Практическое занятие № 3

Тема: Расчет СМО M/G/1.

Цель: Приобретение практических навыков расчета показателей оперативности обработки данных с помощью системы массового обслуживания M/G/1.

Язык программирования, ПО и библиотеки: python 3.x, установленный пакет библиотек Anaconda или отдельно следующие: pandas, numpy, matplotlib. Среда разработки – PyCharm, Spyder 3 или Jypyter.

Порядок выполнения практического занятия:

1. Создайте новый файл с расширением *.py* в проекте.
2. Добавьте в папку с проектом файл *smo_im.py*.
3. Пропишите секцию import в следующем виде:

```
import smo_im
import rand_destribution as rd
import numpy as np
import matplotlib.pyplot as plt
```

4. В ходе практического занятия необходимо сравнить результаты расчетов среднего времени ожидания в системе, полученные по формуле Полячека-Хинчина, с результатами ИМ.

Оценку среднего времени ожидания в системе необходимо произвести в зависимости от коэффициента вариации времени обслуживания ν и коэффициента загрузки системы ρ. График зависимости среднего времени ожидания в системе от коэффициента вариации времени обслуживания необходимо получить при фиксированном значении коэффициента загрузки. Аналогично, для график зависимости среднего времени ожидания в системе от коэффициента загрузки необходимо построить при фиксированном значении коэффициента вариации времени обслуживания.

Фиксируемые значения коэффициентов вариации и загрузки системы определить согласно таблицы по номеру в журнале

	ρ						
		0.65	**0.7**	**0.75**	**0.8**	**0.85**	**0.9**
	0.3	1	2	3	4	5	6
	0.57	7	8	9	10	11	12
ν	**1.2**	13	14	15	16	17	18
	1.5	19	20	21	22	23	24

В ходе практического занятия необходимо:

– Реализовать функцию расчета среднего времени ожидания в системе по формуле Полячека-Хинчина;

– Вывести таблицы с зависимостями среднего времени ожидания в системе M/G/1 для ИМ и рассчитанные по формуле Полячека-Хинчина от коэффициента вариации времени обслуживания при фиксированном коэффициенте загрузки;

– Вывести аналогичные таблицы с зависимостями среднего времени ожидания в системе M/G/1 от коэффициента загрузки при фиксированном коэффициенте вариации времени обслуживания;

– Построить графики по полученным таблицам;

– Построить графики для относительных ошибок ИМ для обоих случаев.

Итоговый отчет должен быть представлен в виде программы. Секции отображения графиков должны быть готовы к отображению по просьбе принимающего. Возможен вариант предварительного формирования графиков – для этого используйте в конце программы вместо *plt.show()* метод *plt.savefig(«имя_файла.jpg»)* для сохранения графика в директорию проекта. При построении графиков необходимо выводить секцию с обозначениями кривых с помощью метода *legend()*

```
ax.plot(roes, w_im, label="ИМ")
ax.plot(roes, w_polychek, label="Числ")
plt.legend()
plt.show()
```

Будьте готовы ответить на **контрольные вопросы** по практическому занятию:

1) Каким образом устроена нотация Кендалла?
2) Какую модель СМО вы исследовали?

3) Как зависит среднее временя ожидания заявок в системе от коэффициента загрузки?

4) Как зависит среднее временя ожидания заявок в системе от коэффициента вариации времени обслуживания?

5) Как зависит точность оценок, полученных с помощью ИМ, от числа обработанных заявок?

6) Как зависит точность оценок от коэффициента загрузки системы?

7) Напишите формулу Полячека-Хинчина. Что вы можете сказать о характере зависимости среднего времени ожидания от коэффициента загрузки и коэффициенте вариации времени обслуживания?

Как пользоваться ИМ

Для запуска ИМ нужно создать экземпляр класса SmoIm, передав ему количество каналов обслуживания *n*:

```
smo = smo_im.SmoIm(n)
```

Далее необходимо задать входной поток с помощью метода *set_sources()*, передав список в виде [*params, type*], где *params* – список с параметрами распределения, *type* – тип распределения, заданный в виде текста. Ниже приведен пример для пуассоновского распределения:

```
smo.set_sources(l, 'M')
```

Распределение времени обслуживания в канале задается аналогично, с помощью метода set_servers(). Ниже приведен пример для распределения Эрланга («E») и гиперэкспоненциального распределения второго порядка («H»).

Для подбора параметров в зависимости от среднего и коэффициента вариации можно воспользоваться методами *get_params_by_mean_and_coev* классов, реализующих данные распределения.

Ниже приведен пример задания каналов обслуживания СМО в зависимости от значения коэффициента вариации. При коэффициенте вариации меньше единицы выбирается распределение Эрланга, иначе – гиперэкспоненциальное распределение.

```
if coev < 1:
    params = rd.Erlang_dist.get_params_by_mean_and_coev(b1, coev)
    smo.set_servers(params, 'E')
```

```
else:
    params = rd.H2_dist.get_params_by_mean_and_coev(b1, coev, is_clx=False)
    smo.set_servers(params, 'H')
```

Чтобы узнать о поддерживаемых типах распределений (в том числе и распределений входного потока), ознакомьтесь с содержимым класса *Server*, содержащемся в файле *smo_im.py*. Ниже приведен его фрагмент:

```
class Server:

    """
    Канал обслуживания
    """
    id = 0

    def __init__(self, params, types):
        """
        params - параметры распределения
        types -  тип распределения
        """
        if types == "M":
            self.dist = rd.Exp_dist(params)
        elif types == "H":
            self.dist = rd.H2_dist(params)
        elif types == "E":
            self.dist = rd.Erlang_dist(params)
        elif types == "C":
            self.dist = rd.Cox_dist(params)
        elif types == "Gamma":
            self.dist = rd.Gamma(params)
        elif types == "Pa":
            self.dist = rd.Pareto_dist(params)
        else:
            raise SetSmoException("Неправильно задан тип распределения
сервера. Варианты M, H, E, C, Pa")
```

Далее необходимо запустить ИМ с помощью метода *run()*, передав количество подлежащих обслуживанию заявок. Начальные моменты времени ожидания можно получить, обратившись к полю ***w*** экземпляра класса. В частности, среднее для получения среднего времени ожидания необходимо вызвать *smo.w[0]*.

```
jobs_count = 100000
smo.run(jobs_count)
w_im.append(smo.w[0])
```

Ниже приведен код для накопления массивов средних времен ожидания, полученных с помощью ИМ и рассчитанных по формуле Полячека-Хинчина в

зависимости от коэффициента вариации времени обслуживания. Также производится накопление массива относительных ошибок ИМ *errors*

```
n = 1
l = 1.0

jobs_count = 1000000
ro_fix = 0.7
coev_fix = 1.2

coevs = np.linspace(0.3, 3, 15)
b1 = ro_fix / l
w_polyachek = []
w_im = []
errors = []

for i in range(len(coevs)):
    smo = smo_im.SmoIm(n)
    smo.set_sources(l, 'M')
    if coevs[i] < 1:
        params = rd.Erlang_dist.get_params_by_mean_and_coev(b1, coevs[i])
        smo.set_servers(params, 'E')
    else:
        params = rd.H2_dist.get_params_by_mean_and_coev(b1, coevs[i],
is_clx=False)
        smo.set_servers(params, 'H')

    smo.run(jobs_count)
    w_im.append(smo.w[0])
    w_polyachek.append(polyachek(l, coevs[i], b1))
    errors.append(100 * (w_im[i] - w_polyachek[i]) / w_polyachek[i])

print("Вариант __")
print("Среднее время ожидания в СМО от "
      "коэффициента вариации времени обслуживания\n")
print("{0:^15s}|{1:^15s}|{2:^15s}".format("coev", "ИМ", "Теор"))
print("-"*45)

for i in range(len(coevs)):
    print("{0:^15.3f}|{1:^15.3f}|{2:^15.3f}".format(coevs[i], w_im[i],
w_polyachek[i]))
```

Предполагается, что вами выше написана функция *polyachek()*, которая принимает на вход три параметра – интенсивность входного потока, коэффициент вариации и среднее время обслуживания и возвращает среднее время ожидания в системе. Аналогично следует накопить значения *w_im*, *w_polyachek, errors* в зависимости от коэффициента загрузки системы.

Построение графиков

Для построения графиков воспользуйтесь библиотекой matplotlib. После формирования массивов, содержащих средние времена пребывания заявок для имитационной модели *w_im* и рассчитанных теоретически *w_polychek*, можно построить график следующим образом

```
fig, ax = plt.subplots()

ax.plot(coevs, w_im, label="ИМ")
ax.plot(coevs, w_polychek, label="Числ")

plt.legend()
plt.show()
```

В данном случае *coevs* представляет собой массив значений коэффициентов вариации времени обслуживания. Ниже приведен пример автоматического заполнения массива 15 значениями в диапазоне от 0.3 до 3 с помощью метода *linspace* библиотеки *numpy*

```
coevs = np.linspace(0.3, 3, 15)
```

После формирования массива относительных ошибок *error*, соответствующий график можно построить так

```
fig, ax = plt.subplots()
ax.plot(coevs, error)
plt.show()
```

Аналогично следует построить график с зависимостью тех же параметров от коэффициента загрузки системы. Массив коэффициентов загрузки системы также можно сформировать с помощью метода *linspace* библиотеки *numpy*

```
roes = np.linspace(0.1, 0.95, 15)
```