Министерство образования Республики Беларусь

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

**Кафедра**

Никончик Даниил Викторович

Отчет по лабораторным работам по курсу

“Математическое моделирование”

студента 2 курса 13 группы

|  |  |
| --- | --- |
| Работа сдана 2021г. | **Преподаватель** |
| зачтена \_\_\_\_\_\_\_\_\_\_ 2021 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись преподавателя) | *Лобач Сергей Викторович*  ассистент кафедры ММАД |
|  |  |

*Минск 2010*  
  
  
  
  
 **Лабораторная работа 1.**

**Условие:**

Смоделировать БСВ (выборку из 100 элементов) мультипликативным конгруэнтным методом и методом Маклорена-Марсальи и реализовать проверку методами Колмогорова и Пирсона.

**Теория:**

**Мультипликативный конгруэнтный метод:**

Псевдослучайная последовательность  строится по следующим рекуррентным формулам:

  

где  - параметры датчика:  - множитель (*<M*), *M* – модуль,  - стартовое значение (нечетное число).

В данной работе брались значения: *M*=2147483648, ==65539.

**Метод Маклорена-Марсальи:**

Пусть  - псевдослучайные последовательности, порожденные независимо работающими датчиками;  - результирующая псевдослучайная последовательность реализация БСВ;

*V={V(0), V(1), …,V(K-1)}* – вспомогательная таблица *K* чисел.

Процесс вычисления  включает следующие этапы:

- первоначальное заполнение таблицы

*V*: 

- случайный выбор из таблицы:



-обновление табличных значений:

.

В данной работе в качестве  бралась последовательность (из 100 элементов), полученная мультипликативным конгруэнтным методом, описанным выше. В качестве , бралась последовательности (из 10000) элементов, полученная аналогичным способом с тем же M и . *K*=100.

** - критерий согласия Пирсона:**

Область возможных значений случайной величины разбивается на интервалы .

Рассматривается следующая статистика,

,

*n* – объем выборки,

 - количество элементов выборки, попавших в *k*-ый интервал,

 - вероятность попадания случайной величины в *k*-ый интервал.

Проверяется условие , где , *G* функция распределения распределения**,**  - уровень значимости (обычно =0.05).

В данной работе отрезок [0;1] разбивался на 10 интервалов.

**Критерий согласия Колмогорова:**

Рассматривается статистика:



где

,



Проверяется условие , где , *K* - функция распределенияраспределения Колмогорова**,**  - уровень значимости.

**Код программы:**

import java.util.Arrays;  
  
public class Lab1 {  
 public static void main(String[] args) {  
 double[] firstSeq = new double[*NumImplements*];  
 double[] secondSeq = new double[*K*];  
 double[] thirdSeq = new double[*NumImplements*];  
 *MultiMethod*(firstSeq, *A0*, *NumImplements*);  
 *MultiMethod*(secondSeq, 2 \* *Beta* + 1, *K*);  
 *MethodMacLarenMarsaglia*(thirdSeq, secondSeq, firstSeq);  
 *TestPirson*(firstSeq, 10);  
 *TestPirson*(thirdSeq, 5);  
 *TestKolmogorov*(firstSeq);  
 *TestKolmogorov*(thirdSeq);  
 }  
  
 public static long *A0* = 65643;  
 public static long *Beta* = *A0*;  
 public static long *M* = 2147483648L;  
 public static int *NumImplements* = 1000;  
 public static int *K* = 256;  
 public static double *CriticalNum* = 16.91898;  
 public static double *CriticalNumD* = 1.63;  
  
 public static void MultiMethod(double[] a, long beta, int n) {  
 double[] aWithStar = new double[n];  
 aWithStar[0] = beta \* beta % *M*;  
 a[0] = aWithStar[0] / *M*;  
  
 for (int i = 1; i < n; i++) {  
 aWithStar[i] = aWithStar[i - 1] \* *Beta* % *M*;  
 a[i] = aWithStar[i] / *M*;  
 }  
 }  
  
 public static void MethodMacLarenMarsaglia(double[] a, double[] b, double[] c) {  
 double[] temp = Arrays.*copyOf*(b, *K*);  
  
 for (int i = 0; i < *NumImplements*; i++) {  
 int s = ((int) (c[i] \* *K*)) % *K*;  
 a[i] = temp[s];  
 temp[s] = c[(i + *K*) % *K*];  
 }  
  
 }  
  
 public static void TestPirson(double[] a, int L) { *// по формуле* Arrays.*sort*(a);  
 double xi = 0;  
 var i = 0;  
  
 for (int j = 1; j <= L; j++) {  
 var count = 0;  
 while ((i < *NumImplements*) && (a[i] < (double) j / L)) {  
 i++;  
 count++;   
 }  
 xi += Math.*pow*(count - (double) (*NumImplements* / L), 2);   
 }  
  
 System.*out*.println("xi^2 = " + xi / (double) (*NumImplements* / L) + " | critical number: " + *CriticalNum*);  
 }  
  
 public static void TestKolmogorov(double[] a) {   
 Arrays.*sort*(a);  
 double D = 0;  
  
 for (int i = 0; i < *NumImplements*; i++) {  
 D = Math.*max*(D, Math.*abs*(((double) i + 1) / *NumImplements*) - a[i]);  
 }  
  
 System.*out*.println("D = " + D \* Math.*sqrt*(1000) + " | critical number: " + *CriticalNumD*);  
 }  
}

**Результаты:**

Мультипликативный конгруэнтный метод:



****

Метод Маклорена-Марсальи:



**Лабораторная работа 2.**

**Условие:**

Смоделировать распределение Пуассона и реализовать проверку методом Пирсона.

**Теория:**

**Распределение Пуассона (с параметром ):**

Случайная величина  принимает только целые неотрицательные значения, причем 

В данной работе, сначала моделировалась последовательность БСВ, а потом по каждой БСВ строился соответствующий элемент выборки распределения Пуассона: отрезок [0;1] разбивался на интервалы длин  проверялось, в какой интервал попадает элемент последовательности БСВ.

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

namespace lab2

{

public class DRV

{

private static int NumberImplementations { get; set; }

internal DRV(int n)

{

NumberImplementations = n;

}

public static List<int> ModelingGeometric(params double[] param)

{

var p = param[0];

var sequence = new List<int>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

sequence.Add( (int) (Math.Log(rnd.NextDouble(), 1 - p) + 1));

}

return sequence;

}

public static List<int> ModelingBernoulli(params double[] param)

{

var p = param[0];

var sequence = new List<int>();

var rnd = new Random();

int st = 0;

for (var i = 0; i < NumberImplementations; i++)

{

sequence.Add(rnd.NextDouble() < p ? 1 : 0);

st++;

}

Console.WriteLine("1", st);

return sequence;

}

public static List<int> ModelingBinomial(params double[] param)

{

var m = param[0];

var p = param[1];

double x = 0;

var sequence = new int[NumberImplementations];

var rnd = new Random();

for(var j = 0; j < NumberImplementations; j++)

{

for(var i = 0; i < m; i++)

{

if (rnd.NextDouble() < p)

{

x++;

}

}

sequence[j] = (int) x;

x = 0;

}

return new List<int>(sequence);

}

public static List<int> ModelingNegBinomial(params double[] param)

{

var r = param[0];

var p = param[1];

double rem = r, x = 0;

var sequence = new int[NumberImplementations];

var rnd = new Random();

for(var j = 0; j < NumberImplementations; j++)

{

while (rem != 0)

{

if (rnd.NextDouble() > p)

{

x++;

}

else

{

rem--;

}

}

sequence[j] = (int) x;

rem = r;

x = 0;

}

return new List<int>(sequence);

}

}

}

Результыт:



**Лабораторная работа 3.**

**Условие:**

Смоделировать распределение Вейбулла-Гнеденко и реализовать проверку методом Колмогорова.

**Теория:**

**Распределение Вейбулла-Гнеденко (с параметрами** *k,* **):**

Распределение имеет плотность



Распределение может быть смоделировано методом обратной функции:

,

*U* – БСВ.

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using Extreme.Mathematics.Calculus;

namespace lab2

{

public class Crv

{

private int NumberImplementations { get; set; }

public static int NumberSection{ get; set; }

internal Crv(int n, int s)

{

NumberImplementations = n;

NumberSection = s;

}

public List<double> ModelingNormal(params double[] param)

{

var p1 = param[0];

var p2 = param[1];

var sequence = new List<double>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

double sum = -6;

for (int j = 0; j < 12; j++)

{

sum += rnd.NextDouble();

}

sequence.Add(p1 + sum \* Math.Sqrt(p2));

}

return sequence;

}

public List<double> ModelingLogNormal(params double[] param)

{

var p1 = param[0];

var p2 = param[1];

var sequence = new List<double>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

double sum = -6;

for (int j = 0; j < 12; j++)

{

sum += rnd.NextDouble();

}

sequence.Add(Math.Exp(p1 + sum \* Math.Sqrt(p2)));

}

return sequence;

}

public List<double> ModelingLogistics(params double[] param)

{

var p1 = param[0];

var p2 = param[1];

var sequence = new List<double>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

var y = rnd.NextDouble();

sequence.Add(p1 + p2 \* Math.Log(y / (1 - y)));

}

return sequence;

}

public List<double> ModelingLaplace(params double[] param)

{

var p = param[0];

var sequence = new List<double>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

var y = rnd.NextDouble();

if (y < 0.5)

sequence.Add(1 / p \* Math.Log(2 \* y));

else

sequence.Add(-1 / p \* Math.Log(2 \* (1 - y)));

}

return sequence;

}

public List<double> ModelingExponential(params double[] param)

{

var p = param[0];

var sequence = new List<double>();

var rnd = new Random();

for (var i = 0; i < NumberImplementations; i++)

{

var y = rnd.NextDouble();

sequence.Add(-Math.Log(y) / p);

}

return sequence;

}

public double NormalFunc(double value, params double[] param)

{

var m = param[0];

var s2 = param[1];

AdaptiveIntegrator integrator = new AdaptiveIntegrator();

Func<double, double> f1 = (x) => Math.Exp(-Math.Pow(x - m, 2) / 2 / s2);

integrator.Integrate(f1, -1000, value);

return integrator.Result / Math.Sqrt(2 \* Math.PI \* s2);

}

public double LogNormalFunc(double value, params double[] param)

{

var m = param[0];

var s2 = param[0];

AdaptiveIntegrator integrator = new AdaptiveIntegrator();

Func<double, double> f1 = (x) =>Math.Exp(-Math.Pow(Math.Log(x) - m, 2)/2/s2);

integrator.Integrate(f1, -1000, value);

return integrator.Result / Math.Sqrt(2 \* Math.PI \* s2);

}

public double LogisticsFunc(double value, params double[] param)

{

var p1 = param[0];

var p2 = param[1];

return 1 / (1 + Math.Exp(-(value - p1) / p2));

}

public double ExpFunc(double value, params double[] param)

{

var p = param[0];

return 1 - Math.Exp(-p \* value);

}

public double LaplaceFunc(double value, params double[] param)

{

var p = param[0];

if (value < 0)

return 0.5 \* Math.Exp(p \* value);

return 1 - 0.5 \* Math.Exp(-p \* value);

}

}

**Результат:**



**Лабораторная работа 4.1.**

**Условие:**

Вычислить интеграл методом Монте-Карло:



**Теория:**

**Метод Монте-Карло приближенного вычисления интеграла:**

Необходимо вычислить .

Пусть  - произвольная случайная величина с плотностью распределения  имеющая конечный момент второго порядка.

Пусть  Тогда 

В качестве приближенного значения *a* можно взять



В данной работе в качестве  бралась случайная величина, равномерно распределенная на [0;1].

**Код программы:**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int n=100;

double\* x=new double[n];

double\* a=new double[n];

int i,j;

double lj;

double sum=0;

for(i=0;i<n;i++)

{

x[i]=rand()/32767.0;

j=0;

a[i]=sin(x[i])/x[i];

sum+=a[i];

cout<<a[i]<<endl;

}

cout<<endl<<endl;

cout<<(sum/n);

cin>>i;

delete[] a;

delete[] x;

return 1;

}

Результат:



**Лабораторная работа 4.2.**

**Условие:**

Решить систему линейных алгебраических уравнений методом Монте-Карло:



**Теория:**

**Метод Монте-Карло приближенного решения системы линейных алгебраических уравнений:**

Необходимо решить систему, представленную в виде , где , собственные значения *A* по модулю меньше 1.

Наша цель – вычислить скалярное произведение вектора решения  с некоторым вектором .

Рассмотрим цепь Маркова с параметрами  такими что





 если 

 если 

Положим



Выберем некоторое натуральное *N* и рассмотрим случайную величину



Где 🡪🡪…🡪 - траекторая цепи Маркова.

*Qm* опряделяется как:



Тогда скалярное произведение вектором *h* и *x* приблизительно равно .

Можем найти *x*, скалярно умножая его на векторы *h* у которых в одной позиции стоит 1, а в остьльных – 0.

В данной работе выбиралось 

**Код программы:**

package Lab4\_2;  
  
import java.util.Random;  
  
public class Lab4\_2 {  
 public static void main(String[] args) {  
 double exactValue = 3.21825;  
 Random rnd = new Random();  
 var res = 0.0;  
 var n = 1000000;  
 for (var i = 0; i < n; i++) {  
 var x = 0.0;  
 var y = 0.0;  
 while(true) {  
 x = rnd.nextDouble() \* 2 \* Math.*sqrt*(3) - Math.*sqrt*(3);  
 y = rnd.nextDouble() \* 2 \* Math.*sqrt*(3) - Math.*sqrt*(3);  
 var value = Math.*pow*(x, 2) + Math.*pow*(y, 2);  
 if (value >= 1 && value < 3) {  
 break;  
 }  
 }  
 res += 2 \* Math.*PI* / (Math.*pow*(x, 2) + Math.*pow*(y, 4));  
 }  
 System.*out*.println("Monte-Carlo: " + res /n);  
 System.*out*.println("Exact Value: " + exactValue);  
 }  
}

**Результат:**

****

**Лабораторная работа 5.**

**Условие:**

Решить систему линейных уравнений, используя метод Монте-Карло.

1. Решить систему линейных алгебраических уравнений  методом Монте-Карло.
2. Сравнить с решением данного уравнения, полученным в произвольном математическом пакете.
3. Построить график зависимости точности решения от длины цепи Маркова и числа смоделированных цепей Маркова.



**Теория:**

**Метод Монте-Карло:**

Процесс описывается математической моделью с использованием генератора случайных величин, модель многократно обсчитывается, на основе полученных данных вычисляются вероятностные характеристики рассматриваемого процесса. Например, чтобы узнать методом Монте-Карло, какое в среднем будет расстояние между двумя случайными точками в круге, нужно взять координаты большого числа случайных пар точек в границах заданной окружности, для каждой пары вычислить расстояние, а потом для них посчитать среднее арифметическое.

**Код программы:**

#include <iostream>

#include <fstream>

#include <time.h>

using namespace std;

double X\_Answer[] = { 1.83801, 2.33645, -0.903427 };

double sqr(double x){

return x\*x;

}

int getMarkovRand(double\*\* P, int curState, int sz) {

double rd = (double)rand() / RAND\_MAX;

for (int i = 0; i < sz; ++i) {

rd -= P[curState][i];

if (rd <= 0)

return i;

}

return sz - 1;

}

double calculate\_coord(double\*\* a, int coord, int n, double\* f, double\*\* P, int MARKOV\_LEN, int MARKOV\_CNT) {

int M\_Prev, M;

double Q\_Prev, Q;

double X = 0;

for (int i = 0; i < MARKOV\_CNT; ++i) {

M\_Prev = coord;

Q\_Prev = 1;

for (int j = 1; j < MARKOV\_LEN; ++j)

{

M = getMarkovRand(P, M\_Prev, n);

if (P[M\_Prev][M]>0)

Q = Q\_Prev \* a[M\_Prev][M] / P[M\_Prev][M];

else

Q = 0;

X += Q \* f[M];

Q\_Prev = Q;

M\_Prev = M;

}

}

return (f[coord] + X / MARKOV\_CNT);

}

void cmpAnswers(double\* X, double\* RealX, int n) {

cout << "Calculated: (" << X[0];

for (int i = 1; i < n; ++i)

cout << ", " << X[i];

cout << ")\nReal answer: (" << RealX[0];

for (int i = 1; i < n; ++i)

cout << ", " << RealX[i];

double diff = 0;

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

diff += sqr(X[i] - RealX[i]);

diff = sqrt(diff);

cout << ")\nDiff: " << diff << endl;

}

void main() {

srand(time(0));

ifstream fin("matrix.txt");

int n;

fin >> n;

double\*\* a = new double\*[n];

double\*\* h = new double\*[n];

double\* f = new double[n];

double\* X = new double[n];

double\*\* P = new double\*[n];

for (int i = 0; i < n; ++i) { //заполняю матрицу переходов цепи

a[i] = new double[n];

h[i] = new double[n];

P[i] = new double[n];

for (int j = 0; j < n; ++j) {

fin >> a[i][j];

h[i][j] = 0;

P[i][j] = 1. / n;

}

fin >> f[i];

for (int j = 0; j < n; ++j)

if (i == j) {

a[i][j] = 1 - a[i][i];

h[i][j] = 1;

}

else

a[i][j] \*= -1;

}

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

X[i] = calculate\_coord(a, i, n, f, P, 50, 10000);

cmpAnswers(X, X\_Answer, n);

system("pause");

//Graphics begin

ofstream fD("D.txt");

ofstream fLen("Len.txt");

ofstream fCnt("Cnt.txt");

int max\_cnt = 1000;

for (int cnt = 10; cnt < max\_cnt; cnt += 200) {

system("cls");

cout << "Generating graphics: " << (int)(sqr((double)cnt / max\_cnt)\*100) << "%.";

for (int len = 2; len < 30; len += 1) {

double diff = 0;

for (int rnd = 0; rnd < 10; ++rnd) {

double cur\_diff = 0;

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

cur\_diff += sqr(calculate\_coord(a, i, n, f, P, len, cnt) - X\_Answer[i]);

diff += sqrt(cur\_diff);

}

fLen << len << endl;

fCnt << cnt << endl;

fD << diff/10 << endl;

}

}

fD.close();

fLen.close();

fCnt.close();

system("python print.py");

//Graphics end;

}

**Результат:**



**Литература**

1. Харин Ю.С., Малюгин В.И., Кирлица В.П., Лобач В.И., Хацкевич Г.А. Основы имитационного и статистического моделирования. Учебное пособие. Минск: ДизайнПРО, 1997 – 228 с.
2. Лобач В.И., Кирлица В.П., Малюгин В.И., Сталевская С.Н. Имитационное и статистическое моделирование. Практикум для студентов математических и экономических специальностей. Минск, БГУ, 2004 –189 с.