g***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №3**

**По дисциплине КМЗИ за IV семестр**

**Тема: «Генерация случайных числе»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Хацкевич М. В.

Брест 2019

Цель работы: изучить методы генерации случайных чисел.

Ход работы

Реализуем генератор LFSR, и проанализируем результаты его работы, а для этого:

* Реализуем сам генератор
* Реализуем визуализация результатов

Листинги:

1. Генератора

#include "pch.h"

#include <iostream>

#include <exception>

#include <iterator>

#include <vector>

#include <set>

#include <utility>

#include <boost/random.hpp>

#include <boost/program\_options.hpp>

#include <functional>

#include <fstream>

namespace po = boost::program\_options;

/\*\* Main class of this project \*\*/

class LFSR

{

public:

LFSR(int nSize = 8);

void SetA(std::initializer\_list<bool> ilA);

void SetS(std::initializer\_list<unsigned int> ilS);

void SetS(std::function<unsigned int()> fnGenerator);

void SetMin(int nMin);

void SetMax(int nMax);

void Forward();

// At the beginning we have

// initializaton values

// If we want to generate different values at the front

// we have to go forward n times where n is size

// of our machine

void Init();

void Write(std::ostream& stream) const;

unsigned int Front() const;

private:

std::vector<unsigned int> m\_vecS;

std::vector<bool> m\_vecA;

int m\_nMin, m\_nMax;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[])

{

// parse command line

po::options\_description desc("Options");

desc.add\_options()

("help,h", "print help")

("from,f", po::value<int>(), "lower border (unsinged int)")

("to,t", po::value<int>(), "higher border (unsinged int)")

("output,0", po::value<std::string>(), "path to file for logging")

("number,n", po::value<int>(), "number of elements");

po::variables\_map vm;

po::store(po::parse\_command\_line(argc, argv, desc), vm);

po::notify(vm);

// variables

unsigned int nMinBorder, nMaxBorder, n;

std::string sFilename;

if (vm.count("help"))

{

std::cout << desc << std::endl;

getchar();

return 0;

}

if (vm.count("from"))

nMinBorder = vm["from"].as<int>();

else nMinBorder = 0;

if (vm.count("to"))

{

nMaxBorder = vm["to"].as<int>();

if (nMaxBorder < nMinBorder)

{

std::cout << "Error! Lower border must be fewer than upper border";

return -1;

}

}

else nMaxBorder = nMinBorder + 100;

if (vm.count("output"))

sFilename = vm["output"].as<std::string>();

if (vm.count("number"))

n = vm["number"].as<int>();

else n = 1;

///////////////////////

// Dealing with LFSR //

///////////////////////

LFSR lfsr(6); // agrument is size

// borders

lfsr.SetMin(nMinBorder);

lfsr.SetMax(nMaxBorder);

// mask

lfsr.SetA({1, 0, 0, 1, 0, 0, 1});

// initial S vector

lfsr.SetS([]() -> unsigned int {

static boost::random::mt19937 gen; // yes, I use random generator to produce random generator

return static\_cast<unsigned int>(gen());

});

// forward n times

// to get rid of initial values

lfsr.Init();

// do we save results into file?

bool fToFile = sFilename.length();

// stream for our output file

std::ofstream fout;

if (fToFile)

{

fout.open(sFilename, std::ios::out | std::ios::trunc);

if (!fout)

{

std::cout << "File cannot be opened!" << std::endl;

fToFile = false;

}

}

// Here we are

// Here is where magic happens

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

{

lfsr.Forward();

if (fToFile)

fout << lfsr.Front() << std::endl;

else

std::cout << lfsr.Front() << std::endl;

}

std::cout << "All done!";

fout.close();

if (fToFile) // show chart of mean value

system(std::string("python chart.py " + sFilename).c\_str());

getchar();

}

/\*\* Class methods definitions \*\*/

LFSR::LFSR(int nSize)

: m\_vecS(nSize), m\_vecA(nSize), m\_nMin(0), m\_nMax(100)

{}

void LFSR::SetA(std::initializer\_list<bool> ilA)

{

int i = 0;

for (bool a : ilA)

{

if (i >= m\_vecA.size()) break;

m\_vecA[i] = a;

i++;

}

}

void LFSR::SetS(std::initializer\_list<unsigned int> ilS)

{

int i = 0;

for (unsigned int S : ilS)

{

if (i >= m\_vecS.size()) break;

m\_vecS[i] = S;

i++;

}

}

void LFSR::SetS(std::function<unsigned int()> fnGenerator)

{

for (unsigned int& S : m\_vecS)

S = fnGenerator();

}

void LFSR::SetMin(int nMin)

{

m\_nMin = nMin;

}

void LFSR::SetMax(int nMax)

{

m\_nMax = nMax;

}

void LFSR::Forward()

{

for (int i = 0; i < m\_vecS.size(); i++)

{

if (i == m\_vecS.size() - 1)

{

m\_vecS[i] = 0;

for (int j = 0; j < m\_vecS.size() - 1; j++)

m\_vecS[i] += m\_vecA[j] \* m\_vecS[j];

}

else

{

m\_vecS[i] = m\_vecS[i + 1];

}

m\_vecS[i] = m\_vecS[i] % (m\_nMax - m\_nMin) + m\_nMin;

}

}

void LFSR::Init()

{

for (int i = 0; i < m\_vecA.size(); i++)

Forward();

}

void LFSR::Write(std::ostream& stream) const

{

std::ostream\_iterator<int> it(stream, " ");

std::copy(m\_vecS.begin(), m\_vecS.end(), it);

std::endl(stream);

}

unsigned int LFSR::Front() const

{

return m\_vecS.front();

}

1. Визуализации

import matplotlib.pyplot as plt

import sys

if len(sys.argv) != 2:

print("Something went wrong while chart creation. (Bad arguments)")

exit(0)

file = open(sys.argv[1], 'r')

values = []

for line in file:

values.append(int(line))

X = [x for x in range(len(values))]

Y = [sum(values[:i])/i for i in range(1, len(X) + 1)]

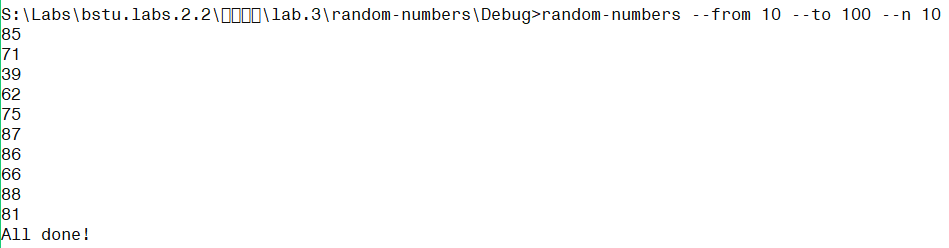
plt.plot(X, Y)

plt.title('Average')

plt.show()

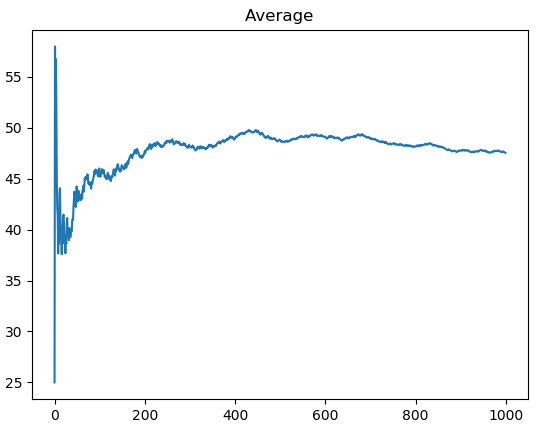
Результаты работы:

1. Вывод генератора



1. Анализ среднего значения для генератора

C:\Users\Ilya\AppData\Local\Microsoft\Windows\INetCache\Content.Word\arguments.png



Насколько видим значения, выдаваемые генератором в среднем, стремятся к значению 50, что говорит о сравнительно хорошем показателе равномерности распределения случайной величины.

Вывод: по ходу лабораторной работы научился производить генерацию случайных чисел с помощью генератора LFSR.