lab1

17 сентября 2022 г.

1 Лабораторная работа №1, Введение в анализ данных с помощью Python

2 ФИО: Жидков Артемий Андреевич

3 группа: R4136c

3.0.1 1. Импортировать библиотеки в Python.

[84]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import random
import scipy
import torch

torch.cuda.synchronize()
torch.cuda.empty_cache()

cuda = torch.device('cuda')
print(torch.cuda.get_device_properties(cuda))
```

```
_CudaDeviceProperties(name='NVIDIA GeForce RTX 3080 Laptop GPU', major=8,
minor=6, total_memory=8191MB, multi_processor_count=48)
```

3.0.2 2. Загрузка и подготовка данных.

[85]:
```
name = random.choice(os.listdir("dataset"))

# name = 'testLab1Var7.csv'

print(f"Dataset: {name}")

dataset = np.genfromtxt(f"dataset/{name}", delimiter=',')

dataset = [dataset[:, i] for i in range(dataset.shape[1])]
title = ["time", "current", "voltage"]
```

```
dataset_dict = dict(zip(title, dataset))
```

```
Dataset: testLab1Var56.csv
```

3.0.3 3. Нарисовать графики тока и напряжения.

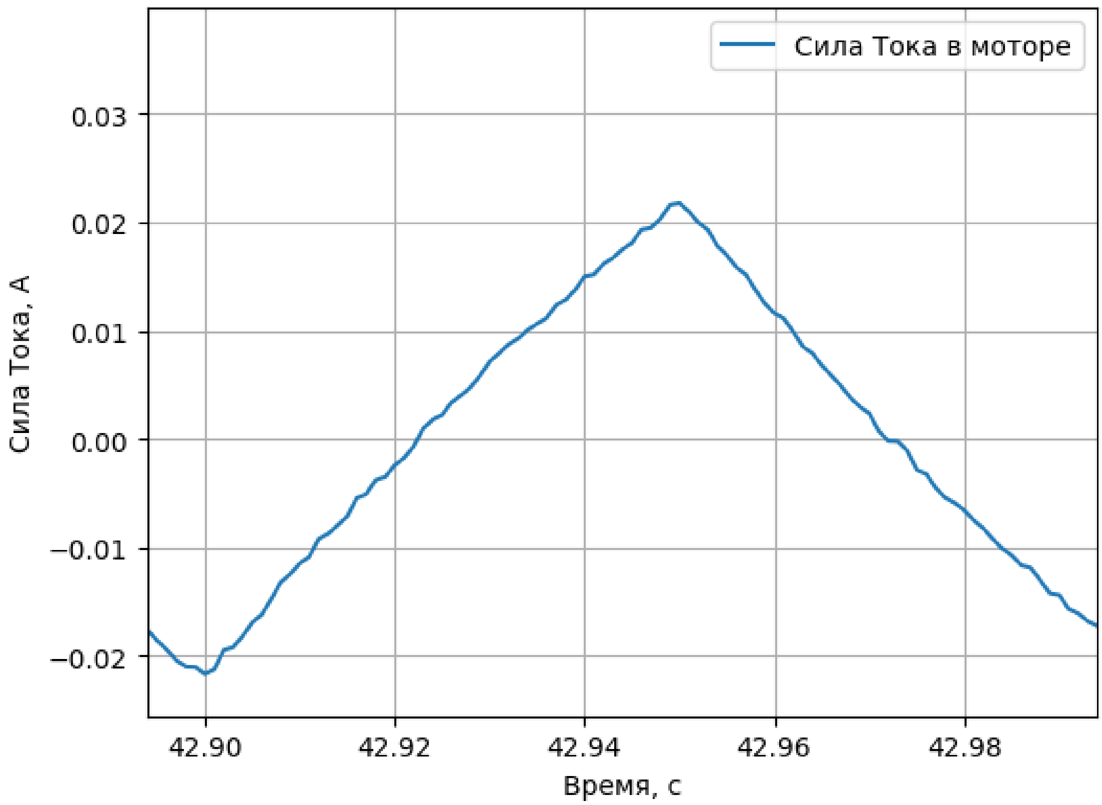
Для удобства отображения отображу не весь график, а некоторый его случайный диапазон заданного размера, установив лимиты на данные.

[86]:
```
"""
Размер интервала
"""
time_period = 0.1
```

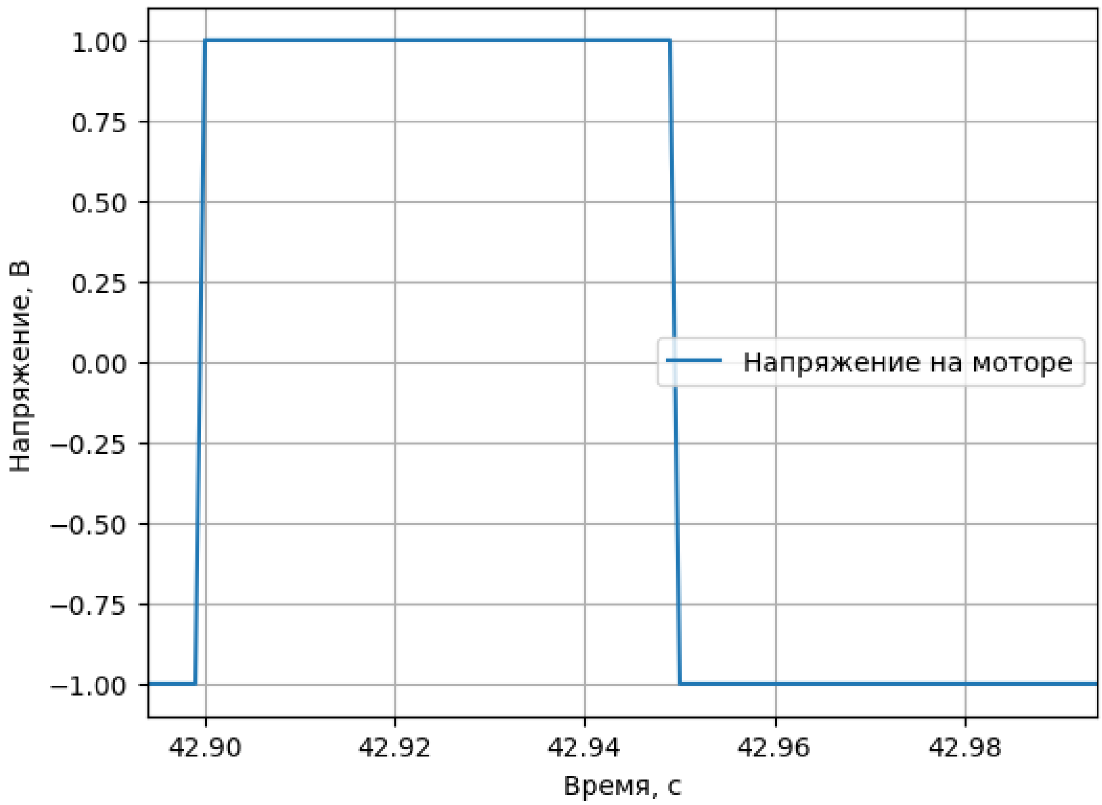
[87]:
```
time_interval = random.random() * (dataset_dict["time"][-1] - time_period)
time_interval = (time_interval, time_interval + time_period)

print(f"Временной интервал {time_interval}")
```

```
Временной интервал (42.89388613861019, 42.99388613861019)
```

[88]:
```
plt.plot(dataset_dict["time"], dataset_dict["current"])
plt.xlim(time_interval)
plt.xlabel('Время, с')
plt.ylabel('Сила Тока, А')
plt.legend(["Сила Тока в моторе"])
plt.grid()
```



[89]:
```
plt.plot(dataset_dict["time"], dataset_dict["voltage"])
plt.xlim(time_interval)
plt.xlabel('Время, с')
plt.ylabel('Напряжение, В')
plt.legend(["Напряжение на моторе"])
plt.grid()
```



3.0.4 4. Рассчитать значения параметров L и R.

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

$$\begin{cases} u = e + R \times i + L \times \dfrac{di}{t} \\ M - M_C = J\dfrac{d\omega}{t} \\ M = C_M \times \Phi \times i \\ e = C_\omega \times \Phi \times \omega \end{cases}$$

где

u - напряжение на якорной обмотке двигателя,

e - электродвижущая сила (ЭДС) якоря,

i - ток якоря,

Φ - поток, создаваемый обмоткой возбуждения,

M - электромагнитный момент двигателя,

M - момент сопротивления движению,

ω - скорость вращения вала двигателя,

R - активное сопротивление якорной цепи,

L - индуктивность якорной цепи,

J - суммарный момент инерции якоря и нагрузки,

$_\omega$ - коэффициент связи между скоростью и ЭДС,

 - коэффициент связи между током якоря и электромагнитным моментом.

Согласно упрощению в данной работе двигатель застопорен, следовательно:

$$\omega = 0 \rightarrow e = C_\omega \times \Phi \times \omega = 0 \rightarrow u = R \times i + L \times \frac{di}{t}$$

Тогда можно получить уравнение электрической составляющей двигателя:

$$L \times \frac{di}{t} = u - R \times i$$

$$\frac{di}{t} = \frac{u}{L} - \frac{R}{L} \times i$$

Что в операторном виде $s = \frac{d}{t}$ будет иметь вид:

$$s \times i = \frac{u}{L} - \frac{R}{L} \times i$$

Теперь нужно составить передаточную функцию, выразив соотношение $G_c = \frac{i}{u}$:

$$G_c(s) = \frac{1}{L \times (s + \frac{R}{L})}$$

Применив Forward Euler (difference) discretization, получу:

$$s = \frac{z-1}{T_d}$$

$$G_d(z) = G_c(s = \frac{z-1}{T_s}) = \frac{1}{L \times (\frac{z-1}{T_s} + \frac{R}{L})}$$

$$G_d(z_i = z^{-1}) = \frac{T_d}{R \times T_d - L + L \times z_i^{-1}}$$

$$G_d(z_i) = \frac{T_d \times z_i}{L - L \times z_i + R \times T_d \times z_i}$$

Данное выражение нужно выразить относительно констант и переменной z^{-1}:

$$G_d(z = z_i^{-1}) = \frac{T_d \times z^{-1}}{L - L \times z^{-1} + R \times T_d \times z^{-1}} = \frac{i(z)}{u(z)}$$

$$T_d \times u(z) * z^{-1} = i(z) * (L - L \times z^{-1} + R \times T_d \times z^{-1})$$

$$T_d \times u(z) * z^{-1} = i(z) * L - i(z) * L \times z^{-1} + i(z) * R \times T_d \times z^{-1}$$

$$i(z) * L = T_d \times u(z) * z^{-1} + i(z) * L \times z^{-1} - i(z) * R \times T_d \times z^{-1}$$

$$i(z) = u(z) * z^{-1} \times (\frac{T_d}{L}) + i(z) * z^{-1} \times (1 - \frac{R \times T_d}{L})$$

$$i(z) = u(z) * z^{-1} \times (\frac{T_d}{L}) - i(z) * z^{-1} \times (\frac{R \times T_d - L}{L})$$

В дискретной системе операция z^{-1} - есть задержка на один такт, следовательно, можно рассчитать значение силы тока в конкретны момент:

$$i_k = u_{k-1} \times (\frac{T_d}{L}) - i_{k-1} \times (\frac{R \times T_d - L}{L})$$

Для упрощения задачи уже линейной аппроксимации функции введу матрицы для хранения и более удобного представления:

$$Y_{n \times 1} = i_{1:end}$$

$$X_{n \times 2} = [i_{0:end-1} | u_{0:end-1}]$$

$$K_{2 \times 1} = [\frac{T_d}{L} | \frac{R \times T_d - L}{L}]^T$$

$$Y = X * K$$

Следующий код подготавливает нужные матрицы и загружает их в память видеокарты:

[90]:
```
X = np.transpose(np.concatenate([np.array([dataset_dict["voltage"], ]), np.
↪array([dataset_dict["current"], ])], axis=0))
Y = np.transpose(np.array([dataset_dict["current"], ]))

# X : n*k
# K : k*1
```

```
# X * K = Y
# [U(k-1);I(k-1)] * K = [I(k)]

X = X[:-1, :]  # U(k-1);I(k-1)
Y = Y[1:, :]  # I(k)

print(X.shape)
print(Y.shape)

X_tensor = torch.tensor(X, device=cuda, dtype=torch.float64)
Y_tensor = torch.tensor(Y, device=cuda, dtype=torch.float64)
```

```
(100000, 2)
(100000, 1)
```

Функция нахождения псевдообратной матрицы Moore–Penrose через Сингулярное Разложение.

Данная матрица позволяет аппроксимировать $Y = X \times K + e$ Методом Наименьших Квадратов:

[91]:
```
def get_pseudoinverse(matrix):
    matrix_svd = torch.svd(matrix)

    matrix_psi = matrix_svd.V
    matrix_psi = torch.mm(matrix_psi, torch.diag(1 / matrix_svd.S))
    matrix_psi = torch.mm(matrix_psi, matrix_svd.U.T)

    return matrix_psi
```

[92]:
```
X_psi = get_pseudoinverse(X_tensor)

print(X_psi.shape)
print(X_tensor.shape)
```

```
torch.Size([2, 100000])
torch.Size([100000, 2])
```

Результатом перемножения псевдообратной матрицы от X и самой матрицы X должна быть матрица, близкая к единичной:

[93]:
```
print(torch.mm(X_psi, X_tensor))
```

```
tensor([[ 1.0000e+00,  4.9975e-19],
        [-1.2334e-15,  1.0000e+00]], device='cuda:0', dtype=torch.float64)
```

$X \times K = Y \rightarrow K = X^{+} \times Y$, где X^{+} - матрица, псевдообратная к X

[94]:
```
K_approx = torch.mm(X_psi, Y_tensor)
print(K_approx)
```

```
tensor([[8.9280e-04],
        [9.9093e-01]], device='cuda:0', dtype=torch.float64)
```

[95]:
```
K = K_approx.cpu()

Td = 0.001
L = Td / K[0]
R = (L - K[1] * L) / Td

print('Вычисленное значение R = ', R.numpy()[0], ' Ом')
print('Вычисленное значение L = ', L.numpy()[0], ' Гн')
```

```
Вычисленное значение R =  10.158316105496112  Ом
Вычисленное значение L =  1.1200701242066806  Гн
```

Также можно посчитать те же данные согласно формулам из исходной лабораторной работы, которые я считаю не самыми корректными

[96]:
```
R = 1 / K[0] * (1 - K[1])
T = -Td / np.log(K[1])
L = T * R

print('R = ', R.numpy()[0], ' Ohm')
print('L = ', L.numpy()[0], ' Hn')
```

```
R =  10.158316105496143  Ohm
L =  1.1149832536880728  Hn
```

3.0.5 5 Рассчитать средние значения и стандартное отклонение.

Для нахождения ошибки между реальным значением Y и его предсказанием моделью $X \times K$, можно просто посчитать их разность $e = Y - X \times K$

Тогда Сумма квадратов ошибки будет:

$$S(K) = \sum e_i^2 = e^T \times e = (Y - X \times K)^T \times (Y - X \times K)$$

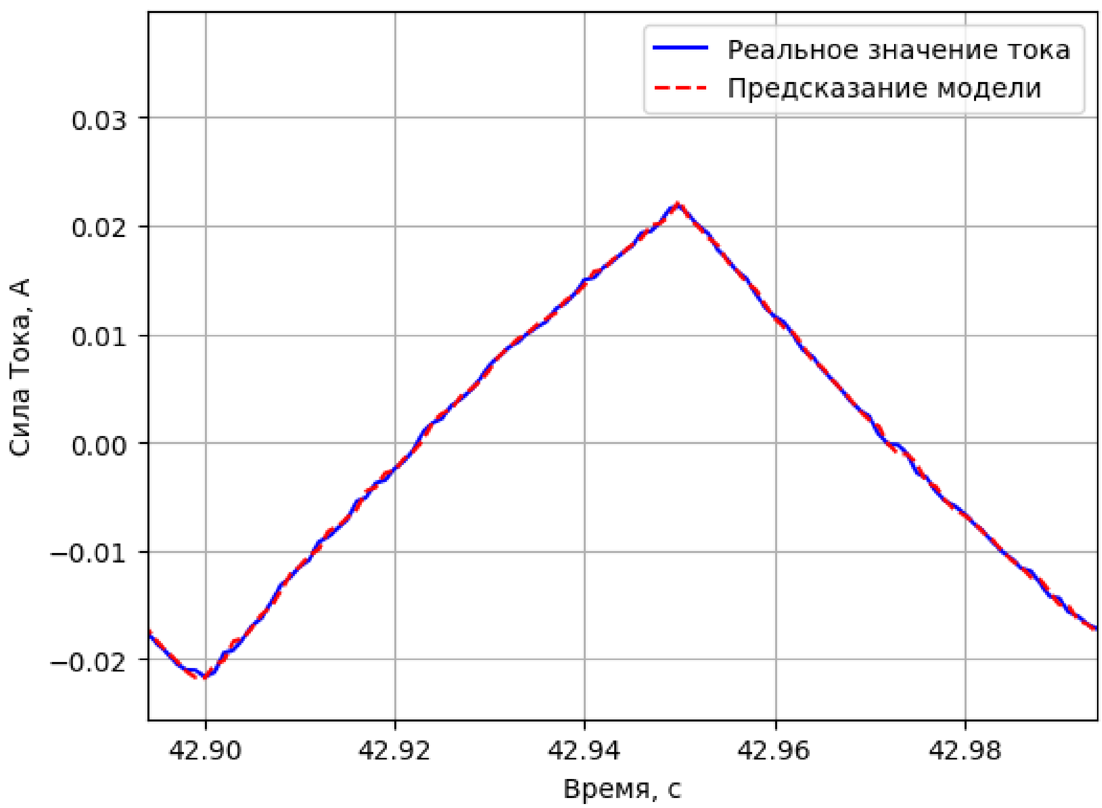
А среднеквадратичное отклонение:

$$\sigma_Y = \sqrt{\frac{S(K)}{n}}$$

[97]:
```
e2_Y = torch.mm(Y_tensor.T - torch.mm(X_tensor, K_approx).T, Y_tensor - torch.
 ↪mm(X_tensor, K_approx))
sigma2_Y = torch.divide(e2_Y, Y_tensor.shape[0])

sigma_Y = torch.sqrt(sigma2_Y)
sigma_Y = sigma_Y.cpu().numpy()[0][0]

print(sigma_Y)
```

```
0.00042367658795242503
```

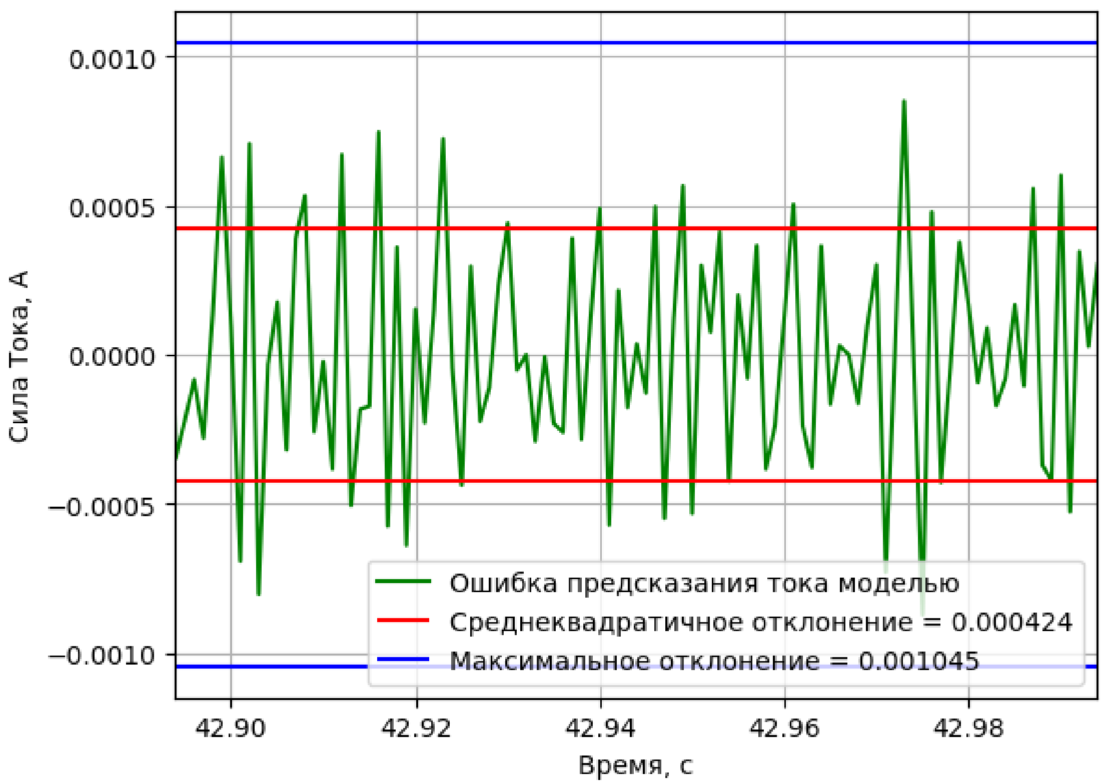
[98]:
```
Y_predict = torch.mm(X_tensor, K_approx)
Y_predict = Y_predict.cpu().numpy()

# print(Y_predict.T[0].shape)
# print(dataset_dict["current"][1:].shape)

plt.plot(dataset_dict["time"][1:], dataset_dict["current"][1:], 'b')
plt.plot(dataset_dict["time"][1:], Y_predict.T[0], 'r--')
plt.xlim(time_interval)
plt.xlabel('Время, с')
plt.ylabel('Сила Тока, А')
plt.legend(["Реальное значение тока", "Предсказание модели"])
plt.grid()
```



Как видно из графика, наша дискретная модель неплохо предсказывает следующее значение силы тока по напряжению и силе тока на предыдущей шаг, но на вопрос, насколько хорошо, способен ответить график ошибки между реальным значением Y и его предсказанием:

[99]:
```
# print(Y_predict.T[0].shape)
# print(dataset_dict["current"][1:].shape)
```

```
max_offset = np.max(np.abs(dataset_dict["current"][1:] - Y_predict.T[0]))

plt.plot(dataset_dict["time"][1:], dataset_dict["current"][1:] - Y_predict.T[0],␣
↪'g')
plt.hlines([-sigma_Y, sigma_Y], dataset_dict["time"][0],␣
↪dataset_dict["time"][-1], 'r')
plt.hlines([-max_offset, max_offset], dataset_dict["time"][0],␣
↪dataset_dict["time"][-1], 'b')
plt.xlim(time_interval)
plt.xlabel('Время, с')
plt.ylabel('Сила Тока, А')
plt.legend(["Ошибка предсказания тока моделью", f"Среднеквадратичное отклонение␣
↪= {sigma_Y:.6f}", f"Максимальное отклонение = {max_offset:.6f}"])
plt.grid()
```



Как видно из графика ошибки предсказания, она находится по большей части в границах среднеквадратичного отклонения, что говорит о корректности вычисления σ_Y, а максимальное отклонение находится не далеко от среднеквадратичного, что говорит об отсутствии серьёзных выбросов в исходных данных.