

Математическое моделирование. Лабораторная работа 3.

Выполнил: Клебанов Д.А, группа 853501, Вариант 10

```
In [1]: from math import factorial as fact
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sts
import seaborn as sns
```

Задание

Рассматривается простейшая одноканальная СМО с ограниченной очередью $m = 2$; работающий канал может иногда выходить из строя (отказывать). Заявка, которая обслуживается в момент отказа канала, становится в очередь, если в ней еще есть свободные места; если нет, она покидает СМО необслуженной. Интенсивность потока заявок X , потока обслуживания Y , потока отказов канала Z , потока восстановлений (ремонтов) R . Перечислить состояния СМО и найти для них финальные вероятности, характеристики эффективности СМО: относительную и абсолютную пропускные способности системы, среднее время пребывания заявки в системе $X = 2$, $Y = 1$, $Z = 0,5$, $R = 1$.

```

In [2]: gSystem:
elf, n, m, lmd, mu, nu, Z, R):

    lmd
    mu
    nu

    = 0
    = 0
    e_log = [(0, 0), ]

    rd_cnt = 0
    e_cnt = 0
    rd_times = []
    ard_times = []
    n = 0

    m_stat_log = []

    self):
    array(self._state_log)

    nt(self):
    (self.backward_times)

    d_intensity(self):
    f.lmd

    rd_intensity(self):
    (self.state, self.n) * self.mu + max(self.state - self.n, 0) * self.nu

    lta_by_intensity(self, intensity):
    lf.state, self.broken, intensity)
    ty == 0:
        np.inf
    random.exponential(1.0 / intensity)

    rward(self, time_delta):
    += time_delta
    rd_cnt += 1
    ate < self.n + self.m:
    tate += 1
        self.broken:
    lf.forward_times.append(self.time)

    efuse_cnt += 1

    ckward(self, time_delta):
        -= 1
    += time_delta
    ard_times.append(self.time)

    ent(self):
    lf.state, self.broken)
    ate == 3 and self.broken:
    RuntimeError()
    lf.state, self.broken)
    forward = self.get_timedelta_by_intensity(self.get_forward_intensity())
    backward = self.get_timedelta_by_intensity(self.get_backward_intensity() if not self.broken else 0)
    forward_broken = self.get_timedelta_by_intensity(self.Z if not self.broken else 0)
    backward_broken = self.get_timedelta_by_intensity(self.R if self.broken else 0)

    ta_forward <= timedelta_backward and timedelta_forward <= timedelta_forward_broken and timedelta_forwa
    rocess_forward(timedelta_forward)
    elta_backward <= timedelta_forward and timedelta_backward <= timedelta_forward_broken and timedelta_b
    rocess_backward(timedelta_backward)
    elta_forward_broken <= timedelta_backward and timedelta_forward_broken <= timedelta_forward and timede
    = 1
    rocess_forward(timedelta_forward_broken)
    elta_backward_broken <= timedelta_backward and timedelta_backward_broken <= timedelta_forward_broken
    rocess_backward(timedelta_backward_broken)

```

```

= 0

RuntimeError()

e_log.append((self.state, self.time))

lf.lmd + self.Z) / self.mu

:
f.nu / self.mu

plier(self, i):
f.p ** i / fact(i)

inator_product(self, i):
prod([(self.n + 1 * self.beta) for l in range(1, i + 1)])

l_final_probabilities(self):
euingSystem(1, 2, 2 + 0.25, 1, 0).theoretical_final_probabilities)
ueuingSystem(1, 2, 2 + 0.25, 1, 0).theoretical_final_probabilities
s = [self._tfp_multiplier(i) for i in range(self.n + 1)]
rt = sum([self.p ** i / self._tfp_denominator_product(i) for i in range(1, self.m + 1)])
(sum(multipliers) + multipliers[self.n] * p0_rest_part)

= list(map(lambda x: x * p0, multipliers))
art = [p0 * multipliers[self.n] * self.p ** i / self._tfp_denominator_product(i) for i in range(1, self.m + 1)]
t(ans_n_part) + ans_rest_part

final_probabilities(self):
p.unique(self.state_log[:, 0], return_counts=True)[1]
nts / np.sum(counts)

l_absolute_bandwidth(self):
lf.lmd + self.Z) * (1.0 - self.theoretical_final_probabilities[self.n + self.m])

absolute_bandwidth(self):
lf.lmd + self.Z) * self.backward_cnt / self.forward_cnt

l_relative_bandwidth(self):
- self.theoretical_final_probabilities[self.n + self.m]

relative_bandwidth(self):
f.backward_cnt / self.forward_cnt

l_average_orders_in_system(self):
([self.n * self.theoretical_final_probabilities[self.n + i] for i in range(1, self.m + 1)]) \
[i * self.theoretical_final_probabilities[i] for i in range(1, self.n + 1)])

average_orders_in_system(self):
sum(list(map(lambda x: min(x, self.n), self.state_log[:, 0]))) / self.state_log.shape[0]

l_average_orders_in_queue(self):
([i * self.theoretical_final_probabilities[self.n + i] for i in range(1, self.m + 1)])

average_orders_in_queue(self):
sum(list(map(lambda x: max(x - self.n, 0), self.state_log[:, 0]))) / self.state_log.shape[0]

l_average_order_time_in_system(self):
f.theoretical_average_orders_in_queue / (self.lmd + self.Z) + self.theoretical_absolute_bandwidth / (self.lmd + self.Z)

average_order_time_in_system(self):

```

```
f.empirical_average_orders_in_queue / (self.lmd + self.Z) + self.empirical_absolute_bandwidth / (self
```

```
In [3]: def emulate(ax, n, m, lmd, mu, nu, Z, R, param_name="", value=None):
n_iter = 10000
system = CustomQueueingSystem(n, m, lmd, mu, nu, Z, R)
for _ in range(n_iter):
    system.process_event()

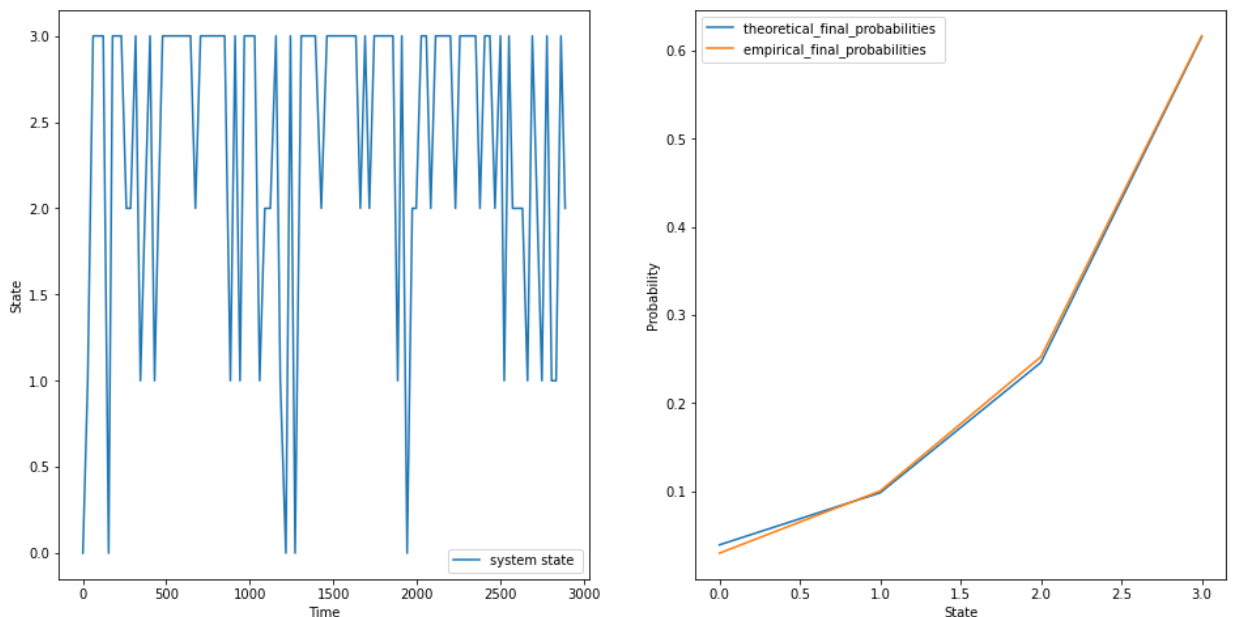
sampled_state_log = system.state_log[:, len(system._state_log) // 100]

emulate_name = ""
if value is not None:
    emulate_name = f'{param_name}={value}'

ax[0].plot(sampled_state_log[:, 1], sampled_state_log[:, 0], label=f'system state {emulate_name}')
ax[0].set_xlabel('Time')
ax[0].set_ylabel('State')
ax[0].legend()

x_values = np.arange(system.n + system.m + 1)
for name in ('theoretical_final_probabilities', 'empirical_final_probabilities'):
    ax[1].plot(x_values, getattr(system, name), label=name + ' ' + emulate_name)
ax[1].set_xlabel('State')
ax[1].set_ylabel('Probability')
ax[1].legend()
return system

# plt.figure()
_, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
system = emulate(ax, 1, 2, 2, 1, 0, 0.5, 1)
plt.show()
```



```
In [4]: print(f'Theoretical absolute bandwidth: {system.theoretical_absolute_bandwidth}')
print(f'Empirical absolute bandwidth: {system.empirical_absolute_bandwidth}')
```

```
Theoretical absolute bandwidth: 0.960591133004926
Empirical absolute bandwidth: 0.9848062447727907
```

```
In [5]: print(f'Theoretical relative bandwidth: {system.theoretical_relative_bandwidth}')
print(f'Empirical relative bandwidth: {system.empirical_relative_bandwidth}')
```

```
Theoretical relative bandwidth: 0.3842364532019704
Empirical relative bandwidth: 0.39392249790911626
```

```
In [6]: print(f'Theoretical average order time in system: {system.theoretical_average_order_time_in_system}')  
        print(f'Empirical average order time in system: {system.empirical_average_order_time_in_system}')
```

```
Theoretical average order time in system: 0.9753694581280787  
Empirical average order time in system: 0.9883430558533218
```

```
In [ ]:
```

```
In [ ]:
```