Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 1
по курсу «Планирование эксперимента»

Студент Сушина А.Д.

Группа ИУ7-81б

Оценка (баллы) _____

Преподаватель Куров А.В,

Москва.
2021 г

Задание на лабораторную работу

Разработать имитационную модель функционирования СМО.
СМО представляет собой одноканальную разомкнутую систему (один генератор заявок и один обслуживающий аппарат). Буфер имеет бесконечную емкость.
В качестве исходных данных пользователь задает интенсивность поступления заявок и интенсивность обслуживания заявок. Программа должна выводить расчетную загрузку системы и фактическую, полученную по результатам моделирования. Пользователь должен иметь возможность задавать время моделирования.
Если параметры законов распределения отличны от интенсивности, то предусмотреть ввод интенсивностей с дальнейшим пересчетом в программе этих величин в параметры закона. В случае двухпараметрических законов пользователь задает интенсивность и ее разброс (среднеквадратическое отклонение).
Построить график зависимости выходного параметра (ср. время ожидания (пребывания) в зависимости от загрузки системы).
Предусмотреть наращивание системы путем добавления новых генераторов и обслуживающих аппаратов.
Вариант 17: Рэлея, Вейбулла с параметром 2

Теоретическая часть

Коэффициент загрузки СМО:

$$\rho=\frac{\lambda}{\mu},$$

где λ - интенсивность входящего потока заявок,
μ - интенсивность обслуживания.

Распределение Рэлея:

$$f(x,\sigma)=\frac{x}{\sigma^2}e^{\frac{-x^2}{2\sigma^2}},x\geq 0,\sigma>0$$

Распределение Вейбулла:

$$f(x)=\frac{a}{\lambda}(\frac{x}{\lambda})^{(k-1)}e^{-(x/\lambda)^k}$$

Реализация

На рисунке 1 представлен пример работы.

```
PS C:\iu7\sem8\experiment-planning\lab1> python main.py
Введите интенсивность прихода послетителей: 10
Введите интенсивность обработки: 11
Загрузка системы(расчетная):  0.9090909090909091
Время работы: 145.3155238083306
Среднее время ожидания:  21.720167162059255
Количество обработанных заявок 1000
PS C:\iu7\sem8\experiment-planning\lab1>
```

Рис 1. Пример работы программы

Эксперимент

Был проведен эксперимент для наблюдения зависимости среднего времени пребывания заявок в очереди от коэффициента загрузки. Полученный график показан на рисунке 2.

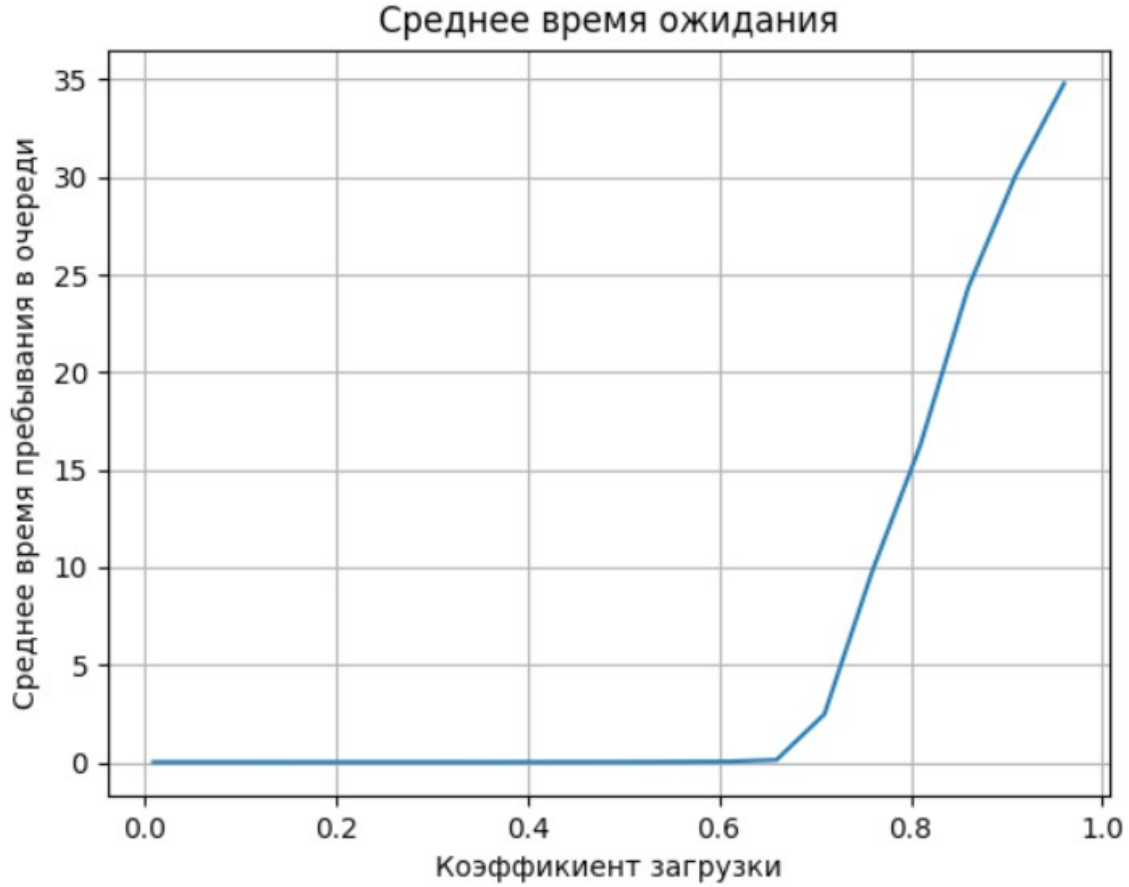

Рис 2. Зависимость времени ожидания в очереди от загрузки системы

Вывод

По результатам эксперимента можно сделать вывод, что среднее время пребывания заявок в системе возрастает с ростом загрузки.

Код программы

Листинг 1. Код файла main.py

```
from Modeller import Modeller
from EventGenerator import Generator
from Distributions import  RayleighDistribution, WeibullDistribution
from Processor import Processor
import math
from matplotlib import pyplot


def view():
   Xdata = list()
   Ydata = list()
```

```
    Ydata_t = list()

    lambda_obr = 100
    k = 2

    for lambda_coming in range(1, lambda_obr+1, 5):
        sigma = (1/lambda_coming) * (math.pi / 2) ** (-1/2)
        lam = (1/lambda_obr) * math.log(2, math.e) ** (-1 / k)

        generators = [
            Generator(
                RayleighDistribution(sigma),
                20000,
            ),
        ]

        operators = [
            Processor(
                WeibullDistribution(k, lam)
            ),
        ]
        for generator in generators:
            generator.receivers = operators.copy()

        model = Modeller(generators, operators)
        result = model.event_mode(12000)
        Xdata.append(lambda_coming/lambda_obr)
        Ydata.append(result['wait_time_middle'])
        # print(lambda_coming/lambda_obr)
        # print(result['wait_time_middle'])
        ro = lambda_coming/lambda_obr
        if ro != 1:
            Ydata_t.append(ro/(1 - ro)/lambda_coming)

    pyplot.title('Среднее время ожидания')
    pyplot.grid(True)
    # pyplot.plot(Xdata, Ydata_t)
    pyplot.plot(Xdata, Ydata)
    pyplot.xlabel("Коэффикиент загрузки")
    pyplot.ylabel("Среднее время пребывания в очереди")
    pyplot.show()



if __name__ == '__main__':
    clients_number = 10000 #Количество клиентов
    proccessed = 1000

    lambda_coming = float(input("Введите интенсивность прихода послетителей: "))
    lambda_obr = float(input("Введите интенсивность обработки: "))

    sigma = (1/lambda_coming) * (math.pi / 2) ** (-1/2)
```

```
   k = 2
   lam = (1/lambda_obr) * math.log(2, math.e) ** (-1 / k)

   generators = [
      Generator(
         RayleighDistribution(sigma),
         clients_number,
      ),
   ]

   operators = [
        Processor(
           WeibullDistribution(k, lam)
        ),
      ]
   for generator in generators:
      generator.receivers = operators.copy()

   model = Modeller(generators, operators)
   result = model.event_mode(proccessed)
   # print(result)
   print("Загрузка системы(расчетная): ", lambda_coming/lambda_obr,
   "\nВремя работы:", result['time'],
   "\nСреднее время ожидания: ", result['wait_time_middle'],
   "\nКоличество обработанных заявок", proccessed)
   # view()
```

Листинг 2. Код файла distributions.py

```
from numpy.random import rayleigh
from scipy.stats import weibull_min
import random
import numpy as np
import math

class RayleighDistribution:
   def __init__(self, sigma: float):
      self.sigma = sigma

   def generate(self):
      return rayleigh(self.sigma)

class WeibullDistribution:
   def __init__(self, k: float,lambd: float):
      self.k = k
      self.lam = lambd

   def generate(self):
      return weibull_min.rvs(self.k, loc=0, scale=self.lam)
```

Листинг 3. Код файла eventGenerator.py

```
class Generator:
```

```
    def __init__(self, generator, count):
        self._generator = generator
        self.receivers = []
        self.num_requests = count
        self.next = 0

    def next_time(self):
        return self._generator.generate()

    def generate_request(self, time):
        if self.num_requests <= 0:
            return None
        self.num_requests -= 1

        # Поиск обработчика с наименьшей очередью
        receiver_min = self.receivers[0]
        min = len(self.receivers[0].queue)
        for receiver in self.receivers:
            if len(receiver.queue) < min:
                min = len(receiver.queue)
                receiver_min = receiver
        receiver_min.receive_request(time)
        return receiver_min
```

Листинг 4. Код файла Processor

```
from EventGenerator import Generator
from numpy import random as nr

class Processor(Generator):
    def __init__(self, generator, max_queue=-1):
        self._generator = generator
        self.processed_requests = 0
        self.received_requests  = 0
        self.queue = []

        self.max_queue_size = max_queue
        self.max_reached_queue_size = 0

        self.next = 0

# обрабатываем запрос, если они есть
    def process_request(self, time):
        if len(self.queue) == 0:
            return -1

        task = self.queue.pop(0)
        wait_time = time - task
        # print(wait_time)

        self.processed_requests += 1

        return wait_time
```

```
# добавляем реквест в очередь
   def receive_request(self, time):
      # print(self.max_queue_size, self.current_queue_size )
      if self.max_queue_size != -1 and self.max_queue_size < len(self.queue):
         return False

      self.received_requests += 1
      self.queue.append(time)

      if self.max_reached_queue_size < len(self.queue):
         self.max_reached_queue_size = len(self.queue)
      return True

   def next_time(self):
      return self._generator.generate()
```

Листинг 5. Код файла Modeller

```
from Distributions import UniformDistribution
from EventGenerator import Generator
from Processor import Processor


class Modeller:
   def __init__(self, generators, operators):
      self._generators = generators
      self._operators = operators

   def event_mode(self, num_requests):
      refusals = 0
      processed = 0
      created = 0
      wait_times =  []

      for g in self._generators:
         g.next = g.next_time()

      self._operators[0].next = self._operators[0].next_time()

      blocks = self._generators + self._operators

      count = 0
      while processed <= num_requests:

         # находим наименьшее время
         current_time = self._generators[0].next
         for block in blocks:
            if 0 < block.next < current_time:
               current_time = block.next

         # для каждого из блоков
         for block in blocks:
            # если событие наступило для этого блока
```

```
      if current_time == block.next:
         if not isinstance(block, Processor):
            # для генератора
            # проверяем, может ли оператор обработать
            next_generator = block.generate_request(current_time)
            if next_generator is not None:
               next_generator.next = \
                  current_time + next_generator.next_time()
               created += 1
            else:
               refusals += 1
            block.next = current_time + block.next_time()
         else:
            wait_time = block.process_request(current_time)
            wait_times.append(wait_time)
            processed += 1
            if len(block.queue) == 0:
               block.next = 0
            else:
               block.next = current_time + block.next_time()

wait_time_middle = 0
for time in wait_times:
   if (time != -1):
      wait_time_middle += time

wait_time_middle /= len(wait_times)

return {
      'time': current_time,
      "wait_time_middle": wait_time_middle
      }
```