**ОТЧЕТ**

по лабораторной работе №6

Анализ и интерпретация результатов моделирования. Планирование эксперимента

Выполнил студент группы АСОИ-122

Силков Е.В.

Проверил: Кушнер А.В.

***Цель работы:*** Изучение методики построения регрессионных моделей  
систем на основе теории планирования эксперимента; реализация метода  
построения регрессионной модели системы по результатам имитационного  
эксперимента

***Исходные данные***



***Код программы***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Windows.Threading;

using ZedGraph;

namespace IMS\_01

{

/// <summary>

/// Логика взаимодействия для Lab\_5.xaml

/// </summary>

public partial class Lab\_6 : Page

{

public const int STORAGE\_DEVICE\_LIMIT = 16;

public const double LAMBDA\_FIRST = 0.4;

public const double LAMBDA\_STEP = 0.22;

public const double LAMBDA\_MIDDLE = 7.3;

public const double LAMBDA\_LAST = 14.2;

public const double MU\_FIRST = 0.4 / 3;

public const double MU\_STEP = 0.18;

public const double MU\_MIDDLE = MU\_FIRST + 6 \* MU\_STEP;

public const double MU\_LAST = MU\_FIRST + 12 \* MU\_STEP;

DispatcherTimer requestForming,

requestProcessing1,

requestProcessing2,

requestProcessing3,

timeRunningOut;

int requestsCount, requestsRealCount,

requestsByChannel1, requestsByChannel2, requestsByChannel3;

double lambda = 0.4;

double mu = 0.4 / 3;

double p1, p2;

int i;

double[,] matr;

double[] s;

List<Table1> tab1;

Queue<Request> source, storageDevice, completedRequests, failedRequests;

Request channel1, channel2, channel3;

private void Page\_Loaded(object sender, RoutedEventArgs e)

{

List<Table0> usl = new List<Table0>();

usl.Add(new Table0("Основной", "0", LAMBDA\_MIDDLE, Math.Round(MU\_MIDDLE, 2)));

usl.Add(new Table0("Верхний", "+1", LAMBDA\_LAST, Math.Round(MU\_LAST, 2)));

usl.Add(new Table0("Нижний", "-1", LAMBDA\_FIRST, Math.Round(MU\_FIRST, 2)));

usl.Add(new Table0("Интервал варьирования", "Δ", LAMBDA\_STEP, MU\_STEP));

dataGrid1.ItemsSource = usl;

}

Request someRequest;

public class Request // Запрос

{

public double Dti { get; } // Интервал времени между поступлениями двух соседних заявок

public double Dtki { get; } // Время обслуживания заявки

public Request(double lambda, double mu)

{

Random r = new Random();

Dti = (-1.0 / lambda) \* Math.Log(r.NextDouble());

Dtki = (-1.0 / mu) \* Math.Log(r.NextDouble());

}

}

public class Table1

{

public int Num { get; }

public int X1 { get; }

public int X2 { get; }

public int X1X2 { get; }

public double Y1 { get; }

public double Y2 { get; }

public double Ymid { get; }

public Table1(int num, int x1, int x2, int x1x2, double y1, double y2, double yMid)

{

Num = num;

X1 = x1;

X2 = x2;

X1X2 = x1x2;

Y1 = y1;

Y2 = y2;

Ymid = yMid;

}

}

public class Table0

{

public string S1 { get; }

public string S2 { get; }

public double S3 { get; }

public double S4 { get; }

public Table0(string s1, string s2, double s3, double s4)

{

S1 = s1;

S2 = s2;

S3 = s3;

S4 = s4;

}

}

public Lab\_6()

{

InitializeComponent();

}

private void StartButton\_Click(object sender, RoutedEventArgs e)

{

resultLabel.Content = "";

i = 0;

lambda = LAMBDA\_LAST;

mu = MU\_FIRST;

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

tab1 = new List<Table1>();

matr = new double[4, 6];

s = new double[4];

sourceLabel.Content = 0;

storageLabel.Content = 0;

completedLabel.Content = 0;

failedLabel.Content = 0;

ch1Label.Content = 0;

ch2Label.Content = 0;

ch3Label.Content = 0;

requestsLeftLabel.Content = 0;

lambdaLabel.Content = "λ= " + string.Format("{0:0.00}", lambda);

muLabel.Content = "μ= " + string.Format("{0:0.00}", mu);

requestForming = new DispatcherTimer();

requestProcessing1 = new DispatcherTimer();

requestProcessing2 = new DispatcherTimer();

requestProcessing3 = new DispatcherTimer();

timeRunningOut = new DispatcherTimer();

requestForming.Tick += RequestForming\_Tick;

requestProcessing1.Tick += RequestProcessing1\_Tick;

requestProcessing2.Tick += RequestProcessing2\_Tick;

requestProcessing3.Tick += RequestProcessing3\_Tick;

timeRunningOut.Tick += TimeRunningOut\_Tick;

requestForming.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing1.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing2.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestProcessing3.Interval = new TimeSpan(0, 0, 0, 0, 100);

timeRunningOut.Interval = new TimeSpan(0, 0, 0, 0, 100);

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

source = new Queue<Request>();

storageDevice = new Queue<Request>();

completedRequests = new Queue<Request>();

failedRequests = new Queue<Request>();

try

{

progressBarT.Maximum = Convert.ToDouble(timeTextBox.Text);

requestsCount = Convert.ToInt32(countTextBox.Text);

}

catch

{

int x;

if (!int.TryParse(timeTextBox.Text, out x)) x = 2500;

progressBarT.Maximum = x;

timeTextBox.Text = progressBarT.Maximum.ToString();

if (!int.TryParse(countTextBox.Text, out requestsCount)) requestsCount = 30;

countTextBox.Text = requestsCount.ToString();

}

someRequest = new Request(lambda, mu);

progressBarS.Maximum = someRequest.Dti \* 1000;

Update();

timeRunningOut.Start();

requestForming.Start();

}

private void TimeRunningOut\_Tick(object sender, EventArgs e)

{

progressBarT.Value += 100;

if (progressBarT.Value >= progressBarT.Maximum)

{

progressBarT.Value -= progressBarT.Maximum;

Update();

}

if (completedRequests.Count + failedRequests.Count == Convert.ToInt32(countTextBox.Text))

{

int h = i / 2;

int x1 = h == 0 | (i / 2) == 2 ? +1 : -1;

int x2 = h > 1 ? +1 : -1;

if (i % 2 == 0) p1 = (double)completedRequests.Count / (completedRequests.Count + failedRequests.Count);

else

{

p2 = (double)completedRequests.Count / (completedRequests.Count + failedRequests.Count);

tab1.Add(new Table1(h + 1, x1, x2, x1 \* x2, p1, p2, (p1 + p2) / 2));

matr[h, 0] = x1; matr[h, 1] = x2; matr[h, 2] = x1 \* x2;

matr[h, 3] = p1; matr[h, 4] = p2; matr[h, 5] = (p1 + p2) / 2;

}

switch(i)

{

case 1:

lambda = LAMBDA\_FIRST;

mu = MU\_FIRST;

break;

case 3:

lambda = LAMBDA\_LAST;

mu = MU\_LAST;

break;

case 5:

lambda = LAMBDA\_FIRST;

mu = MU\_LAST;

break;

}

i++;

Initialize();

if (i == 8)

{

double sSum = 0;

double ynSum = 0, yn1Sum = 0, yn2Sum = 0, yn12Sum = 0;

for (int z = 0; z < 4; z++)

{

s[z] = Math.Pow(matr[z, 3] - matr[z, 5], 2) + Math.Pow(matr[z, 3] - matr[z, 5], 2);

sSum += s[z];

ynSum += matr[z, 5];

yn1Sum += matr[z, 0] \* matr[z, 5];

yn2Sum += matr[z, 1] \* matr[z, 5];

yn12Sum += matr[z, 2] \* matr[z, 5];

}

double a0 = ynSum / 4;

double a1 = yn1Sum / 4;

double a2 = yn2Sum / 4;

double a12 = yn12Sum / 4;

double sMax = s[0], sy = 0;

for (int z = 1; z < 4; z++)

{

if (s[z] > sMax) sMax = s[z];

sy += s[0];

}

sy /= 4;

double sa1 = sy / 8;

double t0 = Math.Abs(a0) / sa1;

double t1 = Math.Abs(a1) / sa1;

double t2 = Math.Abs(a2) / sa1;

double t12 = Math.Abs(a12) / sa1;

double daiPos = 0.9057 \* sa1;

double pNew = 0;

if (t0 >= 0.9057) pNew += a0;

if (t1 >= 0.9057) pNew += a1;

if (t2 >= 0.9057) pNew += a2;

if (t12 >= 0.9057) pNew += a12;

resultLabel.Content = resultLabel.Content.ToString() + string.Format("Коэфф. Корхена: {0} <= 0.9057 ({1})\n", sMax / sSum, sMax / sSum <= 0.9057);

resultLabel.Content = resultLabel.Content.ToString() + "a0 = " + a0 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "a1 = " + a1 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "a2 = " + a2 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "a12 = " + a12 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "Дисперсия Sa1^2 = " + sa1 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "t0 = " + t0 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "t1 = " + t1 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "t2 = " + t2 + "\n";

resultLabel.Content = resultLabel.Content.ToString() + "t12 = " + t12 + "\n";

timeRunningOut.Stop();

dataGrid2.ItemsSource = tab1;

MessageBox.Show("Готово!");

}

}

}

private void RequestProcessing3\_Tick(object sender, EventArgs e)

{

progressBar3.Value += 100;

}

private void RequestProcessing2\_Tick(object sender, EventArgs e)

{

progressBar2.Value += 100;

}

private void RequestProcessing1\_Tick(object sender, EventArgs e)

{

progressBar1.Value += 100;

}

private void RequestForming\_Tick(object sender, EventArgs e)

{

if (requestsRealCount + source.Count == Convert.ToInt32(countTextBox.Text))

{

requestForming.Stop();

progressBarS.Value = 0;

return;

}

progressBarS.Value += 100;

if (progressBarS.Value >= progressBarS.Maximum)

{

progressBarS.Value -= progressBarS.Maximum;

source.Enqueue(someRequest);

someRequest = new Request(lambda, mu);

progressBarS.Maximum = someRequest.Dti \* 1000;

requestsLeftLabel.Content = source.Count.ToString();

}

}

public void Update()

{

// переводим заявку из первого канала к завершенным, если такая в нем имеется и она уже обработана

if (channel1 != null && progressBar1.Value >= progressBar1.Maximum)

{

completedRequests.Enqueue(channel1);

progressBar1.Value = 0;

requestProcessing1.Stop();

channel1 = null;

requestsByChannel1++;

channel1Pic.Fill = new SolidColorBrush(Colors.White);

}

// переводим заявку из второго канала к завершенным, если такая в нем имеется и она уже обработана

if (channel2 != null && progressBar2.Value >= progressBar2.Maximum)

{

completedRequests.Enqueue(channel2);

progressBar2.Value = 0;

requestProcessing2.Stop();

channel2 = null;

requestsByChannel2++;

channel2Pic.Fill = new SolidColorBrush(Colors.White);

}

// переводим заявку из третьего канала к завершенным, если такая в нем имеется и она уже обработана

if (channel3 != null && progressBar3.Value >= progressBar3.Maximum)

{

completedRequests.Enqueue(channel3);

progressBar3.Value = 0;

requestProcessing3.Stop();

channel3 = null;

requestsByChannel3++;

channel3Pic.Fill = new SolidColorBrush(Colors.White);

}

// если 1-й канал свободен, пытаемся заполнить его заявкой...

if (channel1 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel1 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar1.Maximum = channel1.Dtki \* 1000;

requestProcessing1.Start();

channel1Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel1 = source.Dequeue();

progressBar1.Maximum = channel1.Dtki \* 1000;

requestProcessing1.Start();

channel1Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// если 2-й канал свободен, пытаемся заполнить его заявкой...

if (channel2 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel2 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar2.Maximum = channel2.Dtki \* 1000;

requestProcessing2.Start();

channel2Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel2 = source.Dequeue();

progressBar2.Maximum = channel2.Dtki \* 1000;

requestProcessing2.Start();

channel2Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// если 3-й канал свободен, пытаемся заполнить его заявкой...

if (channel3 == null)

{

// ... из накопителя

if (storageDevice.Count > 0)

{

channel3 = storageDevice.Dequeue();

if (storageDevice.Count < STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.White);

progressBar3.Maximum = channel3.Dtki \* 1000;

requestProcessing3.Start();

channel3Pic.Fill = new SolidColorBrush(Colors.Yellow);

}

// ... из источника

else if (source.Count > 0)

{

channel3 = source.Dequeue();

progressBar3.Maximum = channel3.Dtki \* 1000;

requestProcessing3.Start();

channel3Pic.Fill = new SolidColorBrush(Colors.Yellow);

requestsRealCount++;

}

}

// Если все каналы заняты, а заявки в источчнике еще имеются, то

// их перенаправляем в накопитель...

while (source.Count > 0)

{

// ... только, если есть место в этом накопителе,

if (storageDevice.Count <= STORAGE\_DEVICE\_LIMIT - source.Count)

{

storageDevice.Enqueue(source.Dequeue());

if (storageDevice.Count == STORAGE\_DEVICE\_LIMIT) storageDevicePic.Fill = new SolidColorBrush(Colors.Orange);

}

// ... а если нет, то оставляем заявку необслуженной

else

{

failedRequests.Enqueue(source.Dequeue());

}

requestsRealCount++;

}

sourceLabel.Content = requestsRealCount.ToString();

storageLabel.Content = storageDevice.Count.ToString();

completedLabel.Content = completedRequests.Count.ToString();

failedLabel.Content = failedRequests.Count.ToString();

ch1Label.Content = requestsByChannel1.ToString();

ch2Label.Content = requestsByChannel2.ToString();

ch3Label.Content = requestsByChannel3.ToString();

requestsLeftLabel.Content = source.Count.ToString();

}

public void Initialize()

{

requestsCount = requestsRealCount =

requestsByChannel1 = requestsByChannel2 = requestsByChannel3 = 0;

sourceLabel.Content = 0;

storageLabel.Content = 0;

completedLabel.Content = 0;

failedLabel.Content = 0;

ch1Label.Content = 0;

ch2Label.Content = 0;

ch3Label.Content = 0;

requestsLeftLabel.Content = 0;

source = new Queue<Request>();

storageDevice = new Queue<Request>();

completedRequests = new Queue<Request>();

failedRequests = new Queue<Request>();

progressBarT.Maximum = Convert.ToDouble(timeTextBox.Text);

requestsCount = Convert.ToInt32(countTextBox.Text);

someRequest = new Request(lambda, mu);

progressBarS.Maximum = someRequest.Dti \* 1000;

Update();

timeRunningOut.Start();

requestForming.Start();

lambdaLabel.Content = "λ= " + string.Format("{0:0.00}", lambda);

muLabel.Content = "μ= " + string.Format("{0:0.00}", mu);

}

}

}

***Результат***







