Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по лабораторной работе №5**

по дисциплине: «Компьютерные системы моделирования»

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС-22.06  Богданов Н.Р. |
| Проверила:  старший преподаватель кафедры ПО  Корнеева Е.И. |

Тверь 2024

Оглавление

[**Введение** 3](#_Toc197195685)

[**Описание алгоритма** 5](#_Toc197195686)

[**Пост диагностики** 5](#_Toc197195687)

[**АЗС** 6](#_Toc197195688)

[**Программная реализация** 8](#_Toc197195689)

[**Пост диагностики** 8](#_Toc197195690)

[**АЗС** 10](#_Toc197195691)

[**Результаты:** 26](#_Toc197195692)

[**Решение в Excel** 28](#_Toc197195693)

[**Вывод** 33](#_Toc197195694)

[**Ссылка на github** 33](#_Toc197195695)

# **Введение**

Для задач в задании необходимо составить расчеты характеристик СМО. Задачи относятся к одноканальным или многоканальным СМО.

Отчет по заданию должен содержать:

* Постановку задач по варианту;
* Основные действия для решения 2 задач в MS Excel
* Основные действия для решения 2 задач на python
* Основные действия для решения 2 задач на python с помощью библиотеки simpy
* Тестирование и сравнение реализаций c библиотекой simpy и без неё. Провести статистическое исследование генератора при различных значениях выборки: малых n<25, средних , больших n>500.
* Проверку основных исключений и вывод сообщений об ошибках
* GUI интерфейс

**Задание**

**Задача 1.3.** Пост диагностики автомобилей представляет собой одноканальную СМО с отказами. Заявка на диагностику, поступившая в момент, когда пост занят, получает отказ.

Интенсивность потока заявок на диагностику λ (автомобиля в час). Средняя продолжительность диагностики Tобсл. Все потоки событий в системе простейшие.

Определите в установившемся режиме вероятностные характеристики системы.

Тестовые данные для проверки без генератора: λ = 0,5; Tобсл. = 1,2 часа.

Генератор случайных чисел использовать лучший из Лабораторной работы 3. Генерируется λ и Tобсл.

**Задача 2.2.** Рассматривается работа АЗС, на которой имеется n заправочных колонок.

Заправка одной машины длится в среднем Tобсл. В среднем на АЗС каждую λ прибывает машина, нуждающаяся в заправке бензином.

Все машины, вставшие в очередь на заправку, дожидаются своей очереди. Все потоки в системе простейшие. Определите вероятностные характеристики работы АЗС в стационарном режиме.

Тестовые данные для проверки без генератора: λ = 1 машина в минуту;

Tобсл = 3 мин.; n = 4;

Генерация машин и времени обслуживания по Пуассону. Количество колонок для заправки задаются пользователем.

Посчитать для очереди без ограничений и для очереди с 4 местами.

# **Описание алгоритма**

## **Пост диагностики**

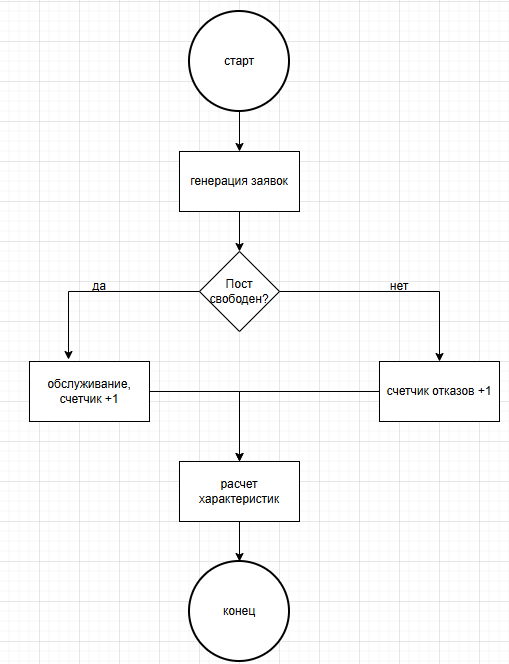
**Алгоритм (аналитическая модель):**

* Заявки обслуживаются по экспоненциальному закону, и при занятом посте происходит отказ.
* Вычисляются параметры:
  + ρ = λ \* Tобсл
  + P₀ = 1 / (1 + ρ) — вероятность простоя
  + Pотк = ρ / (1 + ρ) — вероятность отказа
  + Q = 1 - Pотк — относительная пропускная способность
  + A = λ \* Q — абсолютная пропускная способность

**Алгоритм (имитационная модель с SimPy):**

* Используется одноканальный simpy.Resource.
* Если пост занят, заявка не ждет, теряется.
* События генерации и обслуживания происходят по экспоненциальному распределению.

**Блок-схема:**

****

## **АЗС**

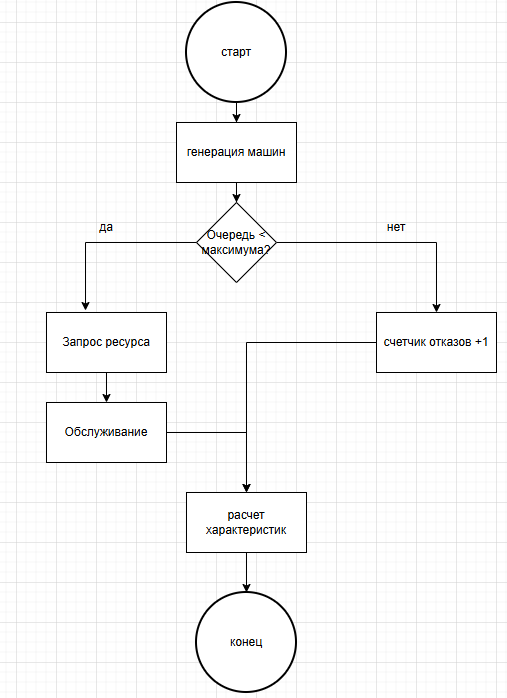
**Аналитическая модель:**

* Вычисляются:
  + P₀ — вероятность простоя
  + Lq — средняя длина очереди
  + Ls — среднее число машин в системе
  + Wq — среднее время ожидания в очереди
  + Pblock — вероятность отказа (только для ограниченной очереди)

**Имитационная модель (SimPy):**

* simpy.Resource моделирует колонки заправки.
* Генерация машин и обслуживание
* В случае ограничения очереди, если очередь заполнена, машина теряется(не обслуживается).

**Блок-схема:**



# **Программная реализация**

## **Пост диагностики**

Решение этой задачи разделено на 2 файла: решение с использованием библиотеки SimPy и без ее использования.

diagnostic.py:

def run\_analytical(lambda\_rate, service\_time):  
 try:  
 ro = lambda\_rate \* service\_time  
 p0 = 1 / (1 + ro)  
 p\_otkaza = ro / (1 + ro)  
 q = 1 - p\_otkaza  
 A = lambda\_rate \* q  
 return {"ro": ro, "p0": p0, "p\_otkaza": p\_otkaza, "q": q, "A": A}  
 except Exception as e:  
 raise RuntimeError("Ошибка при аналитическом расчете: " + str(e))

diagnostic\_simpy.py:

import simpy  
import random  
  
class DiagnosticPost:  
 def \_\_init\_\_(self, env, lambda\_rate, service\_time):  
 self.env = env  
 self.lambda\_rate = lambda\_rate  
 self.service\_time = service\_time  
 self.diagnostic = simpy.Resource(env, capacity=1)  
 self.total\_cars = 0  
 self.served\_cars = 0  
 self.busy\_time = 0.0  
  
 def diagnose(self):  
 with self.diagnostic.request() as request:  
 yield request  
 t = random.expovariate(1 / self.service\_time)  
 self.busy\_time += t  
 yield self.env.timeout(t)  
 self.served\_cars += 1  
  
def run\_simulation(lambda\_rate, service\_time, sim\_time):  
 env = simpy.Environment()  
 post = DiagnosticPost(env, lambda\_rate, service\_time)  
 def car\_generator():  
 while True:  
 yield env.timeout(random.expovariate(lambda\_rate))  
 post.total\_cars += 1  
 env.process(post.diagnose())  
 env.process(car\_generator())  
 env.run(until=sim\_time)  
  
 p\_otkaza = 1 - post.served\_cars / post.total\_cars if post.total\_cars else None  
 q = 1 - p\_otkaza if p\_otkaza is not None else None  
 A = lambda\_rate \* q if q is not None else None  
  
  
 p0 = 1 - post.busy\_time / sim\_time  
  
 return {  
 "p0": p0,  
 "p\_otkaza": p\_otkaza,  
 "q": q,  
 "A": A,  
 "total\_cars": post.total\_cars,  
 "served\_cars": post.served\_cars  
 }

## **АЗС**

Решение этой задачи разделено на 2 файла: решение с использованием библиотеки SimPy и без ее использования.

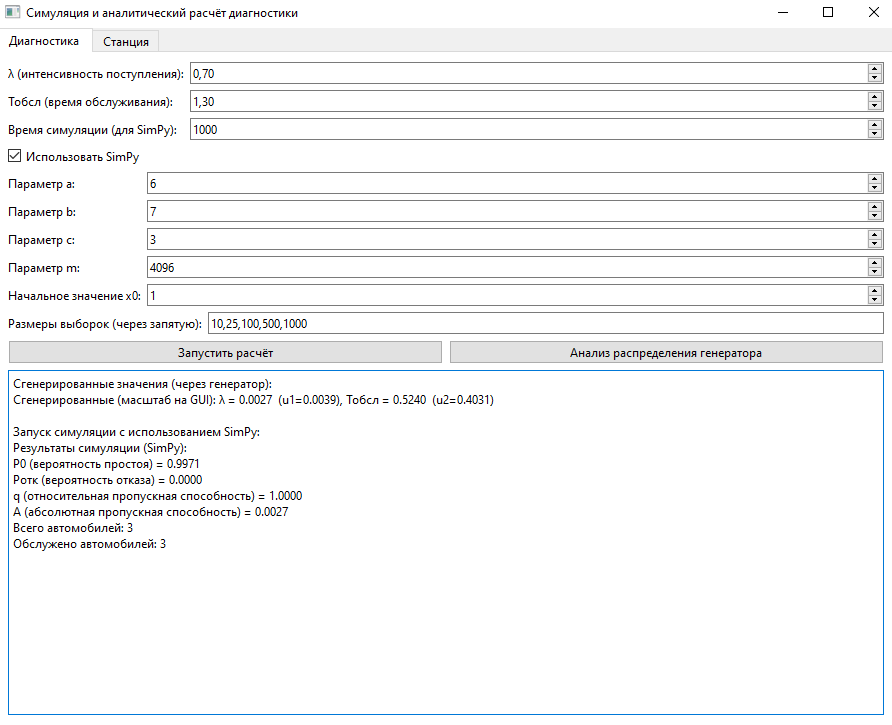
station.py:  
import math  
  
class AnalyticalModel:  
 def \_\_init\_\_(self, arrival\_rate, service\_time, num\_pumps):  
 self.lambda\_ = arrival\_rate  
 self.service\_time = service\_time  
 self.num\_pumps = num\_pumps  
 self.mu = 1.0 / service\_time  
  
 def mmn\_no\_limit\_queue(self):  
 n = self.num\_pumps  
 lambda\_ = self.lambda\_  
 mu = self.mu  
 rho = lambda\_ / (n \* mu)  
  
 sum\_term = sum((n \* rho) \*\* k / math.factorial(k) for k in range(n))  
 p0 = 1.0 / (sum\_term + ((n \* rho) \*\* n / (math.factorial(n) \* (1 - rho))))  
  
 Lq = ((n \* rho) \*\* n \* p0) / (math.factorial(n) \* (1 - rho) \*\* 2)  
 Ls = Lq + lambda\_ / mu  
 Wq = Lq / lambda\_  
  
 return {  
 "P0 (вероятность простоя)": p0,  
 "Lq (среднее число машин в очереди)": Lq,  
 "Ls (среднее число машин в системе)": Ls,  
 "Wq (среднее время ожидания в очереди, мин)": Wq  
 }  
  
 def mmn\_limited\_queue(self, max\_queue):  
 n = self.num\_pumps  
 lambda\_ = self.lambda\_  
 mu = self.mu  
 rho = lambda\_ / (n \* mu)  
 K = n + max\_queue  
  
 sum\_term = sum((n \* rho) \*\* k / math.factorial(k) for k in range(n))  
 sum\_term += sum((n \* rho) \*\* k / (math.factorial(n) \* n \*\* (k - n)) for k in range(n, K + 1))  
 p0 = 1.0 / sum\_term  
 Lq = sum((k - n) \* ((n \* rho) \*\* k / (math.factorial(n) \* n \*\* (k - n))) \* p0 for k in range(n, K + 1))  
 P\_block = ((n \* rho) \*\* K / (math.factorial(n) \* n \*\* (K - n))) \* p0  
 lambda\_eff = lambda\_ \* (1 - P\_block)  
 Ls = Lq + lambda\_eff / mu  
 Wq = Lq / lambda\_  
  
 return {  
 "P0 (вероятность простоя)": p0,  
 "Lq (среднее число машин в очереди)": Lq,  
 "Ls (среднее число машин в системе)": Ls,  
 "Wq (среднее время ожидания в очереди, мин)": Wq,  
 "P\_block (вероятность отказа)": P\_block  
 }  
  
def main():  
 lambda\_arrival = 1  
 service\_time = 3  
 num\_pumps = 4  
 max\_queue = 4  
  
 print("Аналитическая модель (без ограничения очереди):")  
 analytical\_model = AnalyticalModel(lambda\_arrival, service\_time, num\_pumps)  
 results\_no\_limit = analytical\_model.mmn\_no\_limit\_queue()  
 for key, value in results\_no\_limit.items():  
 print(f"{key}: {value:.4f}")  
  
 print("\nАналитическая модель (с ограниченной очередью):")  
 results\_limited = analytical\_model.mmn\_limited\_queue(max\_queue)  
 for key, value in results\_limited.items():  
 print(f"{key}: {value:.4f}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

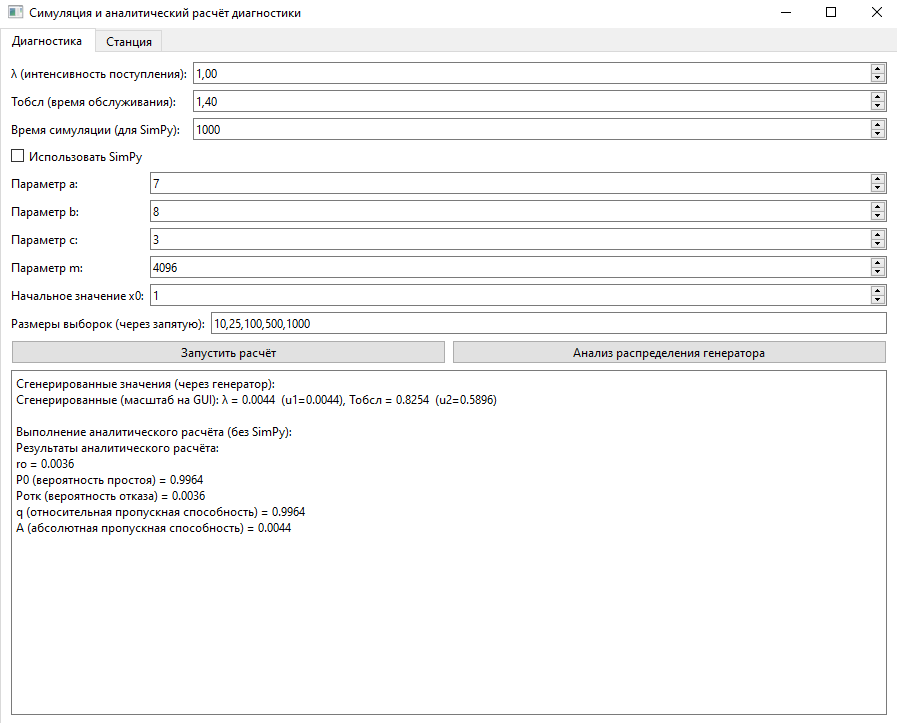
station\_simpy.py:  
import simpy  
import numpy as np  
  
class SimulationModel:  
 def \_\_init\_\_(self, arrival\_rate, service\_time, num\_pumps, simulation\_time=1000, sampling\_interval=0.1):  
 self.arrival\_rate = arrival\_rate  
 self.service\_time = service\_time  
 self.num\_pumps = num\_pumps  
 self.simulation\_time = simulation\_time  
 self.sampling\_interval = sampling\_interval  
  
 def run\_simulation(self, queue\_limit=None):  
 env = simpy.Environment()  
 station = simpy.Resource(env, capacity=self.num\_pumps)  
 wait\_times = []  
 lost\_cars = 0  
 state\_samples = []  
  
 def monitor\_state():  
 while True:  
 state\_samples.append((env.now, len(station.queue), station.count))  
 yield env.timeout(self.sampling\_interval)  
 env.process(monitor\_state())  
  
 def refuel\_car(car\_id):  
 nonlocal lost\_cars  
 arrival\_time = env.now  
 if queue\_limit is not None and len(station.queue) >= queue\_limit:  
 lost\_cars += 1  
 return  
 with station.request() as req:  
 yield req  
 service\_duration = np.random.exponential(self.service\_time)  
 yield env.timeout(service\_duration)  
 wait\_times.append(env.now - arrival\_time)  
  
 def car\_generator():  
 car\_id = 0  
 while True:  
 interarrival\_time = np.random.exponential(1.0 / self.arrival\_rate)  
 yield env.timeout(interarrival\_time)  
 env.process(refuel\_car(car\_id))  
 car\_id += 1  
  
 env.process(car\_generator())  
 env.run(until=self.simulation\_time)  
  
 queue\_lengths = [q for \_, q, \_ in state\_samples]  
 busy\_counts = [b for \_, \_, b in state\_samples]  
 Lq = np.mean(queue\_lengths)  
 Ls = np.mean([q + b for q, b in zip(queue\_lengths, busy\_counts)])  
 idle\_samples = sum(1 for q, b in zip(queue\_lengths, busy\_counts) if (q + b) == 0)  
 P0 = idle\_samples / len(state\_samples) if state\_samples else 0.0  
 Wq = np.mean(wait\_times) if wait\_times else 0.0  
  
 result = {  
 "P0 (вероятность простоя)": P0,  
 "Lq (среднее число машин в очереди)": Lq,  
 "Ls (среднее число машин в системе)": Ls,  
 "Wq (среднее время ожидания в очереди)": Wq,  
 "Число потерянных машин": lost\_cars  
 }  
 return result

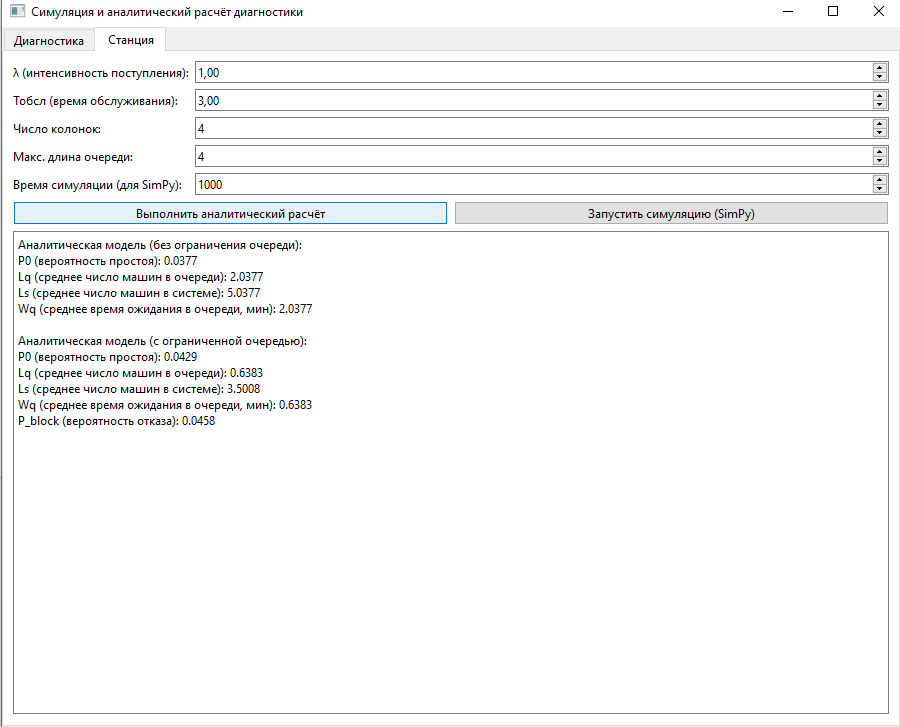
**Интерфейс**

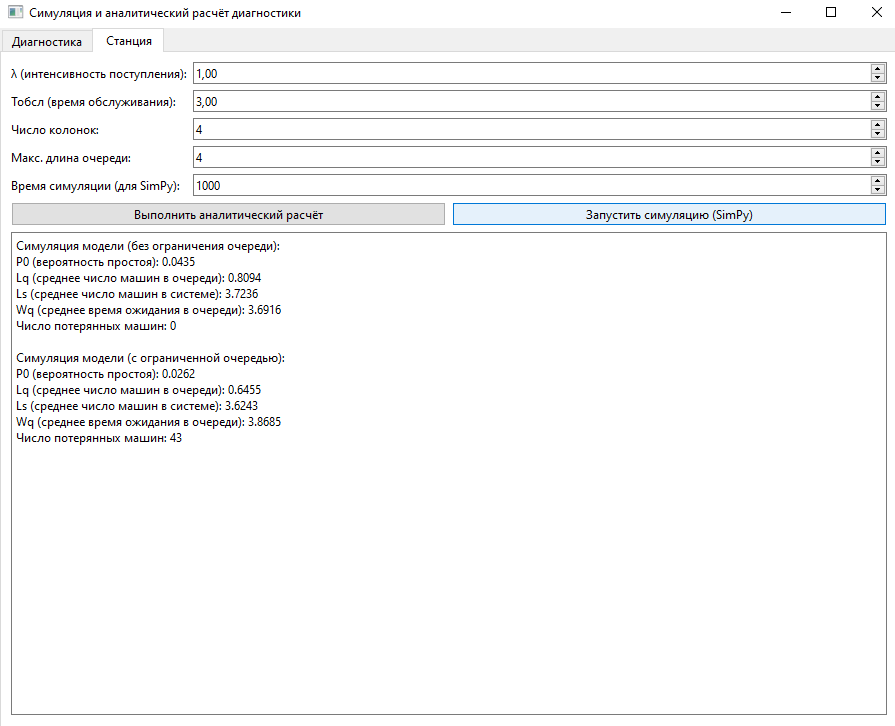
Интерфейс для двух предыдущих задач был написан на PySide6. В интерфейсе присутствуют две вкладки для двух задач, в каждой из которых ты можешь ввести необходимые данные для решения задачи и получить результат.  
import sys  
  
from PySide6.QtWidgets import (  
 QApplication, QMainWindow, QWidget, QTabWidget, QVBoxLayout, QHBoxLayout,  
 QFormLayout, QDoubleSpinBox, QSpinBox, QCheckBox, QPushButton, QTextEdit,  
 QMessageBox, QLineEdit  
)  
from Lab4.analyze import analyze\_distribution  
from Lab4.generator import QuadraticCongruentialGenerator  
from Lab5.diagnostics import run\_analytical  
from Lab5.diagnostics\_simpy import run\_simulation  
from Lab5.station import AnalyticalModel  
from Lab5.station\_simpy import SimulationModel  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Симуляция и аналитический расчёт диагностики")  
 self.resize(900, 700)  
 self.init\_ui()  
  
 def init\_ui(self):  
 self.tab\_widget = QTabWidget()  
 self.setCentralWidget(self.tab\_widget)  
  
 self.simulation\_tab = QWidget()  
 self.tab\_widget.addTab(self.simulation\_tab, "Диагностика")  
 main\_layout = QVBoxLayout(self.simulation\_tab)  
  
 sim\_params\_layout = QFormLayout()  
 self.lambda\_input = QDoubleSpinBox()  
 self.lambda\_input.setRange(0.01, 100.0)  
 self.lambda\_input.setValue(0.5)  
 self.lambda\_input.setSingleStep(0.1)  
 sim\_params\_layout.addRow("λ (интенсивность поступления):", self.lambda\_input)  
  
 self.service\_time\_input = QDoubleSpinBox()  
 self.service\_time\_input.setRange(0.01, 100.0)  
 self.service\_time\_input.setValue(1.2)  
 self.service\_time\_input.setSingleStep(0.1)  
 sim\_params\_layout.addRow("Tобсл (время обслуживания):", self.service\_time\_input)  
  
 self.sim\_time\_input = QSpinBox()  
 self.sim\_time\_input.setRange(1, 10000)  
 self.sim\_time\_input.setValue(1000)  
 sim\_params\_layout.addRow("Время симуляции (для SimPy):", self.sim\_time\_input)  
  
 self.use\_simpy\_checkbox = QCheckBox("Использовать SimPy")  
 self.use\_simpy\_checkbox.setChecked(True)  
 sim\_params\_layout.addRow(self.use\_simpy\_checkbox)  
  
 main\_layout.addLayout(sim\_params\_layout)  
  
 gen\_params\_layout = QFormLayout()  
 self.gen\_a\_input = QSpinBox()  
 self.gen\_a\_input.setRange(0, 10000)  
 self.gen\_a\_input.setValue(6)  
 gen\_params\_layout.addRow("Параметр a:", self.gen\_a\_input)  
  
 self.gen\_b\_input = QSpinBox()  
 self.gen\_b\_input.setRange(0, 10000)  
 self.gen\_b\_input.setValue(7)  
 gen\_params\_layout.addRow("Параметр b:", self.gen\_b\_input)  
  
 self.gen\_c\_input = QSpinBox()  
 self.gen\_c\_input.setRange(0, 10000)  
 self.gen\_c\_input.setValue(3)  
 gen\_params\_layout.addRow("Параметр c:", self.gen\_c\_input)  
  
 self.gen\_m\_input = QSpinBox()  
 self.gen\_m\_input.setRange(1, 1000000)  
 self.gen\_m\_input.setValue(4096)  
 gen\_params\_layout.addRow("Параметр m:", self.gen\_m\_input)  
  
 self.gen\_x0\_input = QSpinBox()  
 self.gen\_x0\_input.setRange(0, 1000000)  
 self.gen\_x0\_input.setValue(1)  
 gen\_params\_layout.addRow("Начальное значение x0:", self.gen\_x0\_input)  
  
 main\_layout.addLayout(gen\_params\_layout)  
  
 test\_params\_layout = QFormLayout()  
 self.sample\_sizes\_input = QLineEdit("10,25,100,500,1000")  
 test\_params\_layout.addRow("Размеры выборок (через запятую):", self.sample\_sizes\_input)  
  
 main\_layout.addLayout(test\_params\_layout)  
  
 buttons\_layout = QHBoxLayout()  
 self.run\_button = QPushButton("Запустить расчёт")  
 self.run\_button.clicked.connect(self.run\_calculations)  
 buttons\_layout.addWidget(self.run\_button)  
  
 self.test\_button = QPushButton("Анализ распределения генератора")  
 self.test\_button.clicked.connect(self.run\_generator\_test)  
 buttons\_layout.addWidget(self.test\_button)  
  
 main\_layout.addLayout(buttons\_layout)  
  
 self.output\_text = QTextEdit()  
 self.output\_text.setReadOnly(True)  
 main\_layout.addWidget(self.output\_text)  
  
 self.station\_tab = QWidget()  
 self.tab\_widget.addTab(self.station\_tab, "Станция")  
 station\_layout = QVBoxLayout(self.station\_tab)  
  
 station\_params\_layout = QFormLayout()  
 self.station\_lambda\_input = QDoubleSpinBox()  
 self.station\_lambda\_input.setRange(0.01, 100.0)  
 self.station\_lambda\_input.setValue(1.0)  
 self.station\_lambda\_input.setSingleStep(0.1)  
 station\_params\_layout.addRow("λ (интенсивность поступления):", self.station\_lambda\_input)  
  
 self.station\_service\_time\_input = QDoubleSpinBox()  
 self.station\_service\_time\_input.setRange(0.01, 100.0)  
 self.station\_service\_time\_input.setValue(3.0)  
 self.station\_service\_time\_input.setSingleStep(0.1)  
 station\_params\_layout.addRow("Tобсл (время обслуживания):", self.station\_service\_time\_input)  
  
 self.station\_num\_pumps\_input = QSpinBox()  
 self.station\_num\_pumps\_input.setRange(1, 100)  
 self.station\_num\_pumps\_input.setValue(4)  
 station\_params\_layout.addRow("Число колонок:", self.station\_num\_pumps\_input)  
  
 self.station\_max\_queue\_input = QSpinBox()  
 self.station\_max\_queue\_input.setRange(0, 100)  
 self.station\_max\_queue\_input.setValue(4)  
 station\_params\_layout.addRow("Макс. длина очереди:", self.station\_max\_queue\_input)  
  
 self.station\_sim\_time\_input = QSpinBox()  
 self.station\_sim\_time\_input.setRange(1, 10000)  
 self.station\_sim\_time\_input.setValue(1000)  
 station\_params\_layout.addRow("Время симуляции (для SimPy):", self.station\_sim\_time\_input)  
  
 station\_layout.addLayout(station\_params\_layout)  
  
 station\_buttons\_layout = QHBoxLayout()  
 self.run\_station\_analytical\_button = QPushButton("Выполнить аналитический расчёт")  
 self.run\_station\_analytical\_button.clicked.connect(self.run\_station\_analytical)  
 station\_buttons\_layout.addWidget(self.run\_station\_analytical\_button)  
  
 self.run\_station\_simulation\_button = QPushButton("Запустить симуляцию (SimPy)")  
 self.run\_station\_simulation\_button.clicked.connect(self.run\_station\_simulation)  
 station\_buttons\_layout.addWidget(self.run\_station\_simulation\_button)  
  
 station\_layout.addLayout(station\_buttons\_layout)  
  
 self.station\_output\_text = QTextEdit()  
 self.station\_output\_text.setReadOnly(True)  
 station\_layout.addWidget(self.station\_output\_text)  
  
 def run\_calculations(self):  
 try:  
 lambda\_rate = self.lambda\_input.value()  
 service\_time = self.service\_time\_input.value()  
 sim\_time = self.sim\_time\_input.value()  
 use\_simpy = self.use\_simpy\_checkbox.isChecked()  
  
 a = self.gen\_a\_input.value()  
 b = self.gen\_b\_input.value()  
 c = self.gen\_c\_input.value()  
 m = self.gen\_m\_input.value()  
 x0 = self.gen\_x0\_input.value()  
  
 gen = QuadraticCongruentialGenerator(a=a, b=b, c=c, m=m, x0=x0)  
  
 u1 = gen.next()  
 u2 = gen.next()  
  
 self.output\_text.clear()  
 self.output\_text.append("Сгенерированные значения (через генератор):")  
 lambda\_gen = u1 \* lambda\_rate  
 t\_obsl\_gen = u2 \* service\_time  
 self.output\_text.append(  
 f"Сгенерированные (масштаб на GUI): λ = {lambda\_gen:.4f} (u1={u1:.4f}), "  
 f"Tобсл = {t\_obsl\_gen:.4f} (u2={u2:.4f})\n"  
 )  
  
 if use\_simpy:  
 self.output\_text.append("Запуск симуляции с использованием SimPy:")  
 sim\_results = run\_simulation(lambda\_gen, t\_obsl\_gen, sim\_time)  
 self.output\_text.append("Результаты симуляции (SimPy):")  
 self.output\_text.append(f"P0 (вероятность простоя) = {sim\_results['p0']:.4f}")  
 if sim\_results['p\_otkaza'] is not None:  
 self.output\_text.append(f"Pотк (вероятность отказа) = {sim\_results['p\_otkaza']:.4f}")  
 self.output\_text.append(f"q (относительная пропускная способность) = {sim\_results['q']:.4f}")  
 self.output\_text.append(f"A (абсолютная пропускная способность) = {sim\_results['A']:.4f}")  
 self.output\_text.append(f"Всего автомобилей: {sim\_results['total\_cars']}")  
 self.output\_text.append(f"Обслужено автомобилей: {sim\_results['served\_cars']}")  
 else:  
 self.output\_text.append("Нет поступивших автомобилей для расчёта Pотк.")  
 else:  
 self.output\_text.append("Выполнение аналитического расчёта (без SimPy):")  
 analytical\_results = run\_analytical(lambda\_gen, t\_obsl\_gen)  
 self.output\_text.append("Результаты аналитического расчёта:")  
 self.output\_text.append(f"ro = {analytical\_results['ro']:.4f}")  
 self.output\_text.append(f"P0 (вероятность простоя) = {analytical\_results['p0']:.4f}")  
 self.output\_text.append(f"Pотк (вероятность отказа) = {analytical\_results['p\_otkaza']:.4f}")  
 self.output\_text.append(f"q (относительная пропускная способность) = {analytical\_results['q']:.4f}")  
 self.output\_text.append(f"A (абсолютная пропускная способность) = {analytical\_results['A']:.4f}")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
  
 def run\_generator\_test(self):  
 try:  
 a = self.gen\_a\_input.value()  
 b = self.gen\_b\_input.value()  
 c = self.gen\_c\_input.value()  
 m = self.gen\_m\_input.value()  
 x0 = self.gen\_x0\_input.value()  
  
 gen = QuadraticCongruentialGenerator(a=a, b=b, c=c, m=m, x0=x0)  
  
 sample\_sizes\_str = self.sample\_sizes\_input.text()  
 sample\_sizes = [int(x.strip()) for x in sample\_sizes\_str.split(",") if x.strip().isdigit()]  
  
 self.output\_text.append("\nЗапуск анализа распределения генератора:")  
 results = analyze\_distribution(gen, sample\_sizes, title="квадратичный конгруэнтный генератор")  
  
 self.output\_text.append("Результаты анализа распределения:")  
 for size, res in results.items():  
 mean, variance, std\_dev, lower\_bound, upper\_bound, expected\_percentage, actual\_percentage = res  
 self.output\_text.append(f"Размер выборки: {size}")  
 self.output\_text.append(f" Среднее: {mean:.4f}, Дисперсия: {variance:.4f}, Стандартное отклонение: {std\_dev:.4f}")  
 self.output\_text.append(f" Интервал: [{lower\_bound:.4f}, {upper\_bound:.4f}]")  
 self.output\_text.append(f" Ожидаемый %: {expected\_percentage:.2f}, Фактический %: {actual\_percentage:.2f}\n")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
  
 def run\_station\_analytical(self):  
 try:  
 lambda\_val = self.station\_lambda\_input.value()  
 service\_time = self.station\_service\_time\_input.value()  
 num\_pumps = self.station\_num\_pumps\_input.value()  
 max\_queue = self.station\_max\_queue\_input.value()  
  
 model = AnalyticalModel(lambda\_val, service\_time, num\_pumps)  
 results\_no\_limit = model.mmn\_no\_limit\_queue()  
 results\_limited = model.mmn\_limited\_queue(max\_queue)  
  
 self.station\_output\_text.clear()  
 self.station\_output\_text.append("Аналитическая модель (без ограничения очереди):")  
 for key, value in results\_no\_limit.items():  
 self.station\_output\_text.append(f"{key}: {value:.4f}")  
 self.station\_output\_text.append("\nАналитическая модель (с ограниченной очередью):")  
 for key, value in results\_limited.items():  
 self.station\_output\_text.append(f"{key}: {value:.4f}")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
  
 def run\_station\_simulation(self):  
 try:  
 lambda\_val = self.station\_lambda\_input.value()  
 service\_time = self.station\_service\_time\_input.value()  
 num\_pumps = self.station\_num\_pumps\_input.value()  
 max\_queue = self.station\_max\_queue\_input.value()  
 sim\_time = self.station\_sim\_time\_input.value()  
  
 sim\_model = SimulationModel(lambda\_val, service\_time, num\_pumps, simulation\_time=sim\_time)  
 results\_no\_limit = sim\_model.run\_simulation(queue\_limit=None)  
 results\_limited = sim\_model.run\_simulation(queue\_limit=max\_queue)  
  
 self.station\_output\_text.clear()  
 self.station\_output\_text.append("Симуляция модели (без ограничения очереди):")  
 for key, value in results\_no\_limit.items():  
 if isinstance(value, int):  
 self.station\_output\_text.append(f"{key}: {value}")  
 else:  
 self.station\_output\_text.append(f"{key}: {value:.4f}")  
  
 self.station\_output\_text.append("\nСимуляция модели (с ограниченной очередью):")  
 for key, value in results\_limited.items():  
 if isinstance(value, int):  
 self.station\_output\_text.append(f"{key}: {value}")  
 else:  
 self.station\_output\_text.append(f"{key}: {value:.4f}")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = MainWindow()  
 window.show()  
 sys.exit(app.exec())

## **Результаты:**





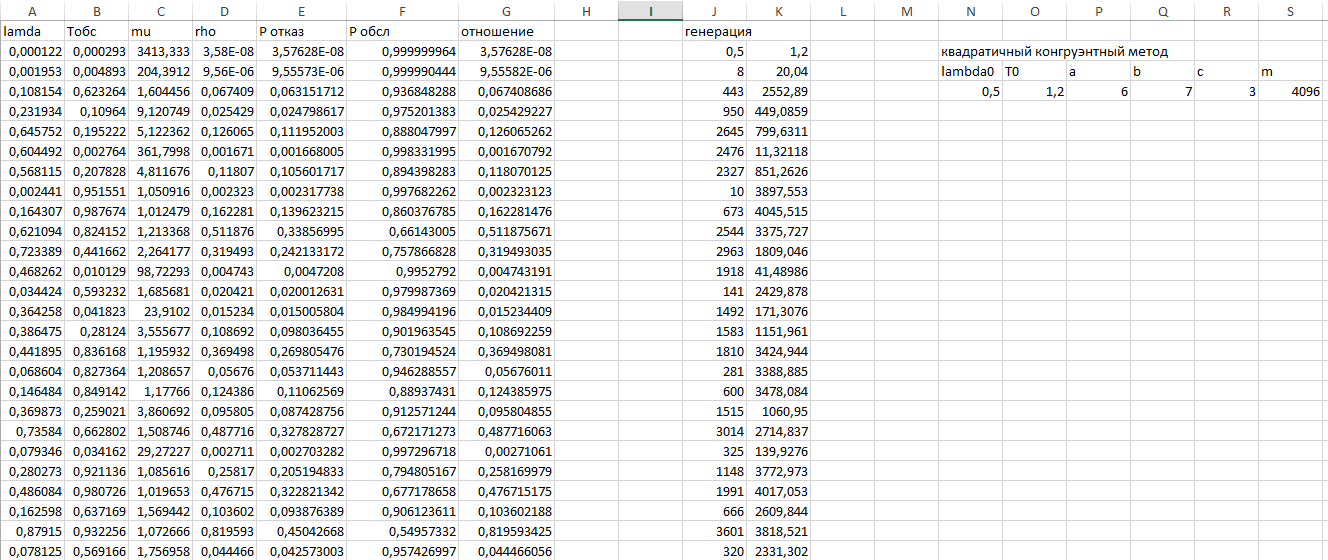




# **Решение в Excel**

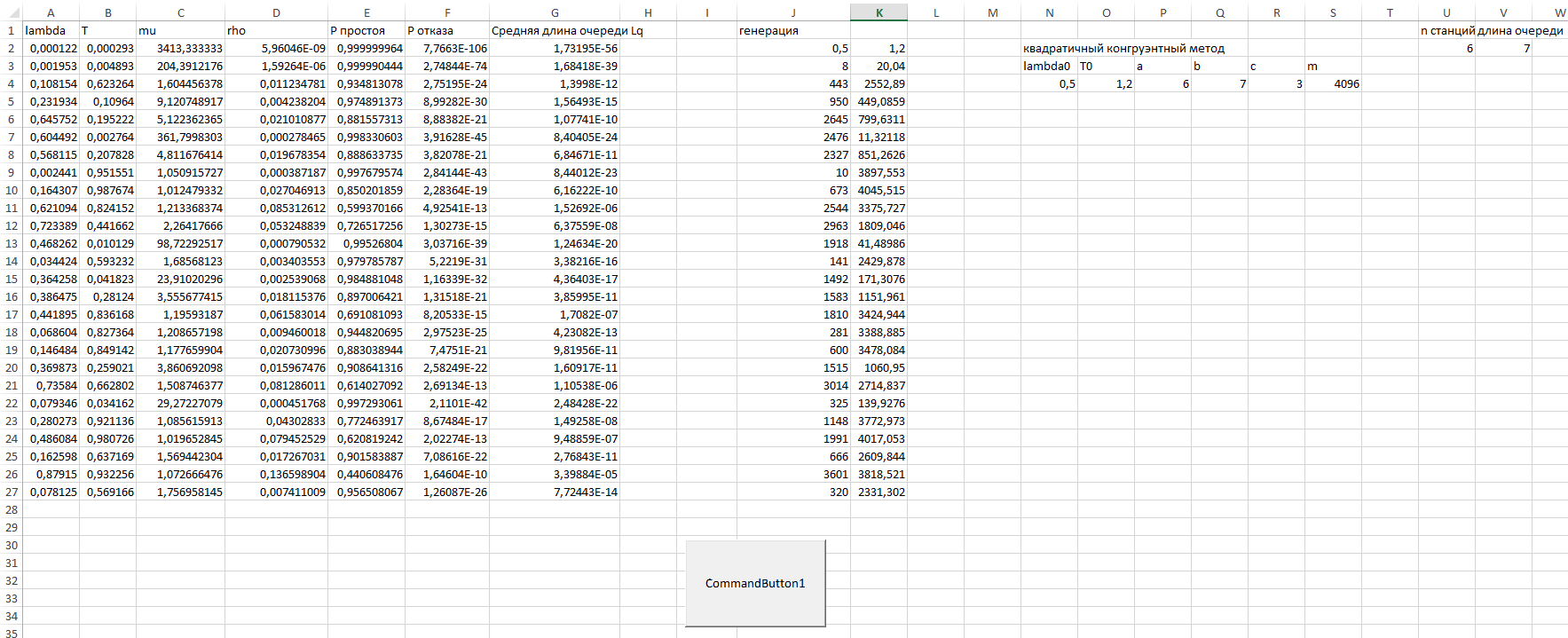
Решения заданий в Excel являются аналитическими решениями для двух задач. lambda и Tобс получаются с помощью квадратичного конгруэнтного метода, а остальные значения рассчитываются с помощью формул, присутствующих в коде выше.

**Пост диагностики**



**АЗС**

Данная задача решена с помощью VBA, что значительно упрощает процесс: нет необходимости разрабатывать отдельные решения для случаев с ограниченной и неограниченной очередью или составлять сложные формулы в Excel*.*



Код на VBA:  
Private Sub CommandButton1\_Click()

Dim ws As Worksheet

Set ws = ActiveSheet

Dim i As Integer

For i = 2 To 27

Dim lambda As Double, T\_service As Double, mu As Double, rho As Double

Dim num As Integer, queue\_capacity As Variant

Dim P0 As Double, P\_reject As Double, Lq As Double

lambda = ws.Cells(i, 1).Value

T\_service = ws.Cells(i, 2).Value

mu = 1 / T\_service

ws.Cells(i, 3).Value = mu

num = ws.Cells(2, 21).Value

queue\_capacity = ws.Cells(2, 22).Value

rho = lambda / (num \* mu)

ws.Cells(i, 4).Value = rho

If rho >= 1 Then

ws.Cells(i, 5).Value = "Ошибка"

ws.Cells(i, 6).Value = "Ошибка"

ws.Cells(i, 7).Value = "Ошибка"

Else

Dim sum\_term As Double

Dim k As Integer

sum\_term = 0

For k = 0 To num - 1

sum\_term = sum\_term + (lambda / mu) ^ k / Factorial(k)

Next k

If IsEmpty(queue\_capacity) Then

P0 = 1 / (sum\_term + ((lambda / mu) ^ num) / (Factorial(num) \* (1 - rho)))

P\_reject = 0

Lq = (P0 \* (lambda / mu) ^ num \* rho) / (Factorial(num) \* (1 - rho) ^ 2)

Else

For k = num To (num + queue\_capacity)

sum\_term = sum\_term + ((lambda / mu) ^ k) / (Factorial(num) \* (num ^ (k - num)))

Next k

P0 = 1 / sum\_term

P\_reject = ((lambda / mu) ^ (num + queue\_capacity)) / (Factorial(num) \* (num ^ queue\_capacity)) \* P0

Lq = 0

For k = num + 1 To num + queue\_capacity

Lq = Lq + (k - num) \* ((lambda / mu) ^ k) / (Factorial(num) \* (num ^ (k - num))) \* P0

Next k

End If

ws.Cells(i, 5).Value = P0

ws.Cells(i, 6).Value = P\_reject

ws.Cells(i, 7).Value = Lq

End If

Next i

End Sub

Function Factorial(n As Integer) As Double

Dim result As Double, i As Integer

result = 1

For i = 1 To n

result = result \* i

Next i

Factorial = result

End Function

# **Вывод**

При использовании чисто аналитической модели на Python расчёты выполняются мгновенно по классическим формулам теории массового обслуживания, это даёт точные значения вероятностей и средних показателей, но применимо лишь к идеализированным случаям. С другой стороны, при моделировании с помощью SimPy строится действительно имитационная система, где генерируются случайные события, формируется очередь, теряются заявки при ограничениях и собираются статистические выборки — такой подход более гибок и ближе к реальной работе системы, однако требует значительно больше ресурсов и даёт результаты с некоторыми колебаниями, которые сглаживается лишь при очень большом времени моделирования. Поскольку для генерации случайных чисел мы использовали квадратичный конгруэнтный генератор — лучший из решений прошлой лабораторной работы 4 — генерация поступления машин и времени обслуживания может выглядеть не совсем наглядно.

# **Ссылка на github**

<https://github.com/TheRomaks/CSM-Labs>