Учреждение образования

Белорусский государственный университет информатики

и радиоэлектроники

Факультет компьютерных технологий ИИТ

Лабораторная работа №4

Моделирование

**Разработка, отладка и исследование программной модели непрерывно-стохастической СМО**

Выполнил: студенты гр.280512

Богданов Н.В.

Шойхет Ю.М.

Проверил: преподаватель Лашкевич Е.М.

Минск, 2015

**Задание**

Построить имитационную модель непрерывно-стохастической СМО и исследовать ее (разработать алгоритм и написать имитирующую программу, предусматривающую сбор и статистическую обработку данных для получения оценок заданных характеристик СМО)

1) Исходные данные

***Вариант 2.***

СМО с отказами состоит из двух последовательных фаз.

Количество мест ожидания первой фазы n1, второй – n2. интенсивность обработки заявок каждого канала µ1 = µ2 = 5.



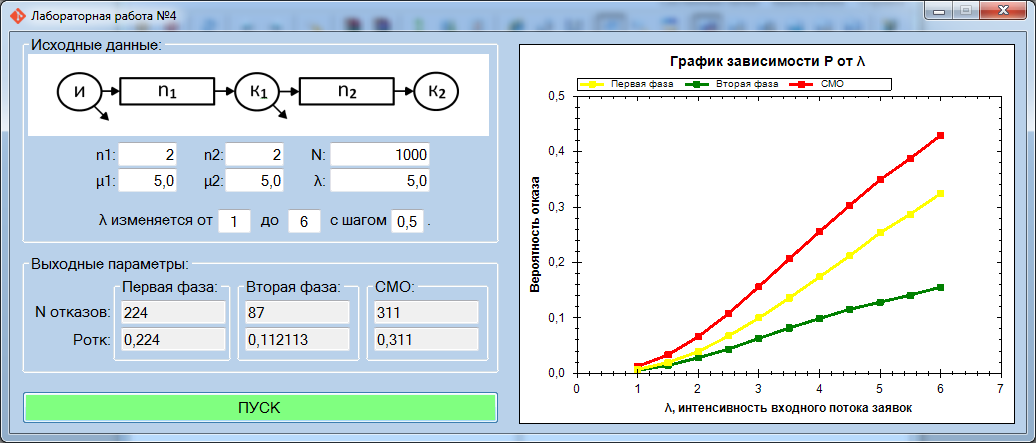
Построить зависимости Ротк, Ротк1, Ротк2 при изменении λ от 1 до 6 с шагом 0.5. Входной поток и потоки обслуживаний - простейшие, n1=2, n2=2.

2) Блок схема

|  |  |  |
| --- | --- | --- |
| kот -denialCount t - currentTime T - requestStream[RequestsNumber] N - RequestsNumber - общее кол-во заявок. |  |  |

outputStream = RequestsNumber - кол-во отказов первой фазы

3) Скиншот программы



4) Код программы

Описание класса фаза

using System;

using System.Collections.Generic;

using System.Text;

namespace lab4

{

class Phase

{

private readonly double \_μ;

private readonly int \_maxQueueCount;

public int DenialCount;

public Phase(double serviceRate, int maxQueueCount){

\_μ = serviceRate;

\_maxQueueCount = maxQueueCount;

}

public List<double> Imitate(List<double> inputStream){

Random rnd = new Random(MyRandom.Next() ^ Environment.TickCount);

double currentTime = 0;

int queueCount = 0;

int denialCount = 0;

List<double> outputStream = new List<double>();

int currentRequest = 0;

while (currentRequest < inputStream.Count){

if (inputStream[currentRequest] < currentTime){

if (queueCount == \_maxQueueCount)

denialCount++; //добовляется количество отказов

else queueCount++; //добовляется количество заявок в очередь

currentRequest++;

}

else{

if (queueCount > 0){

currentTime += -Math.Log(rnd.NextDouble())/\_μ;

queueCount--;

}

else currentTime = inputStream[currentRequest++] + -Math.Log(rnd.NextDouble())/\_μ;

outputStream.Add(currentTime);

}

}

DenialCount = denialCount;

return outputStream;

}

}

}

Обработка результатов

using System;

using System.Collections.Generic;

namespace lab4

{

class QueuingSystem

{

private readonly double \_λ;

private Phase \_phase1, \_phase2;

public int DenialCount1 { get { return \_phase1.DenialCount; } }

public int DenialCount2 { get { return \_phase2.DenialCount; } }

public int DenialCount { get { return DenialCount1 + DenialCount2; } }

public int RequestsNumber;

public int RequestsNumber2;

public double DenialProbability1 { get { return (double)\_phase1.DenialCount / RequestsNumber; } }

public double DenialProbability2 { get { return (double)\_phase2.DenialCount / RequestsNumber2; } }

public double DenialProbability { get { return (double)(\_phase1.DenialCount + \_phase2.DenialCount) / RequestsNumber; } }

public QueuingSystem(double λ, double μ1, double μ2, int maxQueueCount1, int maxQueueCount2){

\_phase1 = new Phase(μ1, maxQueueCount1);

\_phase2 = new Phase(μ2, maxQueueCount2);

\_λ = λ;

}

public void Imitate(int requestsNumber = 100000){

RequestsNumber = requestsNumber;

Random rnd = new Random(MyRandom.Next() ^ Environment.TickCount);

List<double> requestStream = new List<double>();

double currentTime = 0;

requestStream.Add(0);

for (int i = 1; i < requestsNumber; i++){

currentTime += -Math.Log(rnd.NextDouble()) / \_λ;

requestStream.Add(currentTime);

}

List<double> requestStream2 = \_phase1.Imitate(requestStream); /\*Первая фаза\*/

\_phase2.Imitate(requestStream2); /\*Вторая фаза\*/

RequestsNumber2 = requestStream2.Count;

}

}

}

RandomProvider.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace lab4{

public static class MyRandom{

private static readonly Random Rnd = new Random();

public static int Next() { return Rnd.Next(); }

public static int Next(int max) { return Rnd.Next(max); }

public static int Next(int min, int max) { return Rnd.Next(min, max); }

}

}

Form

using System;

using System.Drawing;

using System.Globalization;

using System.Windows.Forms;

using ZedGraph;

namespace lab4{

public partial class Form1 : Form{

public Form1(){

InitializeComponent();

}

private void Imitate(){

int requestsNumber = int.Parse(textBox\_number.Text);

double λ = double.Parse(textBox\_intensity.Text);

double μ1 = double.Parse(textBox\_ch1.Text);

double μ2 = double.Parse(textBox\_ch2.Text);

int maxQueueCount1 = int.Parse(textBox\_queue1.Text);

int maxQueueCount2 = int.Parse(textBox\_queue2.Text);

QueuingSystem qs = new QueuingSystem(λ, μ1, μ2, maxQueueCount1, maxQueueCount2);

qs.Imitate(requestsNumber);

tbdenialCount1.Text = qs.DenialCount1.ToString();

textBox\_denial1.Text = Math.Round(qs.DenialProbability1, 6).ToString();

tbdenialCount2.Text = qs.DenialCount2.ToString();

textBox\_denial2.Text = Math.Round(qs.DenialProbability2, 6).ToString();

tbdenialCount.Text = qs.DenialCount.ToString();

textBox\_denial.Text = Math.Round(qs.DenialProbability, 6).ToString();

}

private void DrawGraph(){

zedGraphControl.Visible = true;

double μ1 = double.Parse(textBox\_ch1.Text);

double μ2 = double.Parse(textBox\_ch2.Text);

int maxQueueCount1 = int.Parse(textBox\_queue1.Text);

int maxQueueCount2 = int.Parse(textBox\_queue2.Text);

double λmin = double.Parse(textBox\_λmin.Text);

double λmax = double.Parse(textBox\_λmax.Text);

double step = double.Parse(textBox\_step.Text);

PointPairList denialProbabilityPoints = new PointPairList();

PointPairList denialProbability1Points = new PointPairList();

PointPairList denialProbability2Points = new PointPairList();

for (double λ = λmin; λ <= λmax; λ += step)

{

QueuingSystem qs = new QueuingSystem(λ, μ1, μ2, maxQueueCount1, maxQueueCount2);

qs.Imitate();

denialProbabilityPoints.Add(λ, qs.DenialProbability);

denialProbability1Points.Add(λ, qs.DenialProbability1);

denialProbability2Points.Add(λ, qs.DenialProbability2);

}

GraphPane pane = zedGraphControl.GraphPane;

pane.XAxis.Title.Text = "λ, интенсивность входного потока заявок";

pane.YAxis.Title.Text = "Вероятность отказа";

pane.Title.Text = "График зависимости P от λ";

pane.CurveList.Clear();

LineItem denial1Curve = pane.AddCurve("Первая фаза", denialProbability1Points, Color.Yellow, SymbolType.Default);

LineItem denial2Curve = pane.AddCurve("Вторая фаза", denialProbability2Points, Color.Green, SymbolType.Default);

LineItem denialCurve = pane.AddCurve("СМО", denialProbabilityPoints, Color.Red, SymbolType.Default);

denialCurve.Line.Width = 3f;

denialCurve.Symbol.Fill = new Fill(Color.Red);

denial1Curve.Line.Width = 3f;

denial1Curve.Symbol.Fill = new Fill(Color.Yellow);

denial2Curve.Line.Width = 3f;

denial2Curve.Symbol.Fill = new Fill(Color.Green);

zedGraphControl.AxisChange();

zedGraphControl.Invalidate();

}

private void buttonSimulate\_Click(object sender, EventArgs e){

Imitate();

DrawGraph();

}

private void textBox\_denial1\_TextChanged(object sender, EventArgs e)

{

}

}

}