БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ и РАДИОЭЛЕКТРОНИКИ

Факультет КСиС

Кафедра ЭВМ ФКСиС

Моделирование

Лабораторная работа № 5

Вариант № 5(а,г)

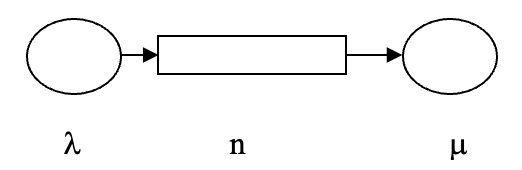
Ст. гр. X5100X Преподаватель:

ФИО Мельник Н.И.

Минск

2022

# Постановка задачи



На вход поступает простейший поток заявок с интенсивностью λ. Время обслуживания – показательное с параметром μ. Работающий (обслуживающий заявки канал ) может выходить из строя (отказывать). Поток отказов – простейший с параметром ν. Ремонт начинается мгновенно после отказа. Время ремонта – показательное с параметром γ.

Найти

* вероятности состояний канала (канал свободен, занят, ремонтируется)
* абсолютную и относительную пропускную способность системы.

λ = 0.5, μ = 1. ν =0.01, γ =0.1, n = 5.

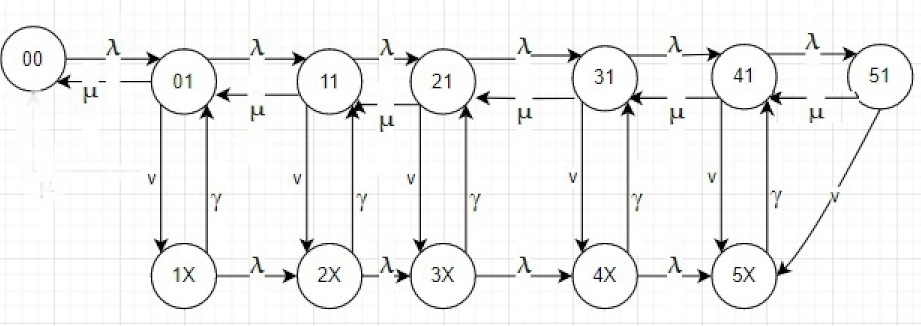
Варианты:

а) Канал может выходить из строя только при обслуживании заявок;

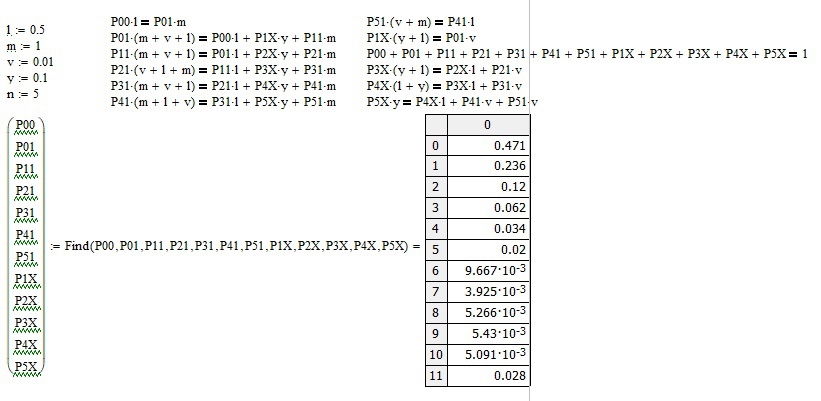
г) Заявка, находившаяся в канале в момент его выхода из строя становится в очередь, если в ней есть места и обслуживается заново;

# Ход выполнения работы

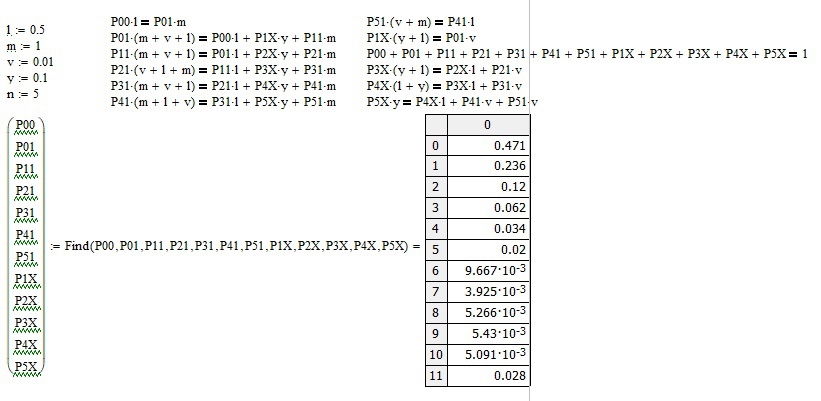
Граф состояний:



Система уравнений по графу состояний:



Найденные вероятности состояний системы:

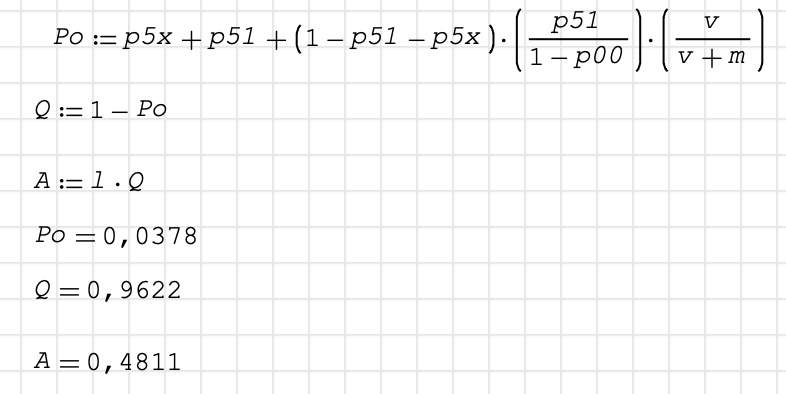


Расчет показателей эффективности:

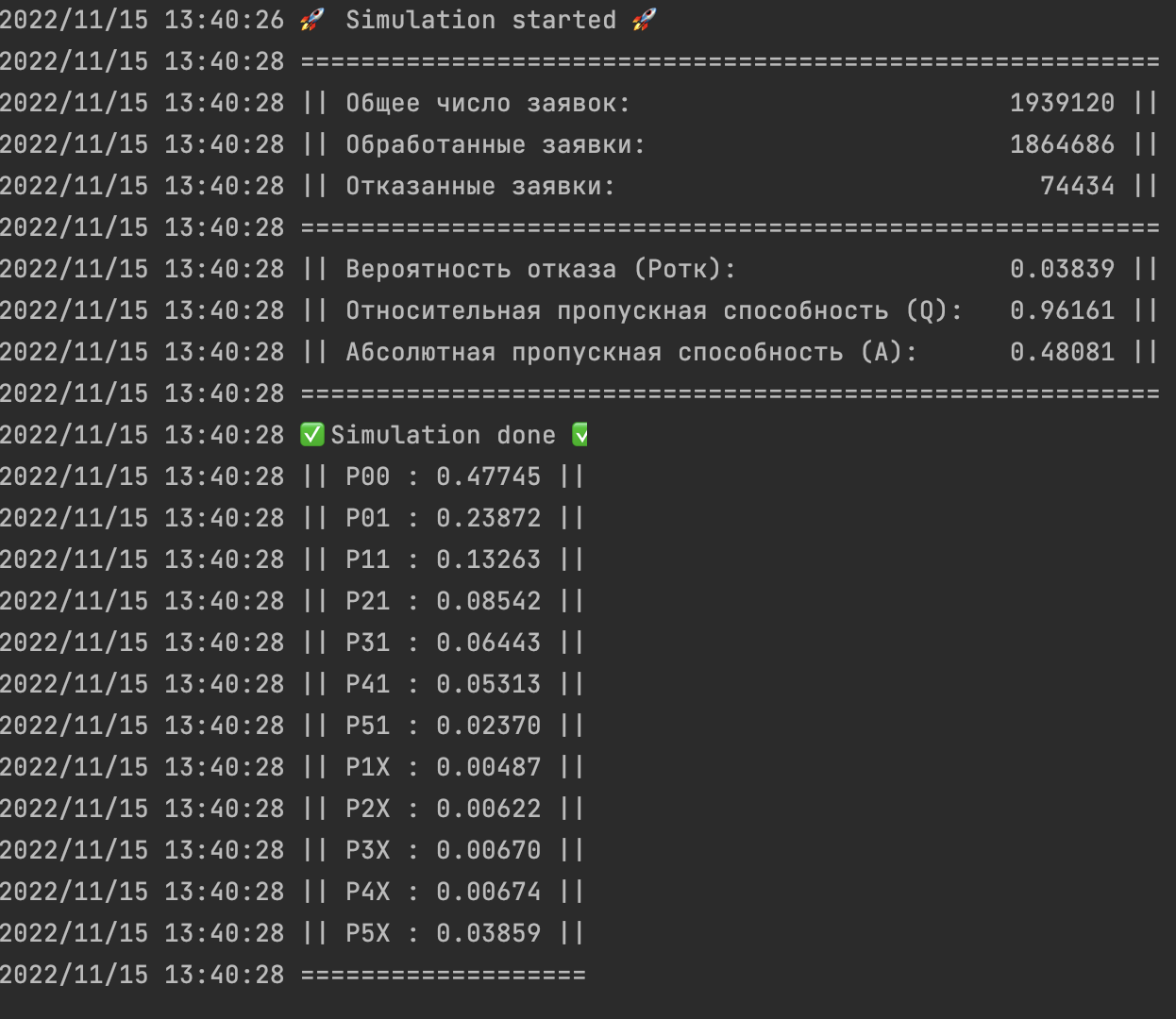
Ротк – вероятность отказа;

Q – относительная пропускная способность;

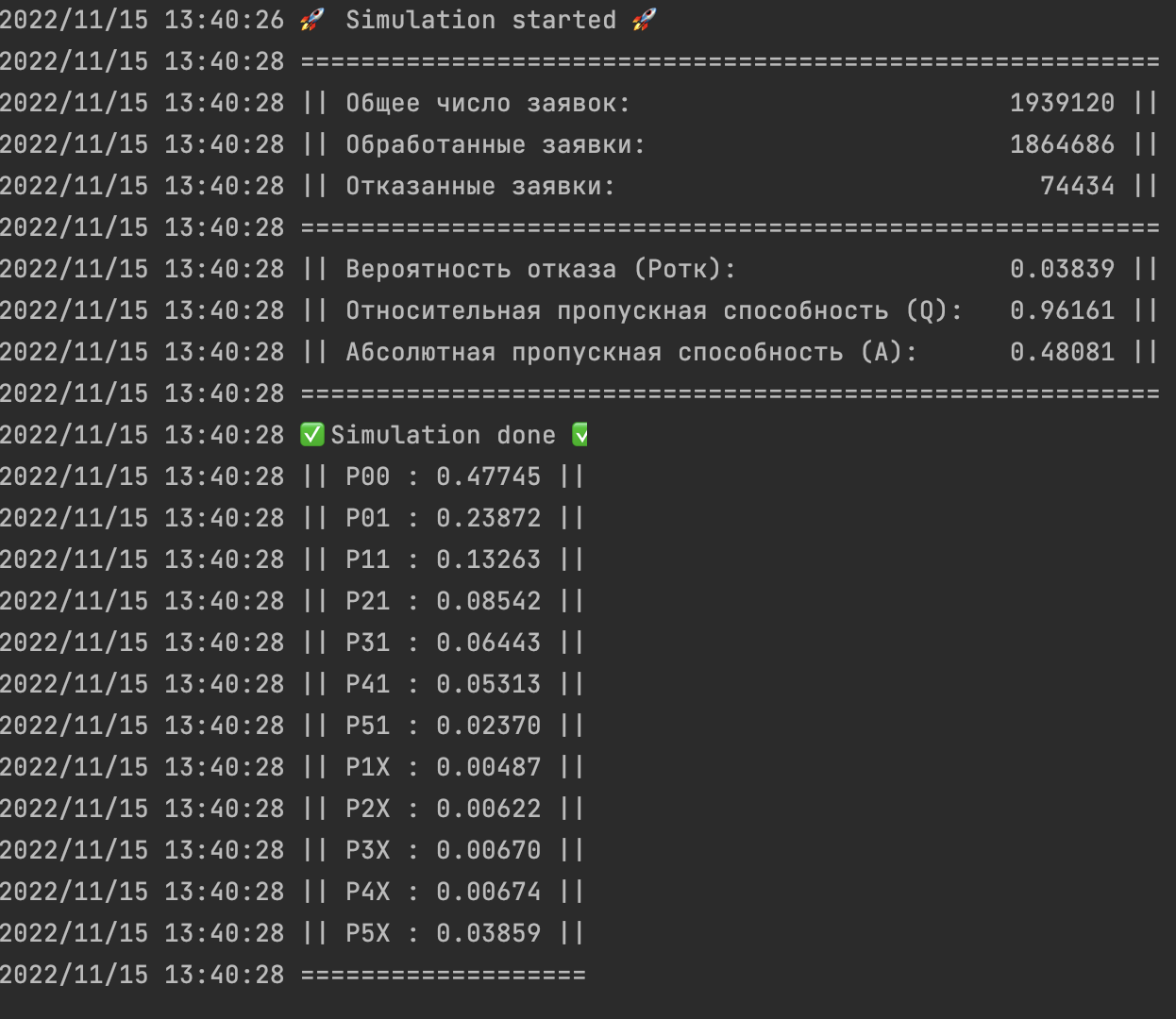
А – абсолютная пропускная способность;



Вероятности состояний системы при имитационном моделировании:



Расчёт показателей эффективности системы при иммитационном моделировании:



Листинг имммитационной модели:

package main

import (

"log"

"sammod\_5-6/internal/service"

)

var machine service.Machine

const (

L = 0.5

Nu = 1.0

V = 0.01

Y = 0.1

N = 5

Count = 10000001

)

func init() {

machine = service.Machine{

NewRequestTime: 1 / L,

DoneRequestTime: Nu,

FailTime: 1 / V,

RepairTime: 1 / Y,

MaxQueue: N,

CurrentQueue: 0,

AllCount: 0,

DoneCount: 0,

FailCount: 0,

IterCount: 1,

RepairStart: 0,

RequestStart: 0,

ChannelState: 0,

States: service.States{},

}

}

func main() {

log.Println("🚀 Simulation started 🚀")

machine.StartSimulating(Count, L)

machine.OutputMain(L)

log.Println("✅ Simulation done ✅")

}

package service

import (

"math"

"math/rand"

"time"

)

type States struct {

P00 int

P01 int

P11 int

P21 int

P31 int

P41 int

P51 int

P1X int

P2X int

P3X int

P4X int

P5X int

}

const (

Free int = 0

Busy = 1

Broken = 2

)

type Machine struct {

//params

NewRequestTime float64

DoneRequestTime float64

FailTime float64

RepairTime float64

//queue

MaxQueue int

CurrentQueue int

//requests count

AllCount int

DoneCount int

FailCount int

//cycle count

IterCount int

RepairStart int

RequestStart int

//channel

ChannelState int // 0 - nothing in, 1 - busy, 2 - broken

States States

}

func (m \*Machine) StartSimulating(n int, l float64) {

for ; m.IterCount < n; m.IterCount++ {

go m.RequestStream()

//go m.ChannelStream()

go m.FailStream()

//go m.CurrentState()

}

m.Validate(n, l)

}

func (m \*Machine) RequestStream() {

m.CurrentState()

if math.Mod(float64(m.IterCount), m.NewRequestTime) == 0 {

m.AllCount++

if m.ChannelState == Free {

m.ChannelState = Busy

m.RequestStart = m.IterCount

} else if m.CurrentQueue < m.MaxQueue {

m.CurrentQueue++

} else {

m.FailCount++

}

}

m.ChannelStream()

}

func (m \*Machine) ChannelStream() {

if m.ChannelState != Broken {

if m.ChannelState == Busy {

if math.Mod(float64(m.IterCount-m.RequestStart), m.DoneRequestTime) == 0 && m.IterCount != m.RequestStart {

m.DoneCount++

m.ChannelState = Free

}

}

if m.ChannelState == Free && m.CurrentQueue > 0 {

m.RequestStart = m.IterCount

m.ChannelState = Busy

m.CurrentQueue--

}

}

}

func (m \*Machine) FailStream() {

if math.Mod(float64(m.IterCount), m.FailTime) == 0 && m.ChannelState == Busy {

m.ChannelState = Broken

m.RepairStart = m.IterCount

if m.CurrentQueue == m.MaxQueue {

m.FailCount++

} else {

m.CurrentQueue++

}

}

m.RepairStream()

}

func (m \*Machine) RepairStream() {

if m.ChannelState == Broken {

if math.Mod(float64(m.IterCount-m.RepairStart), m.RepairTime) == 0 && m.IterCount != m.RepairStart {

m.ChannelState = Free

}

}

}