

double normalDist()

{

double mean = 12.0;

double sigma = 2.0;

const double a = (8 \* (M\_PI - 3)) / (3 \* M\_PI \* (4 - M\_PI));

double x = 2 \* randomize() - 1;

double sign = -std::signbit(x) \* 2 + 1;

double errF\_i = sign \* sqrt(sqrt(pow(2 / (M\_PI \* a) + log(1 - x \* x) / 2, 2) - log(1 - x \* x) / a) -

(2 / (M\_PI \* a) + log(1 - x \* x) / 2));

return mean + sigma \* (sqrt(2) \* errF\_i);

}

double exponentialDist(double lambda)

{

double num = (-1.0 / lambda)\*log(1.0 - randomize());

return num;

}

double randomize()

{

return ((double)(rand()) / (double)RAND\_MAX);

}

void show()

{

cout << setw(5) << "|TT: " << setw(8) << timeTact << left << " | L1: " << setw(8) << l1 << left << " | L2: "

<< setw(8) << l2 << left << " | h: " << setw(8) << h << left << " | S: " << server << " |Q: " << que;

cout << " Que: ";

if (que == 0)

cout << "N ";

for (int i = 0; i < que; i++)

cout << queArr[i] << " ";

cout << endl;

}

unsigned int PseudoRand1() {

return pRnd\_1 = (1103515245 \* pRnd\_1 + 12345) % LECURE\_RAND\_MAX;

}

unsigned int PseudoRand2() {

return pRnd\_2 = (16807 \* pRnd\_2 + 0) % LECURE\_RAND\_MAX;

}

unsigned int LecureRand() {

return (PseudoRand1() - PseudoRand2()) % LECURE\_RAND\_MAX;

}