МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и прикладной математики
Кафедра прикладной математики и экономико-математических методов

ОТЧЁТ

по дисциплине:

«Имитационное моделирование»

на тему:

«Одноканальная модель СМО с бесконечной очередью. Задание №4»

Направление: 01.03.02
Обучающийся: Бронников Егор Игоревич
Группа: ПМ-1901

Санкт-Петербург
2022

Задание

Реализовать одноканальную СМО на языке программирования Python.

Описание модели

СМО с бесконечной очередью – это СМО, в которой всегда есть места в очереди и если требование приходит, в момент, когда обслуживающее устройство занято, то оно не получает немедленного отказа, а может стать в очередь и ожидать освобождения обслуживающего устройства.

На вход одноканальной СМО с бесконечной очередью поступает пуассоновский поток требований с интенсивностью λ.
Интенсивность пуассоновского потока обслуживания – μ.
Дисциплина очереди естественная: кто раньше пришёл, тот раньше и обслуживается.
Число мест в очереди не ограничено.

Реализация модели

Данная модель была реализована на языке программирования Python. Для начала был создан класс *QueuingSystemWithInfinityQueue*, в котором реализованы все необходимые методы.

Данная функция принимает на вход параметр λ, параметр μ и время моделирования системы в условных единицах. (Рисунок 1)

```
class QueuingSystemWithInfinityQueue:
    def __init__(self, *, lambda_: float, mu: float, simulation_time: int):
        self.lambda_ = lambda_                       # интенсивность потока требований
        self.mu = mu                                 # интенсивность обслуживания требований
        self.simulation_time = simulation_time       # время моделирования системы

        self.requirements = []                       # время поступления нового требования
        self.execute_services = []                   # время обслуживания конкретного требования
        self.end_services = []                       # время конца обслуживания конкретного требования
        self.queue = {}                              # очередь на момент подачи i-го требования
```

Рис. 1: Параметры модели

Был реализован метод *generate_requirements*, который генерирует поток требований в соответствии с пуассоновским законом распределения. (Рисунок 2)

```python
def generate_requirements(self) -> List[float]:
    last_requirements_time = 0
    while last_requirements_time < self.simulation_time:
        arrival_time = np.random.exponential(self.lambda_)
        last_requirements_time += arrival_time
        self.requirements.append(last_requirements_time)
    return self.requirements
```

Рис. 2: Реализация метода генерации требований

Был реализован метод *get_service_times*, который в соответствии с пуассоновским законом задаёт каждому требованию его время обработки. (Рисунок 3)

```python
def get_service_times(self) -> List[float]:
    for _ in range(len(self.requirements)):
        service_time = np.random.exponential(self.mu)
        self.execute_services.append(service_time)
    return self.execute_services
```

Рис. 3: Реализация метода задания обработки требований

Был реализован метод *get_service_end*, который считает сколько времени провело требование в системе. (Рисунок 4)

```python
def get_service_end(self) -> List[float]:
    self.end_services.append(self.requirements[0] + self.execute_services[0])
    for requirement in range(len(self.requirements) - 1):
        value = self.requirements[requirement + 1] + self.execute_services[requirement + 1]
        if self.requirements[requirement + 1] < self.end_services[requirement]:
            value += self.end_services[requirement] - self.requirements[requirement + 1]
        self.end_services.append(value)
    return self.end_services
```

Рис. 4: Реализация метода подсчёт времени требования в системе

Был реализован метод *get_queue*, который для каждого требования сопоставляет количество требований, который находятся перед ним в очереди. (Рисунок 5)

```python
def get_queue(self) -> Dict[int, int]:
    self.queue = {}
    for requirement in range(1, len(self.requirements)):
        self.queue[requirement + 1] = (self.requirements[requirement] < np.array(self.end_services[:requireme
    return self.queue
```

Рис. 5: Реализация метода нахождения количества требований в очереди перед текущим требованием

Был реализован метод *get_features*, который рассчитывает основные характеристики модели. (Рисунок 6)

```python
def get_features(self):
    number_of_requirements = len(self.requirements)          # количество требований
    average_service_time = np.mean(self.execute_services)    # среднее время обслуживания
    average_time_spent_in_system = (np.array(self.end_services) - np.array(self.requirements)).mean()    # ср
    middle_length_queue = np.mean(list(self.queue.values()))                          # средняя длина очереди
    requirements_per_unit_of_time = number_of_requirements / self.end_services[-1]    # количество требований
    arriving_per_unit_of_time = number_of_requirements / self.requirements[-1]        # количество требований

    return {"Number of requirements": number_of_requirements,
            "Average service time": average_service_time,
            "Average time spent in system": average_time_spent_in_system,
            "Middle length queue": middle_length_queue,
            "Requirements per unit of time": requirements_per_unit_of_time,
            "Arriving requirements per unit of time": arriving_per_unit_of_time}
```

Рис. 6: Реализация метода нахождения характеристик модели

Характеристики модели:

1. количество требований;
2. среднее время обслуживания;
3. среднее время пребывания в системе требования;
4. средняя длина очереди;
5. количество требований обслуженных в единицу времени;
6. количество требований поступающих в единицу времени;

Если задать параметры модели: $\lambda = 1$, $\mu = 2$, $simulation\_time = 100$, то получаются следующие результаты. (Рисунок 7)

```python
lambda_ = 1
mu = 2
simulation_time = 100

qs = QueuingSystemWithInfinityQueue(lambda_=lambda_,
                                    mu=mu,
                                    simulation_time=simulation_time)
```

```python
requirements = qs.generate_requirements()
service_time = qs.get_service_times()
service_end = qs.get_service_end()
queue = qs.get_queue()

qs.get_features()
```

```
{'Number of requirements': 116,
 'Average service time': 2.2798619976439727,
 'Average time spent in system': 84.5850848144408,
 'Middle length queue': 37.904347826086955,
 'Requirements per unit of time': 0.43519544141659766,
 'Arriving requirements per unit of time': 1.1598994853157558}
```

Рис. 7: Результаты модели при $\lambda = 1$, $\mu = 2$, $simulation\_time = 100$