

:

: R4136

1. Python.

```
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_mem=8592000000)
```

2. .

```
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: testLab1Var11.csv
```

3. .

```
_, _, _ = dataset_dict['time'], dataset_dict['current'], dataset_dict['voltage']
```

```
"""
```

```
"""
```

```
time_period = 0.1
```

```

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

print(f"                {time_interval}")
                (28.65413982391623, 28.75413982391623)

plt.plot(dataset_dict["time"], dataset_dict["current"])
plt.xlim(time_interval)
plt.xlabel('        , ')
plt.ylabel('        , ')
plt.legend(["          "])
plt.grid()

```

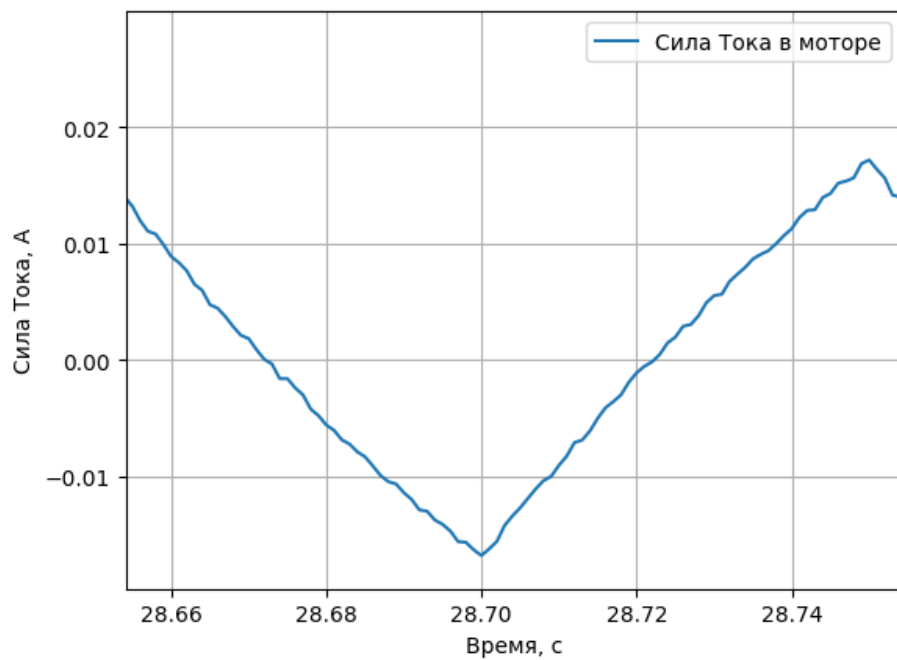


Figure 1: png

```

plt.plot(dataset_dict["time"], dataset_dict["voltage"])
plt.xlim(time_interval)
plt.xlabel('        , ')
plt.ylabel('        , ')
plt.legend(["          "])
plt.grid()

```

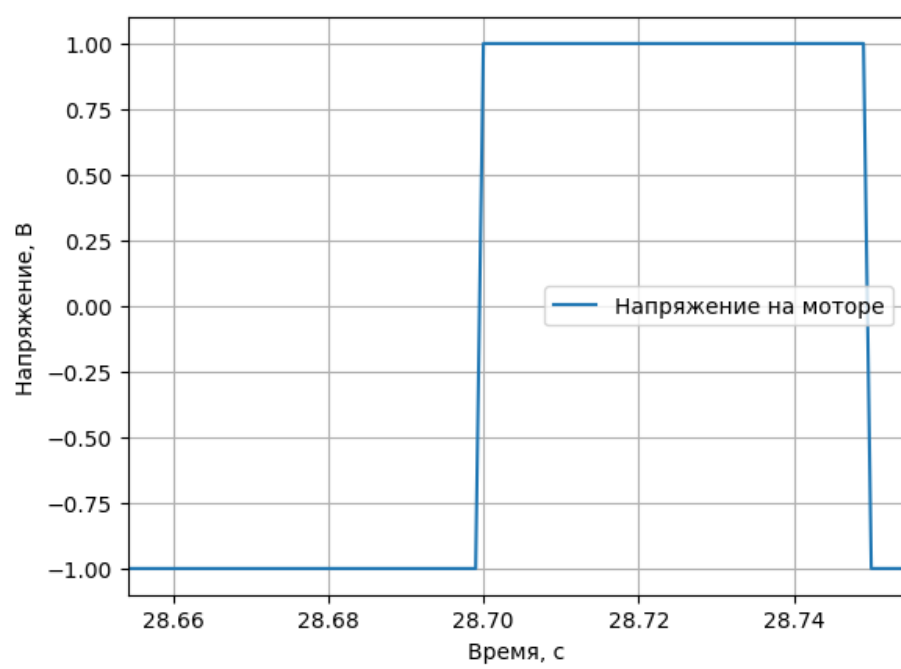


Figure 2: png

4. **L R.**

:

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

u - ,
 e - () ,
 i - ,
 Φ - , ,
 M - ,
 M - ,
 ω - ,
 R - ,
 L - ,
 J - ,
 ω - ,
- .
, :

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

:

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

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

$$s = \frac{d}{t} :$$

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

$$, \quad G_c = \frac{i}{u} :$$

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

Forward Euler (difference) discretization, :

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

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

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

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

$$z^{-1} :$$

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

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

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

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

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

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

$$z^{-1} - \quad , \quad ,$$

:

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

:

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

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

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

$$Y=X*K$$

:

```
X = np.transpose(np.concatenate([np.array([dataset_dict["voltage"], ]), np.array([dataset_d
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)
```

Moore–Penrose .

$$Y = X \times K + e \quad :$$

```
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

X_psi = get_pseudoinverse(X_tensor)

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

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

X X , :

print(torch.mm(X_psi, X_tensor))
tensor([[ 1.0000e+00,  3.7524e-19],
        [-2.7756e-16,  1.0000e+00]], device='cuda:0', dtype=torch.float64)

 $X \times K = Y \rightarrow K = X^+ \times Y$ ,  $X^+ =$ ,  $X$ 

K_approx = torch.mm(X_psi, Y_tensor)
print(K_approx)
tensor([[6.9068e-04],
        [9.9025e-01]], device='cuda:0', dtype=torch.float64)

K = K_approx.cpu()

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

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

R = 14.12081504070506
L = 1.4478479999320457

,

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 = 14.120815040705097 Ohm
L = 1.4407760594432346 Hn

```

5

$$e = Y - X \times K$$

:

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

:

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

```
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)
```

```
0.00030117986995266334
```

```
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('time', 's')
plt.ylabel('Y', 'mV')
plt.legend(["current", "Y"])
plt.grid()
```

```
,
,
,
Y
:
```

```
# 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')
```

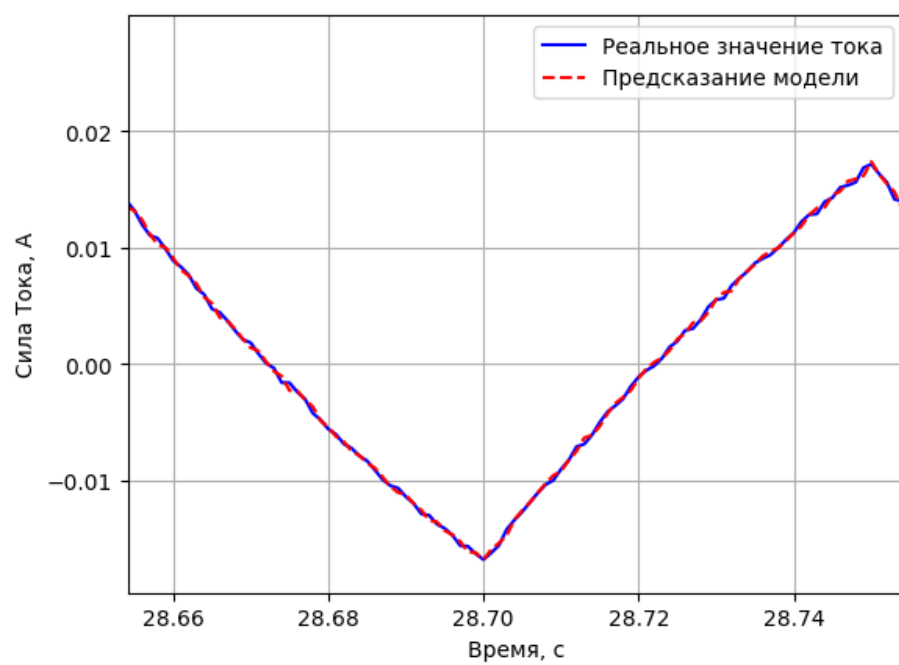



Figure 3: png

```
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(' ', ' ')
plt.ylabel(' ', ' ')
plt.legend(["", f" = {sigma_Y:.6f}", f" = {max_offset:.6f}"])
plt.grid()
```



Figure 4: png

,
 σ_Y ,
.