**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине: «Введение в нейронные сети»

на тему: Анализ временных рядов и аппроксимация

Выполнил:

Студент: группы ИТП-31

Пикулина К.И.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2022

**Цель**: разработать программное обеспечение для решения задачи аппроксимации и анализа временных рядов.

**Ход работы**

# **Задание:**

Задача 1. Для аппроксимации использовать радиально базисную сеть, представленную на рисунке 1 и персептрон – рисунок 2.

Функция для аппроксимации соответствующее варианту.



Количество нейронов на скрытом слое радиально базисной сети и персептрона соответственно: 6 и 8.

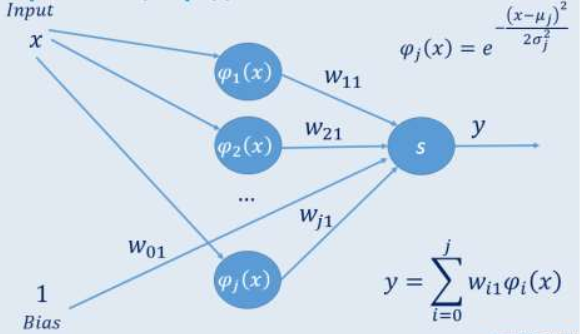


Рисунок 1 – Радиально-базисная сеть

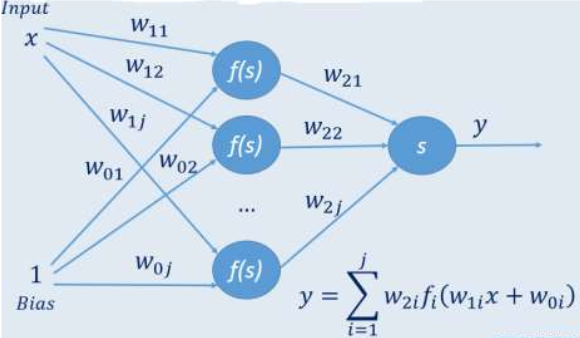


Рисунок 2 – Персептрон

Разработать соответствующее программное обеспечение.

Отобразить результаты аппроксимации графически на одном поле (3 графика).

Дополнить персептрон ещё одним скрытым слоем (количество нейронов выбрать самостоятельно).

Сравнить результаты аппроксимации с помощью радиально-базисной сети, однослойным персептроном и двухслойным персептроном.

Исследовать влияние количества нейронов на скрытом слое на погрешность аппроксимации. Сделать вывод.

Задача 2. Используя метод скользящего окна выполнить экстраполяцию данных на январь месяц 2019 года персептроном (рисунок 3) и сетью Элмана (рисунок 4). Исследовать качество экстраполяции от представления данных, глубины погружения временного ряда и структуры нейронной сети.

Экстраполируемые данные: Курс белорусского рубля к юаню (график представлен на рисунке 7).

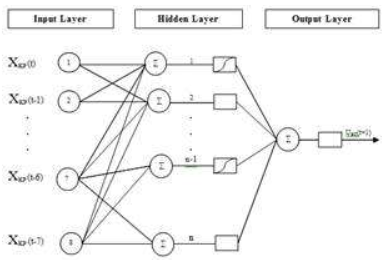


Рисунок 3 – Персептрон со скользящим окном

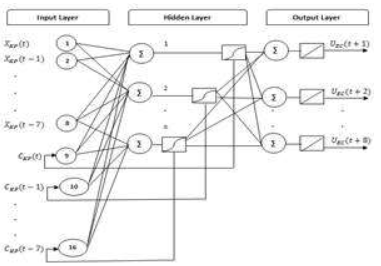


Рисунок 4 – Сеть Элмана

График аппроксимации после обучения сетей из Задания 1 приведён на рисунке 5.

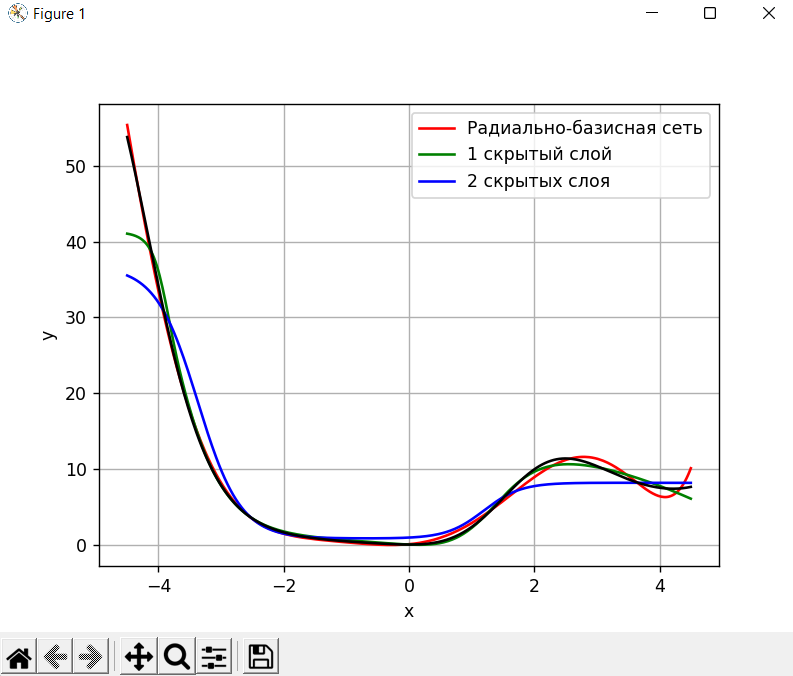


Рисунок 5 – Задание 1. Аппроксимация функции

По результату аппроксимации приведённом на рисунке 5 можно сделать вывод, лучше всего приближает функцию радиально-базисная сеть, на втором месте персептрон с одним скрытым слоем, хуже всего – персептрон с двумя скрытыми слоями (каждый скрытый слой содержит 3 нейрона), так как по графику можно видеть, что он пытается найти мелкие изменения функции и в итоге плохо приближает функцию в общем.

Так как обучение начинается со случайных значений на весах нейронных сетей, то результат обучения может отличаться при разных запусках программы. Это характерно для всех нейронных сетей.

График аппроксимации после обучения сетей, а именно персептронов с разным количеством нейронов на скрытом слое, из Задания 1 приведён на рисунке 6.

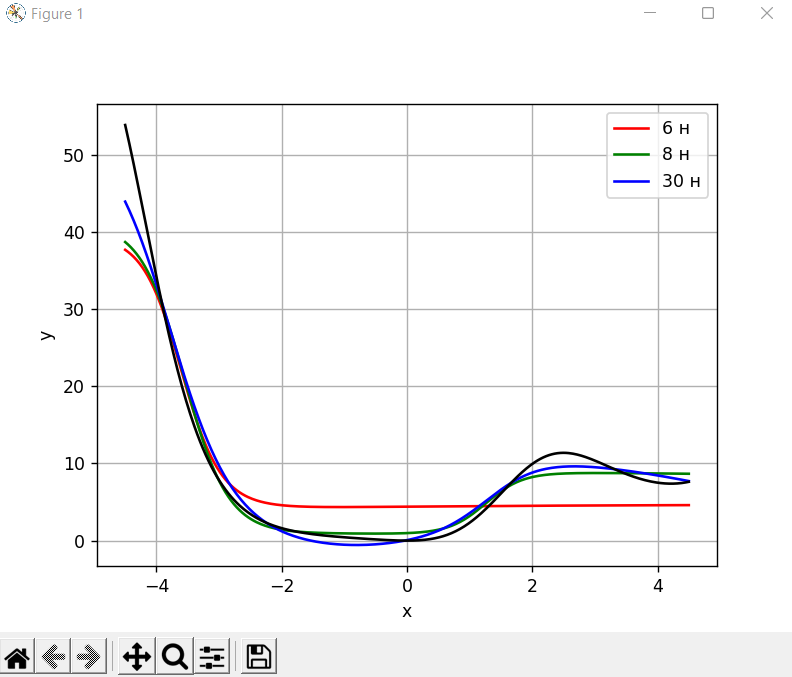


Рисунок 6 – Задание 1. Аппроксимация функции

По результату аппроксимации приведённом на рисунке 6 можно сделать вывод, что персептрона даже с 6 нейронами на скрытом слое достаточно для вполне хорошего приближения заданной функции, однако на краях графика этот персептрон проигрывает персептронам, у которых 8 и 30 нейронов на скрытом слое соответственно.

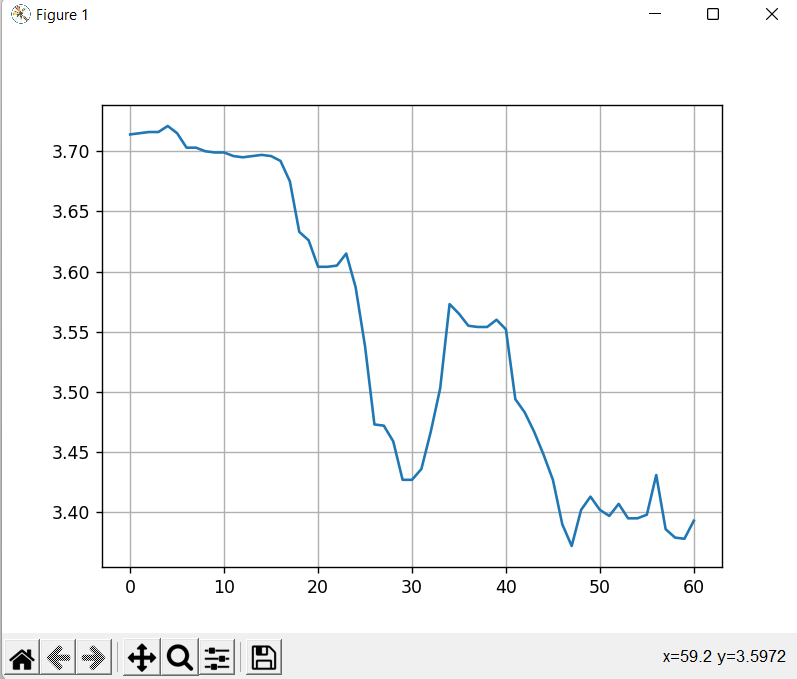


Рисунок 7 – Задание 2. График температуры

График экстраполяции данных Персептроном и сетью Элмана изображён на рисунке 8.

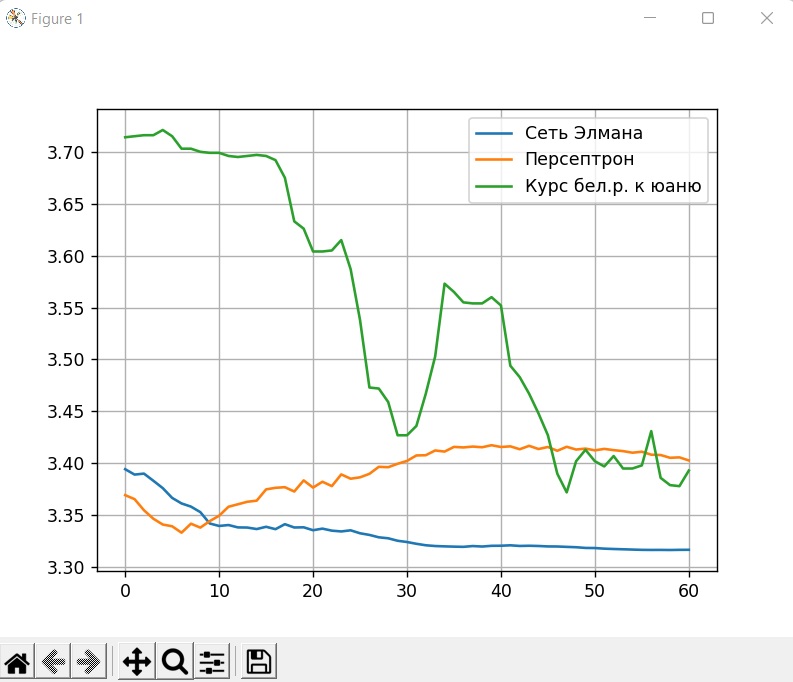


Рисунок 8 – Задание 2. Экстраполяция данных

По результату экстраполяции приведённом на рисунке 8 можно сделать вывод, что нейронные сети, обученные с использованием метода скользящего окна, достаточно хорошо могут выполнять прогноз, то есть выполнять задачу экстраполяции. Глубина погружения – 30 значений. Для предсказания одного нового значения сеть должна получить 30 прошлых значений. Если будет меньше, то сеть не выучит долгосрочные изменения и будет слишком сильно полагаться на последние значения. Если глубина будет больше, то сеть может упустить мелкие изменения или же при недостатке данных такую сеть будет невозможно качественно обучить.

**Вывод:** в результате выполнения лабораторной работы было разработано программное обеспечение для решения задачи экстраполяции и анализа временных рядов.

**Приложение А**

**Листинг программы**

import numpy as np  
  
  
class LinearLayer:  
 def \_\_init\_\_(self, input\_number, output\_number, activation="none", need\_bias=False):  
  
 self.weight = np.random.uniform(low=-np.sqrt(1 / input\_number),  
 high=np.sqrt(1 / input\_number), size=[input\_number, output\_number])  
  
 self.need\_bias = need\_bias  
 self.bias = 0  
 if need\_bias:  
 self.bias = np.random.uniform(low=-np.sqrt(1 / input\_number),  
 high=np.sqrt(1 / input\_number), size=[1, output\_number])  
  
 self.activation = lambda x: x  
 self.der\_activation = lambda x: 1  
  
 if activation == "tanh":  
 self.activation = lambda x: np.tanh(x)  
 self.der\_activation = lambda x: 1 - self.activation(x) \*\* 2  
  
 def forward(self, x):  
 self.input = x  
 self.mid = x @ self.weight  
 if self.need\_bias:  
 self.mid += self.bias  
 self.out = self.activation(self.mid)  
 return self.out  
  
 def backward(self, grad):  
 dout = grad \* self.der\_activation(self.mid)  
 self.dw = self.input.T @ dout  
 if self.need\_bias:  
 self.db = dout  
 dinp = dout @ self.weight.T  
 return dinp  
  
  
class NN:  
 def \_\_init\_\_(self, lr=0.01):  
 self.lr = lr  
 self.layers = []  
 self.optim = self.SGD  
 self.criterion = self.MSELoss  
  
 self.out = 0  
  
 def add\_layer(self, input\_number, output\_number, activation="none", need\_bias=False):  
 self.layers.append(LinearLayer(input\_number, output\_number, activation, need\_bias))  
  
 def forward(self, x):  
 z = x  
 for layer in self.layers:  
 z = layer.forward(z)  
 self.out = z  
 return z  
  
 def \_\_call\_\_(self, \*args):  
 return self.forward(\*args)  
  
 def backward(self, y):  
 grad = self.criterion(y, True)  
 for layer in list(reversed(self.layers)):  
 grad = layer.backward(grad)  
  
 def MSELoss(self, y, der=False):  
 if not der:  
 return np.mean((self.out - y) \*\* 2)  
 return self.out - y  
  
 def update(self):  
 for layer in self.layers:  
 self.optim(layer.weight, layer.dw)  
 if layer.need\_bias:  
 self.optim(layer.bias, layer.db)  
  
 def SGD(self, weight, dw):  
 weight -= self.lr \* dw

import numpy as np  
  
  
class RBFNet:  
 def \_\_init\_\_(self, hidden\_number, sigma=1.0):  
 self.hidden\_number = hidden\_number  
 self.sigma = sigma  
 self.centers = 0  
 self.weights = 0  
  
 def rbf(self, point, center):  
 return np.exp(np.linalg.norm((point - center) \*\* 2 / (2 \* self.sigma \*\* 2)))  
  
 def calculate\_interpolation\_matrix(self, x):  
 g = np.zeros((len(x), self.hidden\_number))  
 for i, point in enumerate(x):  
 for j, center in enumerate(self.centers):  
 g[i, j] = self.rbf(point, center)  
 return g  
  
 def fit(self, x, y):  
 self.centers = x[np.random.choice(len(x), self.hidden\_number)]  
 g = self.calculate\_interpolation\_matrix(x)  
 inv\_g = np.linalg.pinv(g)  
 self.weights = inv\_g @ y  
  
 def predict(self, x):  
 g = self.calculate\_interpolation\_matrix(x)  
 return g @ self.weights

import numpy as np  
  
class RNN:  
  
 def \_\_init\_\_(self, input\_number, hidden\_number, output\_number, lr=0.01):  
 self.lr = lr  
  
 self.w\_ih = np.random.uniform(-np.sqrt(1 / input\_number), np.sqrt(1 / input\_number),  
 size=[input\_number, hidden\_number])  
 self.b\_ih = np.random.uniform(size=[1, hidden\_number])  
  
 self.w\_hh = np.random.uniform(-np.sqrt(1 / hidden\_number), np.sqrt(1 / hidden\_number),  
 size=[hidden\_number, hidden\_number])  
 self.b\_hh = np.random.uniform(size=[1, hidden\_number])  
  
 self.h = np.zeros(shape=[1, hidden\_number])  
 self.h\_t\_1 = np.zeros(shape=[1, hidden\_number])  
  
 self.w = np.random.uniform(-np.sqrt(1 / hidden\_number), np.sqrt(1 / hidden\_number),  
 size=[hidden\_number, output\_number])  
  
 def forward(self, x):  
 self.x = x  
 self.h\_t\_1 = self.h  
 self.h = self.x @ self.w\_ih + self.b\_ih + self.h\_t\_1 @ self.w\_hh + self.b\_hh  
 self.h = np.tanh(self.h)  
 self.out = self.h @ self.w  
 return self.out  
  
 def \_\_call\_\_(self, \*args):  
 return self.forward(\*args)  
  
 def backward(self, y):  
 dloss = self.out - y  
 self.dw = self.h.T @ dloss  
 dh = dloss @ self.w.T  
 grad = (1 - np.tanh(self.h) \*\* 2) \* dh  
 self.dw\_ih = self.x.T @ grad  
 self.db\_ih = 1 \* grad  
 self.dw\_hh = self.h\_t\_1.T @ grad  
 self.db\_hh = 1 \* grad  
  
 def update(self):  
 self.w -= self.lr \* self.dw  
 self.w\_ih -= self.lr \* self.dw\_ih  
 self.b\_ih -= self.lr \* self.db\_ih  
 self.w\_hh -= self.lr \* self.dw\_hh  
 self.b\_hh -= self.lr \* self.db\_hh

import RBFNet as rBFNet  
import RNN as rNN  
  
import numpy as np  
import matplotlib.pyplot as plt  
  
import NeuralNetwork as nn  
  
  
def TraningFunc(fx, rbf, perceptron\_1, perceptron\_2, perceptron\_1x2, perceptron\_1x3):  
 x = np.random.uniform(-4.5, 4.5, size=[75000, 1])  
 y = fx(x)  
 rbf.sigma = np.std(y)  
 rbf.fit(x, y)  
 for i in range(75000):  
 xi = x[i].reshape(1, 1)  
 yi = y[i].reshape(1, 1)  
 perceptron\_1.forward(xi)  
 perceptron\_1.backward(yi)  
 perceptron\_1.update()  
 perceptron\_2.forward(xi)  
 perceptron\_2.backward(yi)  
 perceptron\_2.update()  
 perceptron\_1x2.forward(xi)  
 perceptron\_1x2.backward(yi)  
 perceptron\_1x2.update()  
 perceptron\_1x3.forward(xi)  
 perceptron\_1x3.backward(yi)  
 perceptron\_1x3.update()  
 return True  
  
  
def TraningFunc2(seq\_length, train, rnn, perceptron\_3, mean, std):  
 for epoch in range(500):  
 q = np.random.randint(0, seq\_length)  
 for i in range(q, len(train) - seq\_length, seq\_length):  
 x = (np.array(train[i: i + seq\_length]).reshape(1, seq\_length) - mean) / std  
 y = (np.array(train[i + seq\_length: i + seq\_length + 1]).reshape(1, 1) - mean) / std  
 rnn(x)  
 rnn.backward(y)  
 rnn.update()  
 perceptron\_3.forward(x)  
 perceptron\_3.backward(y)  
 perceptron\_3.update()  
 return True  
  
  
def create\_nets\_for\_task\_1():  
 rbf = rBFNet.RBFNet(10)  
 perceptron\_1 = nn.NN(0.0001)  
 perceptron\_1.add\_layer(1, 3, "tanh", need\_bias=True)  
 perceptron\_1.add\_layer(3, 1)  
 perceptron\_2 = nn.NN(0.0001)  
 perceptron\_2.add\_layer(1, 3, "tanh", need\_bias=True)  
 perceptron\_2.add\_layer(3, 3, "tanh", need\_bias=True)  
 perceptron\_2.add\_layer(3, 1)  
 perceptron\_1x2 = nn.NN(0.0001)  
 perceptron\_1x2.add\_layer(1, 10, "tanh", need\_bias=True)  
 perceptron\_1x2.add\_layer(10, 1)  
 perceptron\_1x3 = nn.NN(0.0001)  
 perceptron\_1x3.add\_layer(1, 30, "tanh", need\_bias=True)  
 perceptron\_1x3.add\_layer(30, 1)  
 return rbf, perceptron\_1, perceptron\_2, perceptron\_1x2, perceptron\_1x3  
  
  
def create\_nets\_for\_task\_2(seq\_length):  
 rnn = rNN.RNN(seq\_length, 40, 1, 0.003)  
 perceptron\_3 = nn.NN(0.003)  
 perceptron\_3.add\_layer(seq\_length, 40, "tanh")  
 perceptron\_3.add\_layer(40, 1)  
 return rnn, perceptron\_3  
  
  
def deserialize(filename):  
 with open(filename, "r") as file:  
 rows = file.read().split()  
 data = []  
 for i, k in enumerate(rows[:-1]):  
 data.append(float(k))  
 return data  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 train = deserialize(r"train.txt")  
 test = deserialize(r"test.txt")  
 mean = np.mean(train)  
 std = np.std(train)  
  
 training\_complete = False  
 fx = lambda x: np.exp(-x) \* (np.arctan(x))  
 rbf, perceptron\_1, perceptron\_2, perceptron\_1x2, perceptron\_1x3 = create\_nets\_for\_task\_1()  
  
 training\_complete\_2 = False  
 seq\_length = 30  
 rnn, perceptron\_3 = create\_nets\_for\_task\_2(seq\_length)  
  
 flag = False  
 k = 0  
 while not flag:  
 print(  
 "1. График аппроксимации\n2. Сравнение сетей с разным кол-вом нейронов на скрытом слое\n3. График температуры\n4. График предсказаний на январь\n")  
 try:  
 k = int(input())  
 except ValueError:  
 pass  
 print()  
 training\_complete = TraningFunc(fx, rbf, perceptron\_1, perceptron\_2, perceptron\_1x2, perceptron\_1x3)  
 training\_complete\_2 = TraningFunc2(seq\_length, train, rnn, perceptron\_3, mean, std)  
 if k == 1:  
 if training\_complete:  
 points = np.linspace(-4.5, 4.5, 400)  
 p1 = []  
 p2 = []  
 for i in points:  
 p1.append(perceptron\_1(i.reshape(1, 1)).reshape(-1))  
 p2.append(perceptron\_2(i.reshape(1, 1)).reshape(-1))  
 plt.plot(points, rbf.predict(points), "r", label="Радиально-базисная сеть")  
 plt.plot(points, p2, "g", label="1 скрытый слой")  
 plt.plot(points, p1, "b", label="2 скрытых слоя")  
 plt.plot(points, fx(points), "k")  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.legend()  
 plt.grid()  
 plt.show()  
 elif k == 2:  
 if training\_complete:  
 points = np.linspace(-4.5, 4.5, 400)  
 p1 = []  
 p1\_x2 = []  
 p1\_x3 = []  
 for i in points:  
 p1.append(perceptron\_1(i.reshape(1, 1)).reshape(-1))  
 p1\_x2.append(perceptron\_1x2(i.reshape(1, 1)).reshape(-1))  
 p1\_x3.append(perceptron\_1x3(i.reshape(1, 1)).reshape(-1))  
 plt.plot(points, p1, "r", label="3 н")  
 plt.plot(points, p1\_x2, "g", label="10 н")  
 plt.plot(points, p1\_x3, "b", label="30 н")  
 plt.plot(points, fx(points), "k")  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.legend()  
 plt.grid()  
 plt.show()  
 elif k == 3:  
 plt.plot(train)  
 plt.grid()  
 plt.show()  
 elif k == 4:  
 if training\_complete\_2:  
 rnn\_y = []  
 perceptron\_y = []  
 for i in train[-seq\_length:]:  
 rnn\_y.append((i - mean) / std)  
 perceptron\_y.append((i - mean) / std)  
 for i in range(len(test)):  
 out\_rnn = rnn(np.array(rnn\_y[i:i + seq\_length]).reshape(1, seq\_length)).reshape(-1)  
 out\_perceptron\_3 = perceptron\_3(  
 np.array(perceptron\_y[i:i + seq\_length]).reshape(1, seq\_length)).reshape(-1)  
 rnn\_y.append(out\_rnn[0])  
 perceptron\_y.append(out\_perceptron\_3[0])  
 plt.plot(np.array(rnn\_y[seq\_length:]) \* std + mean, label="Сеть Элмана")  
 plt.plot(np.array(perceptron\_y[seq\_length:]) \* std + mean, label="Персептрон")  
 plt.plot(test, label="Температура")  
 plt.legend()  
 plt.grid()  
 plt.show()