**Московский авиационный институт**
**(Национальный исследовательский университет)**
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

# Лабораторная работа № 2
по курсу «Нейроинформатика»

Студент: Аксенов А. Е.
Группа: М80-408Б-20
Преподаватель: Горохов М. А.
Оценка:

Москва, 2023

## Цель работы

Целью работы является исследование свойств линейной нейронной сети и алгоритмов ее обучения, применение сети в задачах аппроксимации и фильтрации.

## Основные этапы работы

1. Использовать линейную нейронную сеть с задержками для аппроксимации функции. В качестве метода обучения использовать адаптацию.
2. Использовать линейную нейронную сеть с задержками для аппроксимации функции и выполнения многошагового прогноза.
3. Использовать линейную нейронную сеть в качестве адаптивного фильтра для подавления помех. Для настройки весовых коэффициентов использовать метод наименьших квадратов.

## Оборудование

Процессор : Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
ОЗУ : 16 ГБ

## Программное обеспечение

Python 3.8 + Jupyter Notebook

# Сценарий выполнения работы

1. Использовать линейную нейронную сеть с задержками для аппроксимации функции. В качестве метода обучения использовать адаптацию.

1.1 Создадим обучающее множество согласно варианту.

$$x = \sin(t^2), \quad t \in [0, 5], \, h = 0.025$$

1.2 Создадим функцию для подготовки выборки для последующего обучения. Данная функция будет реализовывать функции TDL – слоя. Начальное приближения этого слоя зададим нулями. Это понизит прогнозирование в начале временной последовательности, но позже качество будет удовлетворительным. Т к решается задача аппроксимации функции, то будет прогнозировать значении функции сейчас по прошлому.

1.3 Инициализируем сеть. Веса зададим небольшими случайными числами. Отобразим структуру сети. Глубина D = 5, lr = 0.01

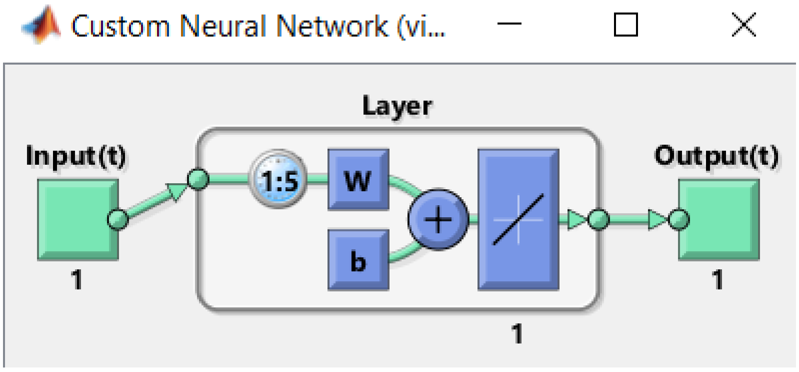


Рис.1 Структура сети

1.4 Обучим сеть по правилу Уидроу-Хоффа. Правило Уидроу-Хоффа формулируется так:

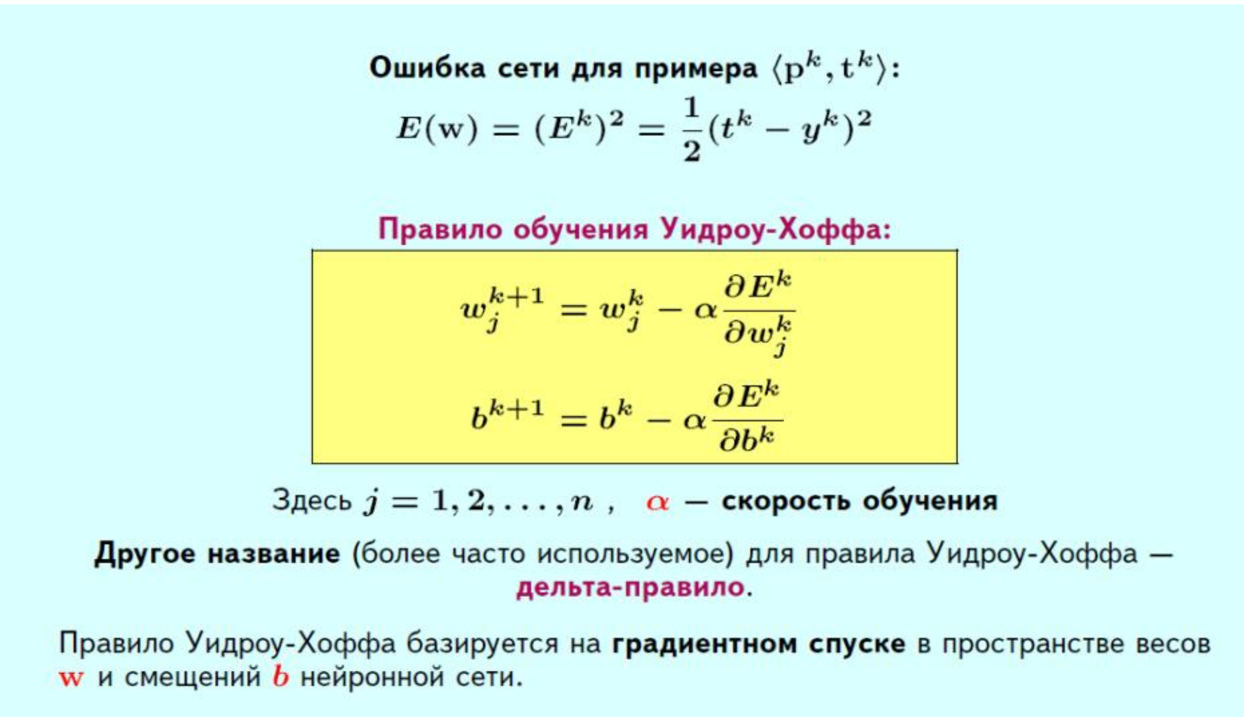


Рис. 2 Правило Уидроу-Хоффа

1.5 Построим график истинной кривой и предсказанной

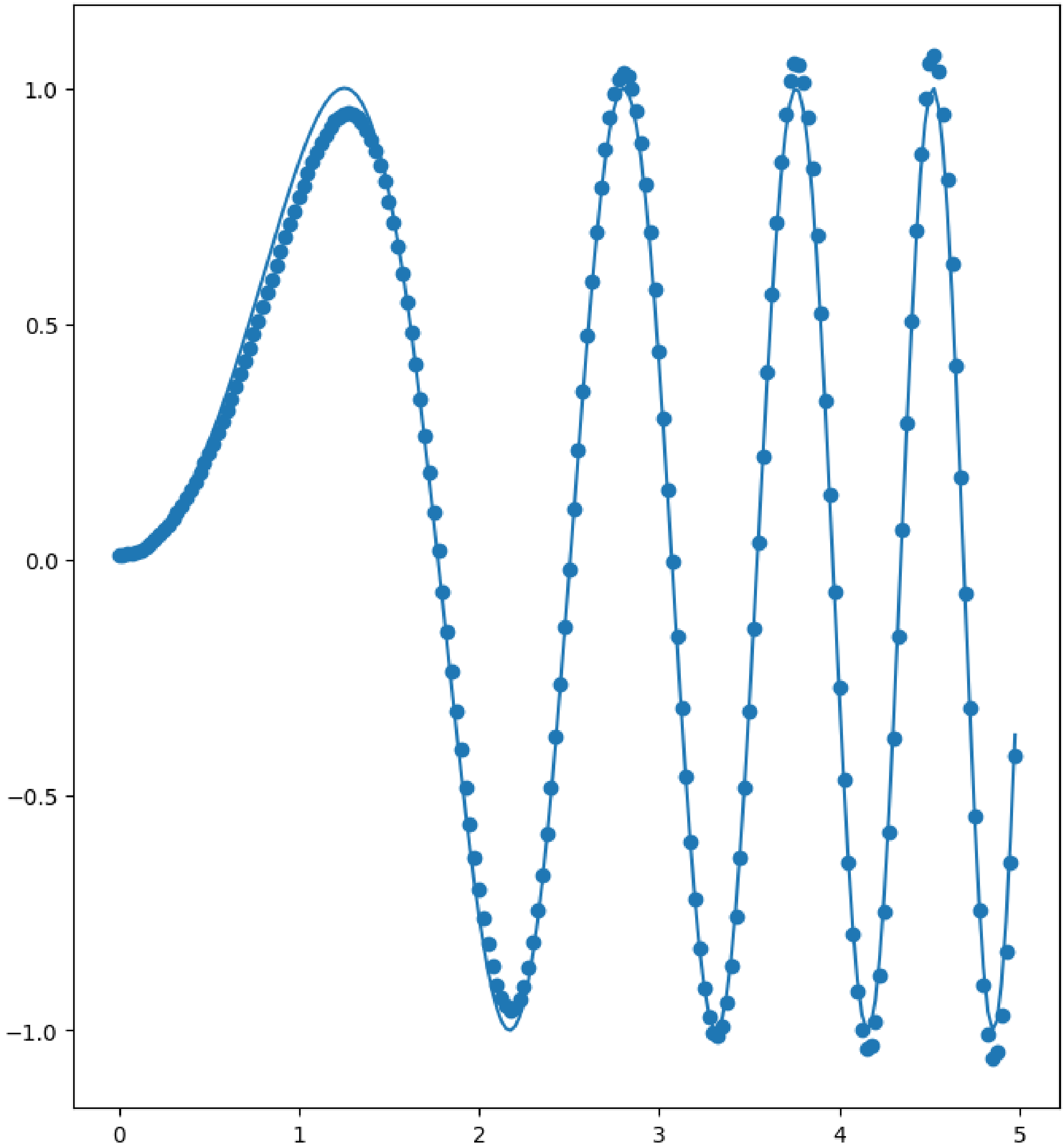


Рис. 3 График предсказанной и истинной прямой

2. Использовать линейную нейронную сеть с задержками для аппроксимации функции и выполнения многошагового прогноза.

2.1 Создадим функцию для подготовки выборки для последующего обучения. Данная функция будет реализовывать функции TDL – слоя. Начальное приближения этого слоя зададим нулями. Это понизит прогнозирование в начале временной последовательности, но позже качество будет удовлетворительным. Т к решается задача многошагового, то будет прогнозировать значении функции потом по сейчас и прошлому.

2.2 Инициализируем сеть. Веса зададим небольшими случайными числами. Отобразим структуру сети. Глубина D = 3, lr = 0.01

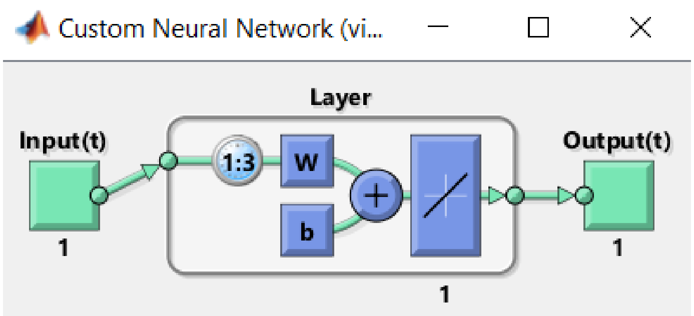


Рис.6 Структура сети

2.3 Обучим модель по правилу правило Уидроу-Хоффа, построим график истинной и предсказанной прямой. Выведим веса модели

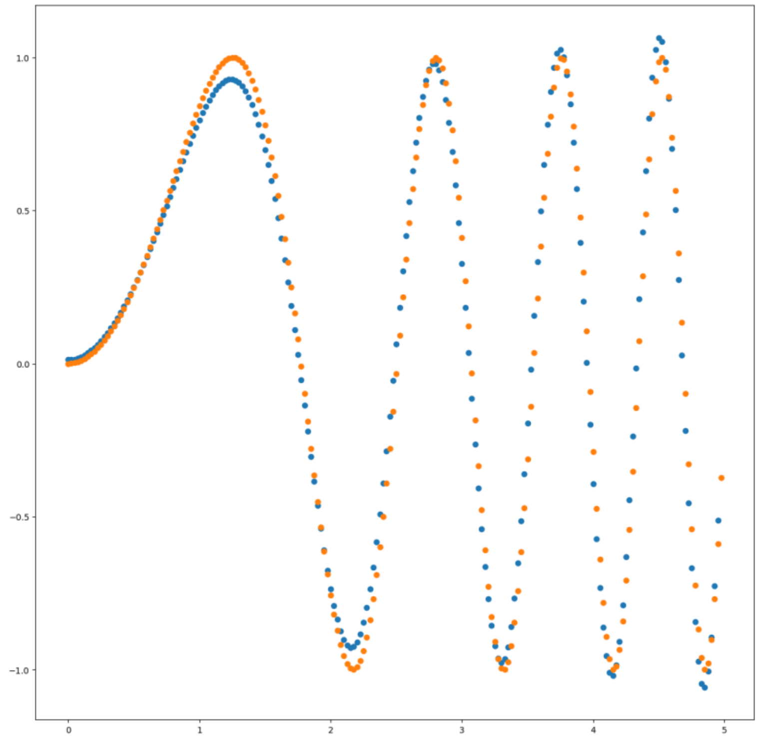

Рис. 7 График предсказанной и истинной прямой

```
# Веса и смещение
model.state_dict()
```

```
OrderedDict([('fc1.weight', tensor([[-0.9886,  0.3176,  1.5701]])),
             ('fc1.bias', tensor([0.0125]))])
```

Рис. 8 Веса модели

2.4 Сделаем прогноза на 10 шагов и построим график, посчитаем ошибку

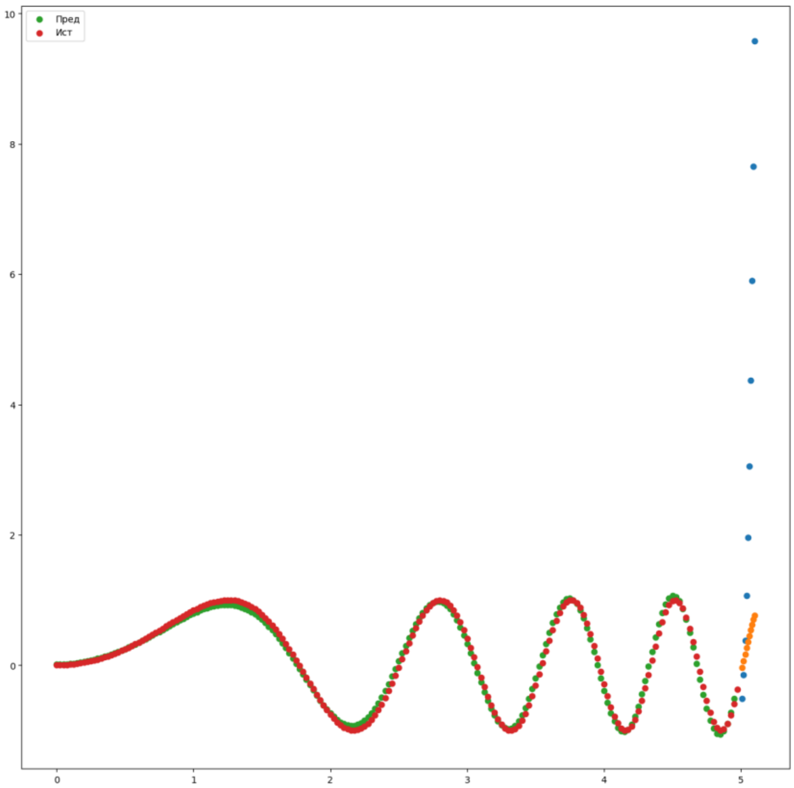


Рис.9 График предсказанной и истинной прямой

```
print('Ошибка для правила Уидроу-Хоффа',np.sqrt(mean_squared_error(y,pred)))
```

```
Ошибка для правила Уидроу-Хоффа 0.05870389416866738
```

Рис. 10 Ошибка

2.5 Отобразим метрики

```
r2_score =  0.9923063407054831
mean_squared_error =  0.0034461471905661
RMSE =  0.05870389416866738
Относительная СКО =  0.0587038877771121
mean_absolute_error =  0.04894274294231336
min absolute error =  2.379427484650032e-05
max absolute error =  0.141512400962775
```

3. Использовать линейную нейронную сеть в качестве адаптивного фильтра для подавления помех. Для настройки весовых коэффициентов использовать метод наименьших квадратов.

3.1 Создадим множество признаков и множество меток согласно варианту. Слева признаки, справа метки. Отрисуем эти множества

$$x = \sin(\frac{2\pi t}{3}), \quad t \in [0,5],\ h = 0.025 \qquad \Big| \qquad y = 0.2\sin(\frac{2\pi t}{3} + \frac{\pi}{2})$$

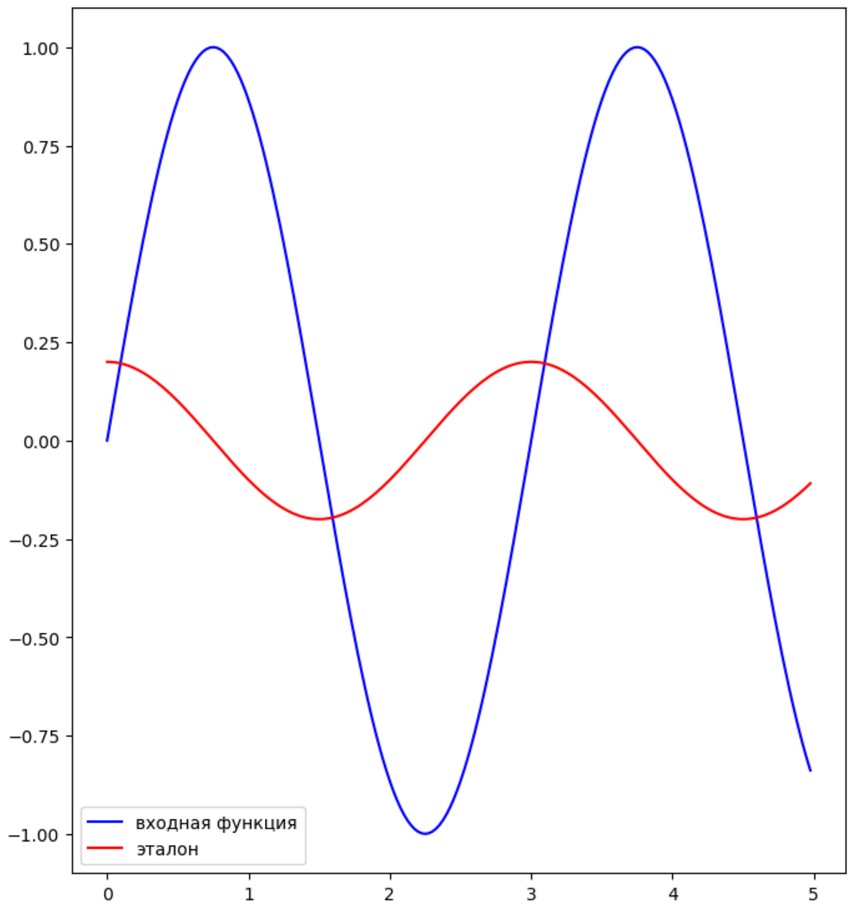


Рис. 13 Входная функция и эталон

3.2 Определим правило обучения LMS так:

**LMS** (Least Mean Square) — алгоритм **обучения с учителем**, предназначенный для обучения сети ADALINE на заданном обучающем наборе.

**Обучающий набор и выходы сети:**

$$\{\langle \mathrm{p}^1, \mathrm{t}^1\rangle, \langle \mathrm{p}^2, \mathrm{t}^2\rangle, \ldots, \langle \mathrm{p}^k, \mathrm{t}^k\rangle, \ldots, \langle \mathrm{p}^L, \mathrm{t}^L\rangle\}$$

$\mathrm{p}^k$ — $k$-й входной вектор, $y^k$ — текущий выход сети; $\mathrm{p}^k \to y^k$

$t^k$ — эталонный (желаемый) выход, отвечающий вектору $\mathrm{p}^k$

**Общая идея алгоритма LMS:**

**Алгоритм LMS** подстраивает **веса и смещения** сети ADALINE таким образом, чтобы минимизировать **среднеквадратическую ошибку**, характеризующую уклонение ***текущего выхода*** сети (при данном текущем значении ее весов и смещений) и ***целевого значения*** этого выхода.

Рис. 14 Правило LMS

3.3 Определим псевдо инверсивное правило так

**Псевдоинверсное правило обучения**

Если матрица $\mathbf{P}$ не является **квадратной**, то вместо правила $\mathbf{W} = \mathbf{T}\mathbf{P}^{-1}$ можно использовать правило

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+},$$

где $\mathbf{P}^{+}$ — псевдообратная матрица, удовлетворяющая условию $\mathbf{P}\mathbf{P}^{+}\mathbf{P} = \mathbf{P}$.

В случае, когда $R > Q$ (число строк в матрице $\mathbf{P}$ больше числа столбцов), псевдообратная матрица может быть вычислена как

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T},$$

следовательно, **псевдоинверсное правило обучения** принимает вид:

$$\mathbf{W} = \mathbf{T}(\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T}$$

Рис. 15 Псевдоинверсное правило

3.3 Обучим модель по правилу псевдоинверсному правилу и LMS. Построим графики истинных и предсказанных кривых, посчитаем ошибки

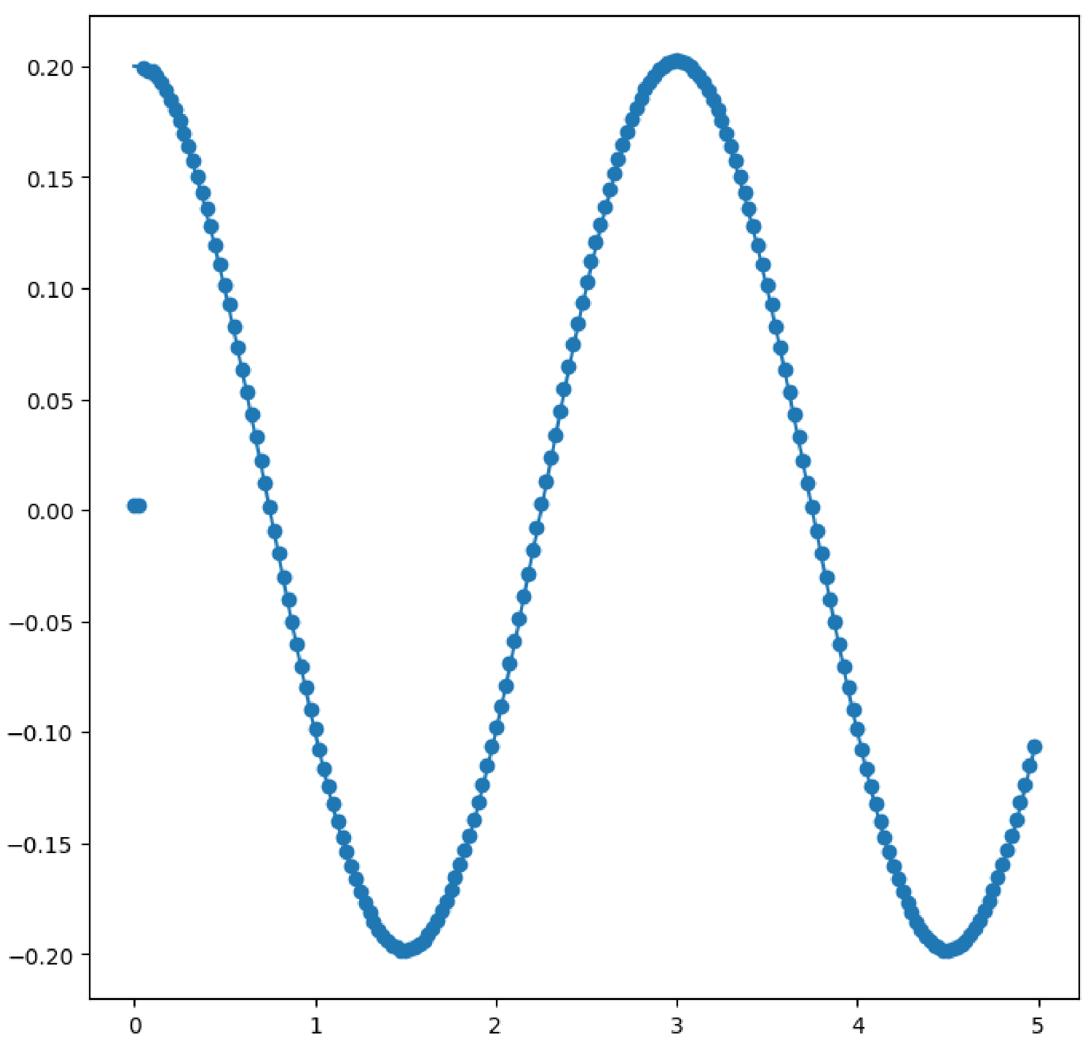

Рис. 16 График истинной и предсказанной кривой модели, обученной по правилу LMS

```
mean_squared_error =  0.00039512446946053724
```

Рис. 17 Ошибка

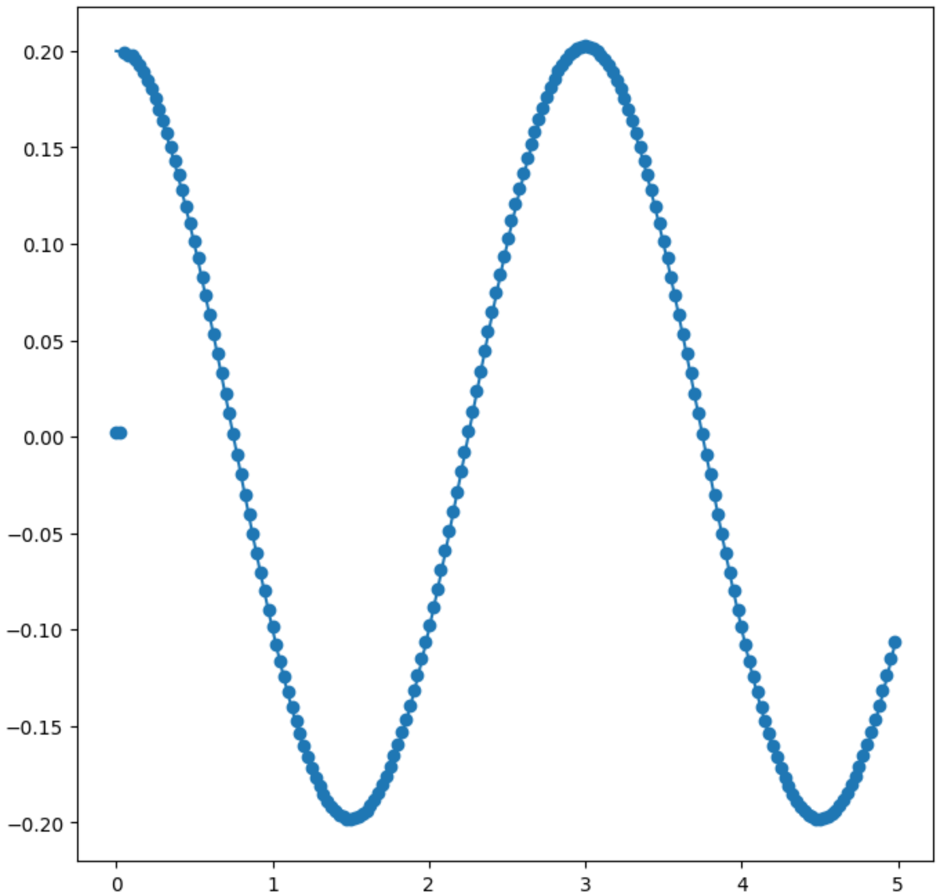


Рис. 18 График истинной и предсказанной кривой модели, обученной по псевдоинверсному правилу

```
mean_squared_error =  0.00039512446946053724
```

Рис. 19  Ошибка модели, обученной по псевдоинверсному правилу

## Код программы

```python
#!/usr/bin/env python
# coding: utf-8

#
# ### Лабораторная 2

# In[1]:


import numpy as np
from sklearn.metrics import accuracy_score, mean_squared_error
from collections import OrderedDict
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
import torch
import pandas as pd
import warnings
warnings.filterwarnings("ignore")


# ### Пункт 1
# - Использовать линейную нейронную сеть с задержками для аппроксимации
функции.

# In[2]:


# Функции
def fun_x_1(t):

    return np.sin(t*t)

def fun_x_2(t):

    return np.sin((2*np.pi*t)/3)

def fun_y(t):

    return 0.2 * np.sin(((2*np.pi*t)/3)+(np.pi/2))


# Пространства
t_1 = np.arange(0, 5, 0.025)
t_2 = np.arange(0, 5, 0.025)

x_1 = torch.tensor(fun_x_1(t_1))
x_2 = torch.tensor(fun_x_2(t_2))
y_1 = fun_y(t_1)
y_2 = fun_y(t_2)


# In[3]:


# Пространства
t_1 = np.arange(0, 5, 0.025)
t_2 = np.arange(0, 5, 0.025)

x_1 = torch.tensor(fun_x_1(t_1))
x_2 = torch.tensor(fun_x_2(t_2))
y_1 = fun_y(t_1)
```

```python
y_2 = fun_y(t_2)


# In[4]:


# Сеть
class Net(nn.Module):
    def __init__(self,D):
        super().__init__()
        self.fc1 = nn.Linear(D, 1)

    def forward(self, x):

        return self.fc1(x)



# Подготовка выборки
def prepare_sample(x, y, D, ):
    back_x = []
    back_y = []


    x = [0 for i in range(D)] + list(x)
    y = [0 for i in range(D)] + list(y)


    for i in range(0,len(x)-D,1):

        back_x.append(x[i:i+D])
        back_y.append(y[i+D])

    back_x = torch.tensor(back_x)
    return back_x, back_y


# Обучение через псевдоинверсивное правило
def train_model_psevda(x_first, y_second, D):
    x, y = prepare_sample(x_first, y_second, D)

    x = list(np.array(x))

    y = np.array(y)
    for i in range(len(x)):
        x[i] = [1] + list(x[i])
    x = np.array(x)

    W = np.dot(y,np.linalg.pinv(x.T))
    model = Net(D=D)
    dict_of_weights = model.state_dict()

    dict_of_weights['fc1.weight'] = torch.tensor(W[1:]).reshape((1,D))

    dict_of_weights['fc1.bias'] = torch.tensor([W[0]])
    model.load_state_dict(dict_of_weights, strict=False)
    x, _ = prepare_sample(x_first, x_first, D)
    x = np.array(x)
    prediction = model(torch.tensor(x).float()).detach().numpy()
    return model, np.sqrt(mean_squared_error(y,prediction)) ,prediction, x, y

# Правило Уидроу-Хоффа
def train_model_Hoffe(x, y, D,  lr=0.01,  epochs = 50):
    model = Net(D=D)
```

```
    x, y = prepare_sample(x, y, D)


    error = 0
    history = {}
    prediction = []
    for i in range(epochs):


        for k,z in zip(x,y):

            error = z - model(k.float())
            dict_of_weights = model.state_dict()
            dict_of_weights['fc1.weight'] = dict_of_weights['fc1.weight'] +
lr * error *(k.float()).T
            dict_of_weights['fc1.bias'] = dict_of_weights['fc1.bias'] +
lr*error
            model.load_state_dict(dict_of_weights, strict=False)

            if i == epochs-1:

                prediction.append(float(model(k.float())))








    final_error = np.sqrt(mean_squared_error(y,prediction))

    return model, final_error, prediction, x, y


# In[5]:


from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error


# In[6]:


# Метрики
def metrics(true, pred):

    true = np.array(true)
    pred = np.array(pred)

    print('r2_score = ', r2_score(true, pred))
    print('mean_squared_error = ', mean_squared_error(true, pred))
    print('RMSE = ', np.sqrt(mean_squared_error(true, pred)))
    print('Относительная СКО = ', np.std(true - pred))
    print('mean_absolute_error = ', mean_absolute_error(true, pred))
    print('min absolute error = ', min(abs(true - pred)))
    print('max absolute error = ', max(abs(true - pred)))




# In[6]:
```

```python
# In[7]:


# Псевдоинверсное правило
model, error_psevdo, pred, x, y = train_model_psevda(x_1, x_1, 5)
plt.figure(figsize=(8,8))
plt.scatter(t_1,pred)
plt.plot(t_1,y)


# In[8]:


metrics(y, pred.reshape(-1))


# In[9]:


print('Ошибка при псевдоинверсном правиле обучения', error_psevdo)


# In[10]:


# Правило Правило Уидроу-Хоффа
model, error_hebb ,pred, x, y= train_model_Hoffe(x_1, x_1,
5,lr=0.01,epochs=50)
plt.figure(figsize=(8,9))
plt.scatter(t_1,pred)
plt.plot(t_1,y)


# In[11]:


metrics(y, pred)


# ###### Псевдоинверсное правило показало себя лучше по метрикам, потому что
применение данного правила дает результат близкий к аналитическому решению

# ### Пункт 2
# - Использовать линейную нейронную сеть с задержками для аппроксимации
функции и выполнения многошагового прогноза.

# In[12]:


# Функция для подготовки выборки
def prepare_sample(x, y, D, ):
    back_x = []
    back_y = []


    x = [0 for i in range(D)] + list(x)
    y = [0 for i in range(D)] + list(y)
```

```
    for i in range(0,len(x)-D-1,1):

        back_x.append(x[i:i+D])
        back_y.append(y[i+D+1])

    back_x = torch.tensor(back_x)
    return back_x, back_y


# In[13]:


# Правило Правило Уидроу-Хоффа

# Функции
def fun_x_1(t):

    return np.sin(t*t)

def fun_x_2(t):

    return np.sin((2*np.pi*t)/3)

def fun_y(t):

    return 0.2 * np.sin(((2*np.pi*t)/3)+(np.pi/2))


# Пространства
t_1 = np.arange(0, 5, 0.025)
t_2 = np.arange(0, 5, 0.025)

x_1 = torch.tensor(fun_x_1(t_1))
x_2 = torch.tensor(fun_x_2(t_2))
y_1 = fun_y(t_1)
y_2 = fun_y(t_2)

model, error_hebb ,pred, x, y= train_model_Hoffe(x_1, x_1,
3,lr=0.001,epochs=600)
plt.figure(figsize=(15,15))
plt.scatter(t_1[:-1],pred)
plt.scatter(t_1,x_1)


# In[13]:





# In[14]:


# Веса и смещение
model.state_dict()


# In[15]:


def get_prediction(model, number, x, D):
    x, y = prepare_sample(x, x ,D)
    learning_part = list(x[-1])
    learning_part = [float(i) for i in learning_part]
```

```
    prediction = []

    for i in range(number):


prediction.append(model(torch.tensor(learning_part)).detach().numpy()[0])
        learning_part = learning_part + [prediction[-1]]

        learning_part = learning_part[1:]

    return prediction


# In[16]:


# Правило Правило Уидроу-Хоффа
new_time = np.array([5 + i/100 for i in range(1,11,1)])
pred_future, y_future = get_prediction(model, 10, x_1, 3), fun_x_1(new_time)
plt.figure(figsize=(15,15))
plt.scatter(new_time,pred_future)
plt.scatter(new_time,y_future)
plt.scatter(t_1[:-1],pred,label='Пред')
plt.scatter(t_1,x_1,label='Ист')
plt.legend()


# In[17]:


metrics(y, pred)


# In[18]:


print('Ошибка для правила Уидроу-Хоффа',np.sqrt(mean_squared_error(y,pred)))


# ##### Прогнозирование выполнено качественно.

# ### Пункт 3
# - Использовать линейную нейронную сеть в качестве адаптивного фильтра для
подавления помех. Для настройки весовых коэффициентов использовать метод
наименьших квадратов

# In[19]:


# Подготовка выборки
def prepare_sample(x, y, D, ):
    back_x = []
    back_y = []


    x = [0 for i in range(D)] + list(x)
    y = [0 for i in range(D)] + list(y)


    for i in range(0,len(x)-D,1):

        back_x.append(x[i:i+D])
        back_y.append(y[i+D])
```

```
    back_x = torch.tensor(back_x)
    return back_x, back_y


# In[20]:


plt.figure(figsize=(8,9))
plt.plot(t_2,x_2,'b',label='входная функция')
plt.plot(t_2,y_2,'r',label='эталон')
plt.legend()


# In[21]:


# МНК
def MNK(x, y, D):

    model = Net(D=D)
    x, y = prepare_sample(x, y, D)

    lstsq_x = []
    for i in x:
        lstsq_x.append([1] + list(i))
    back = np.linalg.lstsq(lstsq_x,y)
    dict_of_weights = model.state_dict()
    dict_of_weights['fc1.weight'] = torch.tensor(back[0][1:]).reshape((1,4))
    dict_of_weights['fc1.bias'] = torch.tensor([back[0][0]])
    model.load_state_dict(dict_of_weights, strict=False)
    prediction = [float(i) for i in
model(torch.tensor(x).float()).detach().numpy()]
    return model, prediction, y

model, pred, y = MNK(x_2, y_2, 4)
plt.figure(figsize=(8,8))
plt.scatter(t_2,pred)
plt.plot(t_2,y)


# In[22]:


metrics(y, pred)


# In[23]:


# Псевдоинверсное правило
model, error_psevdo, pred, x, y = train_model_psevda(x_2, y_2, 4)
plt.figure(figsize=(8,8))
plt.scatter(t_2,pred)
plt.plot(t_2,y)


# In[25]:


metrics(y, pred.reshape(-1))
```

## Выводы

Выполнил лабораторную работу я познакомился с адаптивной фильтрацией реализовал алгоритм Уидроу-Хоффа, МНК и псевдо инверсное правило.

В задаче аппроксимации функции выполнена в целом хорошо. Первые D прогноза показали себя плохо из-за инициализации TDL нулями. Можно было бы инициализировать этот слой первыми числами последовательности, но тогда пришлось бы делать прогноз не с нуля.

В задаче прогнозирования на шаг вперед важным оказалось составление обучающего множества, а именно надо было составить его так, чтоб по информации о сегодня и прошлом мы делали прогноз на завтра.

В задаче подавления шума хоть и не было достигнуто высокое качество на моем варианте, все-таки качество является приемлемым. Так о том, что алгоритмы реализованы правильно, свидетельствует высокое качество прогноза на множестве соседних вариантов задач. В целом прогнозирование шума является интересной и полезной задачей, потому что, приблизив шум, мы можем вычистить его из нашего сигнала и таким образом получить более качественное сообщение.