



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»

ОТЧЕТ
по лабораторной работе № 4
по курсу «Моделирование»
на тему: «Обслуживающий аппарат»
Вариант № 1

Студент ИУ7-71Б
(Группа)

(Подпись, дата)

Мицевич М. Д.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Рудаков И. В.
(И. О. Фамилия)

2022 г.

1 Теоретический раздел

1.1 Равномерное распределение

Равномерное распределение – распределение случайной величины, принимающей значения, принадлежащие некоторому промежутку конечной длины, характеризующееся тем, что плотность вероятности на этом промежутке всюду постоянна. Функция равномерного распределения представлена формулой 1.1.

$$F(x) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & x \geq b \end{cases} \quad (1.1)$$

Функция плотности равномерного распределения представлена формулой 1.2.

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & (x < a) \text{ or } (x > b) \end{cases} \quad (1.2)$$

1.2 Экспоненциальное распределение

Экспоненциальное распределение является частным случаем гамма распределения с параметрами $a = 1$ и $b = \frac{1}{\lambda}$. Функция экспоненциального распределения представлена формулой 1.3.

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (1.3)$$

Функция плотности экспоненциального распределения представлена формулой 1.4.

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (1.4)$$

1.3 Пошаговый принцип (Δt)

Этот принцип заключается в последовательном анализе состояний всех блоков системы в момент $t + \Delta t$. При этом новое состояние блоков определяется в соответствии с их алгоритмическим описанием.

Недостаток: значительные временные затраты на реализацию моделирования системы. А также при недостаточно малом Δt отдельные события в системе могут быть пропущены, что может повлиять на адекватность результатов.

1.4 Событийный принцип

Состояние отдельных устройств изменяются в дискретные моменты времени, совпадающие с моментами времени поступления сообщений в систему, временем окончания обработки задачи и т.д.

При использовании событийного принципа состояние всех блоков системы анализируется лишь в момент проявления какого-либо события. Моменты наступления следующего события определяются минимальным значением из списка событий.

2 Практическая часть

На листинге 2.1 представлен код генератора событий.

Листинг 2.1 – Генератор событий

```
class RequestGenerator:
    def __init__(self, generator):
        self._generator = generator
        self._receivers = set()

    def add_receiver(self, receiver):
        self._receivers.add(receiver)

    def remove_receiver(self, receiver):
        try:
            self._receivers.remove(receiver)
        except KeyError:
            pass

    def next_time_period(self):
        return self._generator.next()

    def emit_request(self):
        for receiver in self._receivers:
            receiver.receive_request()
```

На листинге 2.2 представлен код обслуживающего аппарата.

Листинг 2.2 – Обслуживающий аппарат

```
class RequestProcessor(RequestGenerator):
    def __init__(self, generator, reenter_probability=0):
        super().__init__(generator)
        self._generator = generator
        self._current_queue_size = 0
        self._max_queue_size = 0
        self._processed_requests = 0
        self._reenter_probability = reenter_probability
        self._reentered_requests = 0

    @property
    def processed_requests(self):
        return self._processed_requests
```

```

@property
def max_queue_size(self):
    return self._max_queue_size

@property
def current_queue_size(self):
    return self._current_queue_size

@property
def reentered_requests(self):
    return self._reentered_requests

def process(self):
    if self._current_queue_size > 0:
        self._processed_requests += 1
        self._current_queue_size -= 1
        self.emit_request()
        if nr.random_sample() < self._reenter_probability:
            self._reentered_requests += 1
            self.receive_request()

def receive_request(self):
    self._current_queue_size += 1
    if self._current_queue_size > self._max_queue_size:
        self._max_queue_size += 1

def next_time_period(self):
    return self._generator.next()

```

На листинге 2.3 представлен код генераторов равномерного и экспоненциального распределений.

Листинг 2.3 – Генераторы

```

class UniformGenerator:
    def __init__(self, a, b):
        if not 0 <= a <= b:
            raise ValueError('Параметры должны удовлетворять условию 0 <= a <= b')
        self._a = a
        self._b = b

```

```

def next(self):
    return nr.uniform(self._a, self._b)

class ExponentialGenerator:
    def __init__(self, lmbd):
        self._lambda = 1 / lmbd

    def next(self):
        return nr.exponential(self._lambda)

```

На листинге 2.4 представлен код пошагового принципа.

Листинг 2.4 – Пошаговый принцип

```

def time_based_modelling(self, request_count, dt):
    generator = self._generator
    processor = self._processor

    gen_period = generator.next_time_period()
    proc_period = gen_period + processor.next_time_period()
    current_time = 0
    while processor.processed_requests < request_count:
        if gen_period <= current_time:
            generator.emit_request()
            gen_period += generator.next_time_period()
        if current_time >= proc_period:
            processor.process()
            if processor.current_queue_size > 0:
                proc_period += processor.next_time_period()
            else:
                proc_period = gen_period + processor.
                    next_time_period()
        current_time += dt

    return (processor.processed_requests, processor.
        reentered_requests,
        processor.max_queue_size, current_time)

```

На листинге 2.5 представлен код событийного принципа.

Листинг 2.5 – Событийный принцип

```

def event_based_modelling(self, request_count):
    generator = self._generator

```

```

processor = self._processor

gen_period = generator.next_time_period()
proc_period = gen_period + processor.next_time_period()
while processor.processed_requests < request_count:
    if gen_period <= proc_period:
        generator.emit_request()
        gen_period += generator.next_time_period()
    if gen_period >= proc_period:
        processor.process()
        if processor.current_queue_size > 0:
            proc_period += processor.next_time_period()
        else:
            proc_period = gen_period + processor.
                next_time_period()

return (processor.processed_requests, processor.
        reentered_requests,
        processor.max_queue_size, proc_period)

```

Пример работы системы с использованием пошагового принципа представлен на рисунке 2.1.

СМО с очередью

Генератор

a 1

b 5

ОА

λ 0.5

Система

Количество заявок 10000

Вероятность повторной обработки заявки 0.5

Метод моделирования Δt

Δt 0.01

Результаты

Количество обработанных заявок 10000

Количество повторно обработанных заявок 5028

Максимальная длина очереди 1722

Время работы 20041.98

Моделировать

Рисунок 2.1 – Пошаговый принцип

Пример работы системы с использованием событийного принципа представлен на рисунке 2.2.

СМО с очередью

Генератор

a 1

b 5

ОА

λ 0.5

Система

Количество заявок 10000

Вероятность повторной обработки заявки 0.5

Метод моделирования Событийный ▾

Δt 0.01

Результаты

Количество обработанных заявок 10000

Количество повторно обработанных заявок 5029

Максимальная длина очереди 1514

Время работы 19429.29

Моделировать

Рисунок 2.2 – Событийный принцип