Математическая статистика

Для удобства работы создайте текстовый файл с расширением **.py**, например, **script.py**, и записывайте команды в него.

Для запуска скрипта выполните в **cmd.exe** команду:

```
> python script.py
```

В данной лабораторной работе потребуются библиотеки **numpy** и **matplotlib**. Убедитесь, что они установлены, выполнив в консоли Python команду

```
help('modules')
```

Задание на лабораторную работу

1. Создать в Python переменную, массив, матрицу с заданными, случайными целочисленными, нулевыми, единичными значениями

Обычные переменные в Python создаются так:

```
a = 10
```

Python – динамически типизированный язык, поэтому переменные могут менять свой тип. Например, вот так можно объявить массив:

```
a = [1, 2, 3]
```

При работе с данными очень популярной библиотекой является библиотека NumPy. Она содержит огромное количество математических и статистических функций, функций-генераторов и т.д. Чтобы использовать библиотеку в своих скриптах, необходимо подключить её следующим образом (эта строка пишется в начале скрипта):

```
import numpy as np
```

Тот же массив **a** средствами NumPy можно теперь объявить так:

```
a = np.array([1, 2, 3])
```

Python так же позволяет создавать матрицы. Матрицы создаются как массивы, элементами которых являются другие массивы. Причём размер массивов в элементах может различаться, например:

```
a = [[1], [1, 2], [1, 2, 3]]
```

Следующий код демонстрирует, как создать матрицу, заполненную нулями, с помощью NumPy:

```
a = np.zeros((2, 3))
```

Здесь параметром функции является кортеж (2, 3) – размеры создаваемой матрицы. То есть, будет создана матрица, заполненная нулями, состоящая из 2 строк и 3 столбцов.

Аналогично можно создать матрицу, заполненную единицами, но для этого используется функция **np.ones**, в которую передаётся тот же параметр – размеры создаваемой матрицы.

Чтобы узнать количество строк и столбцов в NumPy-матрице, можно воспользоваться свойством **shape**. Оно возвращает кортеж из нескольких значений. Если мы работаем с двумерной матрицей, то свойство shape будет содержать 2 значения. Первое – количество строк в матрице, второе – количество столбцов:

```
a.shape[0]   # 2
a.shape[1]   # 3
```

Можно изменить размер матрицы, указав новые размеры. Например, если мы хотим из матрицы 2×3 сделать матрицу 1×6, нужно выполнить следующий код:

```
b = a.reshape((1, 6))
```

Или

```
b = a.reshape((1, -1))
```

-1 в данном случае говорит о том, что функция сама подберёт подходящее значение размера (в данном случае 6). Очевидно, что только по одному из измерений допускается установить **-1**.

Если нужно сгенерировать матрицу из определённых значений, например, **7**, можно воспользоваться таким приёмом – умножить каждый элемент единичной матрицы на константу **7**:

```
a = np.ones((2, 3)) * 7
```

Библиотека NumPy позволяет также генерировать числа, массивы и матрицы, заполненные случайными значениями. Например, следующий код генерирует матрицу размера 2×3 со случайными значениями от 0 до 1:

```
a = np.random.random((2, 3))
```

Если необходимо, чтобы значения лежали в диапазоне, например, от 10 до 15, необходимо растянуть диапазон в 5 раз и выполнить смещение на 10. Это можно выполнить так:

```
a = 5 * np.random.random((2, 3)) + 10
```

В некоторых случаях требуется генерировать целые числа. Это выполняется с помощью функции **np.random.randint**. Первые 2 параметра – минимальное и максимальное генерируемые значения. Причём наибольшим значением будет **максимальное - 1**. Третий параметр – размеры создаваемой матрицы. Если передать число, получится массив со случайными значениями, размер которого равен числу. Если передать кортеж (несколько чисел в скобках через запятую), получится матрица соответствующих размеров. Например, вот так можно сгенерировать матрицу размеров 5×5 из целых чисел от 2 до 5:

```
a = np.random.randint(2, 6, (5, 5))
```

Функции **random** и **randint** выдают числа с равномерным распределением. Если требуется сгенерировать числа с нормальным распределением, можно воспользоваться функцией **np.random.normal**. Первые два параметра этой функции – математическое ожидание и среднеквадратическое отклонение генерируемой матрицы. Третий параметр – размер генерируемой матрицы. Вот так можно сгенерировать массив из 1000 элементов в нормальным распределением, математическим ожиданием 3 и среднеквадратическим отклонением 5:

```
data = np.random.normal(3, 5, 1000)
```

2. Импортировать переменную (матрицу) из текстового структурированного файла (файл создать вручную: разделители столбцов – пробелы)

Чтобы импортировать переменную или матрицу из текстового файла, необходимо сначала создать этот файл. Создайте на диске текстовый файл, содержащий число, массив или матрицу. Например, файл **test.txt** с таким содержимым:

```
1 2 3
4 5 6
```

Чтобы загрузить его в матрицу NumPy, необходимо воспользоваться функцией **np.loadtxt**. Параметром функции является путь к файлу:

```
data = np.loadtxt('D:\\test.txt')
```

По умолчанию функция считает, что в файле хранятся вещественные числа. Можно явно указать, что там целые числа, задав параметр **dtype**:

```
data = np.loadtxt('D:\\test.txt', dtype=np.int32)
```

Иногда при работе с Python бывает удобно настроить рабочую директорию, чтобы не писать абсолютный путь к файлу. Для этого можно воспользоваться модулем **os**. Подключите его с помощью команды (пишется в начале скрипта):

```
import os
```

Чтобы задать текущий рабочий каталог, выполните команду (в качестве параметра указывается путь к новому рабочему каталогу):

```
os.chdir('D:')
```

Чтобы убедиться, что рабочий каталог изменился, выполните команду **os.getcwd** (**getcwd** = **get** **c**urrent **w**orking **d**irectory).

3. Загрузить одномерные данные из каталога data/1D из файла с расширением .mat согласно варианта. Оценить параметры одномерной случайной величины, используя команды NumPy: np.max, np.min, np.median, np.mean, np.var, np.std; Команды сохранить в скрипт.

Для загрузки данных из файлов с расширением **.mat** (это формат Matlab) следует воспользоваться функцией **scipy.io.loadmat**. Для этого в начале скрипта необходимо подключить соответствующий модуль:

```
import scipy.io
```

Далее выполняем загрузку одномерных данных:

```
data = scipy.io.loadmat('../data/1D/var1.mat')
```

В качестве параметра в функцию **loadmat** передаётся путь к файлу.

Возвращаемое функцией значение будет представлять собой **словарь Python**. Словарь – это пары «ключ»-«значение». В словаре data будет храниться разная информация: **__version__**, **__globals__**, **__header__** и сам массив с данными. Узнать, какие ключи хранятся в словаре data, можно с помощью функции **keys()**:

```
data.keys()
```

Например, если функция вернула значения

```
dict_keys(['__version__', '__globals__', 'n', '__header__'])
```

то исследуемые одномерные данные будут храниться по ключу **n**. Извлечём эти данные:

```
data = data['n']
```

Выполните расчёт минимального, максимального, медианы, математического ожидания, дисперсии, среднеквадратического отклонения, воспользовавшись функциями соответственно: **np.max**, **np.min**, **np.median**, **np.mean**, **np.var**, **np.std**.

4. Вывести графики одномерных случайных величин и их плотности распределения. Команды сохранить в скрипт. Подобрать вид распределения одномерной случайной величины и его параметры. На одном графике отобразить случайную величину, уровень среднего значения и дисперсию.

Для рисования различных графиков и диаграмм в Python имеется очень мощная библиотека **matplotlib**. Для её использования пропишите в начале скрипта:

```
import matplotlib.pyplot as plt
```

Чтобы нарисовать график, можно воспользоваться простейшим кодом:

```
plt.plot(data)
plt.show()
```

Первая функция выполняет построение графика, вторая выводит его на экран. Вот как может выглядеть получившийся график:

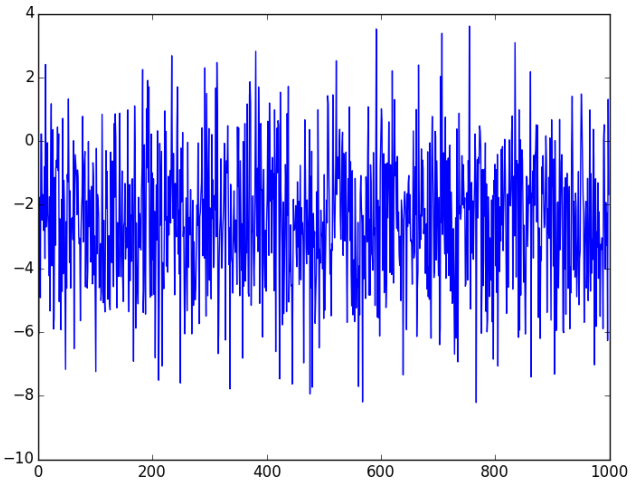

Рисунок 1 График массива, построенный с помощью функции plot

Функция **`plot`** обладает большим числом параметров. Можно управлять цветом и толщиной линий, задавать стиль линий и маркеров, выводить сразу несколько графиков и т.д. Можете самостоятельно ознакомиться с возможностями функции, выполнив в консоли Python команду

```
help('matplotlib.pyplot.plot')
```

или изучив примеры на официальном сайте библиотеки.

Ниже приведён фрагмент кода для рисования значений массива, среднего значения и дисперсии на одном графике:

```
mean = np.mean(data) * np.ones(len(data))
var = np.var(data) * np.ones(len(data))
plt.plot(data, 'b-', mean, 'r-', mean-var, 'g--', mean+var, 'g--')
plt.grid()
plt.show()
```

Получается следующий график:

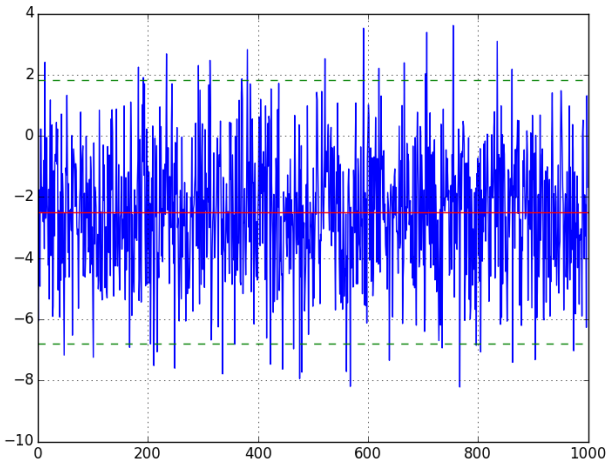

Рисунок 2 График массива с нанесёнными поверх линиями среднего и дисперсии

Функция **`plt.grid()`** отображает сетку на графике.

Ещё один из вариантов построения предыдущего графика – использование функции **`plt.hlines()`**.

Для построения гистограммы (закона распределения случайной величины) необходимо воспользоваться функцией **`plt.hist()`**. В функцию передаётся массив с данными и параметр **`bins`** – количество столбцов в гистограмме. Например, гистограмму как на рисунке ниже можно построить с помощью такого кода:

```
plt.hist(data, bins=20)
plt.grid()
plt.show()
```

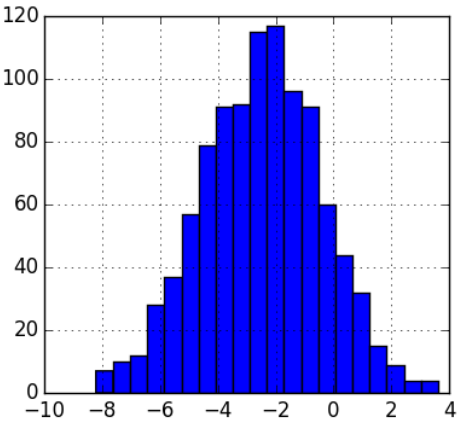

Рисунок 3 Гистограмма (закон распределения) массива данных

5. Построить и вывести на графике автокорреляцию заданной одномерной случайной величины. Команды сохранить в скрипт.

Для решения задачи можно воспользоваться функцией **`np.correlate`**. Но недостатком этой функции является то, что вместо коэффициента корреляции Пирсона она вычисляет альтернативный коэффициент корреляции, который не нормирован от -1 до -1.

Коэффициент корреляции Пирсона вычисляет функция **`np.corrcoef`**. В качестве результата она возвращает матрицу корреляции размера 2×2. На главной диагонали матрицы расположены единицы. Искомые коэффициенты корреляции соответствуют элементам матрицы с индексами **`[0, 1]`** и **`[1, 0]`** (так как матрица симметрична).

Ниже приведена реализация функции автокорреляции на основании функции **`np.corrcoef`**. Наберите код этой функции в своём скрипте.

```
def autocorrelate(a):
  n = len(a)
  cor = []
  for i in range(n//2, n//2+n):
    a1 = a[:i+1]   if i < n else a[i-n+1:]
    a2 = a[n-i-1:] if i < n else a[:2*n-i-1]
    cor.append(np.corrcoef(a1, a2)[0, 1])
  return np.array(cor)
```

Важное замечание

В функцию np.correlate должны передаваться одномерные массивы. При выполнении лабораторной работы возможна ситуация, когда массив data представляет собой одномерную матрицу, состоящую, например, из 1000 строк и 1 столбца. Проверить это можно с помощью свойства shape:

```
>>> data.shape
(1000, 1)
```

Чтобы превратить одномерную матрицу в одномерный массив, используйте функцию **`np.ravel`**:

```
>>> data = np.ravel(data)
>>> data.shape
(1000,)
```

После этого можно выполнять корреляцию и строить график. Пример кода приведён ниже:

```
cor = autocorrelate(data)
plt.plot(cor)
plt.show()
```

Вот примерный вид графика автокорреляции:

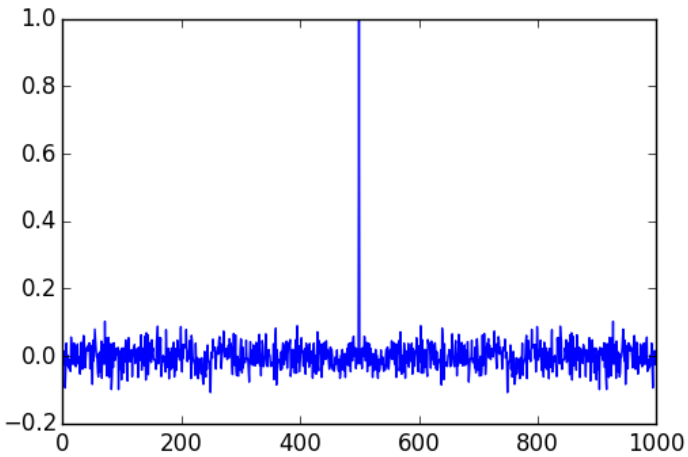

Рисунок 4 График автокорреляции

График имеет необычный вид. Рисунок ниже поясняет такую форму графика.

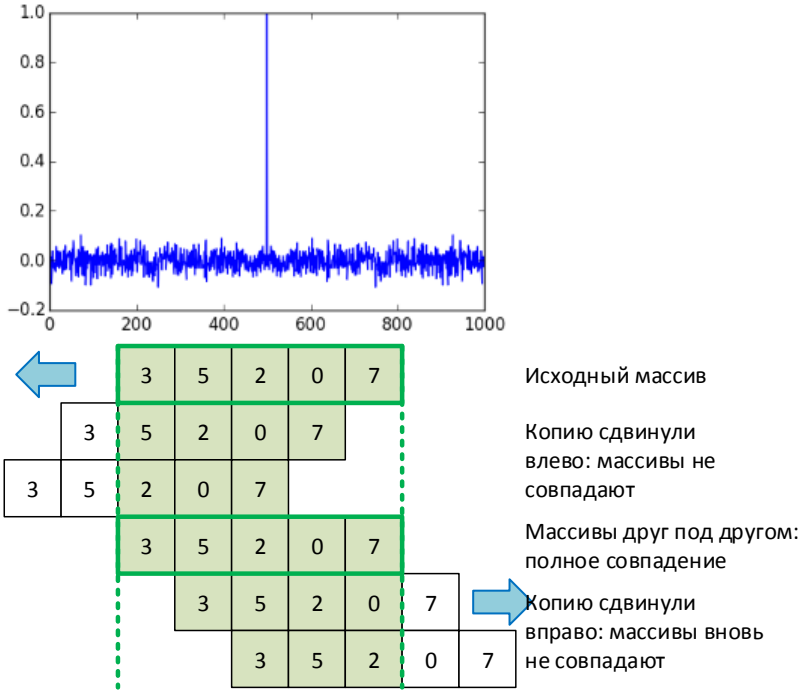

Рисунок 5 Объяснение графика автокорреляции

При автокорреляции исходный массив движется сам относительно себя. Когда сдвинутая копия находится точно под исходной, массивы полностью совпадают, и коэффициент корреляции Пирсона равен 1. Но стоит только сдвинуть копию влево или вправо, исходный массив и копия перестают совпадать, и коэффициент корреляции резко уменьшается.

6. Импортировать из файла *.mat многомерные данные. Первые 5 столбцов считать входными значениями (измерения датчиков, наблюдения и т.д.), последний 6 столбец – выходная величина, для которой требуется установить зависимость от входных величин.

Для импорта используйте подход, описанный в разделе 3.

7. Построить матрицу корреляции для всех входных и выходных величин. Сделать выводы о зависимости/независимости выходной величины от каждой из входных компонент. Отобразить точечный график для случайных величин, коэффициент корреляции Пирсона для которых по модулю больше 0.8.

Загруженные в предыдущем разделе данные представляют собой матрицу из 1000 строк и 6 столбцов. Чтобы построить матрицу корреляции, нужно скоррелировать попарно каждый столбец с каждым.

Общий подход к решению этой задачи следующий:

1. Пусть данные в пункте 6 загружены в матрицу data. Необходимо определить количество столбцов в матрице (в общем виде их не всегда будет 6). Для этого можно воспользоваться свойством shape. Пусть количество столбцов сохранено в переменной **n**.

2. Необходимо создать квадратную матрицу размера n×n, заполненную нулями. Для этого стоит воспользоваться функцией np.zeros. Пусть созданная матрица называется **`corr_matrix`**.

3. Реализовать цикл вида:

```
for i in range(0, n):
```

```
  for j in range(0, n):
    # выбор i-го столбца и j-го столбца
    # код вычисления корреляции
    # запись коэффициента корреляции в соответствующую ячейку corr_matrix
```

Чтобы в Python выбрать определённый столбец матрицы, нужно воспользоваться кодом:

```
col = data[:, 3]
```

где 3 – индекс столбца. Двоеточие в данном случае означает, что будут взяты все строки и 4-й столбец. Аналогично можно взять заданную строку из матрицы:

```
row = data[5,:]
```

Более того, Python и библиотека NumPy позволяют выбирать диапазоны значений, например:

```
range = data[2:5, 0:3]
```

В данном случае будет выбран фрагмент матрицы, соответствующий строкам 2-4 и столбцам 0-2.

Индекс -1 означает, что будет выбран последний элемент (последняя строка или столбец), например:

```
data[:,-1]   # будет выбран последний столбец матрицы data
```

Для вычисления корреляции используйте уже знакомую вам функцию **`np.corrcoef`**.

Запишите вычисленный коэффициент корреляции в соответствующую ячейку матрицы **`corr_matrix`**.

4. Выведите матрицу на экран. Вывод выполняется функцией **`print`**. Для того, чтобы выводить данные с точностью до 2 знаков после запятой, используйте функцию **`np.set_printoptions`**:

```
np.set_printoptions(precision=2)
print(corr_matrix)
```

Должен получиться примерно следующий вывод:

```
[[ 1.    0.01 -0.06 -0.01  0.    0.19]
 [ 0.01  1.   -0.04 -0.02 -0.02  0.42]
 [-0.06 -0.04  1.   -0.02  0.02 -0.89]
 [-0.01 -0.02 -0.02  1.    0.08 -0.07]
 [ 0.   -0.02  0.02  0.08  1.   -0.03]
 [ 0.19  0.42 -0.89 -0.07 -0.03  1.  ]]
```

В матрице найдите элементы, которые по модулю больше **`0.8`**. В данном примере это число **`-0.89`**, являющееся коэффициентом корреляции столбцов 2 и 5.

5. Выведите точечный график значений столбцов с высоким коэффициентом корреляции. Вот примерный код построения графика:

```
plt.plot(data[:, 2], data[:, 5], 'b.')
plt.grid()
plt.show()
```

Получившийся график приведён ниже

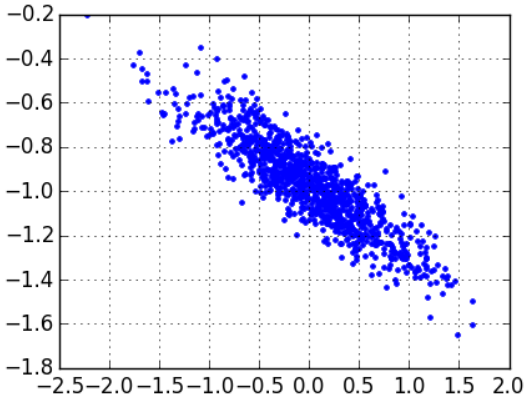

Рисунок 6 Точечный график автокорреляции двух столбцов