# Практическая работа 3. Введение в анализ данных с помощью Python

**Обзор**

Почему Python?

Python - это язык общего назначения. У Python есть несколько библиотек, которые пригодятся для исследований и вычислений:

• SciPy — это библиотека с научными инструментами.

• NumPy — это расширение, которое добавляет поддержку матриц и многомерных массивов, а также математических функций для работы с ними.

• Matplotlib - библиотека для работы с 2D и 3D графикой.

Многие ученые выбирают Python из-за его библиотек и простоты изучения языков.

Python используется для разработки встроенных систем для различных устройств.

Какие знания и навыки вы приобретете, выполнив эту лабораторную работу?

Выполнив эту лабораторную работу, вы получите следующие навыки:

• Возможность загрузки данных в Python

• Умение работать с массивами данных

• Возможность обрабатывать и визуализировать данные

• Понимание процесса оценки параметров модели исследуемого объекта.

## Практическое задание

Одним из основных элементов киберфизических систем, например роботов, являются двигатели. Часто бывает необходимо решить задачу оценки параметров двигателя каждого звена манипулятора при проектировании системы управления роботом. Мы рассмотрим решение задачи анализа данных.

У каждого студента есть свой набор данных, который содержит информацию о напряжении, токе и времени измерения. Период измерения 0,001 с. Период тестового сигнала 0,1 с. Количество периодов тестового сигнала - 1000. Тип двигателя - двигатель постоянного тока.

Этапы работы:

1. Импортировать библиотеки в Python.

2. Загрузка и подготовка данных.

3. Нарисовать графики тока и напряжения.

4. Рассчитать значения параметров L и R.

5. Рассчитать средние значения и стандартное отклонение расчетных значений параметров L и R.

**Теория**

**Упрощённая модель двигателя постоянного тока.** Модель двигателя постоянного тока описывается следующей системой дифференциальных уравнений:

 (1)

где u - напряжение питания на обмотках двигателя, i - ток в обмотках двигателя, - скорость вращения двигателя, Md - возмущающий момент, R - сопротивление обмоток двигателя, L - индуктивность двигателя. обмоток, *ce* - постоянная противо-ЭДС, J - момент инерции.

Требуемая скорость вращения Ω может быть достигнута определенным способом управления напряжением питания u. Это одна из решаемых задач, например, в робототехнике. Сформировать необходимый контроль можно, зная параметры L, R, ce, J. Задача оценки этих параметров также решается при диагностике неисправностей двигателя. В этой лабораторной работе мы рассмотрим решение упрощенной задачи. Рассмотрим оценку параметров L и R по тестовым данным, где Ω = 0. В данной задаче нас интересует только первое уравнение:

 (2)

Напряжение изменяется с периодом Td = 0,001 с (период выборки), ток измеряется с таким же периодом. Уравнение переходного процесса для любого временного интервала имеет вид:

 (3)

и для двух соседних интервалов измерения:

 (4)

где n - номер интервала измерения.

**Оценка параметров L и R обычным методом наименьших квадратов (МНК).** Формула (4) - это линейная регрессия. Его можно переписать следующим образом:

 (5)

where



Если в датчике присутствует шум измерения, то регрессию (5) можно записать следующим образом:

, (6)

где  - шум измерения датчика.

Матрица факторов записывается следующим образом:



А вектор рассматриваемой переменной выглядит следующим образом:



Уравнение (6) для серии измерений может быть записано следующим образом:



Вектор параметров k можно найти методом 0LS:



И соответственно значения неизвестных параметров находятся следующим образом:



**Руководство по лабораторной работе**

**1. Импорт библиотек.** Во-первых, нам нужно импортировать библиотеки, необходимые для выполнения задачи. Нам понадобится библиотека numpy, которая содержит различные инструменты для работы с данными и для математического анализа. Это можно сделать с помощью следующей команды:

import numpy as np

Команда import загружает библиотеку, имя которой следует за этой командой. Слово as является необязательным и означает, что мы будем обращаться к инструментам библиотеки не через имя «numpy», а через новое имя «np». Это полезно, если, например, имя библиотеки очень длинное.

Кроме того, мы загрузим объект linalg отдельно от библиотеки numpy, чтобы не писать длинный вызов numpy.linalg. Он содержит необходимые нам матричные инструменты. Итак, добавим строку:

from numpy import linalg as LA

И снова, чтобы не писать linalg, переименуем его в LA.

Последняя библиотека, которая нам понадобится, — это библиотека matplotlib. Из него мы загрузим только один объект pyplot, который требуется для рисования графиков. И для удобства назовем его plt:

import matplotlib.pyplot as plt

Всё вместе это будет выглядеть следующим образом:

import numpy as np  
from numpy import linalg as LA  
import matplotlib.pyplot as plt

**2. Загрузка и подготовка данных.** Библиотеки загружены. Теперь нам нужно загрузить данные из файла. У каждого из вас будет свой файл с собственным набором данных. Давайте воспользуемся функцией genfromtxt ('filename', delimiter = ',') из библиотеки numpy. Здесь filename - имя файла, delimiter - обозначение разделителя данных в этом файле (в нашем файле это ','). Затем пишем:

data\_read = np.genfromtxt(**'test.csv'**,delimiter=**','**)

Выполняя эту строку, мы загружаем данные из файла test.csv, разделенные знаком, в переменную data\_read. Переменной можно присвоить любое имя, поэтому ваша задача - назвать ее по-своему. Имя необходимо начинать с буквы, не использовать пробелы и специальные символы. Все буквы в переменных должны быть латинскими.

Данные загружаются в нашу переменную data\_read (но у вас есть собственное имя для этой переменной). Теперь необходимо выделить из них время, ток и напряжение, с которыми мы потом будем работать. Давайте сначала выделим время:

time = data\_read[:,0]

Квадратные скобки после переменной data\_read указывают, какие элементы мы перенесем во временную переменную. Пишем номер строки, а затем номер столбца, через запятую. В нашей записи «:» означает, что мы выбираем все строки массива, которые находятся в столбце 0. Все элементы в Python нумеруются, начиная с 0.

Теперь временная переменная содержит данные о времени всех измерений, но, к сожалению, в виде ряда. Для дальнейших операций нам нужно преобразовать эти данные в вектор-столбец. Для этого нам нужно добавить еще одно измерение к нашему вектору и сделать его вертикальным. Мы сделаем нашу переменную матрицей с одним столбцом. Добавление нового измерения выполняется параметром newaxis библиотеки numpy следующим образом:

time = time[:, np.newaxis]

Здесь вектор времени преобразуется в матрицу с одним столбцом. Результат преобразования записывается в новую переменную time, а старые серии перезаписываются.

То же самое можно сделать для векторов, содержащих данные о токе (назовем его current) и напряжении (назовем его voltage). Все вместе это будет выглядеть так:

time = data\_read[:,0]

time = time[:, np.newaxis]

current = data\_read[:,1]  
current = current[:, np.newaxis]  
voltage = data\_read[:,2]  
voltage = voltage[:, np.newaxis]

Таким образом, у нас есть три вектора-столбца, которые содержат данные о времени, токе и напряжении.

**3. Построение графиков тока и напряжения.** Давайте теперь нарисуем графики и посмотрим на данные. Нам понадобится два графика: график напряжения и график тока. Мы разместим их друг под другом и сделаем одинаковый масштаб по оси x. Создадим картинку fig, на котором два отдельных графика будут в своих осях координат ay1 и ay2:

fig, (ay1,ay2) = plt.subplots(2, 1, sharex=True)

Функция subplot библиотеки matplotlib.plot выводит 2 строки и 1 столбец отдельных систем координат для графиков, то есть у нас будут отдельные участки 2x1, которые будут расположены вертикально. «Sharex = True» сообщает функции plt () (matplotlib.plot ()), что графики будут иметь одинаковый масштаб и диапазон по оси x. Это будет выглядеть так:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Figure 1**

Теперь нам нужно построить график зависимости напряжения от времени в верхней системе координат ay1:

ay1.plot(time,voltage)

Функция plot () строит зависимость данных в векторе напряжения от данных во временном векторе по осям ay1:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Figure 2**

Данных слишком много, поэтому пока ничего не понятно. Давайте отобразим только два периода тестового сигнала, чтобы увидеть, как все это выглядит. Заменим строку следующим образом:

T\_per = 0.1  
ay1.plot(time[time<2\*T\_per],voltage[time<2\*T\_per])

Теперь внутри скобок [] мы указали только те элементы векторов времени и напряжения, которые удовлетворяют временам «time <2 \* T\_per», где T\_per - новая переменная, равная 0,1 с (указанный период тестового сигнала во время эксперимента). График теперь выглядит так:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Figure 3**

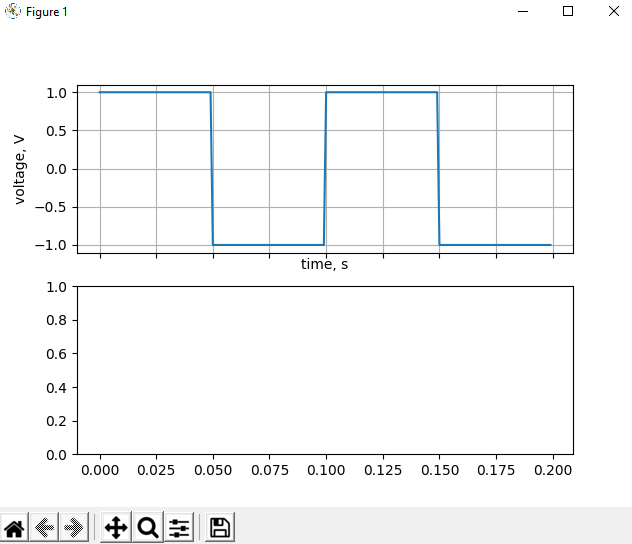
Данные стали выглядеть намного лучше. Но неплохо было бы нарисовать сетку и разметить оси. Сетка отображается на оси ay1 функцией grid () следующим образом:

ay1.grid()

И метки создаются с помощью функций setx\_label ('label') и sety\_label ('label'), где label - это желаемое имя оси. Вывод по оси ay1 выполняется следующим образом:

ay1.set\_xlabel(**'time, s'**)  
ay1.set\_ylabel(**'voltage, V'**)

Теперь наши графики выглядят следующим образом:



**Figure 4**

Осталось вывести график тока по оси ay2. Это делается аналогичным образом, и весь код принимает вид:

fig, (ay1,ay2) = plt.subplots(2, 1, sharex=True)  
T\_per = 0.1  
ay1.plot(time[time<2\*T\_per],voltage[time<2\*T\_per])  
ay1.grid()  
ay1.set\_xlabel(**'time, s'**)  
ay1.set\_ylabel(**'voltage, V'**)  
ay2.plot(time[time<2\*T\_per],current[time<2\*T\_per])  
ay2.grid()  
ay2.set\_xlabel(**'time, s'**)  
ay2.set\_ylabel(**'current, A'**)  
plt.show()

Здесь функция show () библиотеки matplotlib.plot (plt) отображает все графики на экране:

Изображение выглядит как текст, карта

Автоматически созданное описание

**Figure 5**

Графики, отображаемые на fig 5, можно сохранить в файл с помощью функции savefig ('filename') библиотеки matplot.lib:

fig.savefig(**'Recieved data(part)'**)

Графики будут сохранены в файл ‘*Recieved data(part).png***’.**

Полный код этой части лабораторной работы будет выглядеть так:

fig, (ay1,ay2) = plt.subplots(2, 1, sharex=True)  
T\_per = 0.1  
ay1.plot(time[time<2\*T\_per],voltage[time<2\*T\_per])  
ay1.grid()  
ay1.set\_xlabel(**'time, s'**)  
ay1.set\_ylabel(**'voltage, V'**)  
ay2.plot(time[time<2\*T\_per],current[time<2\*T\_per])  
ay2.grid()  
ay2.set\_xlabel(**'time, s'**)  
ay2.set\_ylabel(**'current, A'**)  
plt.show()  
fig.savefig(**'Recieved data(part)'**)

**4. Расчет вероятных значений параметров L и R.** Согласно теории, сначала необходимо вычислить матрицу факторов X:



Она состоит из двух столбцов, которые заполнены данными напряжения и тока от индекса 1 до n-1. Чтобы объединить столбцы в матрицу, мы будем использовать функцию concatenate () библиотеки numpy (np):

X = np.concatenate([voltage[0:len(voltage)-2],current[0:len(current)-2]], axis = 1)

**Len** («имя набора данных») возвращает количество элементов в массиве n. Напомню, что нумерация в Python начинается с нуля, поэтому нужно вычесть 2 из количества элементов.

Массивы можно объединять по строкам или столбцам. Если мы соединим наши два вектора-столбца строка за строкой, мы получим один большой вектор-столбец. Нам нужно объединить по столбцам, поэтому параметру оси присвоено значение 1.

Теперь нам нужно сформировать вектор нашей переменной:



Для этого необходимо сделать следующее:

Y = current[1:len(current)-1]

Нумерация начинается не с 2, а с 1, потому что в Python индексы нумеруются, начиная с 0.

И найдем параметры линейной регрессии OLS:



Здесь нам нужно познакомиться сразу с тремя функциями: транспонированием, умножением матриц и вычислением обратной матрицы.

Для транспонирования не нужно выполнять никаких преобразований. Транспонированная матрица создается сразу после инициализации исходной матрицы и сохраняется в том же объекте, что и элемент name\_of\_matrix.T. Например, вызов транспонированной матрицы X выглядит так:

X.T

Умножение матриц выполняется функцией dot (first\_matrix, second\_matrix) библиотеки numpy (np). Например, расчет выглядит так:

np.dot(X.T,X)

Обратная матрица вычисляется с помощью функции inv (matrix) библиотеки numpy.linalg (LA). Например, расчет выглядит так:

LA.inv(np.dot(X.T,X))

Далее вычисление  будет выглядеть следующим образом:

K = np.dot(np.dot(LA.inv(np.dot(X.T,X)),X.T),Y)

Умножение матриц всегда выполняется только слева направо.

На основании рассчитанных оценок параметров регрессии k можно рассчитать оценки индуктивности и сопротивления:



В Python это будет выглядеть так:

R = 1 / K[0] \* (1 - K[1])  
T = -Td / np.log(K[1])  
L = T\*R

Единственная функция, которую мы еще не знаем, - это натуральный логарифм. Логарифм x вычисляется функцией log (x) библиотеки numpy (np).

Получив оценки параметров L и R, хочу сравнить, насколько наша модель с найденными параметрами соответствует объекту, с которого были получены данные. Для этого по полученным параметрам регрессии рассчитываем силу тока  и сравниваем их с исходными данными:

current\_est = X.dot(K)  
plt.plot(time[time<T\_per],current[time<T\_per])  
plt.plot(time[time<T\_per],current\_est[time[0:len(current)-2]<T\_per])  
plt.show()

В результате мы получим следующий график:

Изображение выглядит как текст, карта

Автоматически созданное описание

**Figure 6**

Здесь синий график - это исходные данные, красный график - результат оценки. Графики близки друг к другу. Полный код этой части лабораторной работы будет выглядеть так:

X = np.concatenate([voltage[0:len(voltage)-2],current[0:len(current)-2]], axis = 1)  
Y = current[1:len(current)-1]  
K = np.dot(np.dot(LA.inv(np.dot(X.T,X)),X.T),Y)  
  
Td = 0.001  
R = 1 / K[0] \* (1 - K[1])  
T = -Td / np.log(K[1])  
L = T\*R

current\_est = X.dot(K)  
  
fig, ax = plt.subplots(1, 1)  
plt.plot(time[time<T\_per],current[time<T\_per])  
plt.plot(time[time<T\_per],current\_est[time[0:len(current)-2]<T\_per])  
ax.grid()  
ax.set\_xlabel(**'time, s'**)  
ax.set\_ylabel(**'current, A'**)  
plt.show()  
fig.savefig(**'Compared data(part)'**)

**5. Вычисление средних значений и стандартного отклонения прогнозируемых значений параметров L и R.** Значения параметров R и L являются случайными из-за наличия случайных ошибок измерения. Поэтому при расчете параметров среднее значение этих случайных величин и разброс (стандартное отклонение) их значений также представляют интерес для того, чтобы понять, в каком интервале может быть реальное значение параметров R и L.

Зададим пустые массивы R\_est и L\_est, а затем добавим к ним вычисленные значения оценок R\_est и L\_est:R\_est = []  
L\_est = []

Рассчитаем значения для каждого периода тестового сигнала. В наших данных всего 1000 таких периодов. Зададим количество периодов n и начнем вычислять оценки в цикле:

for i in range(0, n-1, 1):  
 ind = (time>=T\_per\*i) & (time <= T\_per\*(i+1))  
 new\_current = current[ind]  
 new\_current = new\_current[:, np.newaxis]  
 new\_voltage = voltage[ind]  
 new\_voltage = new\_voltage[:, np.newaxis]  
  
 X = np.concatenate([new\_voltage[0:len(new\_voltage) - 2], new\_current[0:len(new\_current) - 2]], axis=1)  
 Y = new\_current[1:len(new\_current) - 1]  
 K = np.dot(np.dot(LA.inv(np.dot(X.T, X)), X.T), Y)  
  
 if K[1] > 0:  
 R = 1/K[0]\*(1-K[1])  
 T = -Td / np.log(K[1])  
 R\_est.append(R)  
 L\_est.append(T\*R)

Цикл определяется с помощью for var in range. Все, что находится внутри цикла, будет выполнено для каждого значения var из диапазона значений диапазона. Диапазон значений задается функцией range (0, n-1,1) от 0 до n-1 с шагом 1. В Python нет специальных конструкций для указания начала и конца тела цикла. Поэтому все, что нужно повторить в цикле, пишется с дополнительным отступом от левого края страницы.

Внутри цикла нам нужно загрузить часть данных из каждого периода тестового сигнала в переменные new\_current и new\_voltage. Индексы выбираются следующей конструкцией:

ind = (time>=T\_per\*i) & (time <= T\_per\*(i+1))

это означает, что при каждом i-м выполнении цикла мы берем данные с начала i-го периода до начала (i + 1) -го периода.

Основные расчеты производятся так же, как и в предыдущем пункте. Следует отметить, что в нашей ситуации натуральный логарифм вычисляется только от положительных чисел. То есть оценка может быть рассчитана только для параметров K [1]> 0. Для этого используется условный оператор if:

if K[1] > 0:

Все, что должно быть выполнено по этому условию, пишется с дополнительным отступом от левого края страницы. Отступы указаны в таблице.

После выполнения цикла у нас есть два массива чисел R\_est и L\_est. Для выполнения функций библиотеки numpy (np) вам необходимо преобразовать эти массивы в массивы данных этой библиотеки с помощью функции array (name\_of\_array):

R\_est = np.array(R\_est)  
L\_est = np.array(L\_est)

Среднее значение рассчитывается с помощью функции mean (array), а стандартное отклонение рассчитывается с помощью функции std (array) библиотеки numpy (np). Вывод этих значений на консоль может быть выполнен с помощью функции print (value1, value2, ...), которая выводит в рабочую область все, что ей передано:

print(**'Mean value of R: '**,np.mean(R\_est),**' Ohm'**)  
print(**'Standart deviation of R: '**,np.std(R\_est))  
print(**'Mean value of L = '**,np.mean(L\_est),**' Hn'**)  
print(**'Standart deviation of R: '**,np.std(L\_est))

Полный код этой части лабораторной работы будет выглядеть так:

R\_est = []  
L\_est = []  
  
n = 1000  
for i in range(0, n-1, 1):  
 ind = (time>=T\_per\*i) & (time <= T\_per\*(i+1))  
 new\_current = current[ind]  
 new\_current = new\_current[:, np.newaxis]  
 new\_voltage = voltage[ind]  
 new\_voltage = new\_voltage[:, np.newaxis]  
  
 X = np.concatenate([new\_voltage[1:len(new\_voltage) - 1], new\_current[0:len(new\_current) - 2]], axis=1)  
 Y = current[1:len(new\_current) - 1]  
 K = np.dot(np.dot(LA.inv(np.dot(X.T, X)), X.T), Y)  
  
 if K[1] > 0:  
 R = 1/K[0]\*(1-K[1])  
 T = -Td / np.log(K[1])  
 R\_est.append(R)  
 L\_est.append(T\*R)  
  
R\_est = np.array(R\_est)  
L\_est = np.array(L\_est)  
  
print(**'Mean value of R: '**,np.mean(R\_est),**' Ohm'**)  
print(**'Standart deviation of R: '**,np.std(R\_est))  
print(**'Mean value of L = '**,np.mean(L\_est),**' Hn'**)  
print(**'Standart deviation of R: '**,np.std(L\_est))