Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5
По дисциплине: «ОМО»
Тема:” Нелинейные ИНС в задачах регрессии”

Выполнил:
Студент 3-го курса
Группы АС-66
Савинец М.Д.
Проверил:
Крощенко А.А.

Брест 2025

Цель: Выполнить моделирование прогнозирующей нелинейной ИНС.

Вариант 11

Задание:

1. Выполнить моделирование прогнозирующей нелинейной ИНС. Для генерации обучающих и тестовых данных использовать функцию

$$y = a\cos(bx) + c\sin(dx) \ .$$

Варианты заданий приведены в следующей таблице:

№ варианта	a	b	c	d	Кол-во входов ИНС	Кол-во НЭ в скрытом слое
1	0.1	0.1	0.05	0.1	6	2
2	0.2	0.2	0.06	0.2	8	3
3	0.3	0.3	0.07	0.3	10	4
4	0.4	0.4	0.08	0.4	6	2
5	0.1	0.5	0.09	0.5	8	3
6	0.2	0.6	0.05	0.6	10	4
7	0.3	0.1	0.06	0.1	6	2
8	0.4	0.2	0.07	0.2	8	3
9	0.1	0.3	0.08	0.3	10	4
10	0.2	0.4	0.09	0.4	6	2
11	0.3	0.5	0.05	0.5	8	3

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import pandas as pd

# Параметры
b = 0.5
c = 0.05
d = 0.5

n_inputs = 8
n_hidden = 3


# Генерация временного ряда
def generate_series(a, N=2000):
    i = np.arange(N)
    y = a * np.cos(b * i) + c * np.sin(d * i)
    return y


# Формирование выборки
def create_dataset(series, look_back=8):
    X, Y = [], []
    for i in range(look_back, len(series)):
```

```
        X.append(series[i - look_back:i])
        Y.append(series[i])
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)


# Модель MLP
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MLP, self).__init__()
        self.hidden = nn.Linear(input_size, hidden_size)
        self.output = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.hidden(x))
        return self.output(x)


# Подбор параметра a
a_values = np.arange(0.1, 0.51, 0.05)
best_a = None
min_test_mse = float('inf')
results = []

for a in a_values:
    y_full = generate_series(a, N=2000)
    X, Y = create_dataset(y_full, look_back=n_inputs)

    # ИСПРАВЛЕННОЕ разделение на обучающую и тестовую выборки
    train_size = int(0.8 * len(X))
    X_train, X_test = X[:train_size], X[train_size:]
    Y_train, Y_test = Y[:train_size], Y[train_size:]

    X_train_t = torch.tensor(X_train)
    Y_train_t = torch.tensor(Y_train).unsqueeze(1)
    X_test_t = torch.tensor(X_test)
    Y_test_t = torch.tensor(Y_test).unsqueeze(1)

    model = MLP(n_inputs, n_hidden)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)

    losses = []
    for epoch in range(1500):
        model.train()
        optimizer.zero_grad()
        pred = model(X_train_t)
        loss = criterion(pred, Y_train_t)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    model.eval()
    with torch.no_grad():
        pred_test = model(X_test_t).numpy().flatten()

    test_mse = float(np.mean((Y_test - pred_test) ** 2))
    results.append({'a': round(a, 2), 'test_mse': test_mse})

    if test_mse < min_test_mse:
        min_test_mse = test_mse
        best_a = round(a, 2)
        best_model = model
        best_losses = losses
```

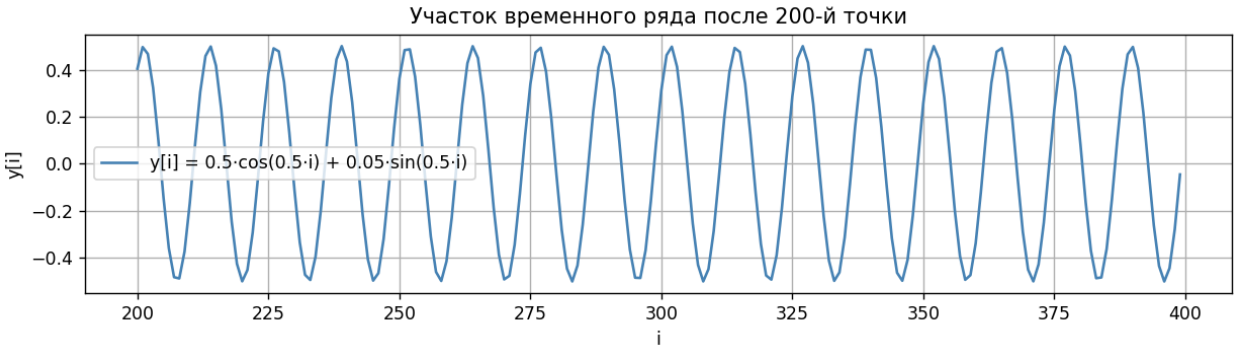
```
        best_split_data = (X_train, Y_train, X_test, Y_test, pred_test)

print(f"Оптимальное a = {best_a} (Test MSE = {min_test_mse:.8f})")

# Результаты для best_a
a = best_a
X_train, Y_train, X_test, Y_test, pred_test = best_split_data

X_train_t = torch.tensor(X_train)
Y_train_t = torch.tensor(Y_train).unsqueeze(1)
X_test_t = torch.tensor(X_test)
Y_test_t = torch.tensor(Y_test).unsqueeze(1)

model = best_model
with torch.no_grad():
    pred_train = model(X_train_t).numpy().flatten()

# График временного ряда после 200-й точки
y_full = generate_series(a, N=2000)
y_plot = y_full[200:400]
plt.figure(figsize=(10, 3))
plt.plot(np.arange(200, 400), y_plot,
         label=f'y[i] = {a}·cos({b}·i) + {c}·sin({d}·i)',
         color='steelblue')
plt.title('Участок временного ряда после 200-й точки')
plt.xlabel('i')
plt.ylabel('y[i]')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# График ошибки обучения
plt.figure(figsize=(8, 4))
plt.plot(best_losses, color='darkorange')
plt.title(f'Ошибка (MSE) при обучении (a = {best_a})')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.yscale('log')
plt.grid(True)
plt.tight_layout()
plt.show()

# Таблица обучения
train_df = pd.DataFrame({
    'Эталонное значение': Y_train[:10],
    'Полученное значение': pred_train[:10],
    'Отклонение': Y_train[:10] - pred_train[:10]
})
print("\n=== РЕЗУЛЬТАТЫ ОБУЧЕНИЯ (первые 10) ===")
print(train_df.round(6).to_string(index=False))

# Таблица прогнозирования
test_df = pd.DataFrame({
    'Эталонное значение': Y_test[:10],
    'Полученное значение': pred_test[:10],
    'Отклонение': Y_test[:10] - pred_test[:10]
})
print("\n=== РЕЗУЛЬТАТЫ ПРОГНОЗИРОВАНИЯ (первые 10) ===")
print(test_df.round(6).to_string(index=False))

# Метрики
train_mse = np.mean((Y_train - pred_train) ** 2)
test_mse = np.mean((Y_test - pred_test) ** 2)
```

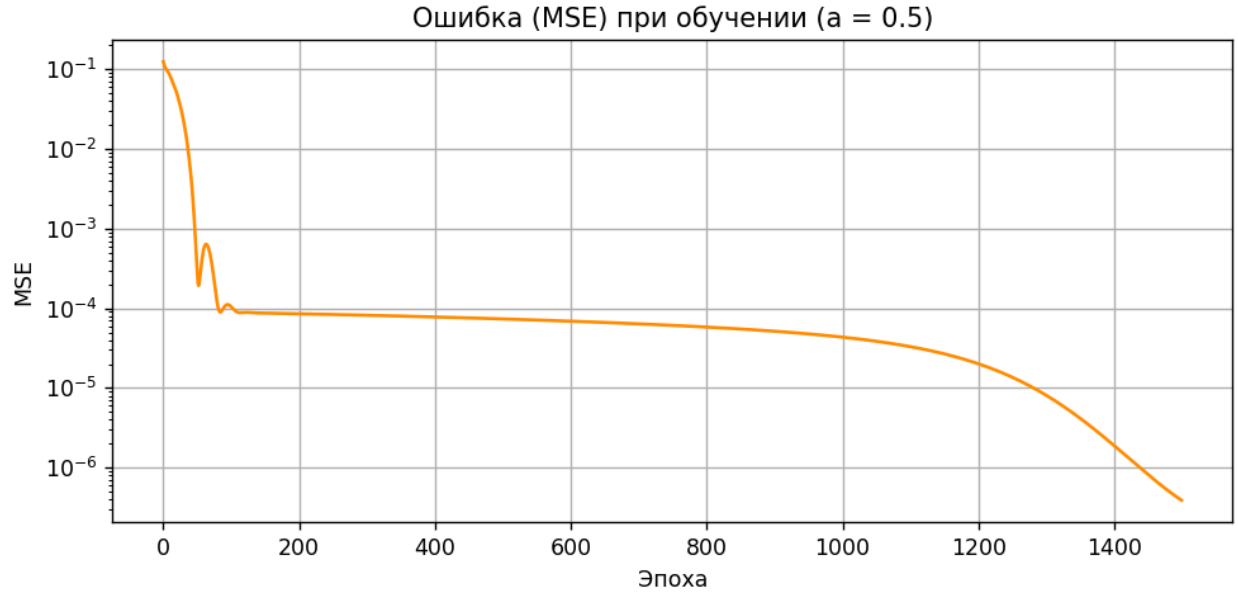
```
train_mae = np.mean(np.abs(Y_train - pred_train))
test_mae = np.mean(np.abs(Y_test - pred_test))

print(f"\nОценка при a = {best_a}:")
print(f"Train → MSE: {train_mse:.8f}, MAE: {train_mae:.8f}")
print(f"Test  → MSE: {test_mse:.8f}, MAE: {test_mae:.8f}")

# Сравнение эталона и прогноза на обучающей и тестовой выборках
plt.figure(figsize=(12, 6))

# Обучающая выборка (первые 100 точек)
plt.subplot(2, 1, 1)
plt.plot(Y_train[:100], label='Эталон (обучение)', color='blue',
linewidth=1.5)
plt.plot(pred_train[:100], label='Прогноз (обучение)', color='red',
linestyle='--', linewidth=1)
plt.title('Сравнение эталона и прогноза на обучающей выборке (первые 100
точек)')
plt.xlabel('Номер точки в обучающей выборке')
plt.ylabel('y')
plt.legend()
plt.grid(True)

# Тестовая выборка (первые 100 точек)
plt.subplot(2, 1, 2)
plt.plot(Y_test[:100], label='Эталон (тест)', color='green', linewidth=1.5)
plt.plot(pred_test[:100], label='Прогноз (тест)', color='orange',
linestyle='--', linewidth=1)
plt.title('Сравнение эталона и прогноза на тестовой выборке (первые 100
точек)')
plt.xlabel('Номер точки в тестовой выборке')
plt.ylabel('y')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

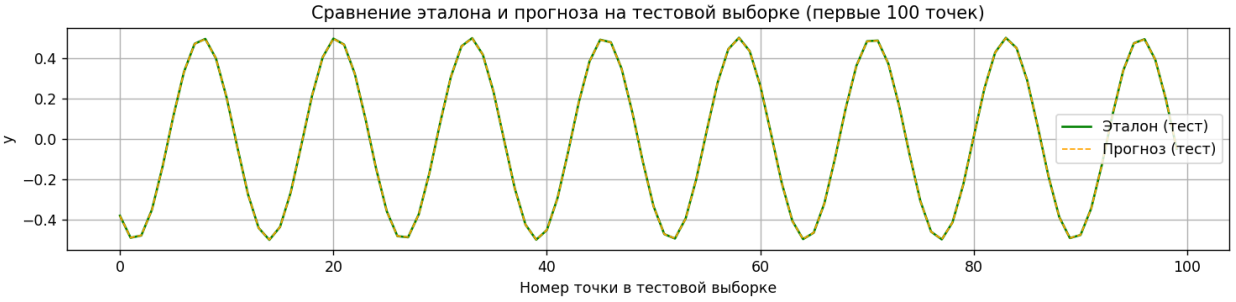
Результаты:

```
=== РЕЗУЛЬТАТЫ ОБУЧЕНИЯ (первые 10) ===
 Эталонное значение  Полученное значение  Отклонение
          -0.103204           -0.102691   -0.000514
          -0.069956           -0.070128    0.000172
          -0.019580           -0.020119    0.000539
           0.035590            0.035279    0.000311
           0.082046            0.082102   -0.000056
           0.108415            0.108665   -0.000250
           0.108240            0.108585   -0.000345
           0.081564            0.081822   -0.000258
           0.034918            0.034733    0.000185
          -0.020277           -0.020888    0.000611

=== РЕЗУЛЬТАТЫ ПРОГНОЗИРОВАНИЯ (первые 10) ===
 Эталонное значение  Полученное значение  Отклонение
          -0.053702           -0.054256    0.000554
          -0.094141           -0.094050   -0.000091
          -0.111531           -0.110853   -0.000679
          -0.101615           -0.101142   -0.000473
          -0.066820           -0.067037    0.000217
          -0.015665           -0.016204    0.000539
           0.039326            0.039043    0.000283
           0.084688            0.084763   -0.000076
           0.109315            0.109574   -0.000259
           0.107179            0.107527   -0.000348
```

```
Оценка при a = 0.1:
Train → MSE: 0.00000015, MAE: 0.00034101
Test  → MSE: 0.00000015, MAE: 0.00034176
```







Вывод: На практике выполнили моделирование прогнозирующей нелинейной ИНС.