МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

ОТЧЁТ

По лабораторной работе № \_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил:

Студент группы ИИ-22

Копанчук Евгений Романович

Проверил\_\_:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Брест – 2023

**Ход работы**

*Код программы*

import numpy as np

from tqdm import tqdm

import matplotlib.pyplot as plt

a, b, d, n\_input = 4, 7, 0.2, 4

def f(a, b, d, x):

    return a \* np.sin(b \* x) + d;

amount = 30

tspan = [0, 0.89]

step = (tspan[1] - tspan[0]) / amount

X = np.arange(tspan[0], tspan[1], step)

y = np.array([f(a, b, d, \_) for \_ in X])

n = y.shape[0] - n\_input - 1

X\_train = np.empty((n, n\_input), dtype=float)

y\_train = np.empty(n, dtype=float)

for i in range(n):

    for j in range(0, n\_input, 1):

        X\_train[i][j] = y[i + j]

    y\_train[i] = y[i + n\_input]

X = np.array([tspan[1] + \_ \* step for \_ in range(0, amount, 1)])

y = np.array([f(a, b, d, \_) for \_ in X])

X\_test = np.empty((n, n\_input), dtype=float)

y\_test = np.empty(n, dtype=float)

for i in range(n):

    for j in range(0, n\_input, 1):

        X\_test[i][j] = y[i + j]

    y\_test[i] = y[i + n\_input]

print(X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape)

class LinearModel:

    def \_\_init\_\_(self):

        self.w = None

        self.lr = None

        self.n\_input = None

    def test(self, y\_test):

        y\_pred = []

        for \_ in range(y\_test.shape[0]):

            y\_pred.append(model.predict(X\_test[\_]))

        print(f"mean |y-e| = {np.mean(np.abs(y\_test - y\_pred))}")

        plt.plot(X, y, color='green')

        plt.plot(X[n\_input:-1], y\_pred, color='red')

        plt.show()

    def fit(self, X\_train, y\_train, mode='classic', opt='const', lr=0.01, epoches=100, e=1e-3, batch\_size=4):

        self.lr = lr

        self.n\_input = X\_train.shape[1]

        self.epoches = epoches

        self.w = np.random.uniform(0, 1, self.n\_input + 1)

        loss = []

        if mode == 'online-learning' and opt == 'const':

            for ep in range(epoches):

                y\_pred = np.empty(y\_train.shape, dtype=float)

                for i in range(len(X\_train)):

                    y\_pred[i] = self.predict(X\_train[i])

                    self.w -= lr \* self.grad(X\_train[i], y\_pred[i], y\_train[i])

                loss.append(self.mse(y\_pred, y\_train))

                if loss[-1] < e:

                    break

        if mode == 'online-learning' and opt == 'adapted':

            for ep in range(epoches):

                y\_pred = np.empty(y\_train.shape, dtype=float)

                for i in range(len(X\_train)):

                    y\_pred[i] = self.predict(X\_train[i])

                    lr = 1 / (1 + np.sum(np.square(X\_train[i])))

                    self.w -= lr \* self.grad(X\_train[i], y\_pred[i], y\_train[i])

                loss.append(self.mse(y\_pred, y\_train))

                if loss[-1] < e:

                    break

        if mode == 'batch-learning' and opt == 'const':

            n = y\_train.shape[0] // batch\_size \* batch\_size

            for ep in range(epoches):

                y\_pred = np.empty(n , dtype=float)

                for b in range(0, n, batch\_size):

                    grad = np.zeros(self.n\_input + 1, dtype=float)

                    for i in range(batch\_size):

                        y\_pred[b + i] = self.predict(X\_train[b + i])

                        grad += self.grad(X\_train[b + i], y\_pred[b + i], y\_train[b + i])

                    self.w -= lr \* grad / batch\_size

                loss.append(self.mse(y\_pred, y\_train[:n]))

                if loss[-1] < e:

                    break

        if mode == 'batch-learning' and opt == 'adapted':

            n = y\_train.shape[0] // batch\_size \* batch\_size

            for ep in range(epoches):

                y\_pred = np.empty(n , dtype=float)

                for b in range(0, n, batch\_size):

                    grad = np.zeros(self.n\_input + 1, dtype=float)

                    for i in range(batch\_size):

                        y\_pred[b + i] = self.predict(X\_train[b + i])

                        grad += self.grad(X\_train[b + i], y\_pred[b + i], y\_train[b + i])

                    a = [(y\_pred[b + p] - y\_train[b + p]) \* (1 + np.sum(np.square(X\_train[b:b+batch\_size]))) for p in range(batch\_size)]

                    tlr = np.sum([(y\_pred[b + p] - y\_train[b + p]) \* a[p] for p in range(batch\_size)])

                    blr = np.sum([a[p] \*\* 2 for p in range(batch\_size)])

                    lr = tlr / blr

                    self.w -= lr \* grad

                loss.append(self.mse(y\_pred, y\_train[:n]))

                if loss[-1] < e:

                    break

        plt.plot(range(len(loss)), loss)

        plt.show()

    def mse(self, y\_pred, y\_real):

        return 1 / 2 \* np.sum(np.square(y\_pred - y\_real))

    def predict(self, X):

        if X.shape[0] < self.n\_input + 1:

            X = np.concatenate((X, np.ones(1)), axis=0)

        return np.dot(X, self.w)

    def grad(self, X, y\_pred, y\_real):

        if X.shape[0] < self.n\_input + 1:

            X = np.concatenate((X, np.ones(1)), axis=0)

        return (y\_pred - y\_real) \* X

    def W(self):

        return self.w

model = LinearModel()

model.fit(X\_train, y\_train, mode='online-learning', opt='const', lr=0.045, epoches=100, e=1e-3)

model.test(y\_test)

model = LinearModel()

model.fit(X\_train, y\_train, mode='online-learning', opt='adapted', epoches=100, e=1e-3)

model.test(y\_test)

model = LinearModel()

model.fit(X\_train, y\_train, mode='batch-learning', opt='const', batch\_size=4, lr=0.045, epoches=100, e=1e-3)

model.test(y\_test)

model = LinearModel()

model.fit(X\_train, y\_train, mode='batch-learning', opt='adapted', lr=0.03, batch\_size=4, epoches=100, e=1e-3)

model.test(y\_test)

|  |  |
| --- | --- |
|  |  |
| mean |y-e| = 0.014821360749135003 | |

|  |  |
| --- | --- |
|  |  |
| mean |y-e| = 0.014821360749135003 | |

|  |  |
| --- | --- |
|  |  |
| mean |y-e| = 0.0062694886961516105 | |

|  |  |
| --- | --- |
|  |  |
| mean |y-e| = 0.007170062830437176 | |