|  |  |
| --- | --- |
|  | **Министерство образования и науки 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) |

ОТЧЕТ

По лабораторной работе №1

По курсу «Планирование эксперимента»

Студент: Иванов Д. М.

Группа: ИУ7-81

Преподаватель: Куров А.В.

Москва, 2021.

Цель работы

Разработать имитационную модель функционирования одноканальной разомкнутой СМО с одним типом заявок. Буфер бесконечный. Для законов распределения с двумя параметрами предусмотреть возможность ввода только самих параметров. Для законов распределения с одним параметром:

* предусмотреть возможность ввода интенсивности;
* предусмотреть возможность ввода параметра распределения.
* предусмотреть возможность добавления генераторов

Законы распределения:

|  |  |
| --- | --- |
| Генератор | Обслуживающий аппарат |
| Экспоненциальный | Нормальный |

1. Аналитическая часть

Коэффициент загрузки СМО:

𝜌=𝜆/𝜇,

где λ - интенсивность входящего потока заявок,

μ - интенсивность обслуживания.

При загрузке меньше 1, достижимо устойчивое состояние и вычислимо среднее время ожидания заявок. При загрузке, превышающей 1, устойчивое состояние становится недостижимым, в следствие чего данные, полученные экспериментально, теряют физический смысл.

Среднее время ожидания должно устремляться к бесконечности при загрузке, превышающей единицу.

1. Технологическая часть

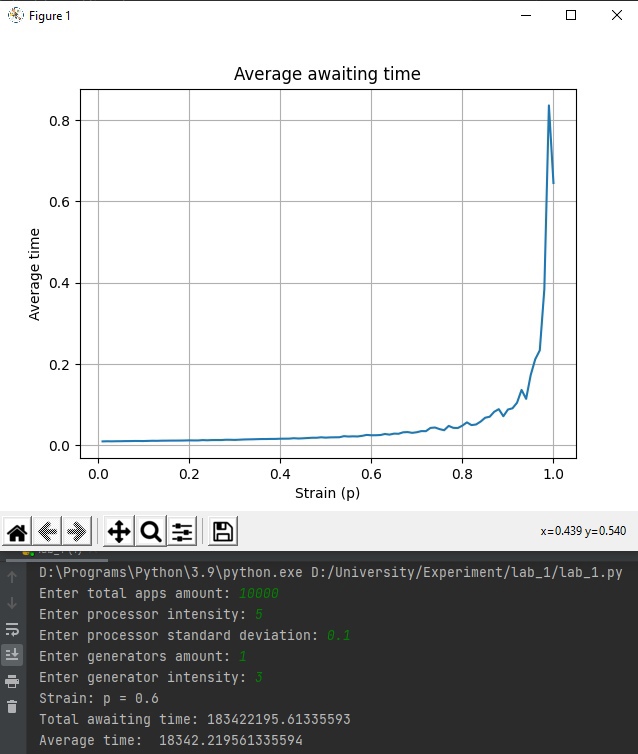
Программа выполнена на языке Python. Для получения законов распределения использовалась библиотека numpy. Для вывоа графика была использована библиотека matplotlib.

Листинг 1. Исходный код программы.

from numpy.random import exponential, normal  
from matplotlib import pyplot  
  
def exp\_by\_intensity(params):  
 return exponential(1 / params[0])  
  
  
def norm\_by\_intensity(params):  
 res = -1  
 while res < 0:  
 res = normal(1 / params[0], 1 / params[1])  
 return res  
  
  
class Generator:  
 def \_\_init\_\_(self, func, params):  
 self.law = func  
 self.params = params  
  
 def generation\_time(self):  
 return self.law(self.params)  
  
  
class EventModel:  
 def \_\_init\_\_(self, generators, processor, total\_apps=0):  
 self.generators = generators  
 self.processor = processor  
 self.total\_apps = total\_apps  
  
 def proceed(self):  
 processed = 0  
 self.queue = []  
 self.events = []  
 self.totally\_waited = 0  
 i = 0  
 for generator in self.generators:  
 self.events.append([generator.generation\_time(), 'g', 0])  
 i += 1  
 self.free = True  
 while processed < self.total\_apps:  
 event = self.events.pop(0)  
 if event[1] == 'g':  
 self.\_generate(event)  
 elif event[1] == 'p':  
 processed += 1  
 self.\_process(event[0])  
  
 return self.totally\_waited  
  
 def \_add\_event(self, event: list):  
 i = 0  
 while i < len(self.events) and self.events[i][0] < event[0]:  
 i += 1  
 self.events.insert(i, event)  
  
 def \_generate(self, event):  
 self.queue.append(event[0])  
 self.\_add\_event([event[0] + self.generators[event[2]].generation\_time(), 'g', event[2]])  
 if self.free:  
 self.\_process(event[0])  
  
 def \_process(self, time):  
 if len(self.queue) > 0:  
 processing\_time = self.processor.generation\_time()  
 self.totally\_waited += processing\_time + time - self.queue.pop(0)  
 self.\_add\_event([time + processing\_time, 'p'])  
 self.free = False  
 else:  
 self.free = True  
  
  
def view(total\_apps, proc\_dev):  
 max\_int = 100  
 proc = Generator(exp\_by\_intensity, (max\_int, proc\_dev))  
 Xdata = list()  
 Ydata = list()  
  
 for intensity in range(1, max\_int + 1):  
 gen = Generator(exp\_by\_intensity, (intensity,))  
 model = EventModel([gen], proc, total\_apps)  
 Xdata.append(intensity / max\_int)  
 Ydata.append(model.proceed() / total\_apps)  
  
 pyplot.title('Average awaiting time')  
 pyplot.grid(True)  
 pyplot.plot(Xdata, Ydata)  
 pyplot.xlabel("Strain (p)")  
 pyplot.ylabel("Average time")  
 pyplot.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 total\_apps = int(input("Enter total apps amount: "))  
 if total\_apps < 1:  
 print("Input error.\n")  
 exit(1)  
  
 proc\_int = float(input("Enter processor intensity: "))  
 proc\_dev = float(input("Enter processor standard deviation: "))  
  
 gen\_amount = int(input("Enter generators amount: "))  
 gens = []  
 gen\_int\_sum = 0  
 for i in range(gen\_amount):  
 gen\_int = float(input("Enter generator intensity: "))  
 gens.append(Generator(exp\_by\_intensity, (gen\_int,)))  
 gen\_int\_sum += gen\_int  
  
 proc = Generator(norm\_by\_intensity, (proc\_int, proc\_dev))  
 model = EventModel(gens, proc, total\_apps)  
  
 time = model.proceed()  
 print("Strain: p =", gen\_int\_sum / proc\_int, "\nTotal awaiting time:", time, "\nAverage time: ", time / total\_apps)  
  
 view(total\_apps, proc\_dev)

1. Экспериментальная часть

На рисунке 1 показан график, являющийся результатом работы программы после обработки 10000 заявок.

  
Рисунок 1. Результат эксперимента