Белорусский Госдарственный Университет Информатики и Радиоэлектроники

Кафедра ПОИТ

Лабораторная работа №3

«Построение и исследование аналитической модели дискретно -стохастической системы массового обслуживания»

Вариант 21

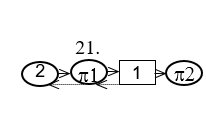
|  |  |
| --- | --- |
| Выполнил:  студент гр.551002 | Романовский Ю. С. |
| Проверил: | Мельник Н.И. |

Минск 2018

1. ЦЕЛЬ РАБОТЫ

Построение аналитической модели дискретно-стохастической системы и расчет параметров данной системы с использованием построенной модели.

1. ИСХОДНЫЕ ДАННЫЕ К РАБОТЕ

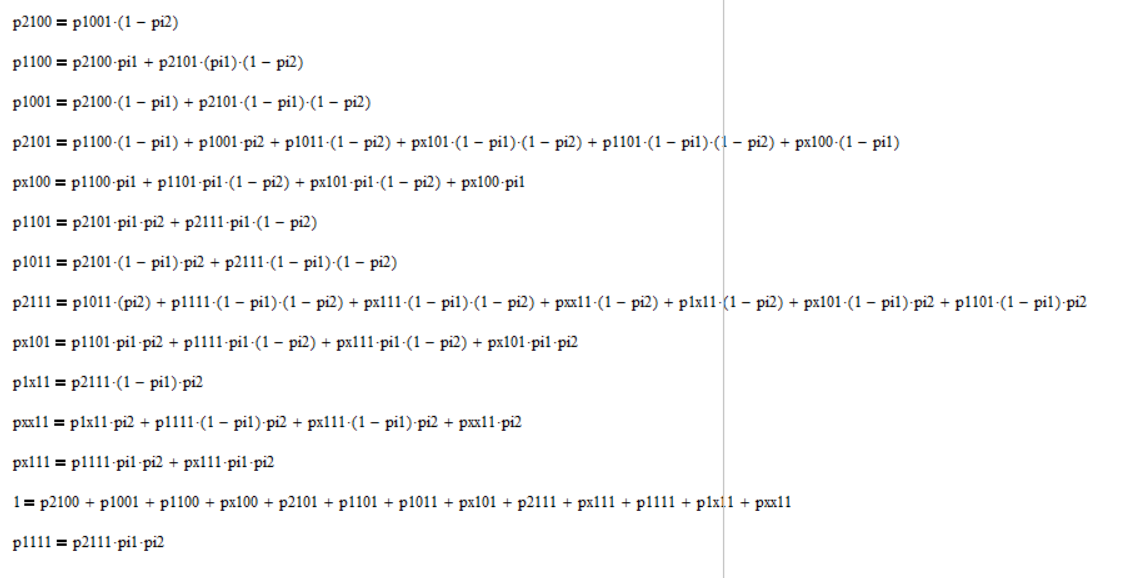


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | ρ | π1 | π2 | Цель исследования |
| 21 | - | 0,4 | 0,5 | А, Lоч , Wс |

1. ХОД РАБОТЫ



На основе графа переходов построим СЛАУ:

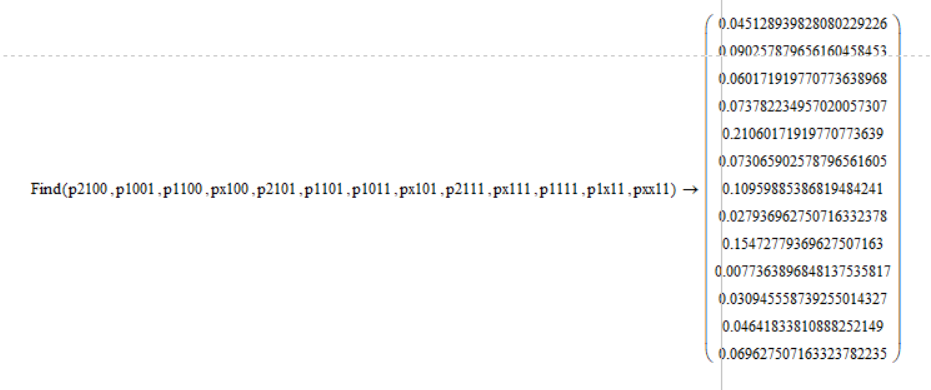


где pi1=0.4, pi2=0.5

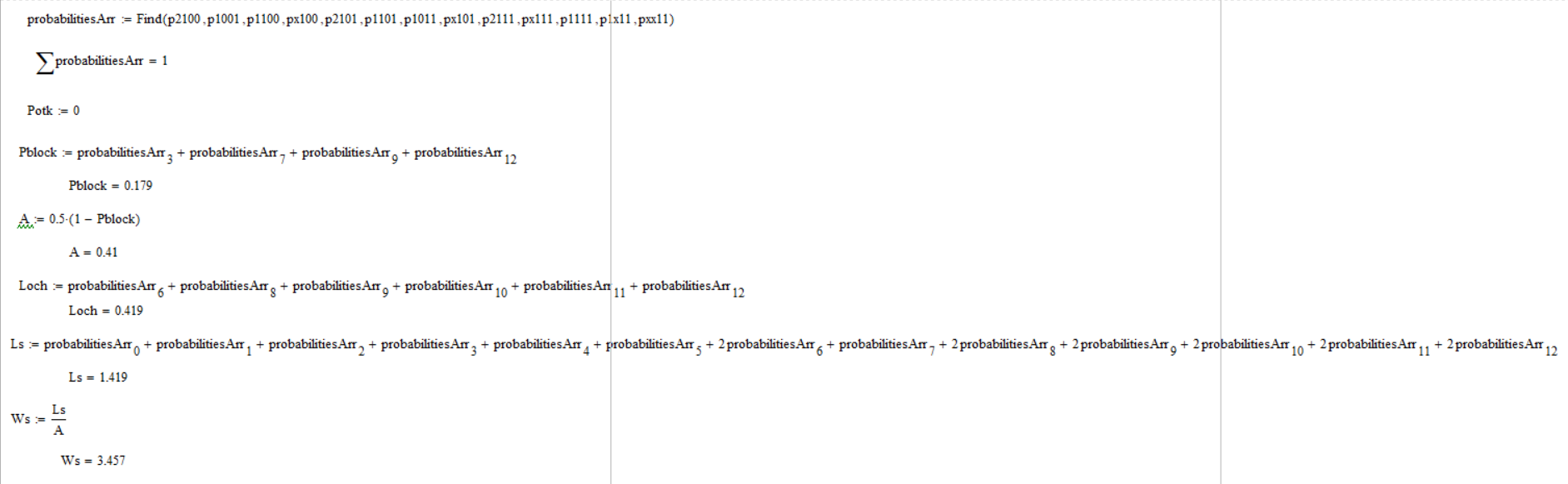
Воспользовавшись нормировочным уравнением:



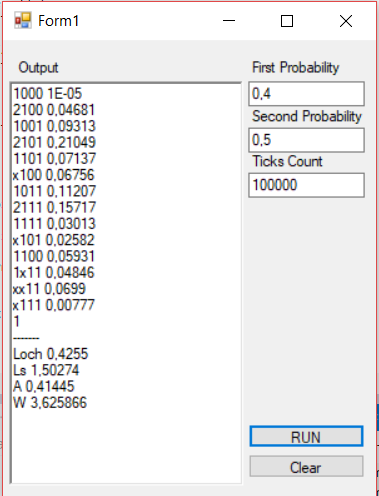
и заменив в исходной системе любое уравнение данным, найдем решение СЛАУ при заданных pi1 и pi2:



Рассчитаем показатели эффективности:



1. РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ



1. ВЫВОДЫ

В данной работе была исследована дискретно-стохастическая модель СМО. Для нее был построен граф состояний, на основе которого была составлена СЛАУ и рассчитаны вероятности нахождения системы в каждом из состояний. Произведен расчет данных показателей эффективности: A, Wc, Lоч. Проведено имитационное моделирование системы. В ходе сравнения полученных статистических показателей с расчетными показателями сильных отклонений не выявлено. Текущие отклонения связаны со статистической погрешностью.

1. КОД ПРОГРАММЫ

class Manager

{

List<Element> elements = new List<Element>();

List<Element> elementsInverse = new List<Element>();

Dictionary<string, int> statesCount = new Dictionary<string, int>();

TraficCounter traficCounter;// = new TraficCounter();

Generator generator;// = new Generator(1, true);

Processor processor1;// = new Processor(.4f, true);

Queue queue;// = new Queue(1);

Processor processor2;// = new Processor(.5f, true);

StringBuilder sb = new StringBuilder();

int stateCount = 0;

string state;

int iterationsCOunt = 0;

int generalVocationsCount = 0;

int generalQueueLemght = 0;

string State

{

get

{

sb.Clear();

return generator.State + processor1.State + queue.State + processor2.State;

}

}

public float MeanQueueLength

{

get

{

return ((float)generalQueueLemght / iterationsCOunt);

}

}

public float MeanVocationsCount

{

get

{

return ((float)generalVocationsCount / iterationsCOunt);

}

}

public float MeanTrafic

{

get

{

return ((float)elements[0].TraficCounter?.VocationsCount / iterationsCOunt);

}

}

void Initialize(float probability1 = 0.0f, float probability2 = 0.0f)

{

traficCounter = new TraficCounter();

generator = new Generator(1, true);

processor1 = new Processor(probability1, true);

queue = new Queue(1);

processor2 = new Processor(probability2, true);

generator.Init(null, processor1);

processor1.Init(generator, queue);

queue.Init(processor1, processor2);

processor2.Init(queue, null, traficCounter);

elements.Add(processor2);

elements.Add(queue);

elements.Add(processor1);

elements.Add(generator);

elementsInverse.Add(generator);

elementsInverse.Add(processor1);

elementsInverse.Add(queue);

elementsInverse.Add(processor2);

}

public Manager(int iterationsCOunt = 100000, float probability1 = 0.0f, float probability2 = 0.0f)

{

Initialize(probability1, probability2);

this.iterationsCOunt = iterationsCOunt;

}

public void ProcessTick()

{

elements.ForEach(item => item.ProcessTick(true));

generalQueueLemght += queue.CurrentCapacity;

generalVocationsCount += (processor1.State == "1") ? 1 : 0;

generalVocationsCount += ( processor2.State == "1") ? 1 : 0;

state = State;

if (statesCount.TryGetValue(state, out stateCount))

{

statesCount[state]++;

}

else

{

statesCount.Add(state, 1);

}

}

public void ProcessManyTicks()

{

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

{

ProcessTick();

}

}

public string PrintAll()

{

string result = "";

float sum = 0f;

foreach(var item in statesCount)

{

result += item.Key + " " + ((float)item.Value / iterationsCOunt).ToString() + "\n";

sum += ((float)item.Value / iterationsCOunt);

}

result += sum.ToString() + "\n-------\n";

result += "Loch "+ MeanQueueLength.ToString() + "\n";

result += "Ls " + MeanVocationsCount.ToString() + "\n";

result += "A " + MeanTrafic.ToString() + "\n";

result += "W " +(MeanVocationsCount / MeanTrafic ).ToString();

return result;

}

public void Clear()

{

Initialize();

statesCount.Clear();

}

}