

Лабораторная работа №1

Задача: Создать, используя nn tool НС для аппроксимации функции.

Значения для входа и цели вычислить в заданном диапазоне. Привести схему сети (simulink), графики , характеризующие качество обучения сети и ее работу : окно состояния обучения, изменение ошибки сети в процессе обучения, окно линейной регрессии между выходом НС и целями, пояснить графики и сделать выводы.

Проверить работу сети, используя входные данные из заданного диапазона, но не использовавшиеся для обучения сети.

Сравнить результаты, используя nnstart (fitting app) и соответственно функцию fitnet для решения той же задачи. Привести графики, сделать выводы.

Вариант

$$y = x_1 \cdot \sin(x_2), \quad x_1, x_2 \in [-1;1]$$

Ход работы:

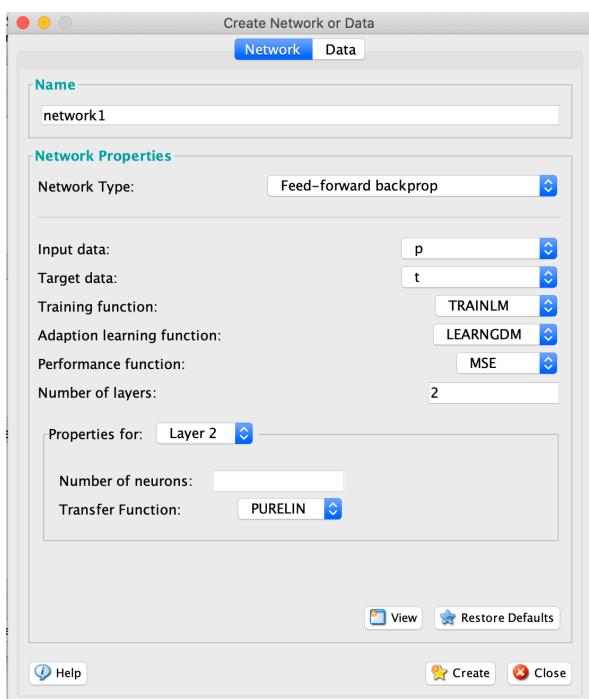
Создадим входные данные x1, x2 и данные цели

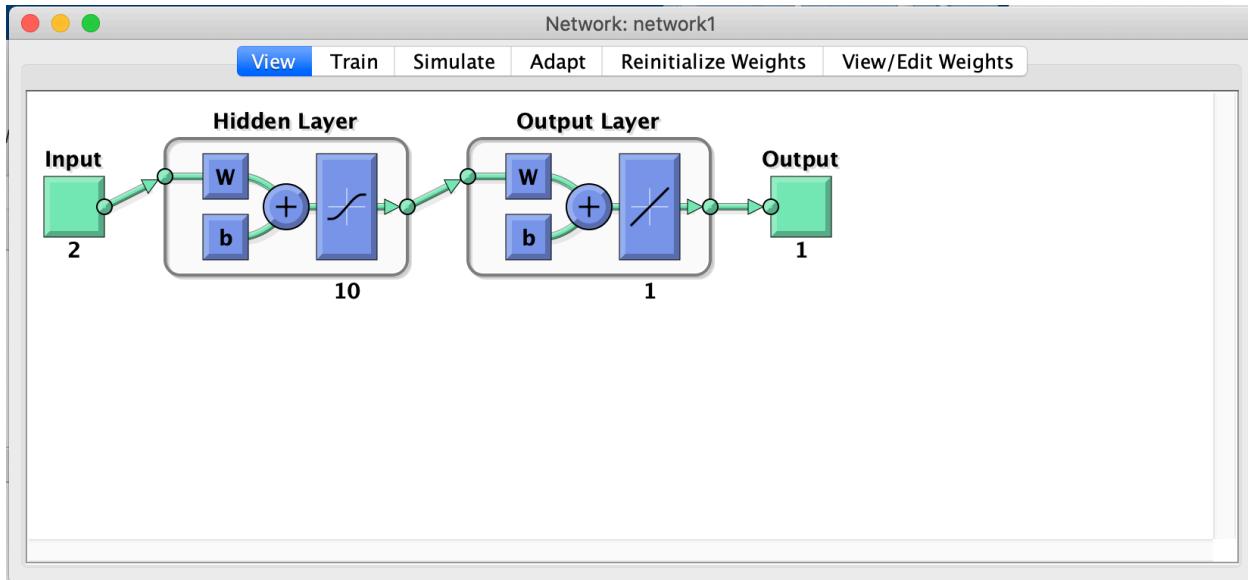
```
1 |from __future__ import division
2 |import random
3 |import math
4 |
5 |x1 = []
6 |x2 = []
7 |y = []
8 |for i in range(100):
9 |    x1.append(round(random.uniform(-1, 1), 2))
10 |   x2.append(round(random.uniform(-1, 1), 2))
11 |   y.append(round(math.sqrt(x2[i] ** 2 + x1[i] ** 2), 2))
12 |
13 |x1_str = ''
14 |x2_str = ''
15 |y_str = ''
16 |for i in range(100):
17 |    x1_str += ' ' + str(x1[i])
18 |    x2_str += ' ' + str(x2[i])
19 |    y_str += ' ' + str(y[i])
20 |x12_input = '[' + x1_str + ';' + x2_str + ']'
21 |y_input = '[' + y_str + ']'
22 |
23 |input = open('input.txt', 'w')
24 |input.write(x12_input)
25 |input.write('\n')
26 |input.write(y_input)
27 |input.close()
28 |
29 |print(math.sqrt(0.7 ** 2 + 0.3 ** 2))
30 |print(math.sqrt(0.3 ** 2 + 0.2 ** 2))
31 |print(math.sqrt(0.9 ** 2 + 0.9 ** 2))
32 |
```

```
p = [ -0.56 0.81 -0.52 -0.7 0.47 -0.55 -0.86 -0.03 -0.1 0.87 -0.3 0.87 -0.52 -0.53 -0.46 -0.44
-0.38 -0.08 0.63 -0.05 -0.92 -0.18 -0.09 -0.1 -0.11 0.35 0.18 0.49 -0.51 0.06 -0.34 0.21 0.5 0.86
-0.84 0.88 -0.57 -0.85 0.87 -0.75 0.74 -0.41 0.16 0.75 0.95 -0.93 -0.18 -0.25 -0.04 0.69 -0.07
-0.23 0.79 -0.36 0.35 -0.34 0.18 -0.75 -0.54 0.85 0.68 -0.1 0.91 -0.18 -0.73 -0.39 0.05 -0.78
-0.24 -0.38 0.12 0.21 0.3 -0.33 0.62 0.74 -0.02 0.63 -0.55 0.04 -0.04 -0.37 -0.1 -0.38 0.5 -0.48
-0.56 -0.99 0.96 0.02 0.82 0.56 0.15 -0.26 0.22 -0.96 -0.39 0.9 -0.47 -0.44; -0.16 0.76 -0.31 0.97
0.31 -0.3 0.65 0.66 0.07 -0.09 0.02 0.3 -0.95 -0.5 0.66 0.45 0.42 -0.55 -0.98 0.39 -0.63 0.72 0.5
0.0 -0.03 -0.42 0.3 0.38 0.65 0.4 0.06 0.5 0.73 0.22 0.07 0.21 0.16 0.28 -0.44 0.56 0.26 0.22
-0.22 -0.59 -0.75 0.76 0.25 -0.68 -0.86 0.68 -0.85 0.69 -0.73 -0.74 -0.34 0.6 -0.73 0.56 -0.5 -0.9
-0.52 0.75 -0.8 -0.36 0.39 -0.1 0.89 -0.73 0.63 -0.3 -0.31 -0.37 -0.56 0.15 -0.98 -0.48 0.83 -0.52
0.95 -0.57 0.96 0.52 0.8 -0.19 0.9 -0.21 -0.84 0.57 -0.46 -0.47 0.06 0.9 -0.48 0.54 0.06 -0.97
0.43 0.38 -0.17 0.71]

t = [ 0.58 1.11 0.61 1.2 0.56 0.63 1.08 0.66 0.12 0.87 0.3 0.92 1.08 0.73 0.8 0.63 0.57 0.56 1.17
0.39 1.12 0.74 0.51 0.1 0.11 0.55 0.35 0.62 0.83 0.4 0.35 0.54 0.88 0.89 0.84 0.9 0.59 0.89 0.97
0.94 0.78 0.47 0.27 0.95 1.21 1.2 0.31 0.72 0.86 0.97 0.85 0.73 1.08 0.82 0.49 0.69 0.75 0.94
0.74 1.24 0.86 0.76 1.21 0.4 0.83 0.4 0.89 1.07 0.67 0.48 0.33 0.43 0.64 0.36 1.16 0.88 0.83
0.82 1.1 0.57 0.96 0.64 0.81 0.42 1.03 0.52 1.01 1.14 1.06 0.47 0.82 1.06 0.5 0.6 0.23 1.36 0.58
0.98 0.5 0.84]
```

Создадим нейронную сеть





Зададим параметры на обучение

Network: network1

View Train Simulate Adapt Reinitialize Weights View/Edit Weights

Training Info Training Parameters

Training Data		Training Results	
Inputs	p	Outputs	network1_outputs
Targets	t	Errors	network1_errors
Init Input Delay States	(zeros)	Final Input Delay States	network1_inputStates
Init Layer Delay States	(zeros)	Final Layer Delay States	network1_layerStates

Train Network

Network: network1

View Train Simulate Adapt Reinitialize Weights View/Edit Weights

Training Info Training Parameters

showWindow	true	mu	0.001
showCommandLine	false	mu_dec	0.1
show	25	mu_inc	10
epochs	1000	mu_max	10000000000
time	Inf		
goal	0		
min_grad	1e-010		
max_fail	6		

Train Network

Обучим сеть

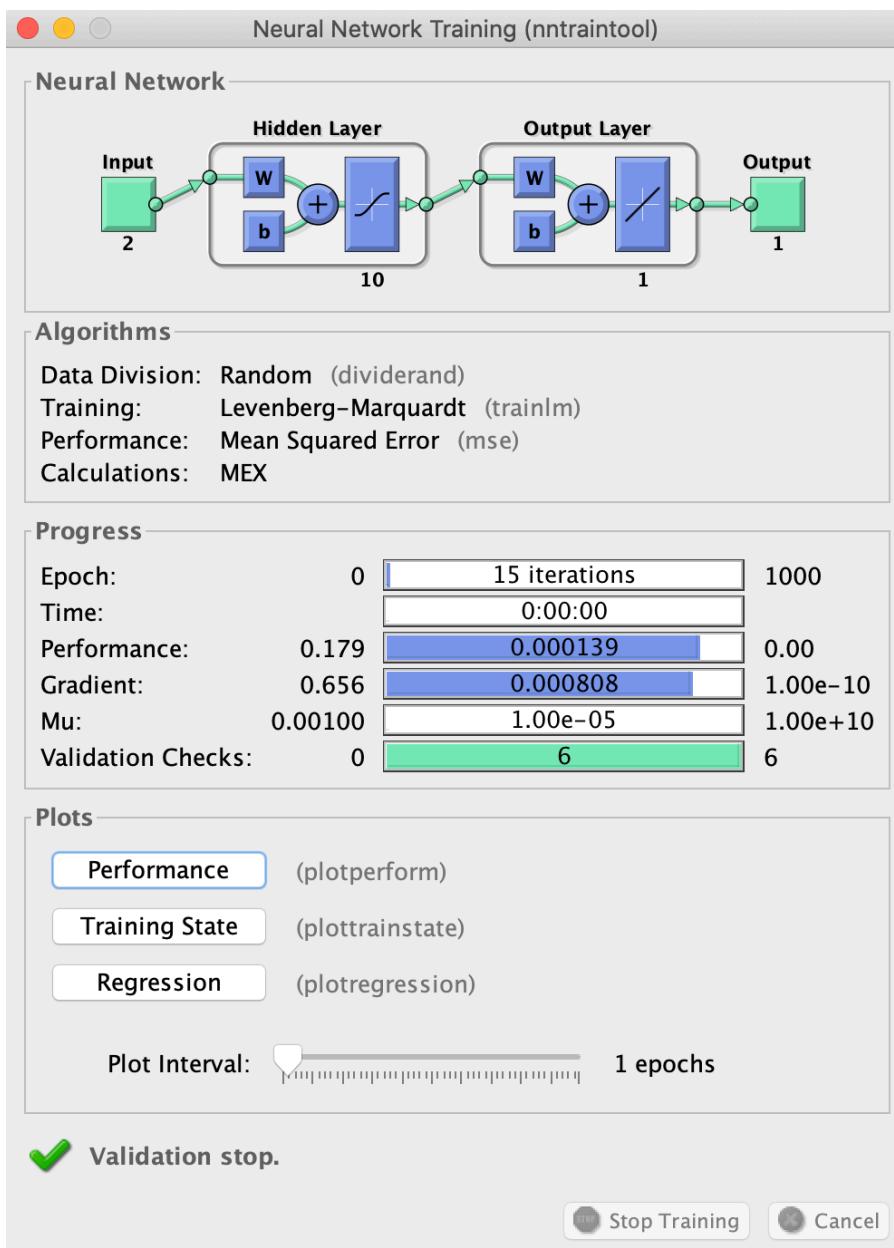


График производительности сети

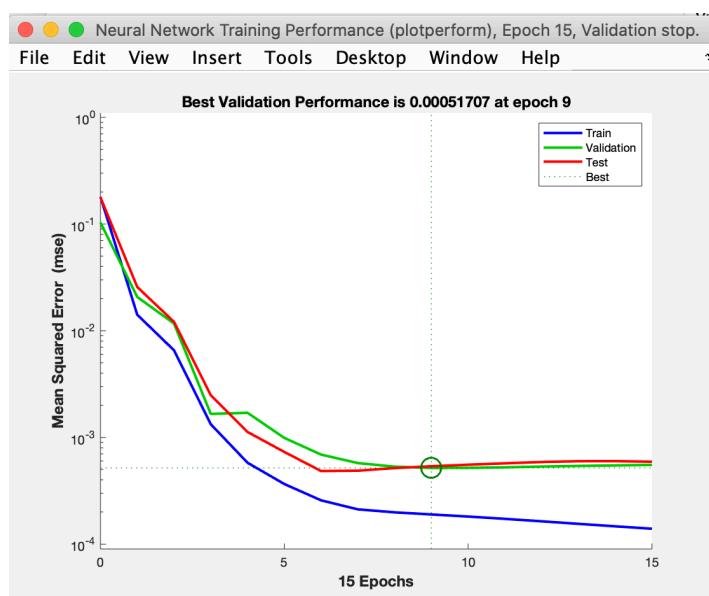
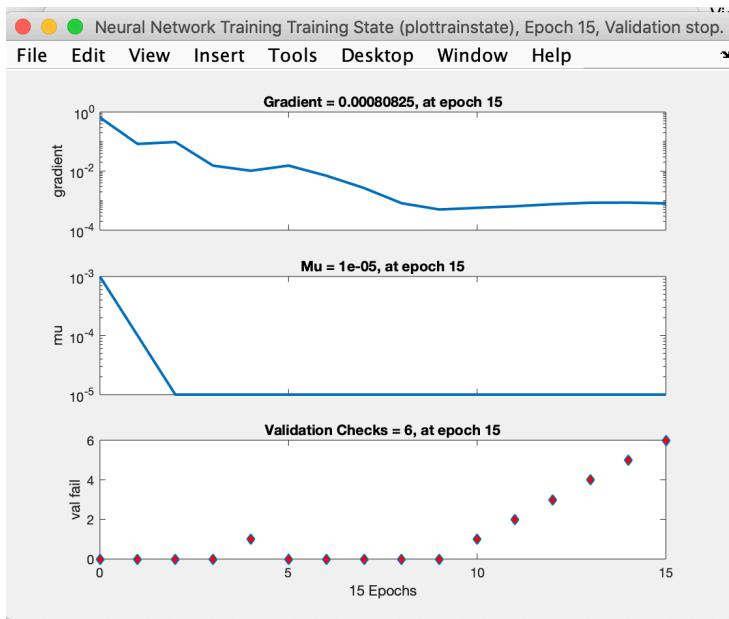
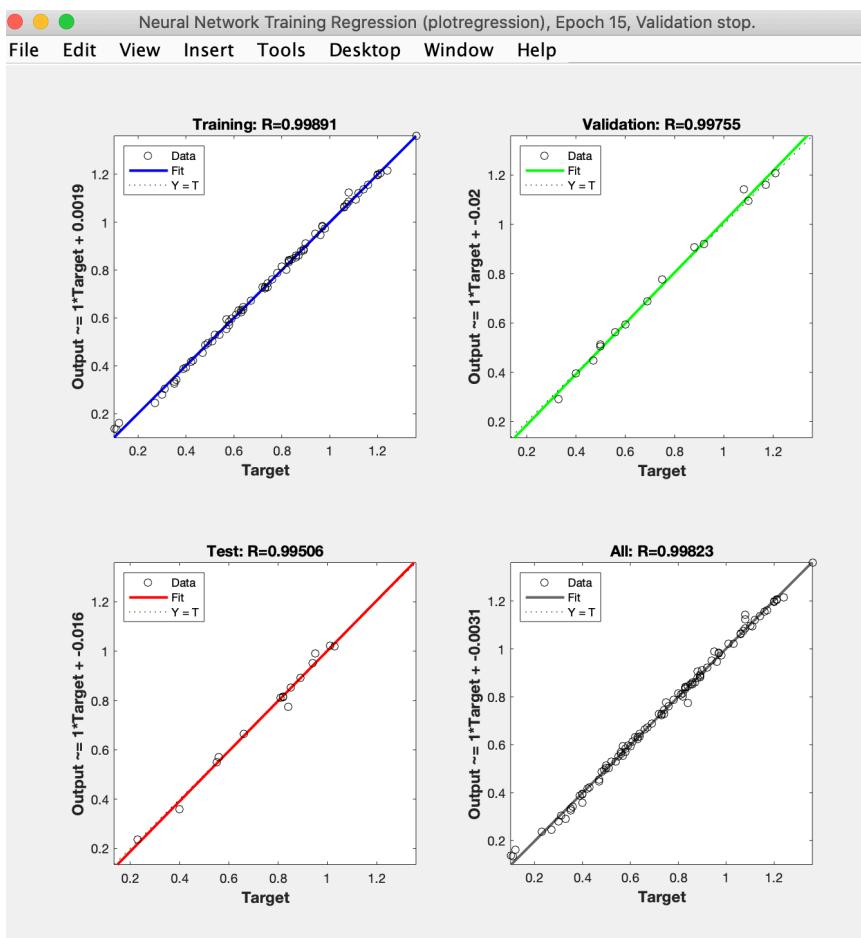


График процесса обучения сети



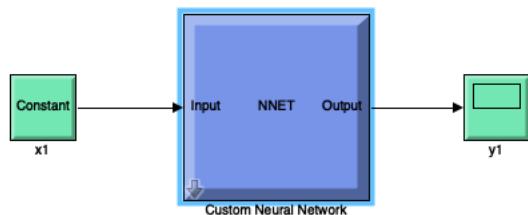
Процесс регрессии



Значения выходов сети

```
[0.5833 1.0937 0.61414 1.1992 0.57082 0.63142 1.0873 0.66432 0.16149 0.85931 0.27906  
0.92154 1.1426 0.72229 0.81459 0.62372 0.55528 0.56283 1.1608 0.38844 1.1222 0.744  
0.5035 0.1369 0.13473 0.55104 0.33418 0.63305 0.84032 0.39457 0.32486 0.52869 0.90675  
0.89141 0.77537 0.91004 0.59577 0.88121 0.98201 0.95132 0.78696 0.44713 0.24442 0.98988  
1.2034 1.1961 0.30318 0.72792 0.86024 0.98498 0.85159 0.72767 1.1238 0.81585 0.49427  
0.68896 0.7761 0.95132 0.72855 1.2136 0.85211 0.76152 1.2067 0.35894 0.83932 0.3922  
0.88782 1.0754 0.67282 0.48575 0.29101 0.42337 0.64528 0.34188 1.1556 0.88001 0.83474  
0.80092 1.0954 0.59499 0.94689 0.63404 0.8105 0.417 1.0207 0.53042 1.0223 1.1362 1.0638  
0.45562 0.81388 1.0619 0.51384 0.59393 0.23602 1.359 0.57054 0.97245 0.50417 0.84438]
```

Модель нейронной сети в среде Simulink



Вывод:

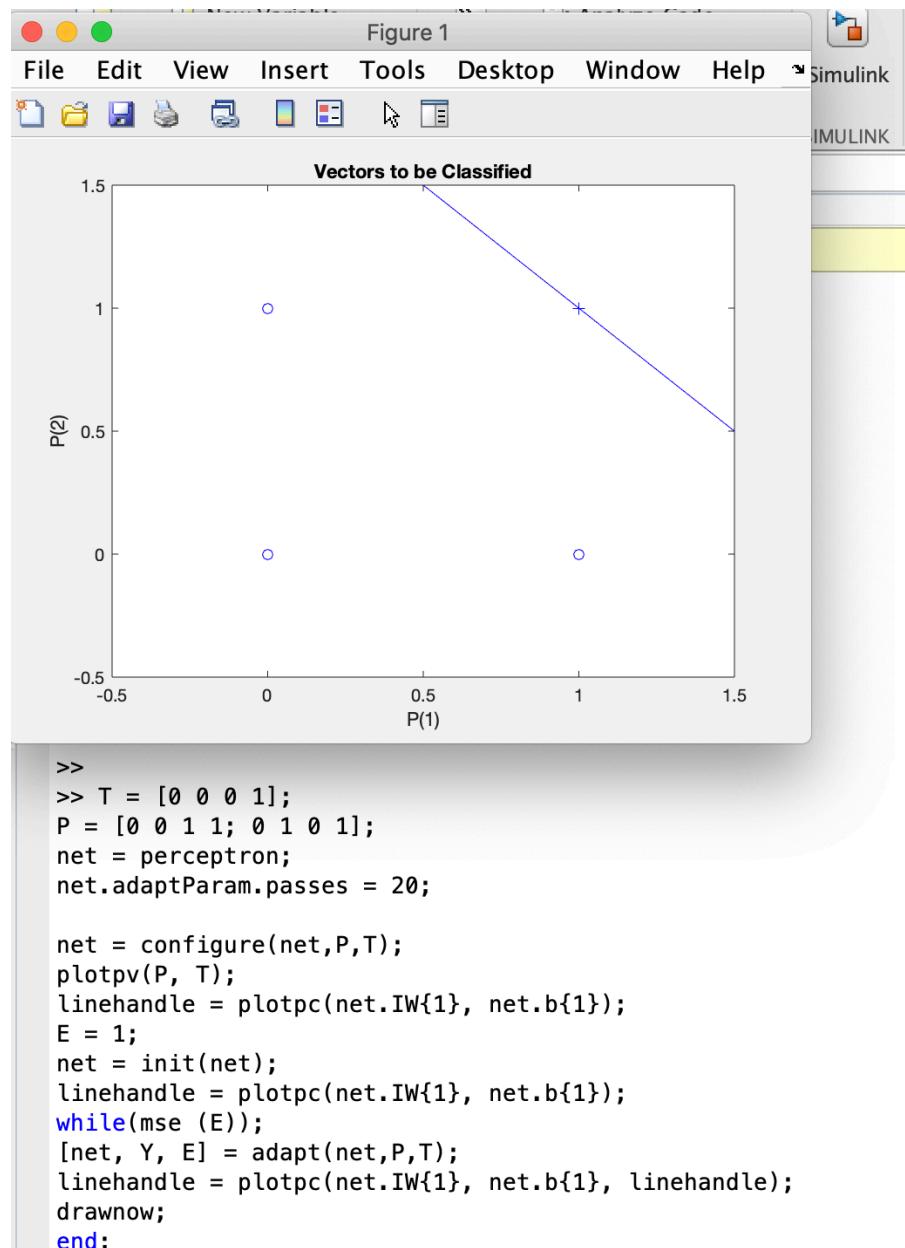
Познакомился с основными методами работы GUI Matlab. Создала нейронную сеть априориизирующую заданную функцию.

Лабораторная работа №2

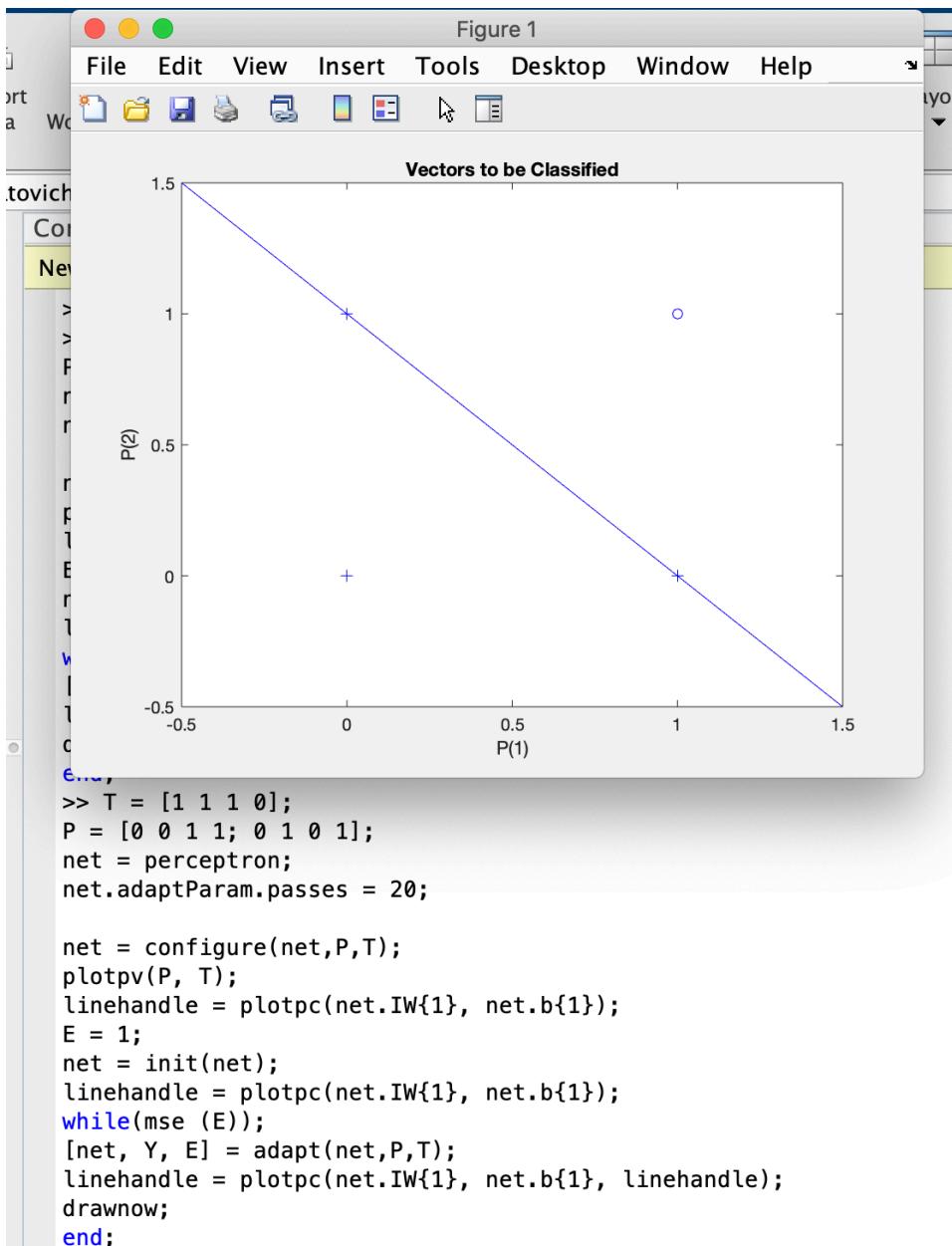
Задача: Реализовать логические функции с помощью персептрона (функция perceptron), изобразить входные векторы и разделяющую прямую на плоскости, сделать выводы о реализуемости логических функций

Ход работы:

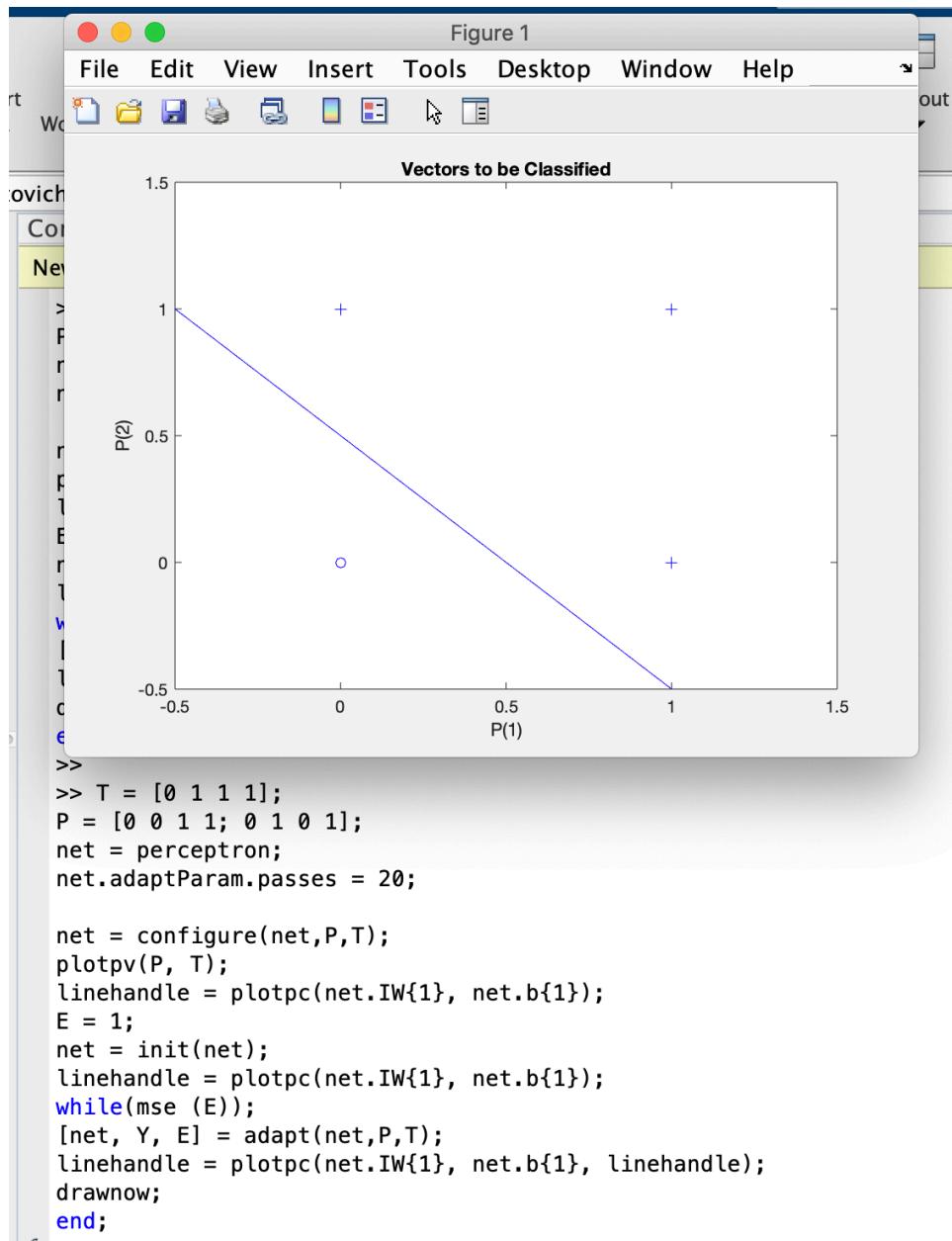
AND



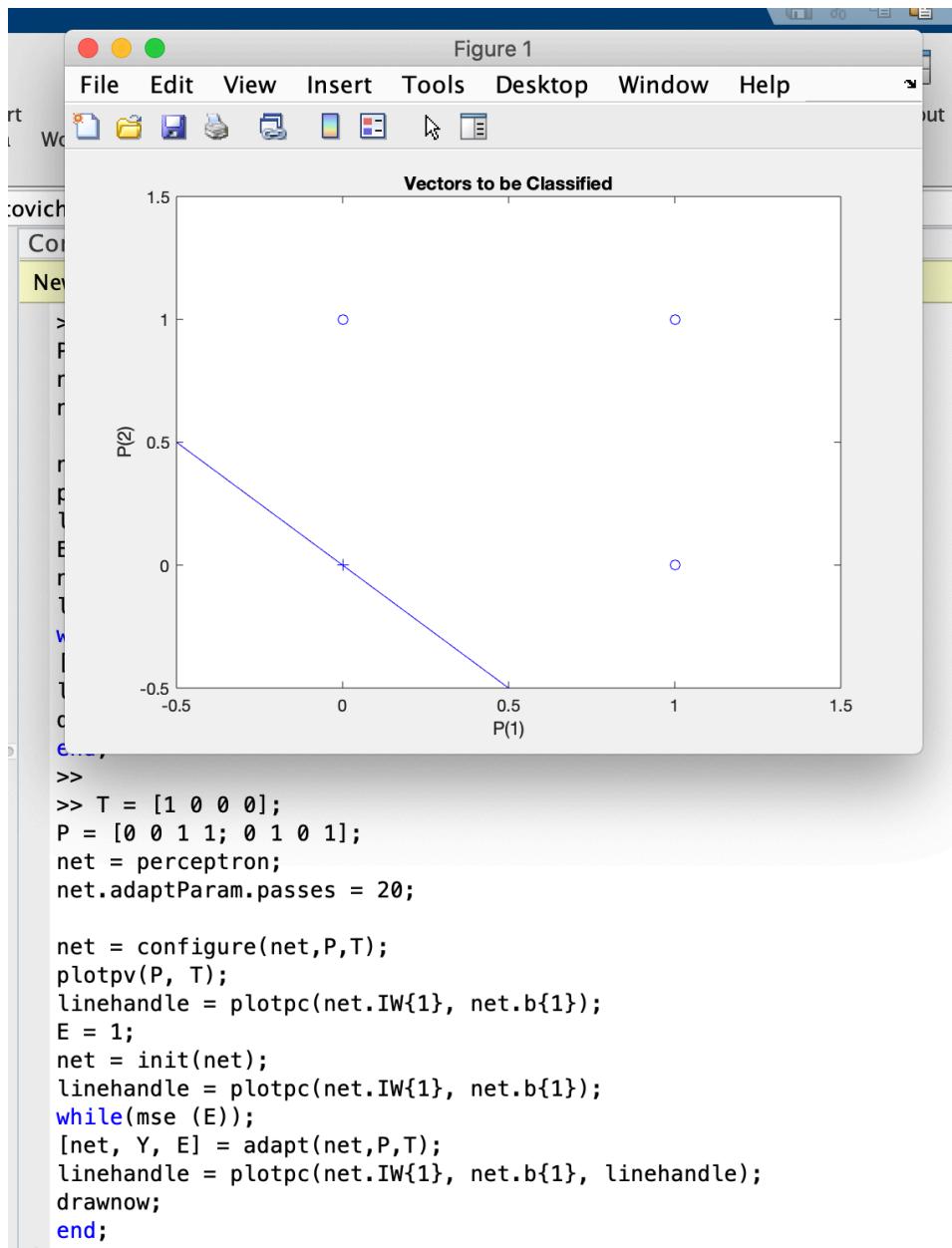
NAND



OR



NOR



Вывод:

Обучила простейшую нейронную сеть модели перцептрана определять и разделять простейшие логические функции. Определила возможности модели простейшего нейрона

Лабораторная работа №3

Задача: Произвести исследования следующих алгоритмов обучения НС: adapt, trainlm, leargd, traingd, leargdm, traingdm, traingda, traingdx, trainrp на примере обучения сети feedforwardnet (представить схему в simulink). Привести окна обучения, графики сигналов выхода и цели, графики поверхности целевой функции и функции аппроксимируемой НС, линейной регрессии между выходом и целями, изменения ошибки.

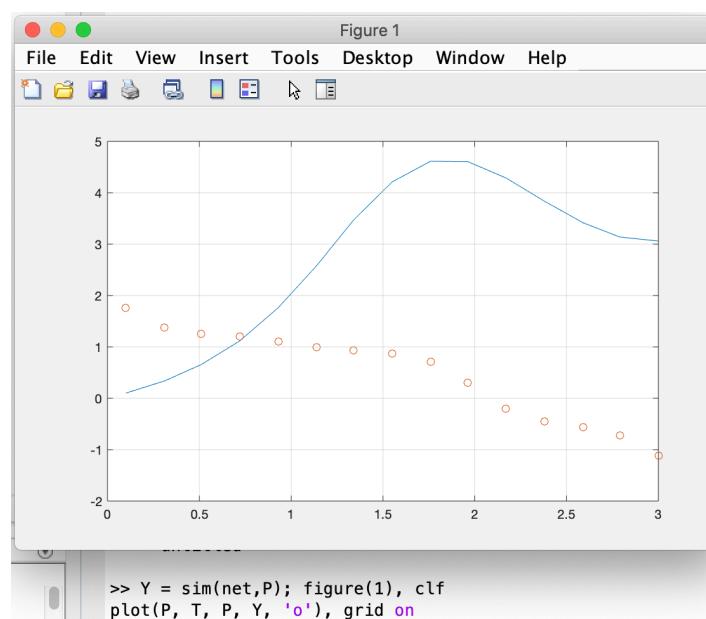
Сделать выводы о качестве и производительности алгоритмов обучения

Ход работы:

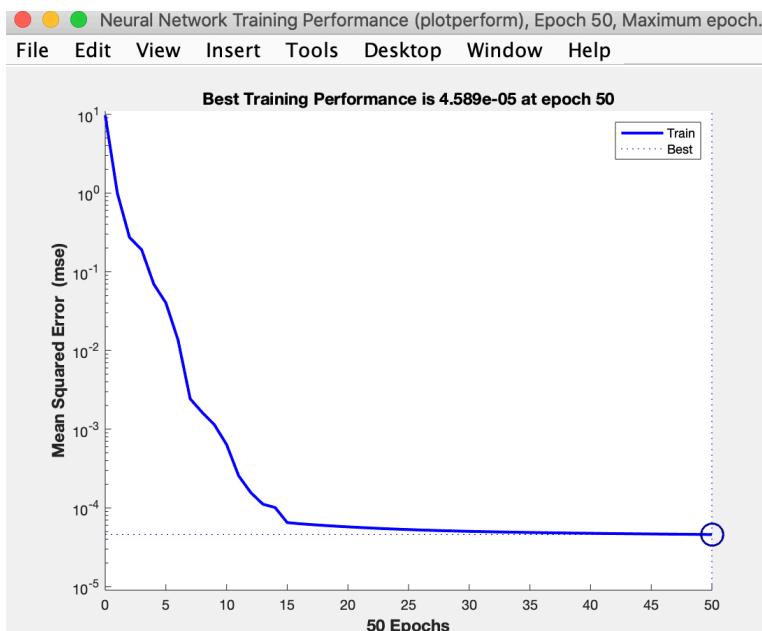
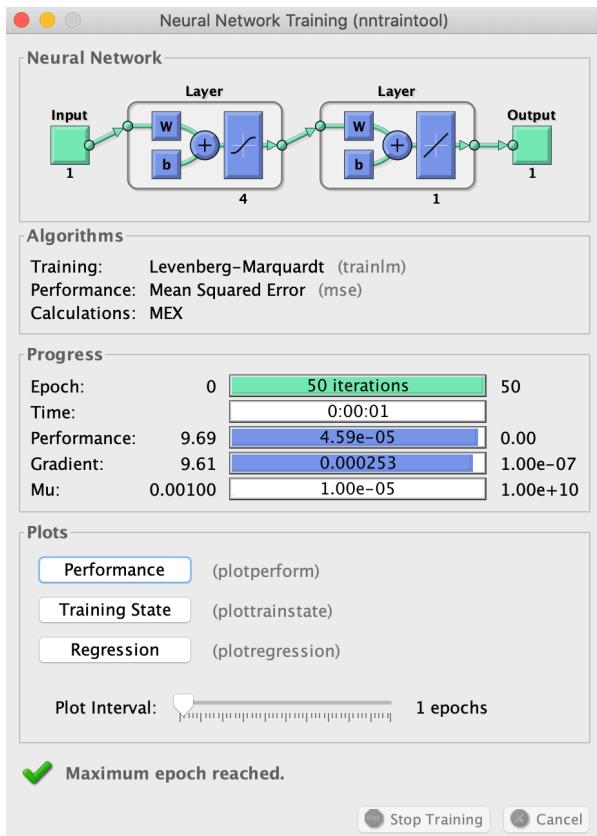
Создать нейронную сеть, чтобы обеспечить следующее отображение последовательности входа Р в последовательность целей Т:

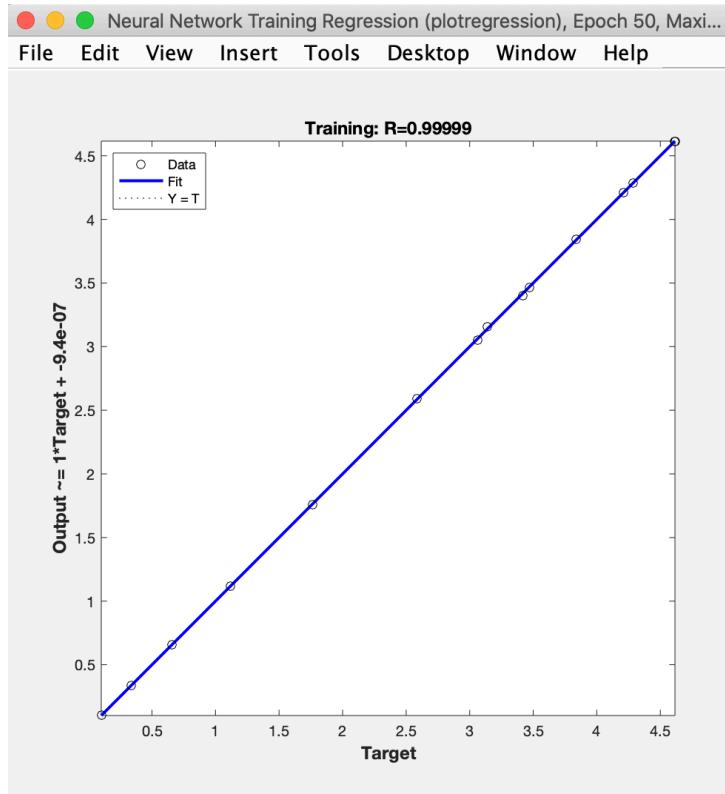
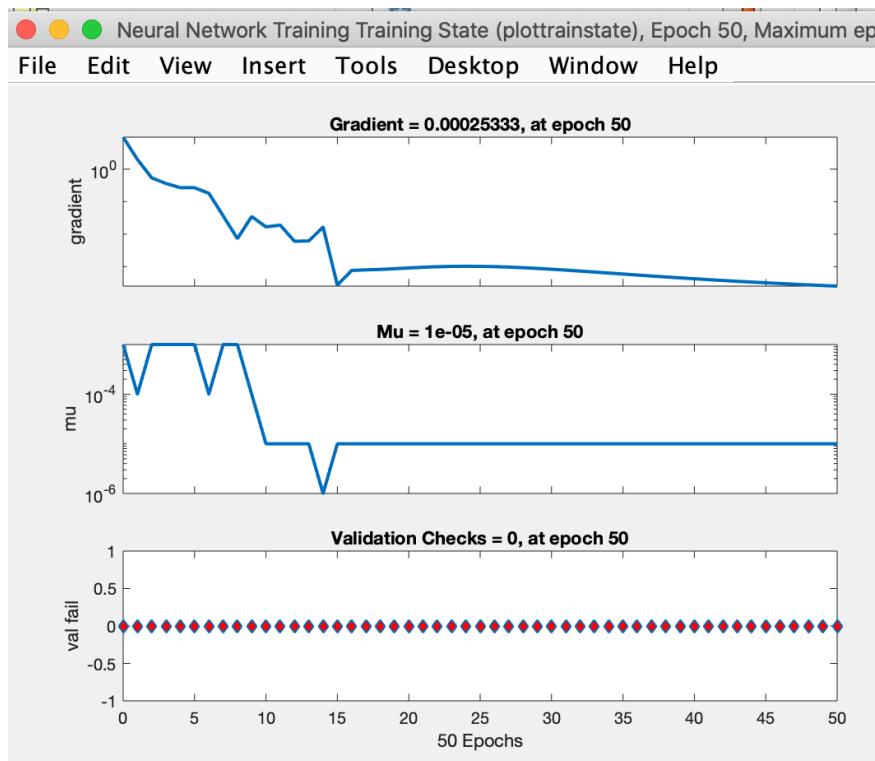
```
>> P = [0.10 0.31 0.51 0.72 0.93 1.14 1.34 1.55 1.76 1.96 2.17 2.38 2.59 2.79 3.00]
P =
Columns 1 through 11
    0.1000    0.3100    0.5100    0.7200    0.9300    1.1400    1.3400    1.5500    1.7600    1.9600    2.1700
Columns 12 through 15
    2.3800    2.5900    2.7900    3.0000
>> T = [0.1010 0.3365 0.6551 1.1159 1.7632 2.5847 3.4686 4.2115 4.6152 4.6095 4.2887 3.8349 3.4160 3.1388 3.0603]
T =
Columns 1 through 11
    0.1010    0.3365    0.6551    1.1159    1.7632    2.5847    3.4686    4.2115    4.6152    4.6095    4.2887
Columns 12 through 15
    3.8349    3.4160    3.1388    3.0603
>> net = newff([0 3],[4 1],{'tansig' 'purelin'})
```

Выполним моделирование сети и построим графики сигналов выхода и цели:



Обучим сеть в течение 50 циклов:





Выполним моделирование сформированной двухслойной сети, используя обучающую последовательность входа

Figure 1

File Edit View Insert Tools Desktop Window Help

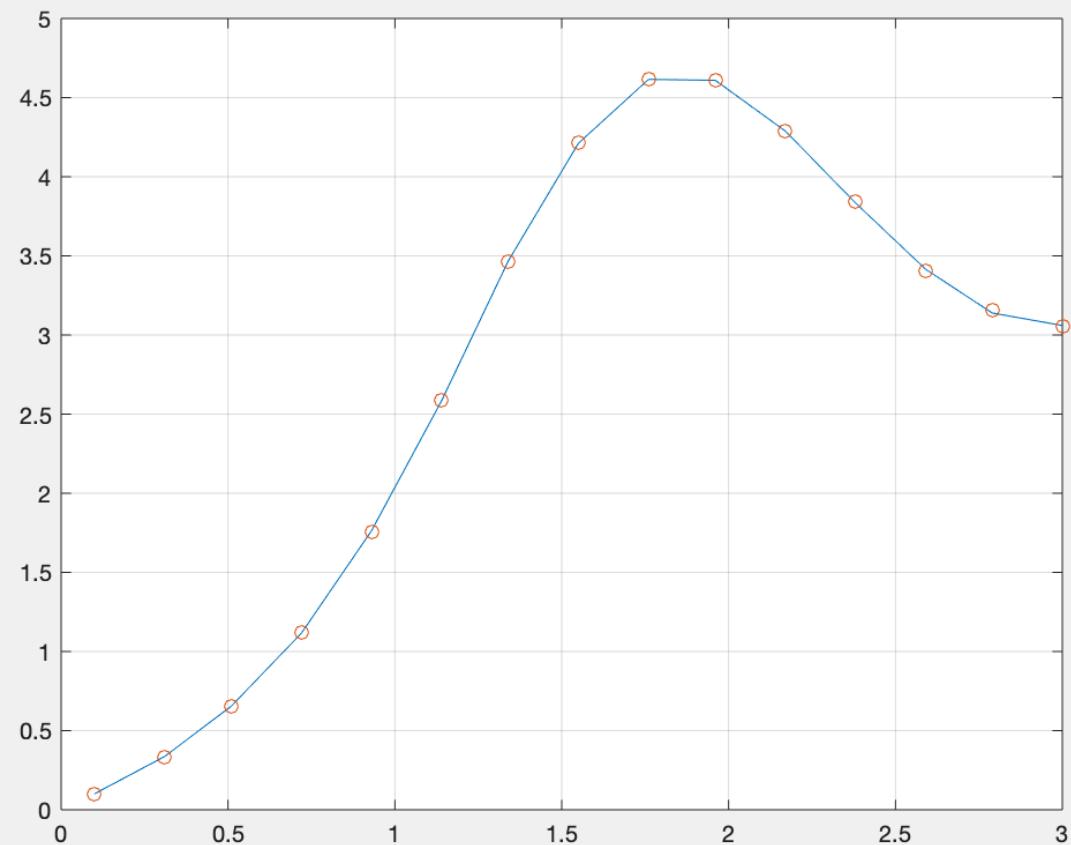


График поверхности целевой функции

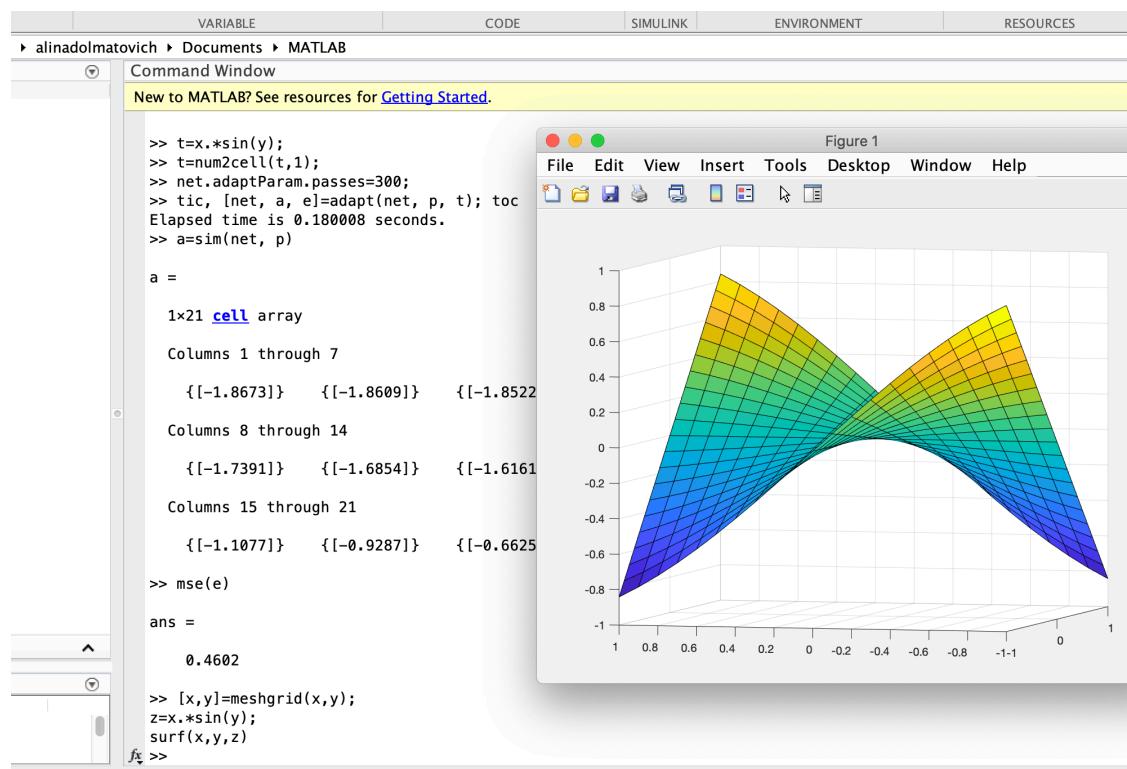
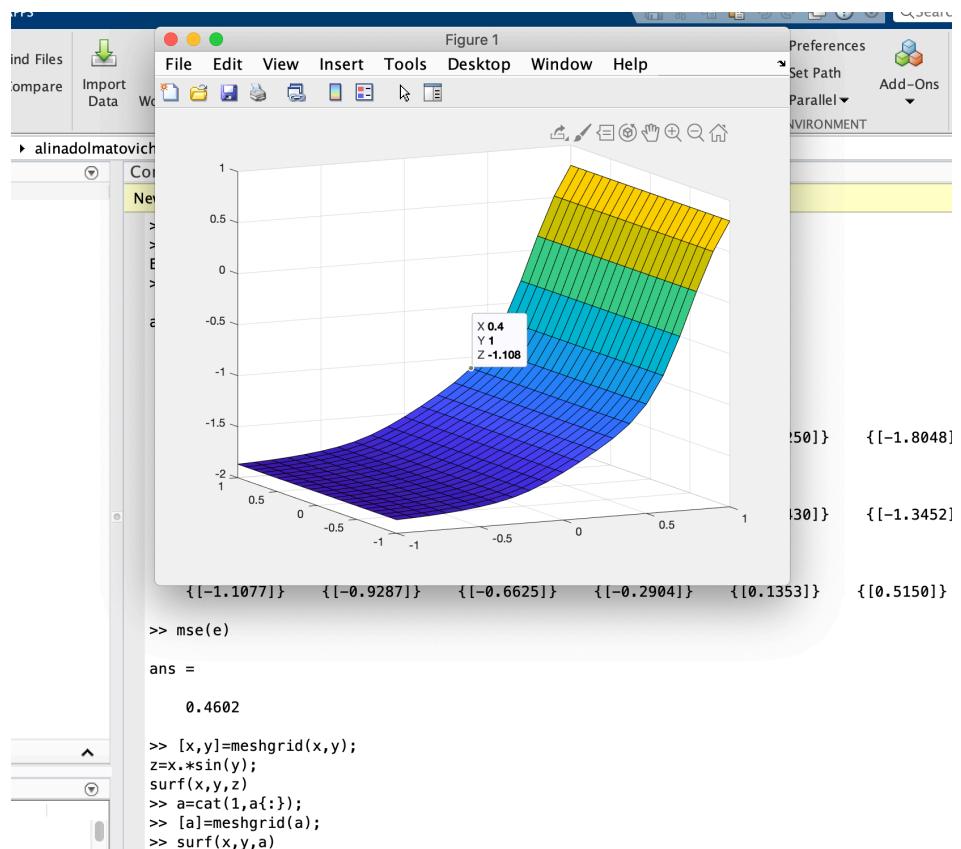


График поверхности функции аппроксимируемой нейронной сетью



Групповое обучение.

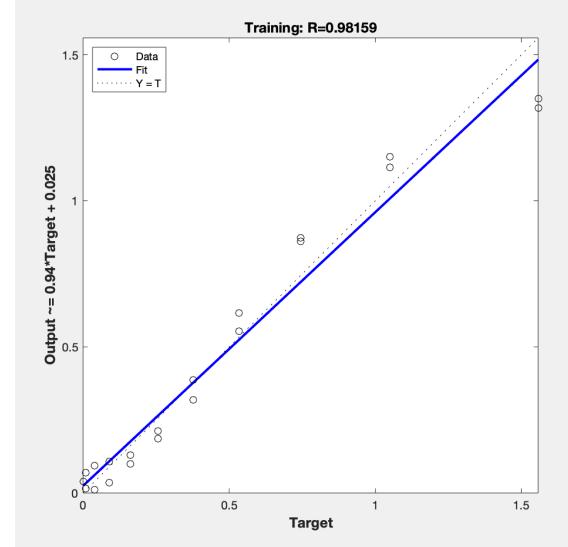
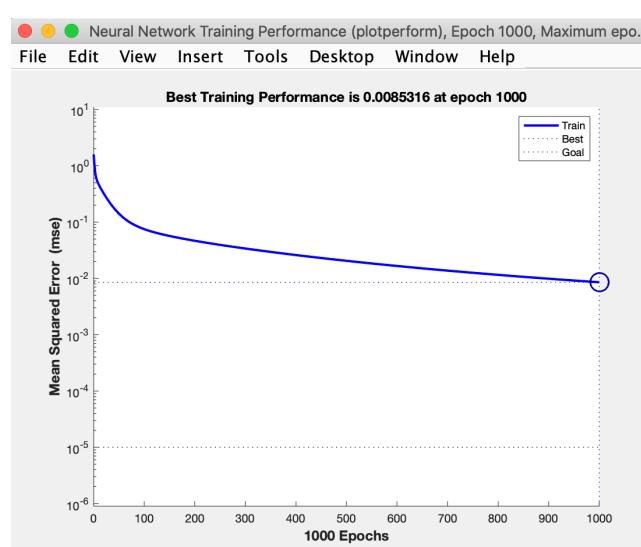
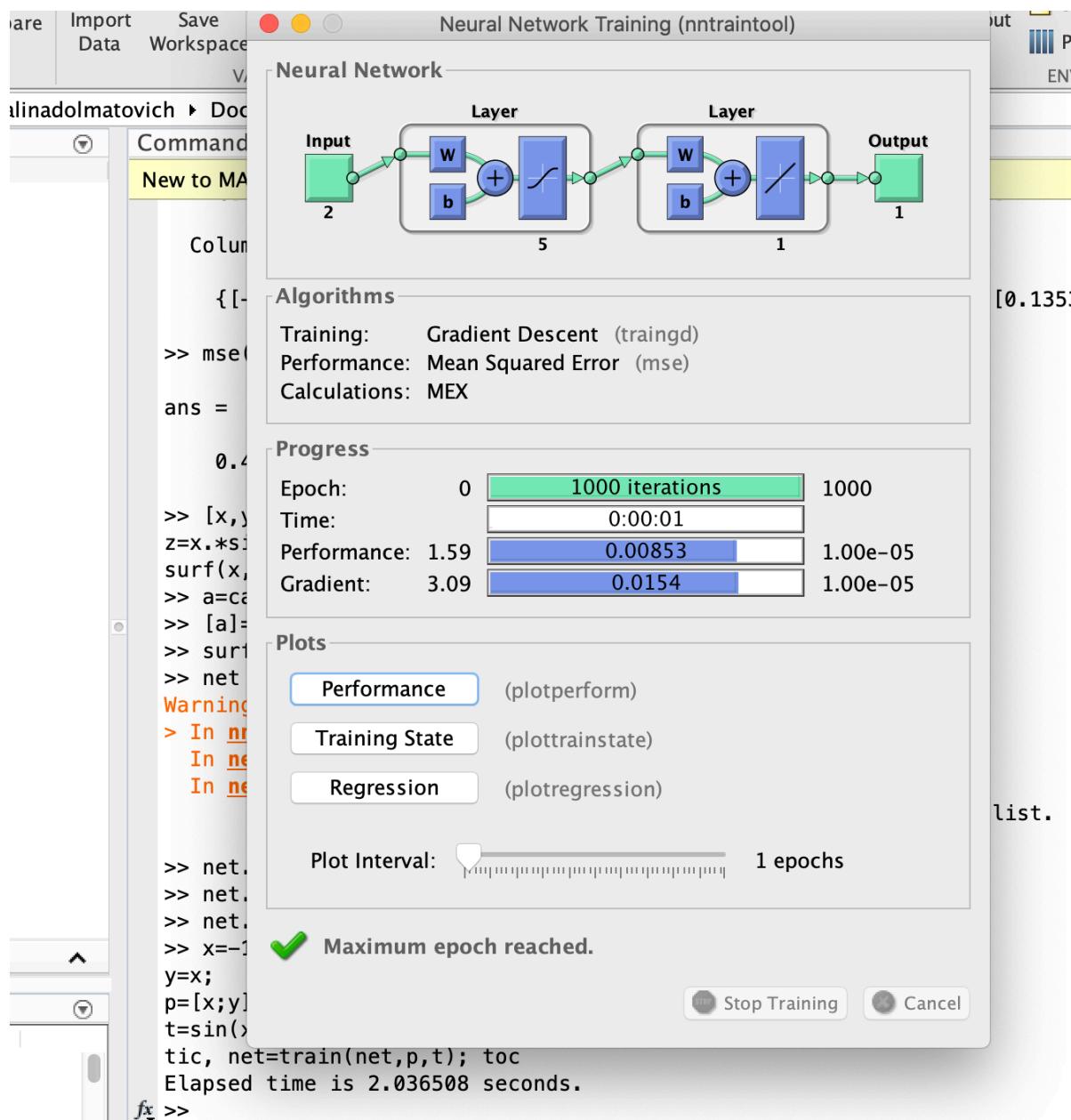
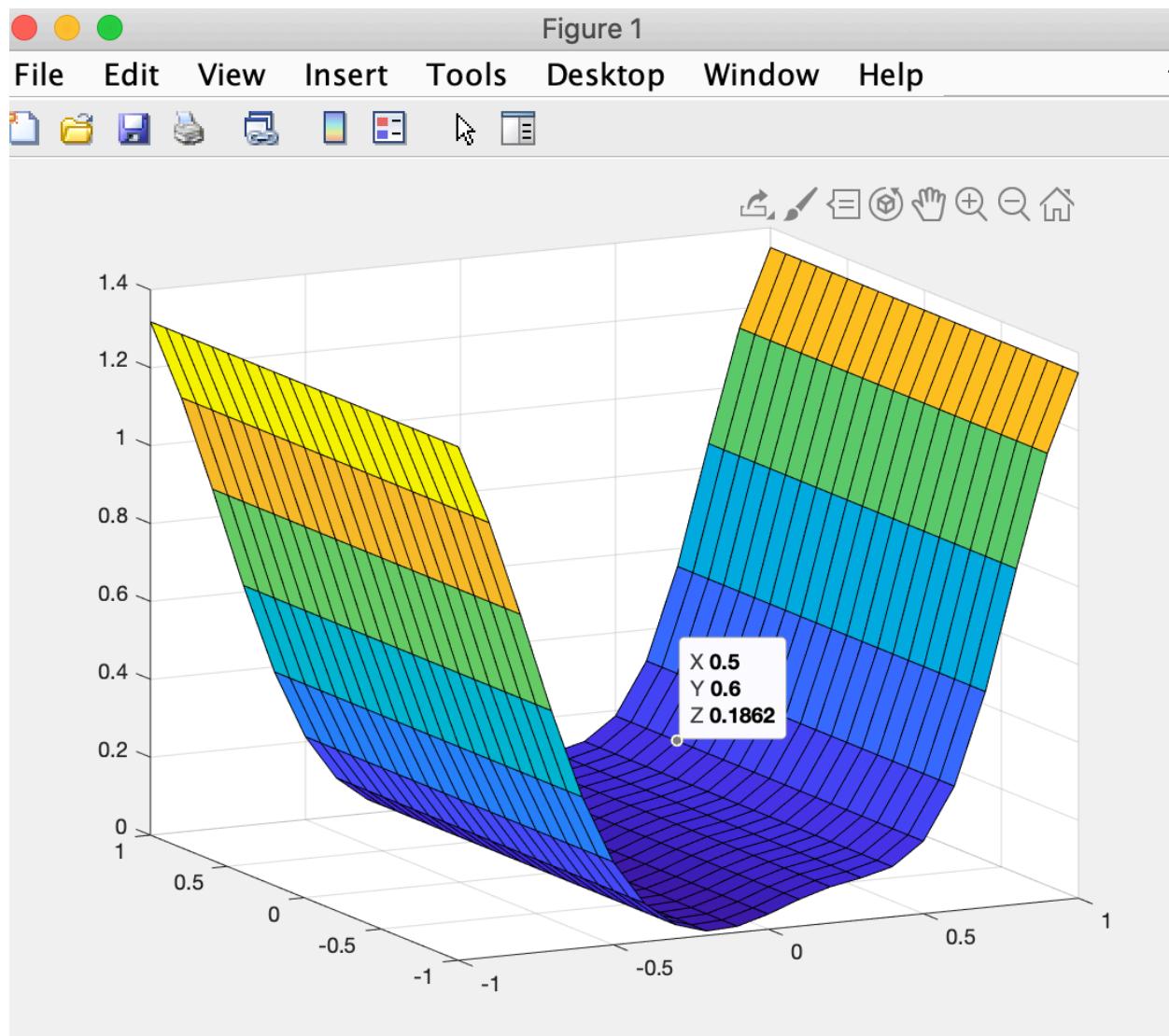
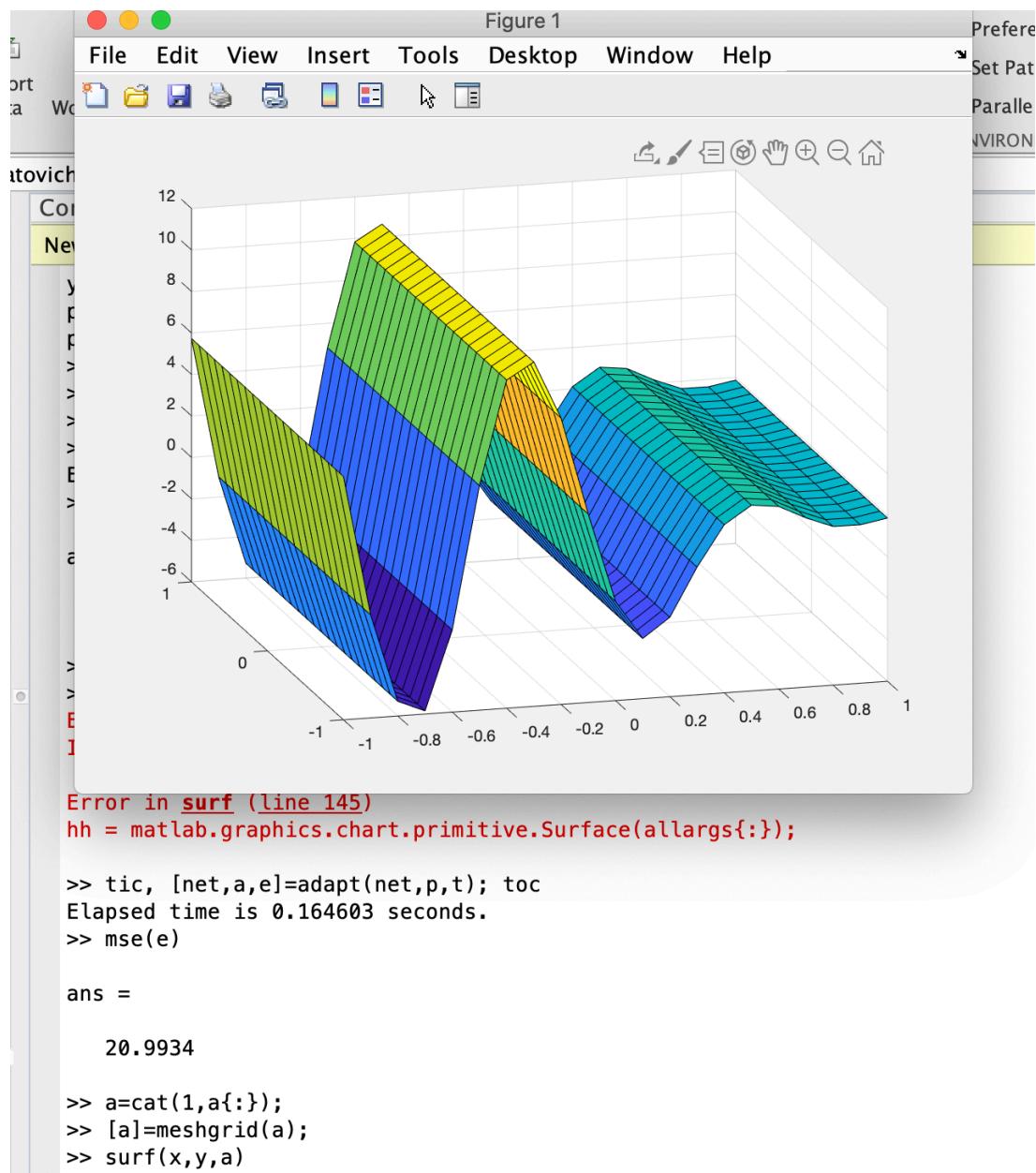


График поверхности функции аппроксимируемой нейронной сетью



Алгоритм GDM.



Вопросы:

В: Каким алгоритмом обучаются многослойные нейронные сети?

О: Обратное распространение ошибки

В: Из каких основных этапов состоит алгоритм обратного распространения ошибки?

О: Вычисление прямого распространения, последовательное вычисление ошибки и корректировка весов, согласно ошибке.

В: Почему алгоритм обратного распространения ошибки относится к классу алгоритмов градиентного спуска?

О: Потому что при вычислении градиентный шаг приближает предсказанные значения к минимуму градиента, изменения веса нейронных связей.

В: Как влияет функция принадлежности на правило изменения весов в обратном алгоритме распространения ошибки?

О: Каждая функция принадлежности обладает собственным правилом при корректировке весов. При градиентном спуске в задаче классификации функция принадлежности влияет на скорость спуска.

Вывод: Изучила метод обратного распространения ошибки, а также методы функционирования многослойных нейронных сетей

Лабораторная работа №4

Задача: Применение нейронных сетей для аппроксимации функций и предсказания временного процесса

Ход работы:

Задание 1(Вариант 6)

Построить модель нейронной сечи, аппроксимирующей полином

$$Y = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$$

На интервале [-1,1]

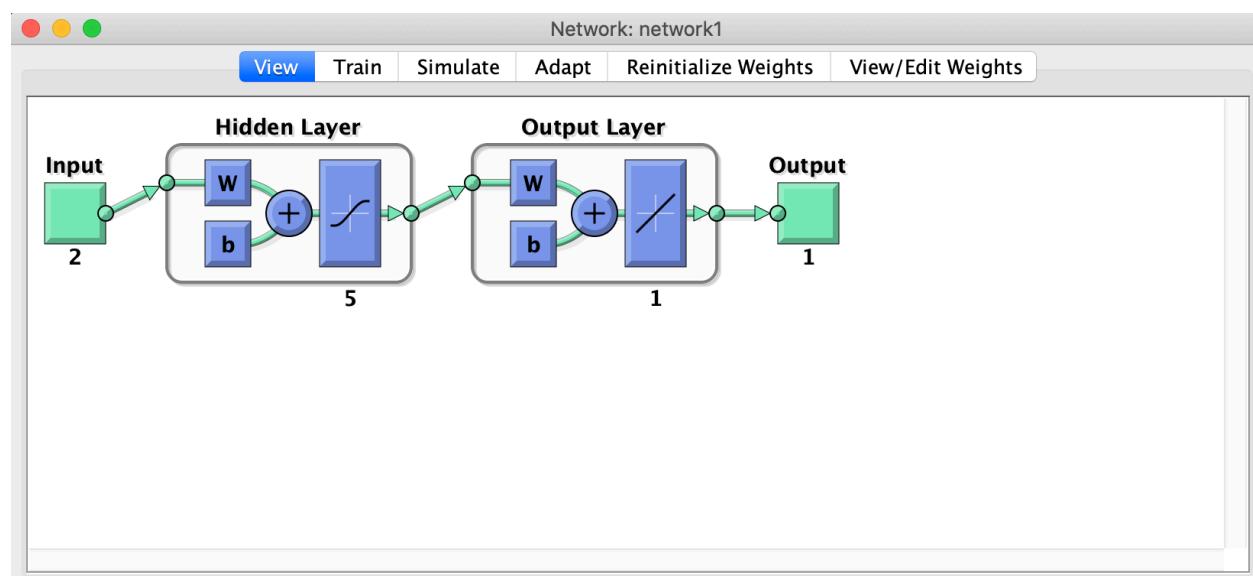
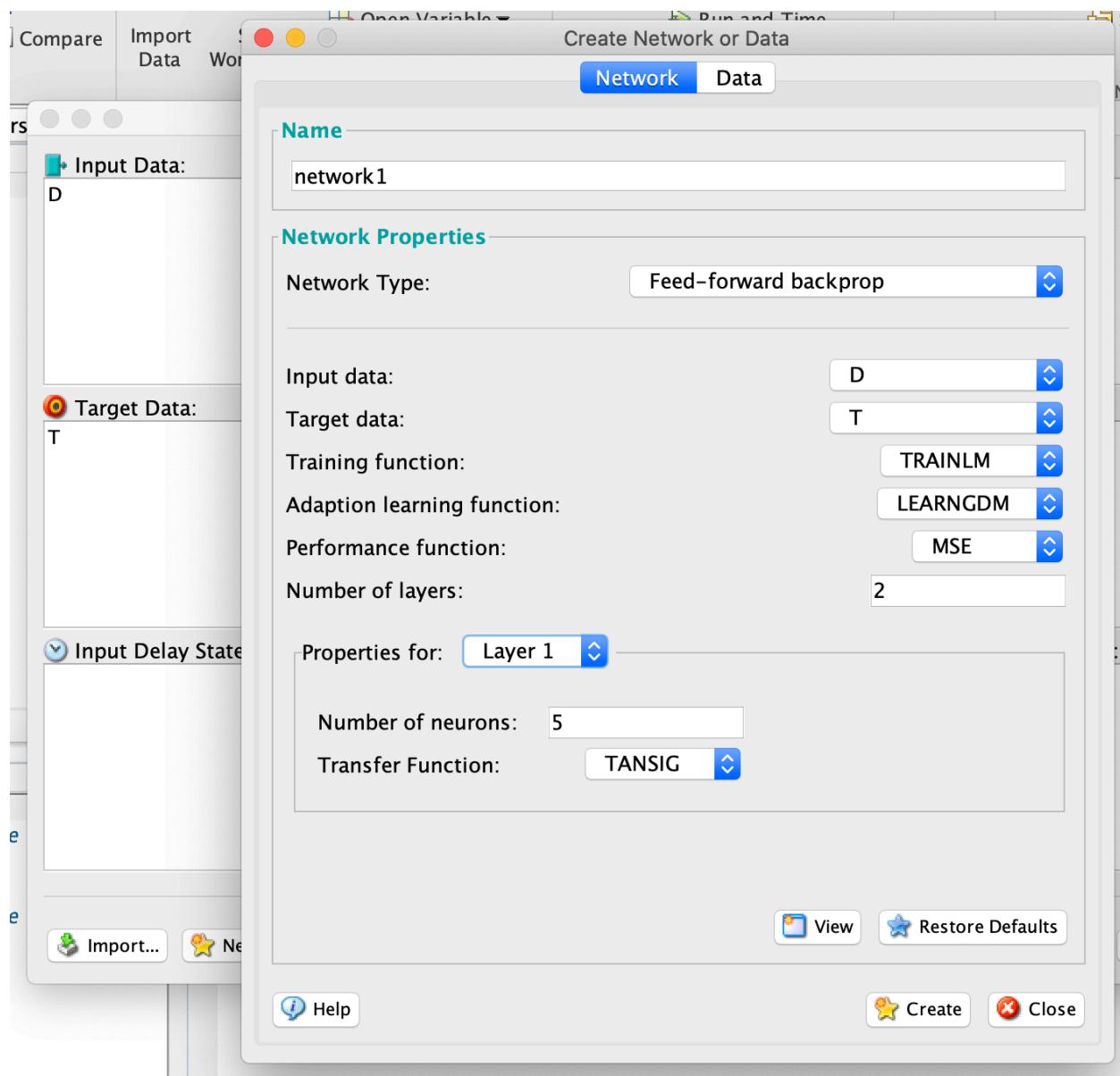
Коэффициенты полинома [-1, -1, -1, 1, 1, 1]

Сгенерируем входные и выходные данные в рабочее окружение.

```
trial License -- to use to evaluate

>> x = 1;
>> for i = 0:.1:1;
for j = 0:.1:1;
D(1,x) = i;
D(2,x) = j;
T(x) = -1*i*i-1*j*j-1*i*j+i+j+1;
x = x+1;
end;
end;
Unrecognized function or variable 'J'.

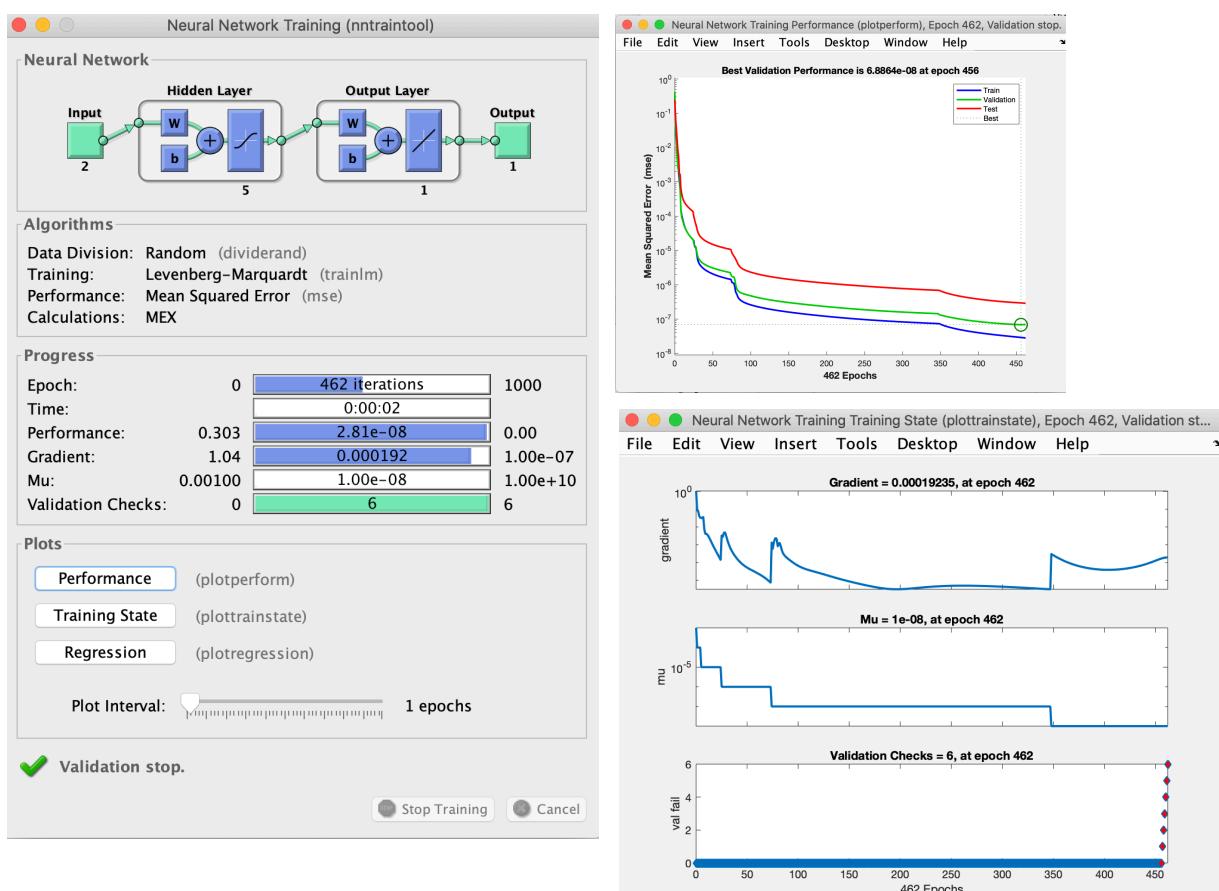
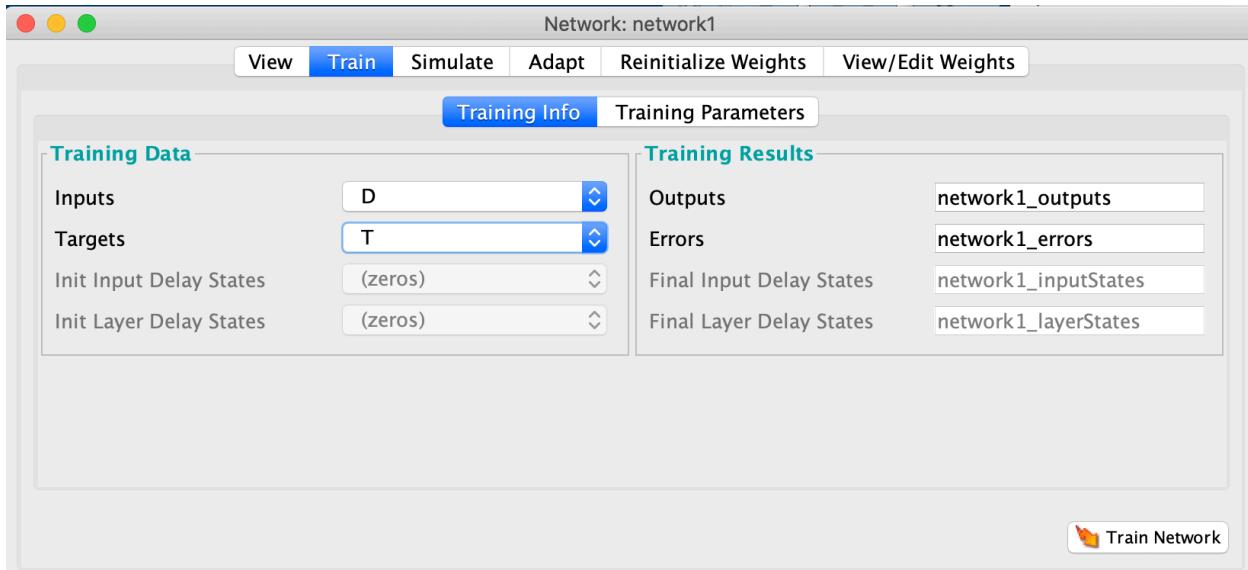
>> for i = 0:.1:1;
for j = 0:.1:1;
D(1,x) = i;
D(2,x) = j;
T(x) = -1*i*i-1*j*j-1*i*j+i+j+1;
x = x+1;
end;
end;
```

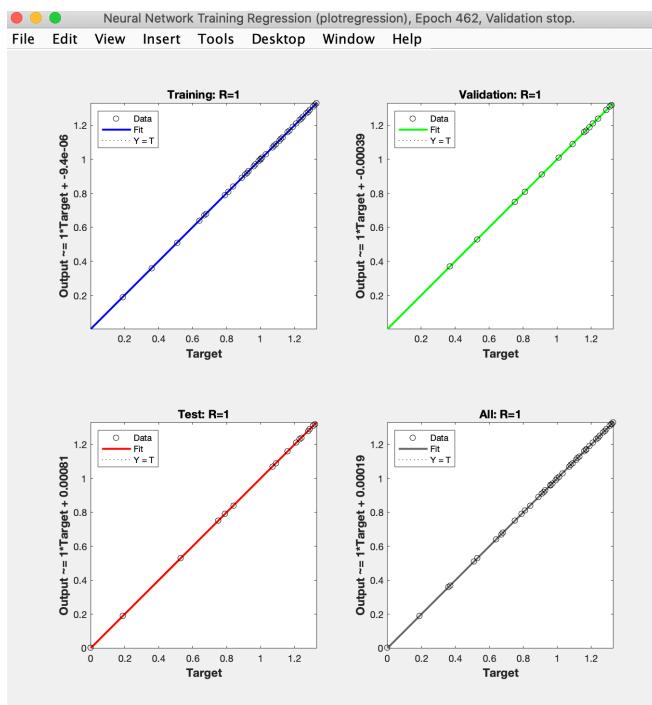


Создадим нейронную сеть с двумя слоями:

1. 5 нелинейных нейронов
2. 1 с линейной функцией активации

Обучим сеть, при этом в качестве входных данных необходимо использовать матрицу D, а в качестве цели вектор T





Обученную сеть симулируем вектором данных, получив выход сети `network2_outputs` на основе входных данных матрицы D

Network: network1

View Train Simulate Adapt Reinitialize Weights View/Edit Weights

Simulation Data

Inputs: D
Targets: (zeros)

Init Input Delay States: (zeros)
Init Layer Delay States: (zeros)

Supply Targets:

Simulation Results

Outputs: network2_outputs
Final Input Delay States: network1_inputStates
Final Layer Delay States: network1_layerStates
Errors: network1_errors

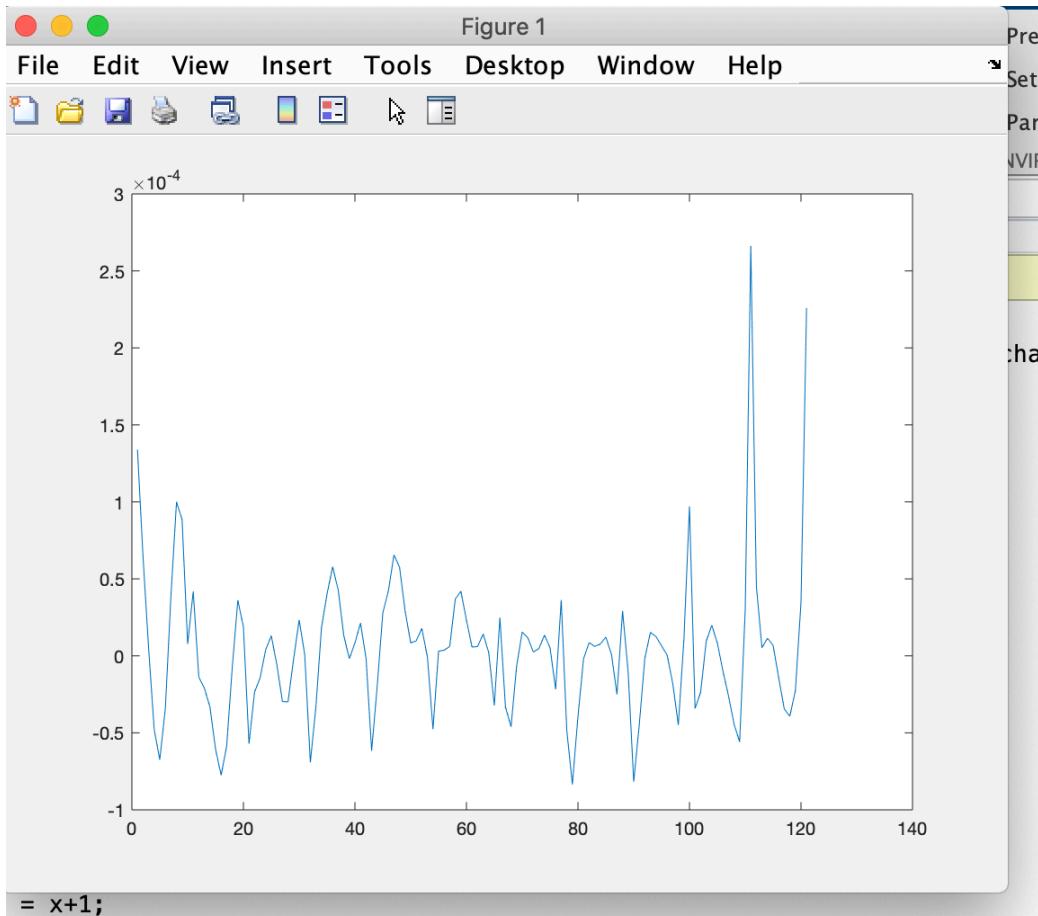
Neural Network/Data Manager (nntool)

Input Data: D
Target Data: T
Input Delay States:
Networks: network1
Output Data: network1_outputs
network2_outputs
Error Data: network1_errors
Layer Delay States:

Simulate Network

Import... New... Open... Export... Delete Help Close

График ошибки



```
= x+1;
nd;
nd;
> nntool
> error = network2_outputs-T;
nrecognized function or variable 'network2_outputs'.

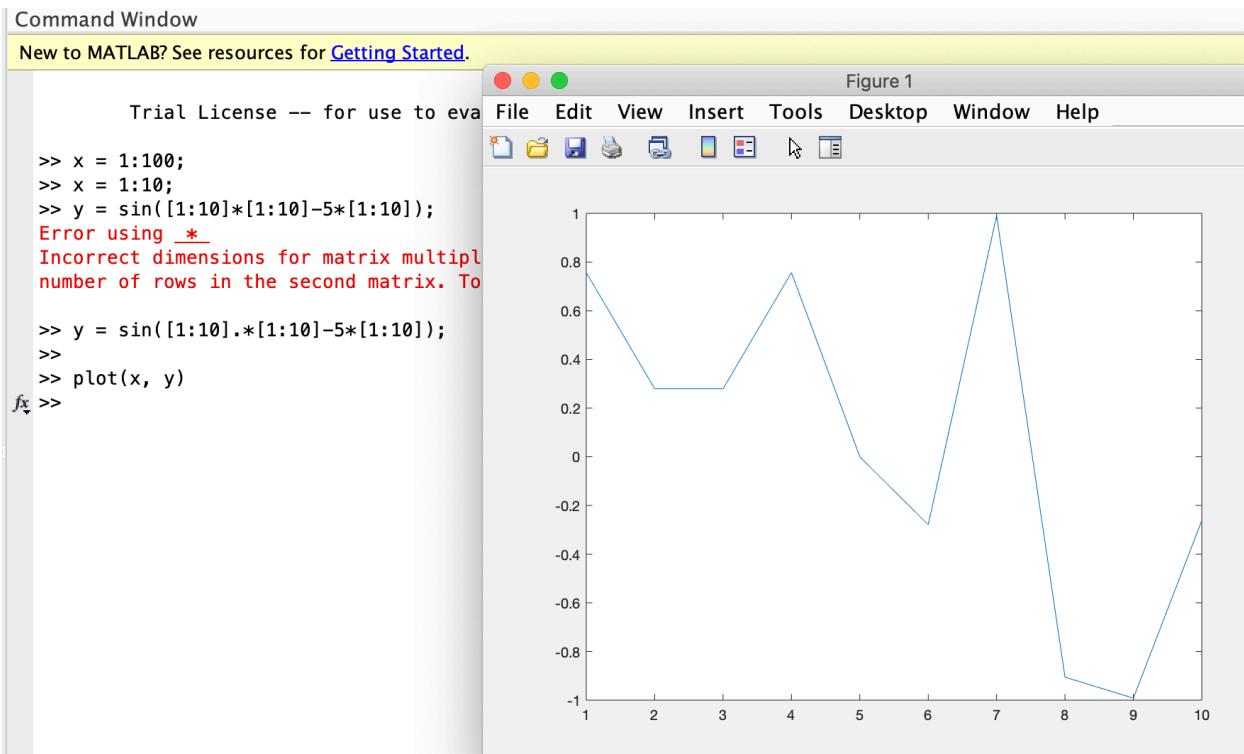
> nntool
> error = network2_outputs-T;
> plot(error)
>
```

Задание 2(Вариант 6)

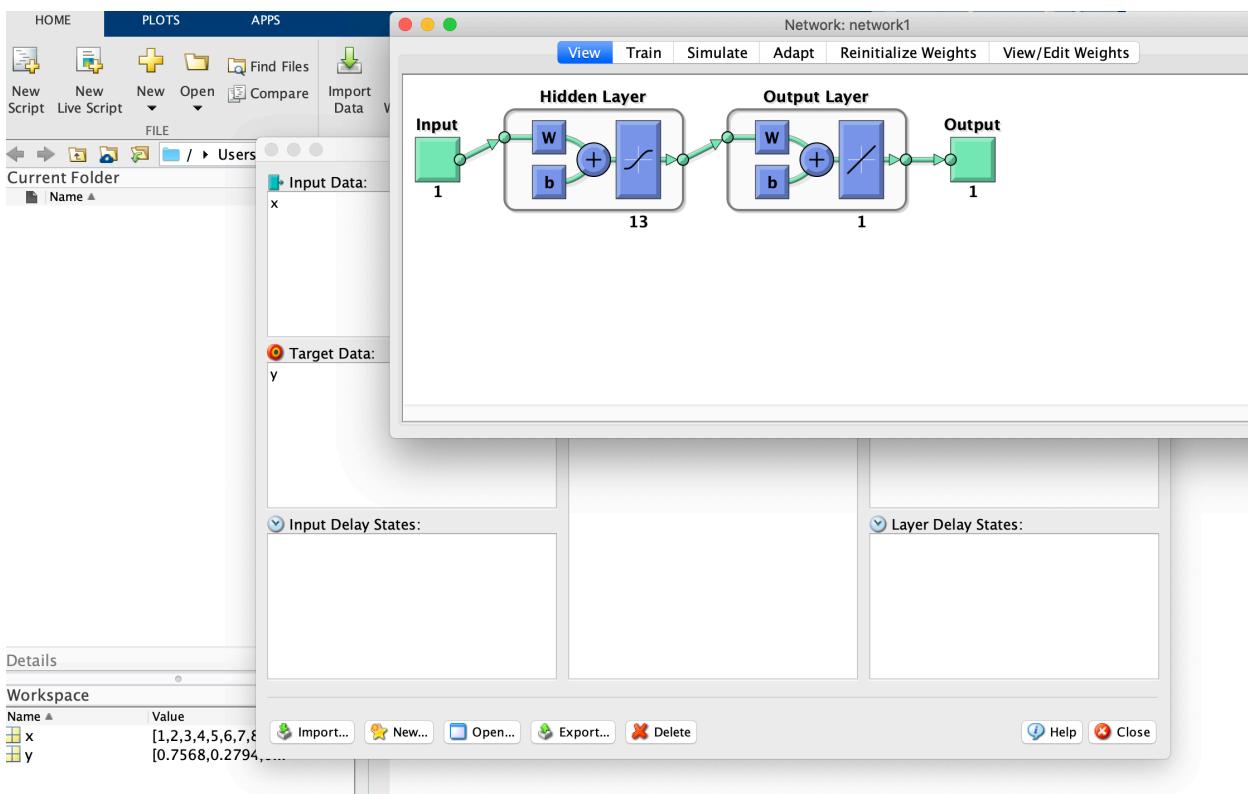
Построить модель нейронной сети для аппроксимации функции $f(t)$ на промежутке $[0, 10]$:

$$\underline{\sin(t^2 - 5 \cdot t)}$$

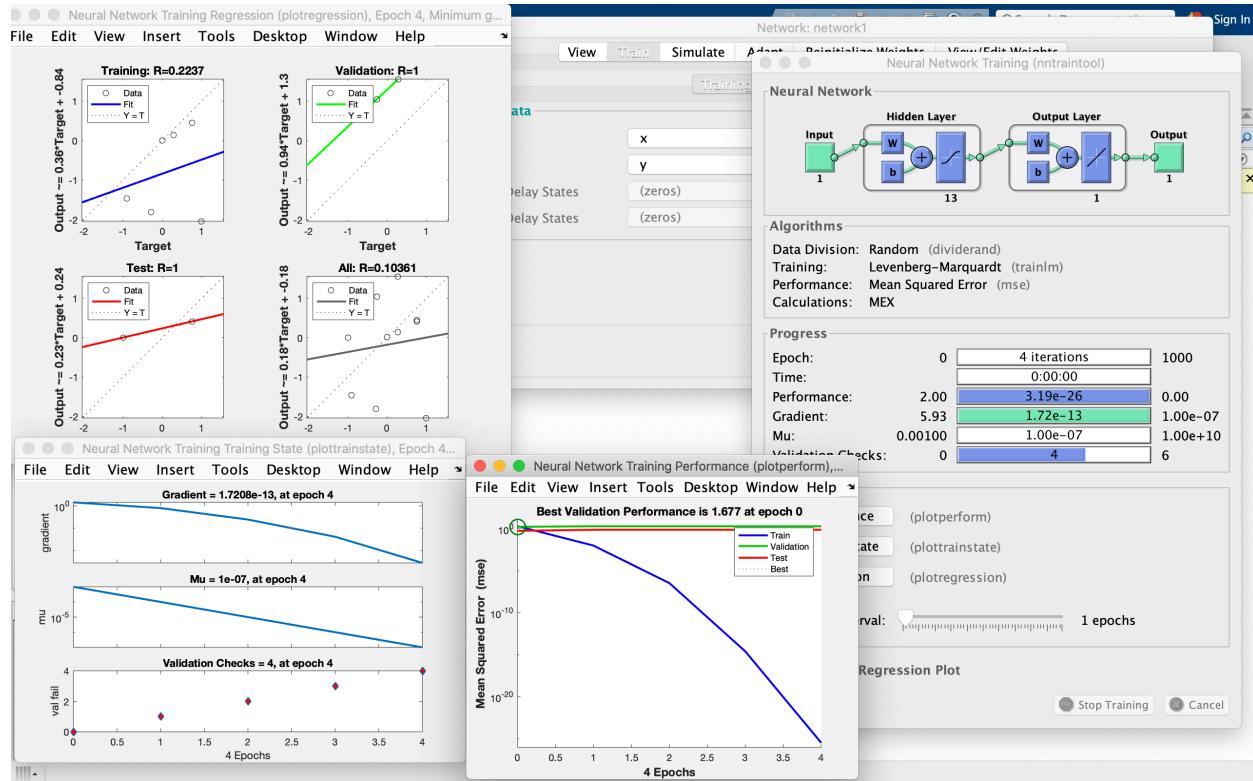
График функции



Архитектура сети для решения задачи аппроксимации



Обучим сеть



Построим графики функции и выхода нейронной сети в одной системе координат:

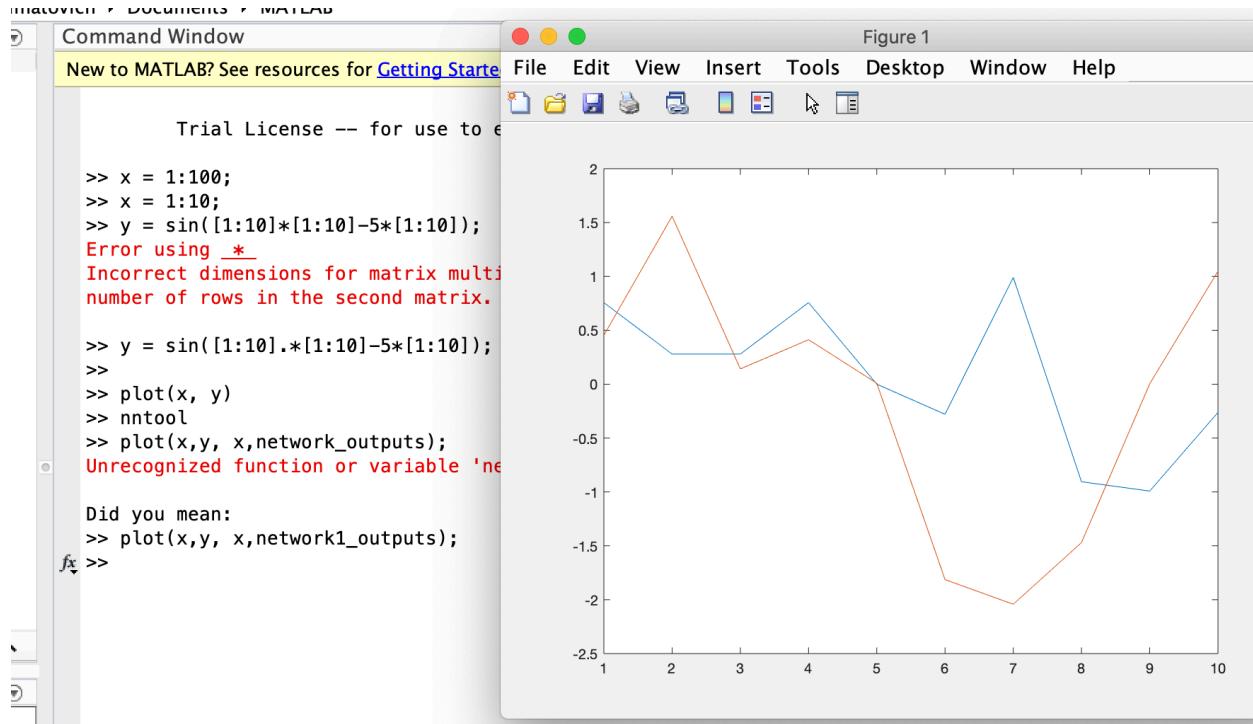
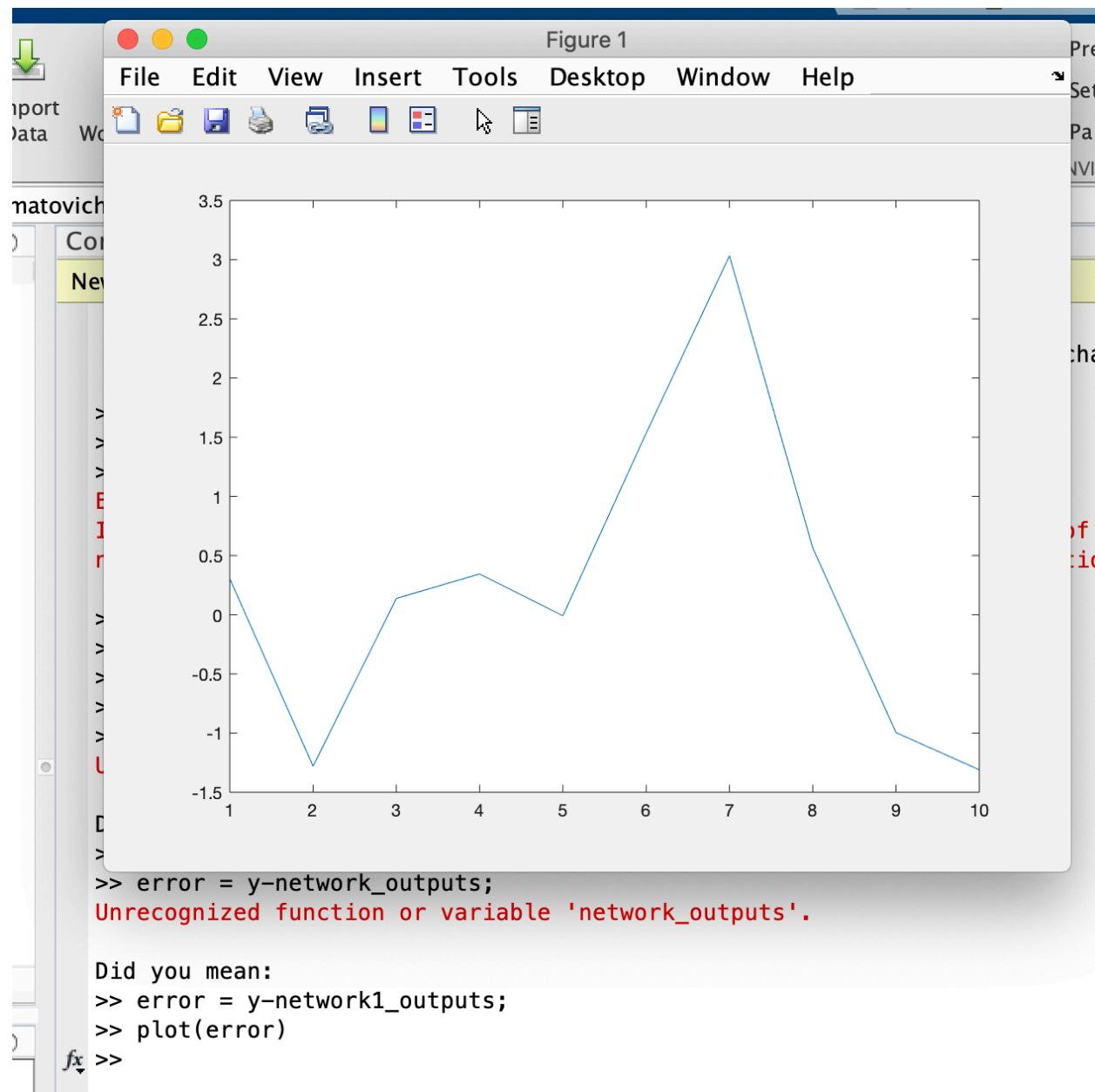


График ошибки приближения



Вывод:

Построила модель нейронной сечи, аппроксимирующей полином на интервале [-1,1],

Построила модель нейронной сети для аппроксимации функции $f(t)$ на промежутке [0,10],
заданной согласно номеру варианта

Лабораторная работа №5

Задача: Распознавание образов. Анализ работы многослойной нейронной сети по распознаванию букв методом обратного распространения ошибки.

Ход работы:

восстановим шаблон для i-й буквы алфавита

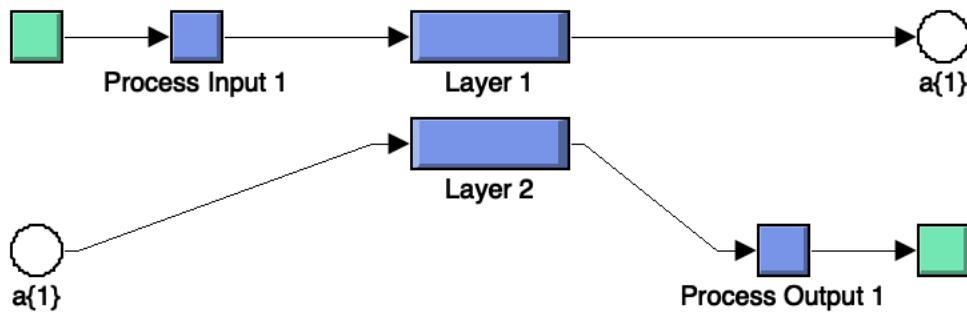
```
>> import nnet.*  
import network.*  
>> [alphabet, targets] = prprob;  
>> i = 10;  
>> ti = alphabet(:, i);  
>> letter{i} = reshape(ti, 5, 7)';  
>> letter{i}  
  
ans =  
  
    1     1     1     1     1  
    0     0     1     0     0  
    0     0     1     0     0  
    0     0     1     0     0  
    0     0     1     0     0  
    1     0     1     0     0  
    0     1     0     0     0
```

fix >> |

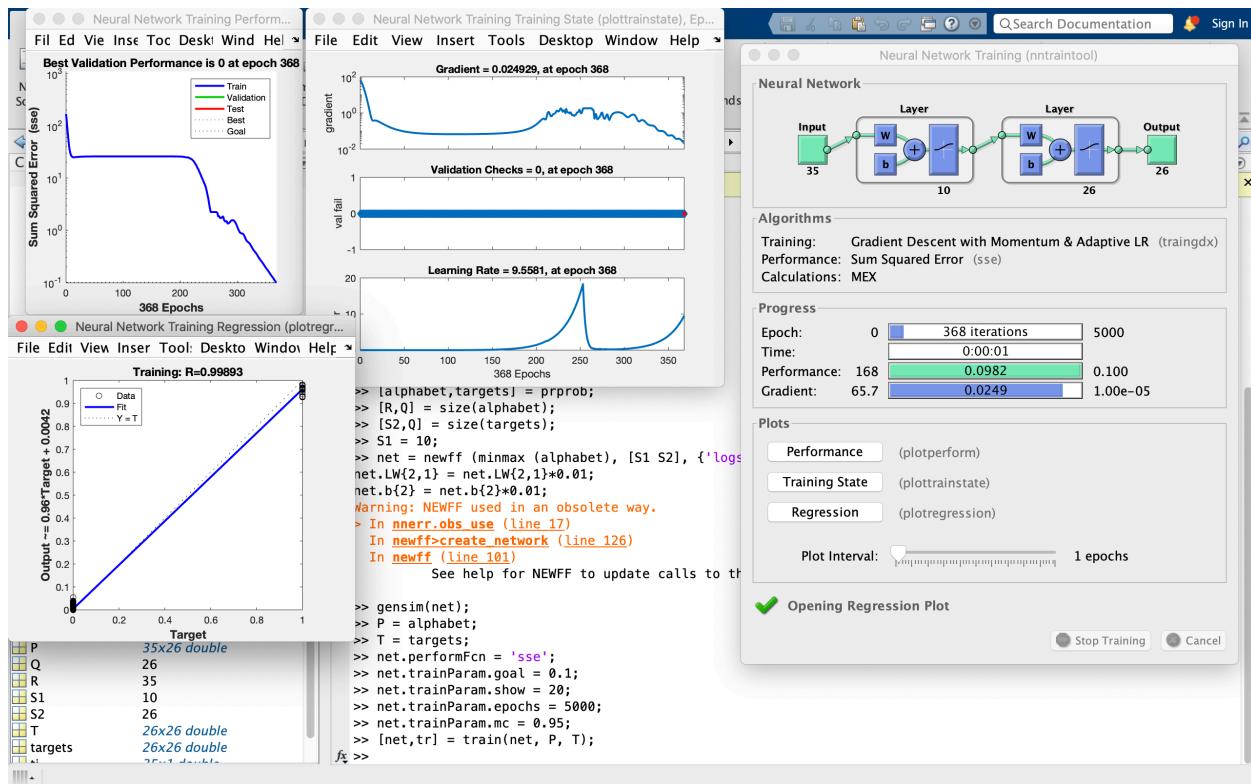
Инициализация сети.

```
>> [alphabet,targets] = prprob;  
>> [R,Q] = size(alphabet);  
>> [S2,Q] = size(targets);  
>> S1 = 10;  
>> net = newff (minmax (alphabet), [S1 S2], {'logsig' 'logsig'}, 'traingdx');  
net.LW{2,1} = net.LW{2,1}*0.01;  
net.b{2} = net.b{2}*0.01;  
Warning: NEWFF used in an obsolete way.  
> In nnerr.obs use (line 17)  
  In newff>create network (line 126)  
  In newff (line 101)  
    See help for NEWFF to update calls to the new argument list.  
  
>> gensim(net);
```

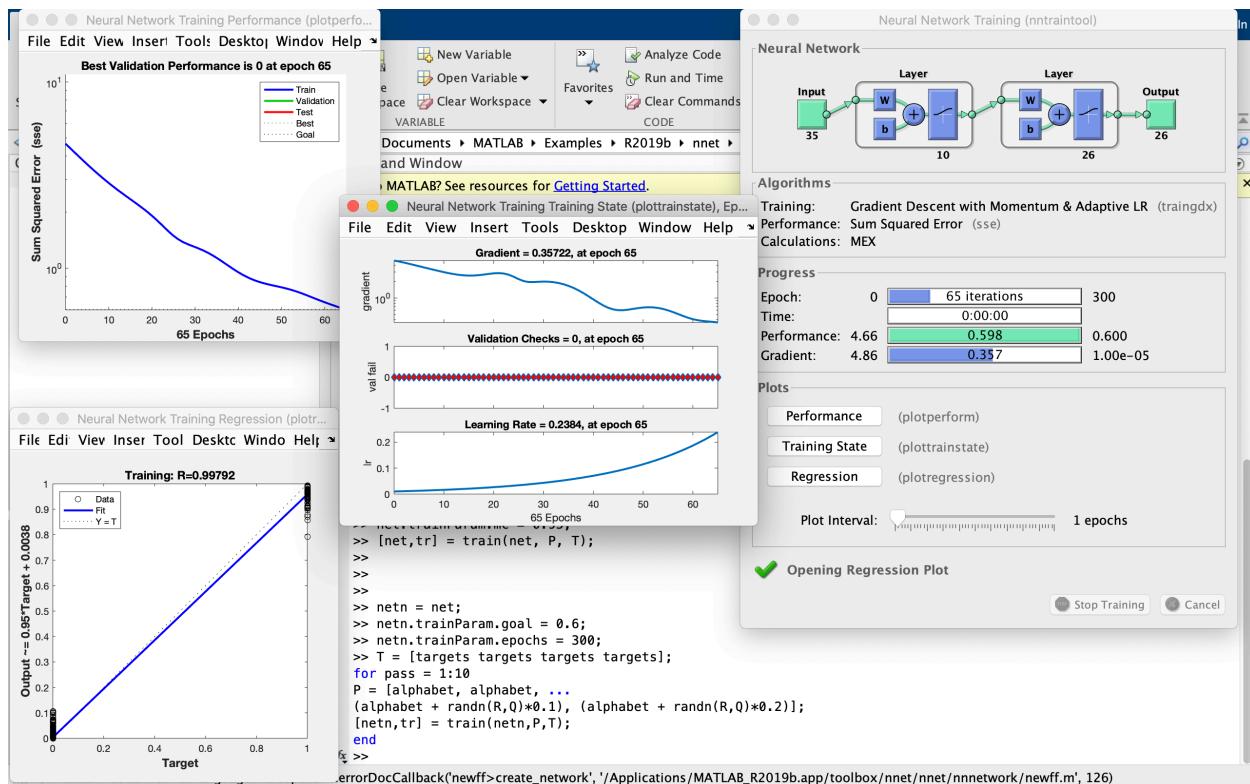
Структура нейронной сети



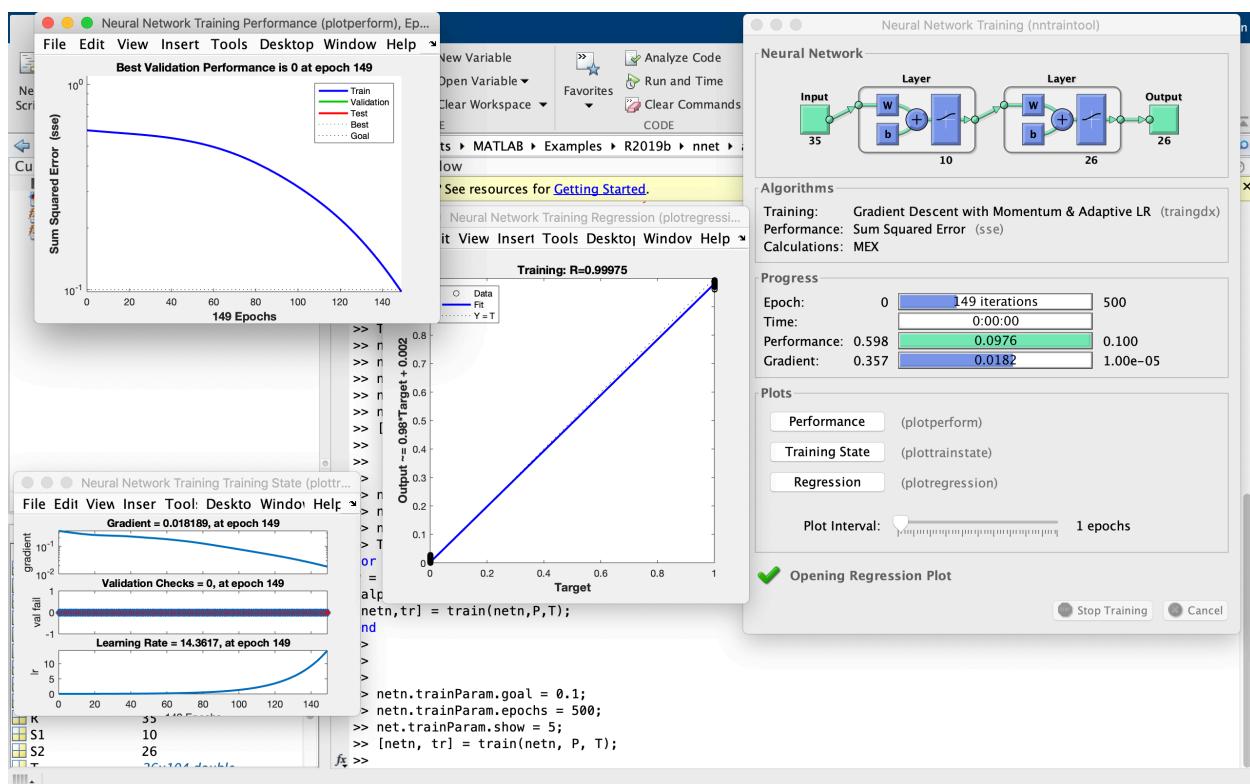
Обучение сети распознаванию образов в отсутствии шума



Обучение сети распознаванию образов в присутствии шума

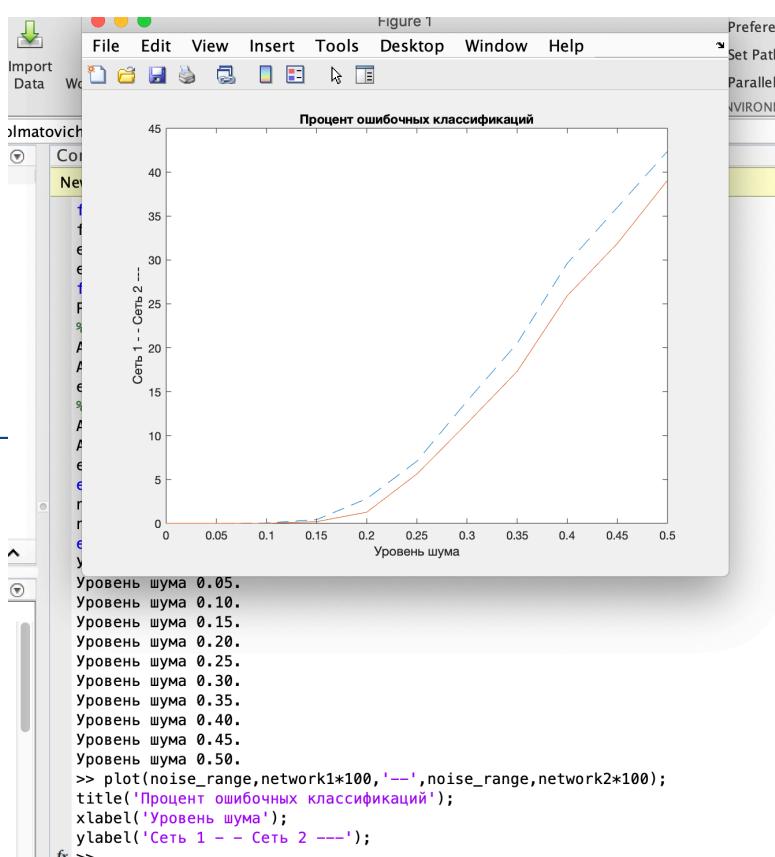


Повторное обучение в отсутствие шума

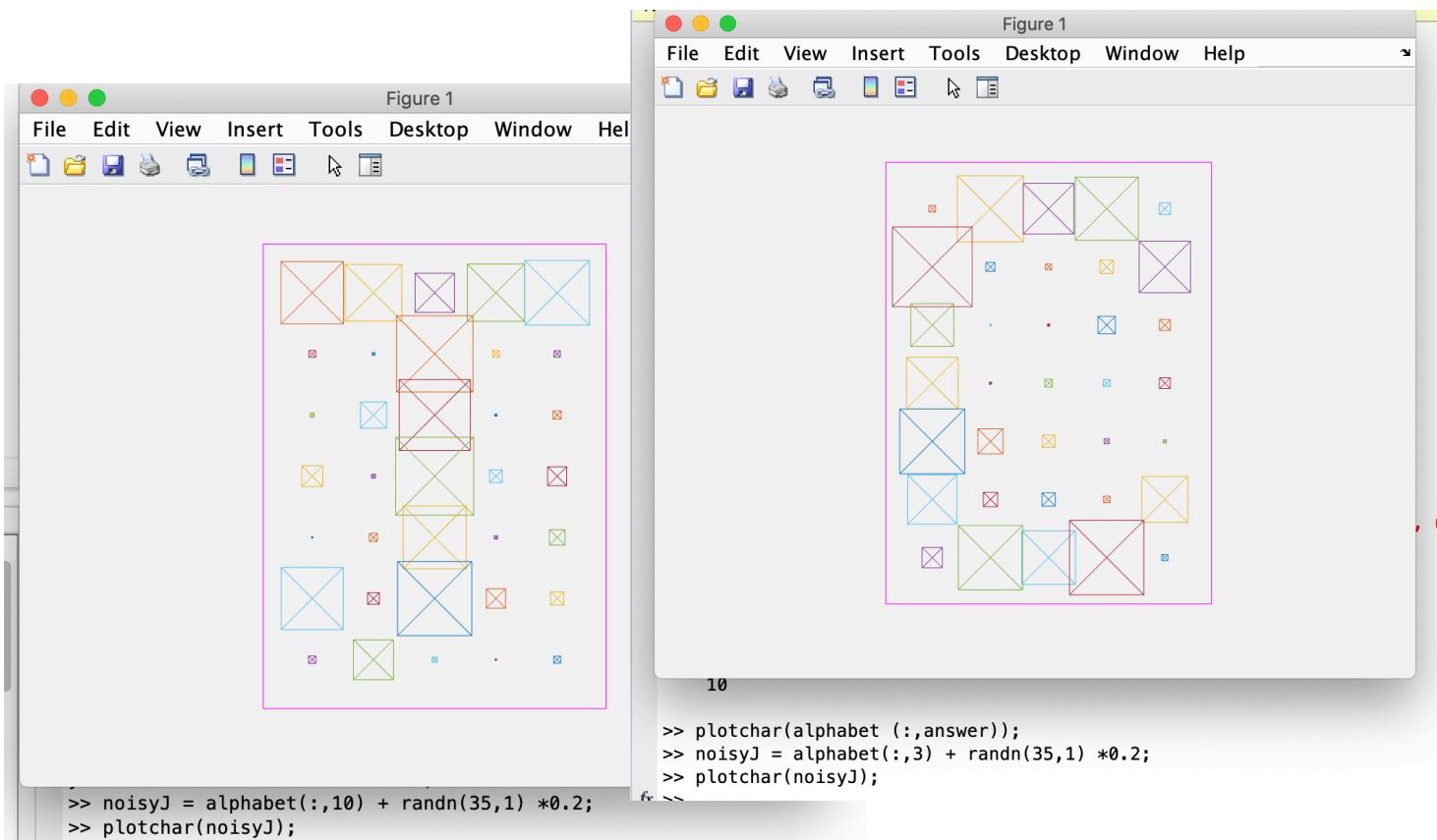


Проверка эффективности функционирования сети

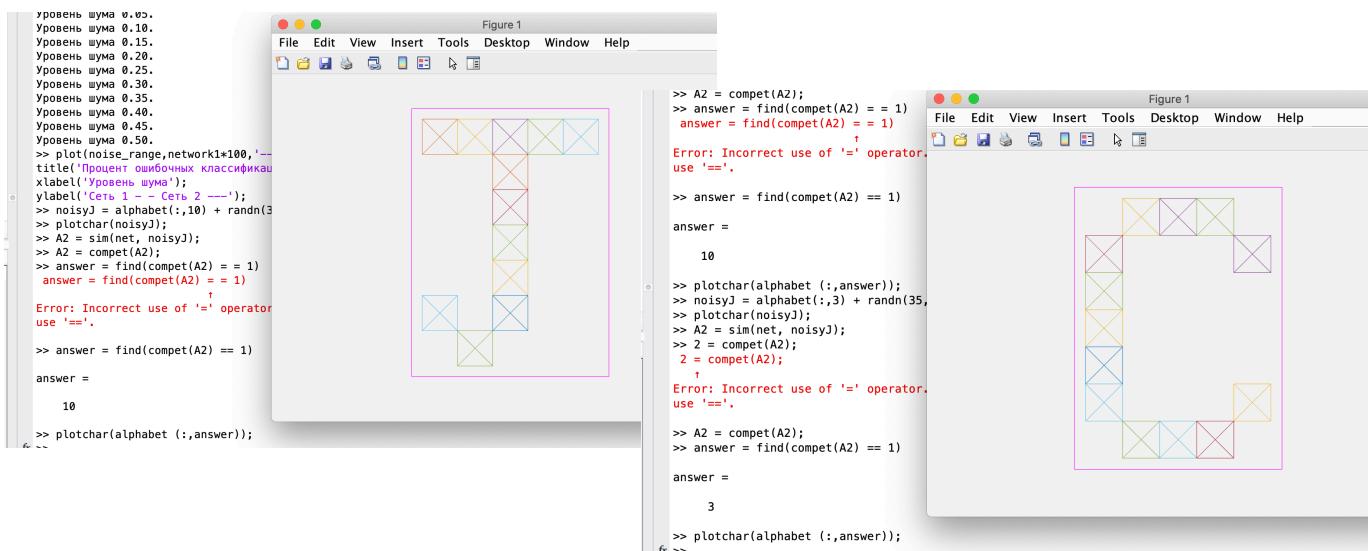
```
>> noise_range = 0:.05:.5;
max_test = 100;
network1 = [];
network2 = [];
T = targets;
for noiselevel = noise_range
    fprintf('Уровень шума %.2f.\n',noiselevel);
    errors1 = 0;
    errors2 = 0;
    for i = 1:max_test
        P = alphabet + randn(35,26)*noiselevel;
        % Тест для сети 1
        A = sim(net,P);
        AA = compet(A);
        errors1 = errors1+sum(sum(abs(AA-T)))/2;
        % Тест для сети 2
        An = sim(netn,P);
        AAn = compet(An);
        errors2 = errors2+sum(sum(abs(AAn-T)))/2;
    end
    network1 = [network1 errors1/26/100];
    network2 = [network2 errors2/26/100];
end
Уровень шума 0.00.
Уровень шума 0.05.
Уровень шума 0.10.
Уровень шума 0.15.
Уровень шума 0.20.
Уровень шума 0.25.
Уровень шума 0.30.
Уровень шума 0.35.
Уровень шума 0.40.
Уровень шума 0.45.
Уровень шума 0.50.
```



Проверим работу нейронной сети для распознавания символов. Сформируем зашумленный вектор входа для символа J(C):



Распознанные символы



Вывод:

Эта задача демонстрирует, как может быть разработана простая система распознавания изображений. Заметим, что процесс обучения не состоял из единственного обращения к обучающей функции. Сеть была обучена несколько раз при различных векторах входа. Обучение сети на различных наборах зашумленных векторов позволило обучить сеть работать с изображениями, искаженными шумами, что характерно для реальной практики.

Лабораторная работа №6

Задача: Изучение радиальных базисных, сетей регрессии, вероятностных нейронных сетей

1) Создание и исследование моделей сетей GRNN: согласно варианта задания подготовить массивы входных векторов Р и целей Т для нейронной сети GRNN, создать и обучить сеть GRNN для аппроксимации функции.

Значения вектора входа: [2 -1 1 -1 2]

Значения вектора выхода: [-1 1 1 1 3]

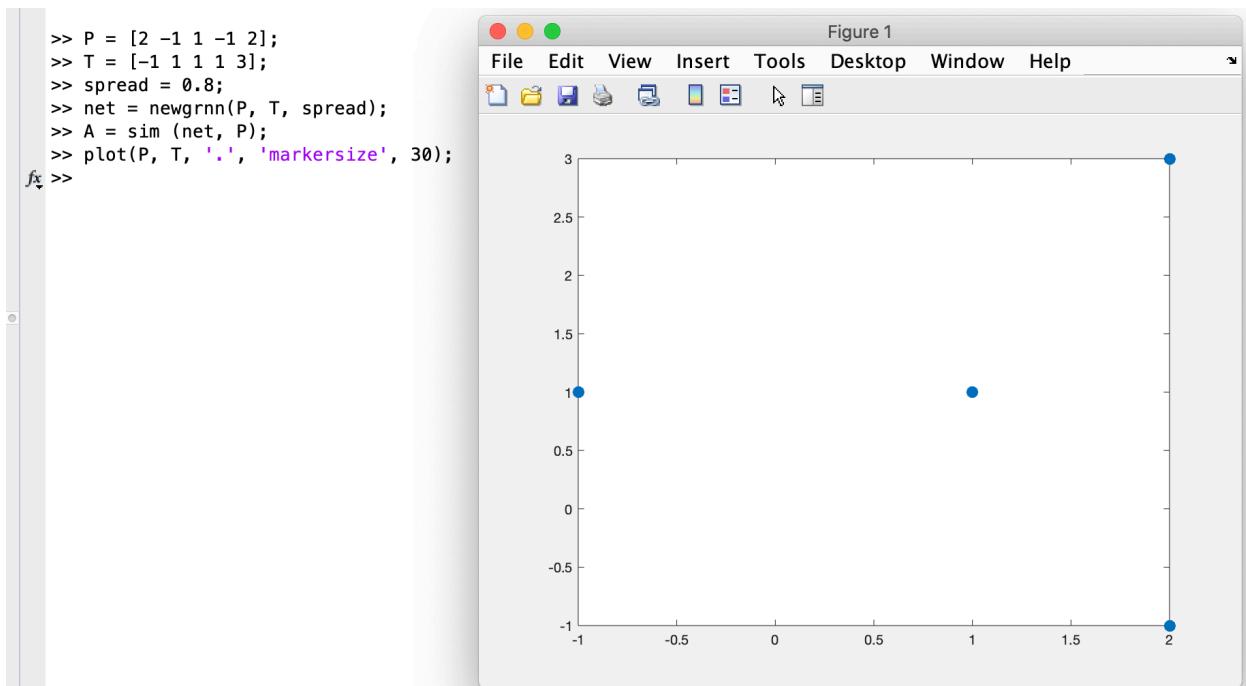
2) Построить модель радиальной базисной сети, аппроксимирующую функцию

$$z = e^x + 3y, x \in [0,1], y \in [-2,1]$$

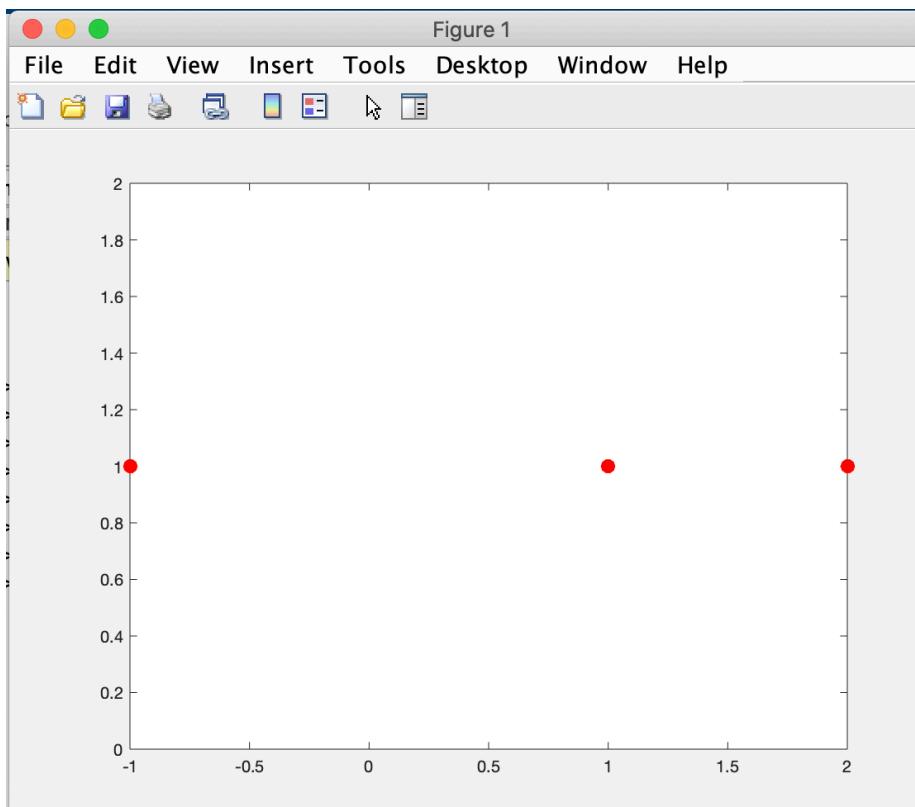
Ход работы:

Создание и обучение нейронной сети регрессии

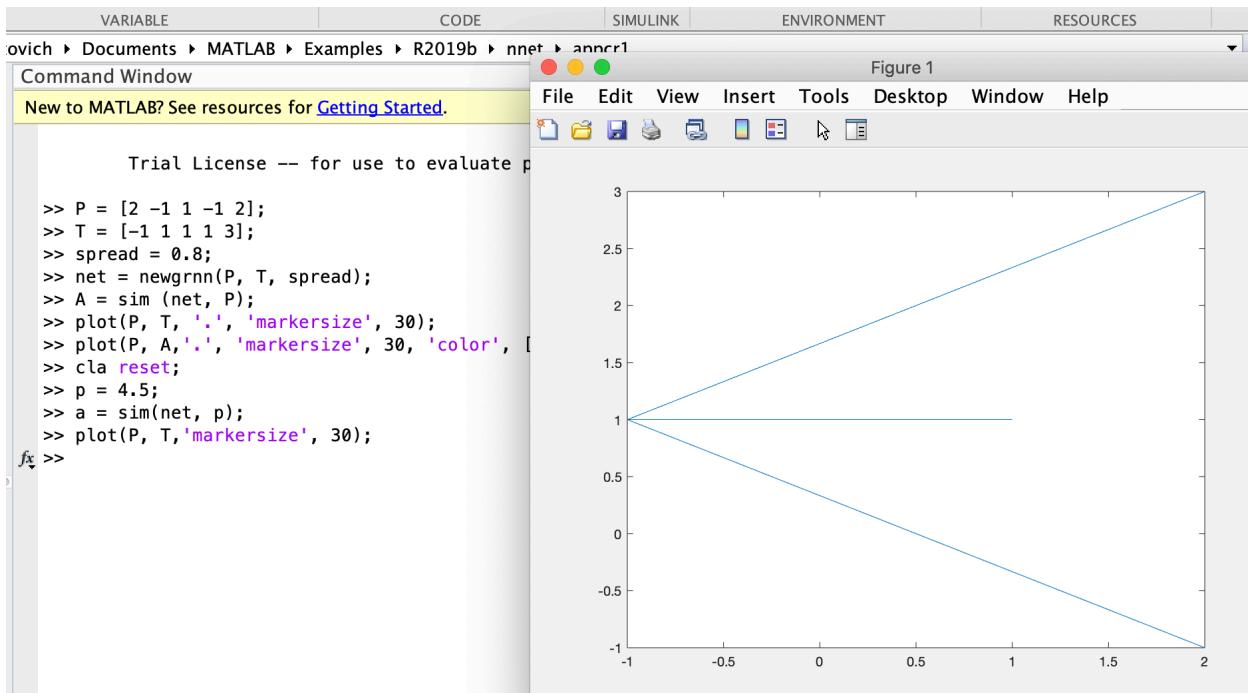
изображение аппроксимируемой функции



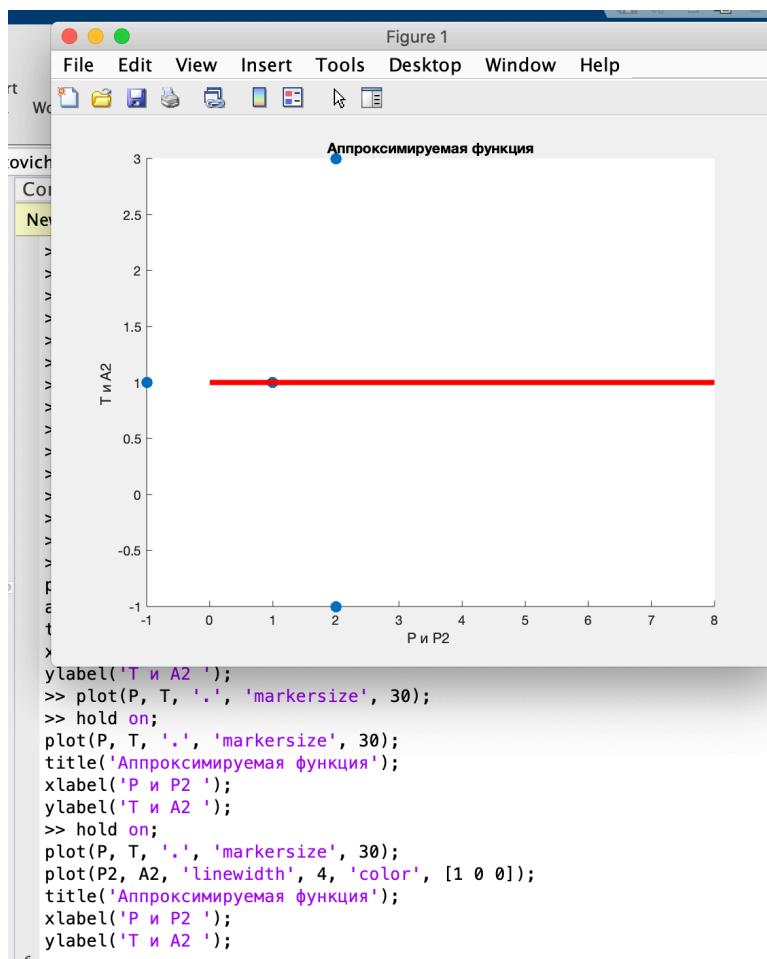
изображение работы необученной НС регрессии



Изображение аппроксимируемой функции

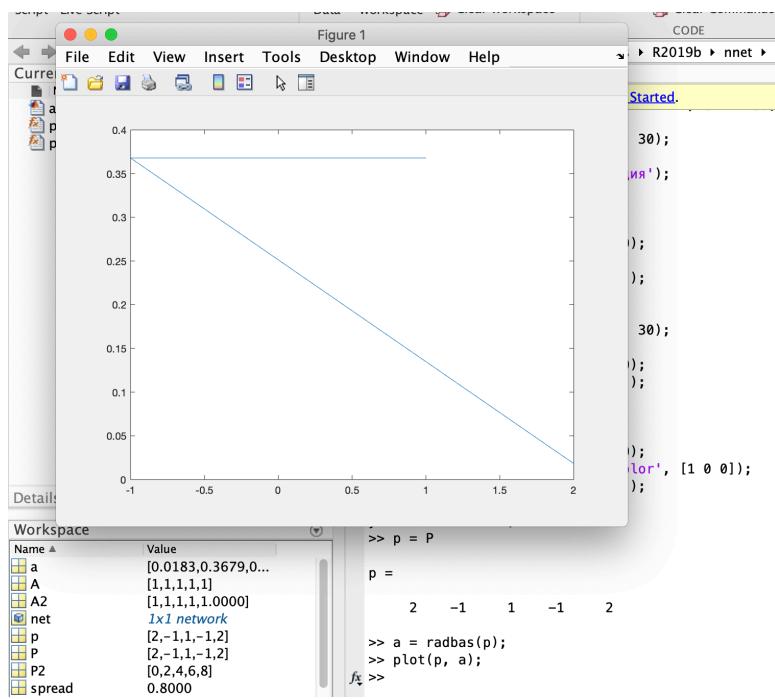


Апроксимация точек с помощью нейронной сети регрессии

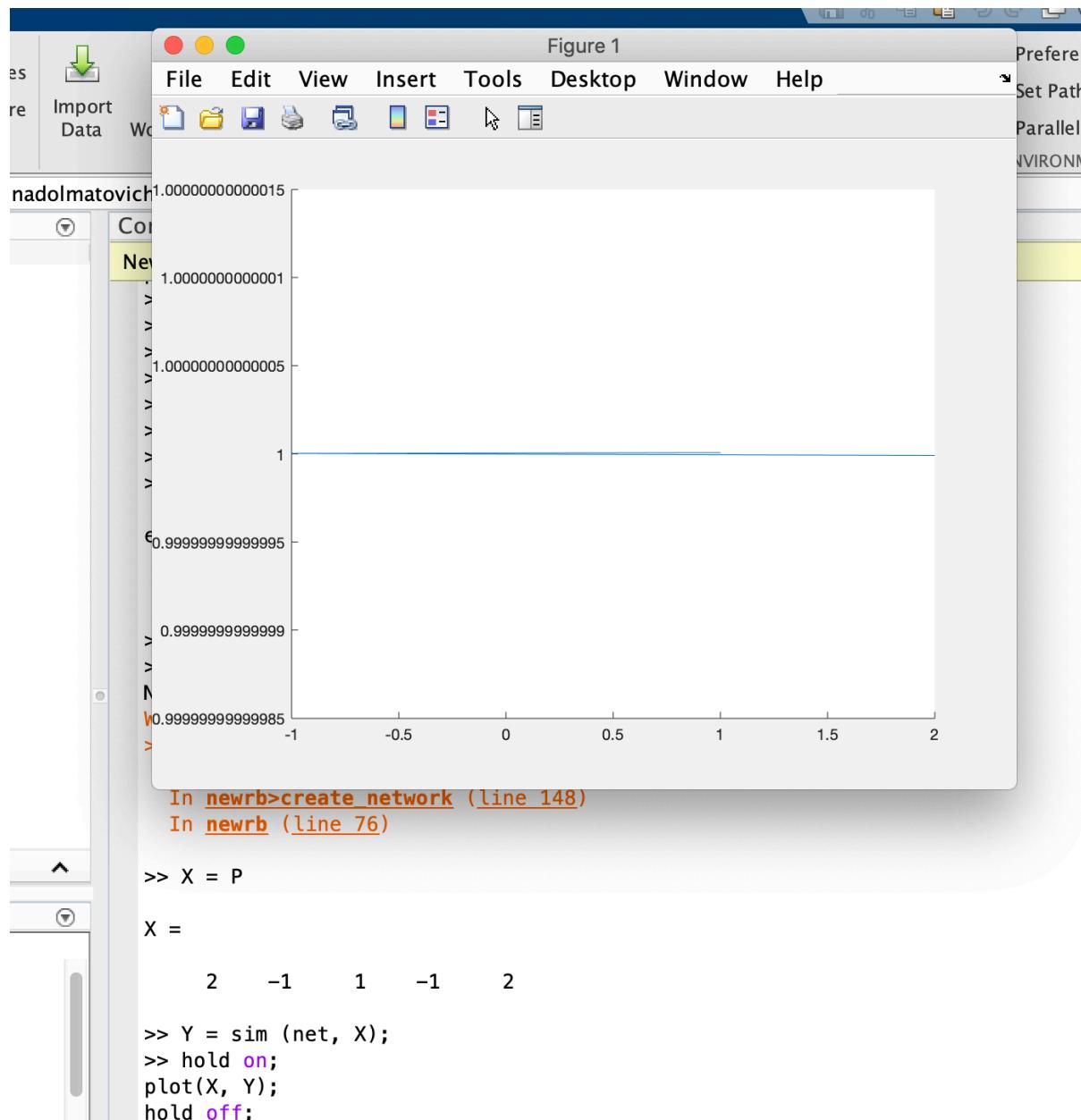


Рассмотрим аппроксимацию функций на основе радиальной базисной сети.

РБФ

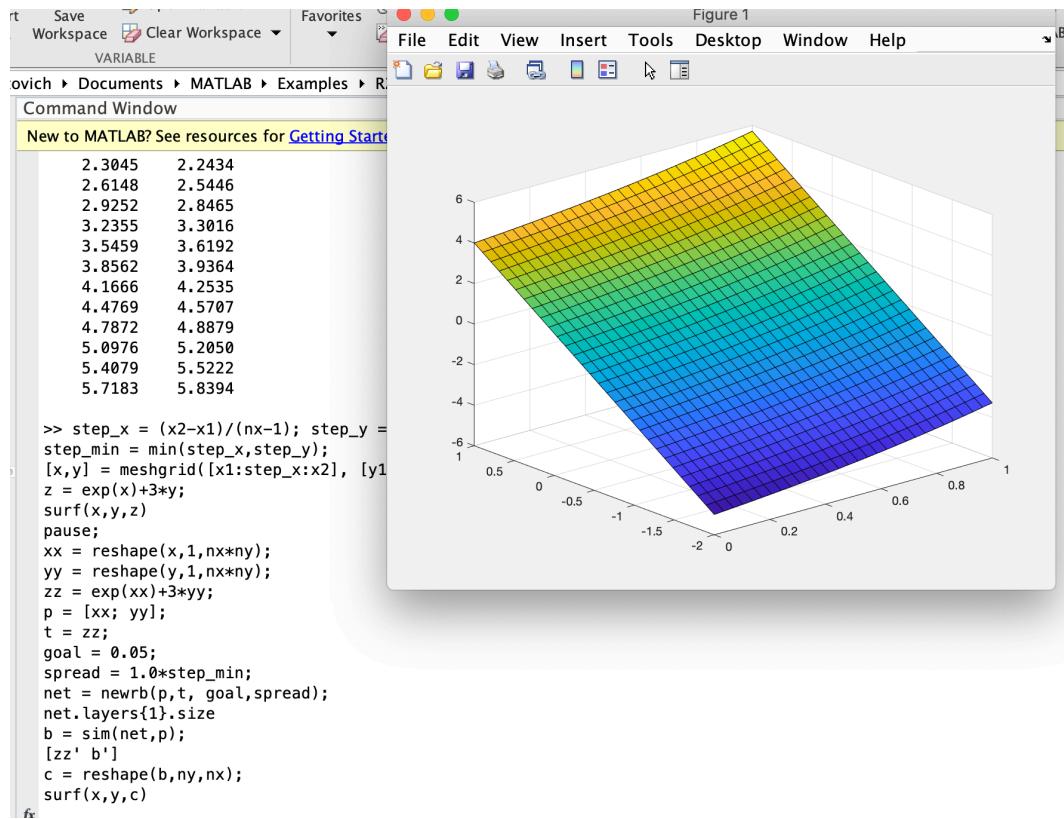


Результат аппроксимации с РБНС

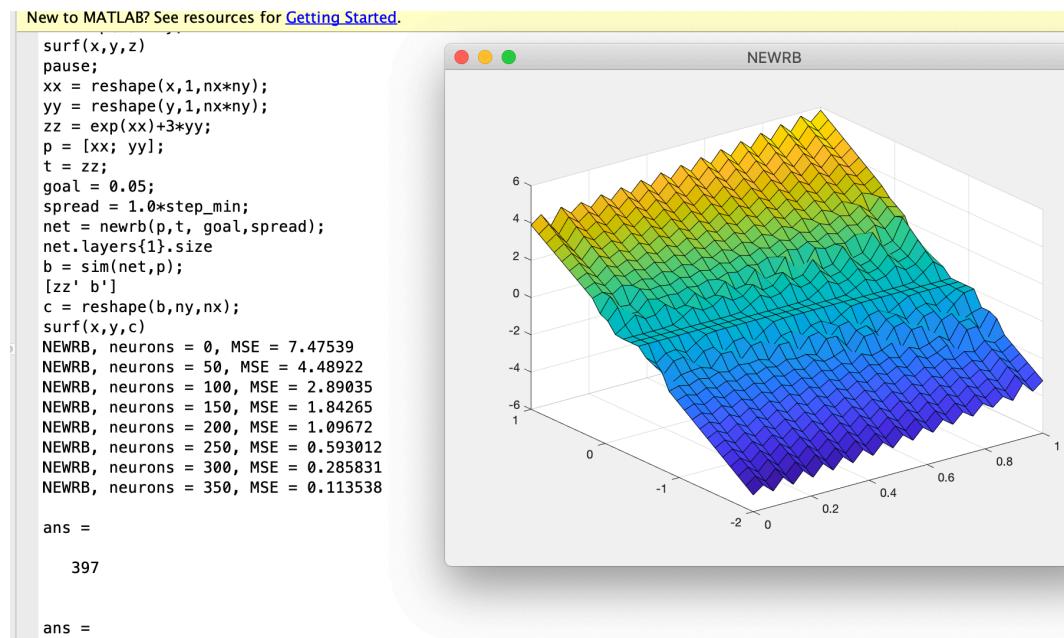


решение аппроксимации функции двух переменных

График исходной функции двух переменных



результат аппроксимации нелинейной зависимости



Вывод: Изучила радиальных базисных, сетей регрессии, вероятностных нейронных сетей

Лабораторная работа №7

Задача: Исследование сети Кохонена и алгоритма обучения без учителя

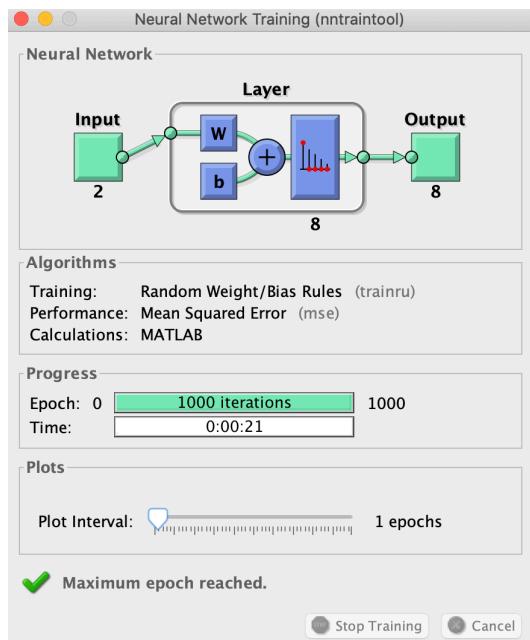
Ход работы:

создадим однослойную нейронную сеть Кохонена.

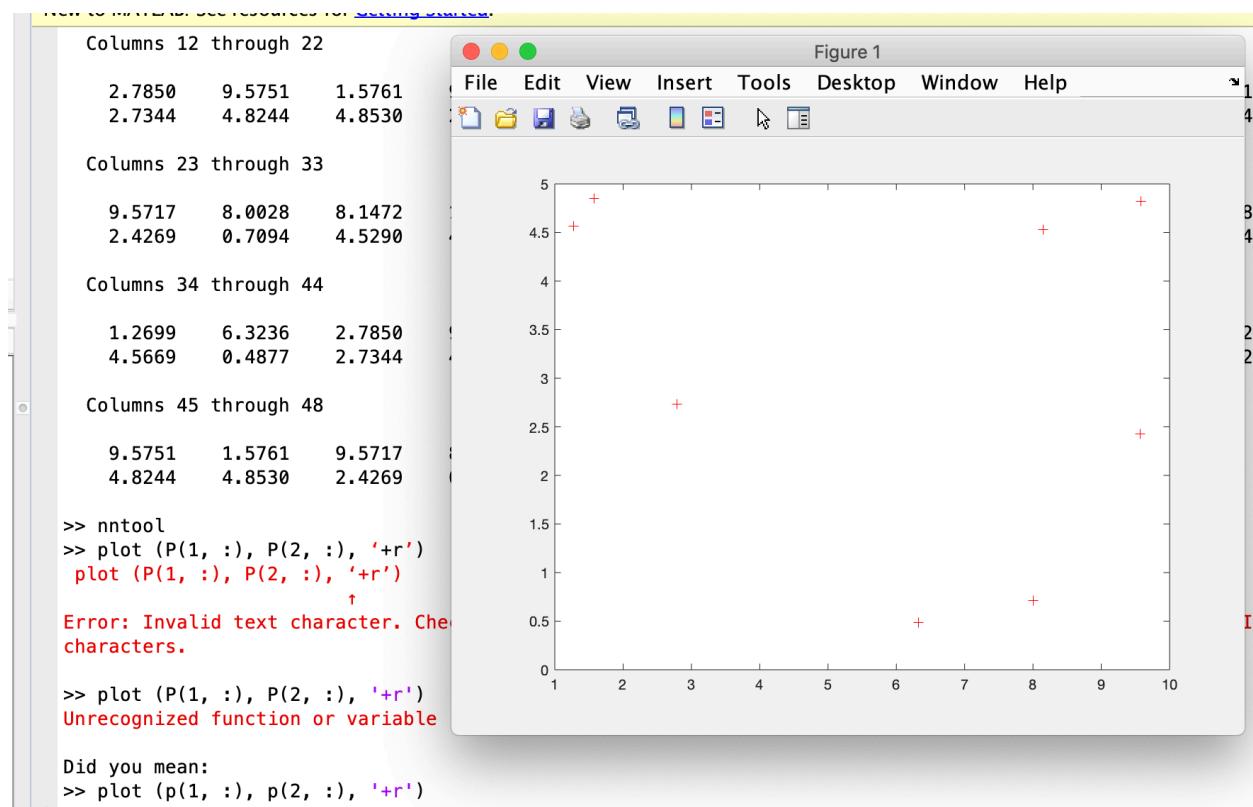
The screenshot shows the MATLAB environment with the following components:

- Toolbar:** Standard MATLAB toolbar with File, Variable, Code, and Community tabs.
- Current Folder:** Shows files `appcr1.m`, `plotchar.m`, and `prprob.m`.
- Workspace:** Displays variables `c`, `d`, `maxv`, `minv`, `n`, `p`, `q`, `r`, `t`, `v`, and `x` with their respective values.
- Command Window:** Contains MATLAB code for generating input data `p` and target data `t`. It includes a warning about matrix dimensions and displays numerical data for columns 1 through 33.
- Network Properties:** A dialog box for creating a Kohonen network. It shows the network is named `network1` and is of type `Competitive`. It specifies `Input data: p`, `Number of neurons: 8`, `Kohonen learning rate: 0.1`, and `Conscience learning rate: 0.001`.
- Network Diagram:** A schematic diagram titled "Layer" showing an input node (labeled 2) connected to a processing unit. The unit contains weight matrix `W`, bias vector `b`, and a summation node (`+`). The output of this unit is then passed through another processing unit (labeled 8) to an output node (labeled 8).
- Bottom Navigation:** Buttons for View, Train, Simulate, Adapt, and Reinit.

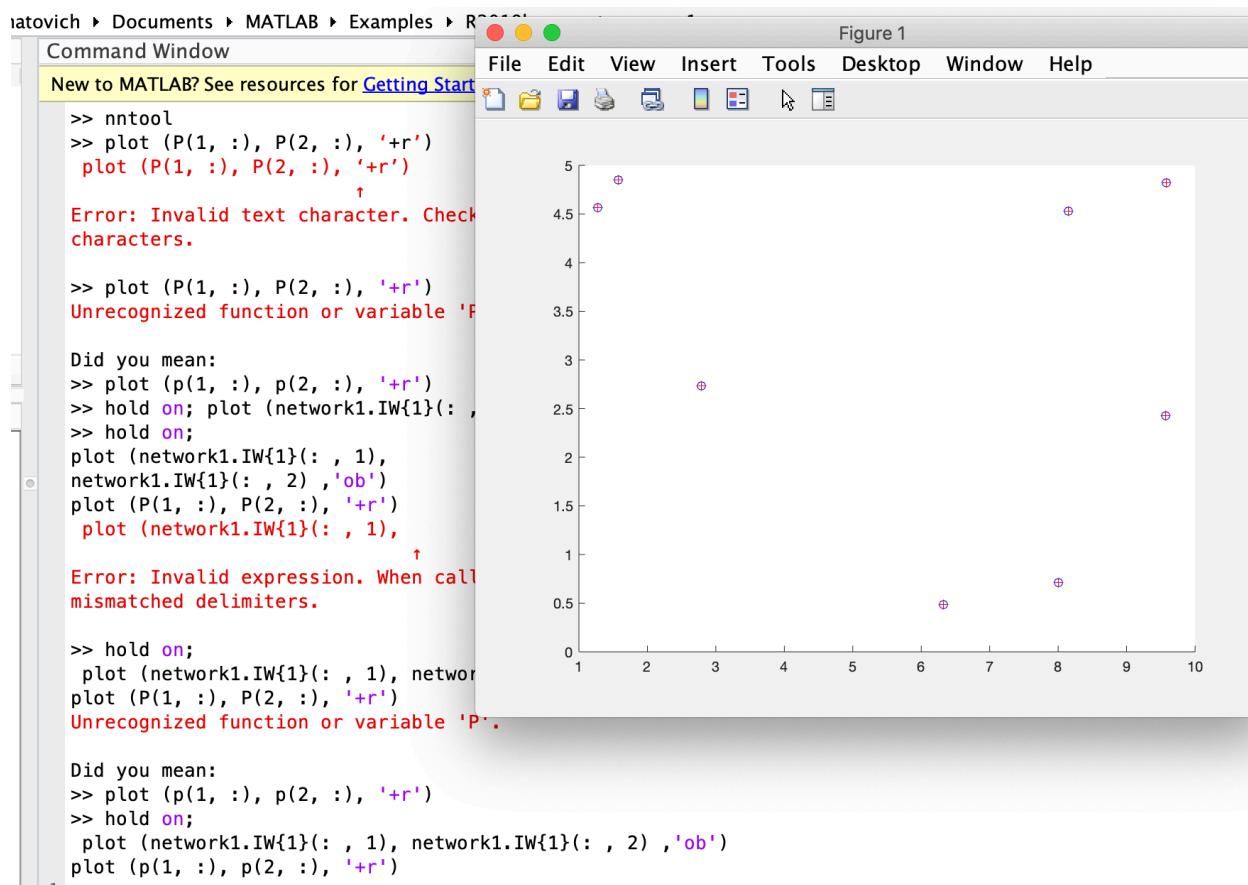
Проведем инициализацию весов сети, установим диапазон изменения входных данных и проведем обучение сети.



Для иллюстрации обучения построим расположение входных данных.



Построим центры классифицируемых областей.



Создание и использование соперничающего слоя в командном окне

```
>> P = [.2 .5 .1 .8; .3 .7 .2 .6];
>> net = newc ([0 1; 0 1], 2);
>> net = train (net, P);
>> Y = sim (net, P)

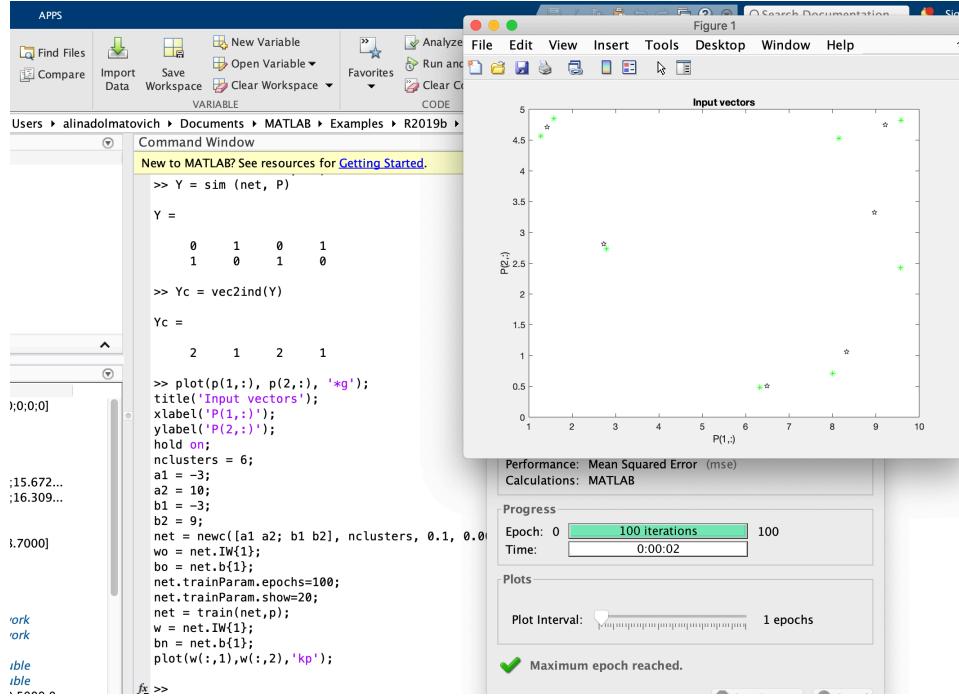
Y =
0      1      0      1
1      0      1      0

>> Yc = vec2ind(Y)

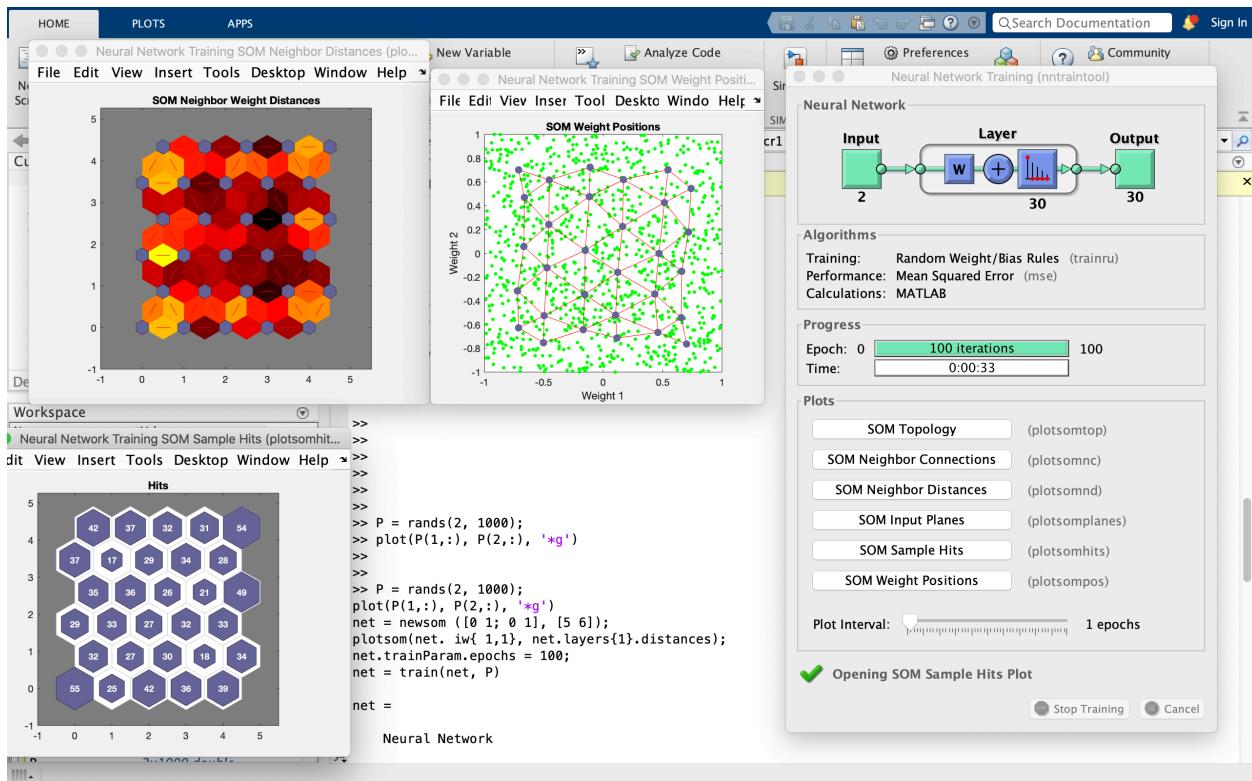
Yc =
2      1      2      1

fx >>
```

обучения самоорганизующейся нейронной сети Кохонена. Исходные данные - снежинки, полученные центры кластеров — звезды)



Применение SOM для кластеризации векторов

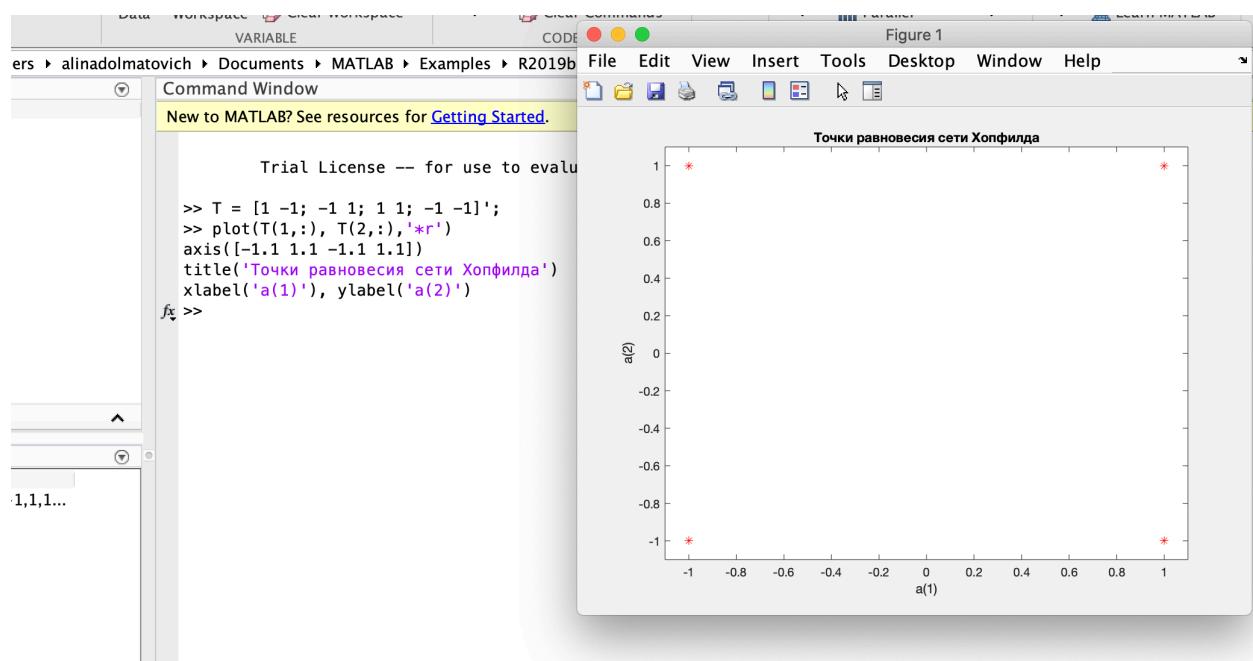


Лабораторная работа №8

Задача: Исследование сети хопфилда. исследовать устойчивость сети Хопфилда и её сходимость к аттракторам, применение сети Хопфилда для распознавания образов.

Ход работы:

Создадим нейронную сеть Хопфилда с четырьмя нейронами и определим 4 точки равновесия в виде следующего массива целевых векторов



Рассчитаем веса и смещения модифицированной сети Хопфилда,

```
>> net = newhop(T);  
W = net.LW{1,1}  
b = net.b{1,1}
```

W =

```
1.1618      0  
0      1.1618
```

b =

```
1.0e-16 *  
-0.1797  
-0.1797
```

```
f x >>
```

Проверим, принадлежат ли вершины квадрата сети Хопфилда:

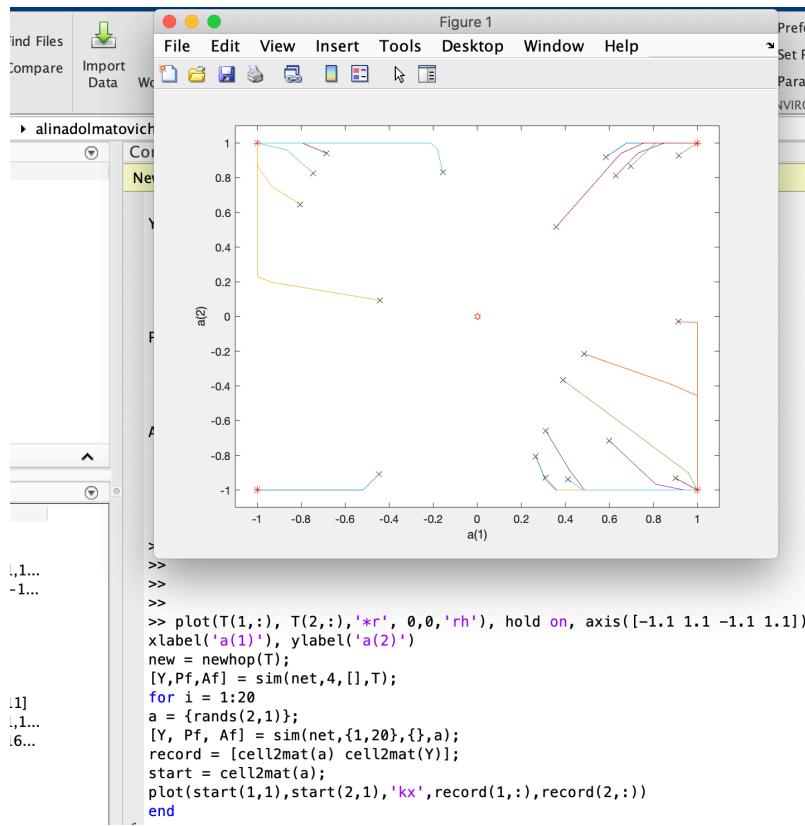
```
>> Ai = T;
[Y,Pf,Af] = sim(net,4,[],Ai)

Y =
1   -1   1   -1
-1   1   1   -1

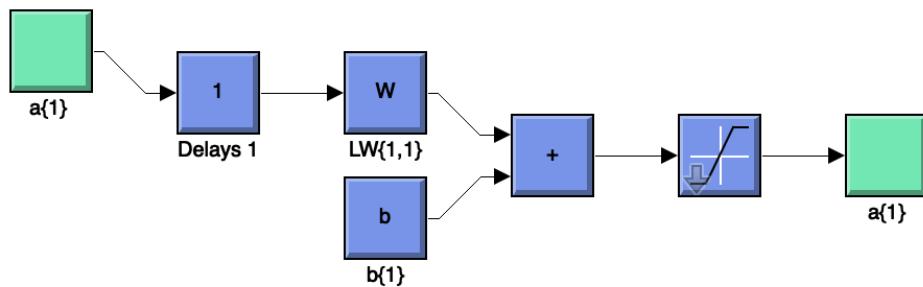
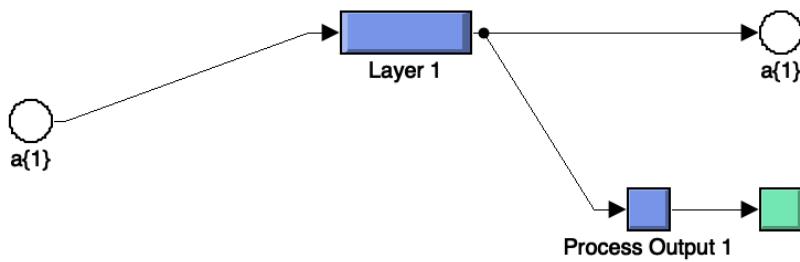
Pf =
0x0 empty cell array

Af =
1x1 cell array
{2x4 double}
```

Теперь проверим поведение сети при случайных начальных условиях.



Создадим сеть Хопфилда с двумя устойчивыми точками в трехмерном пространстве



Проверим сходимость сети при произвольном входном векторе Ai

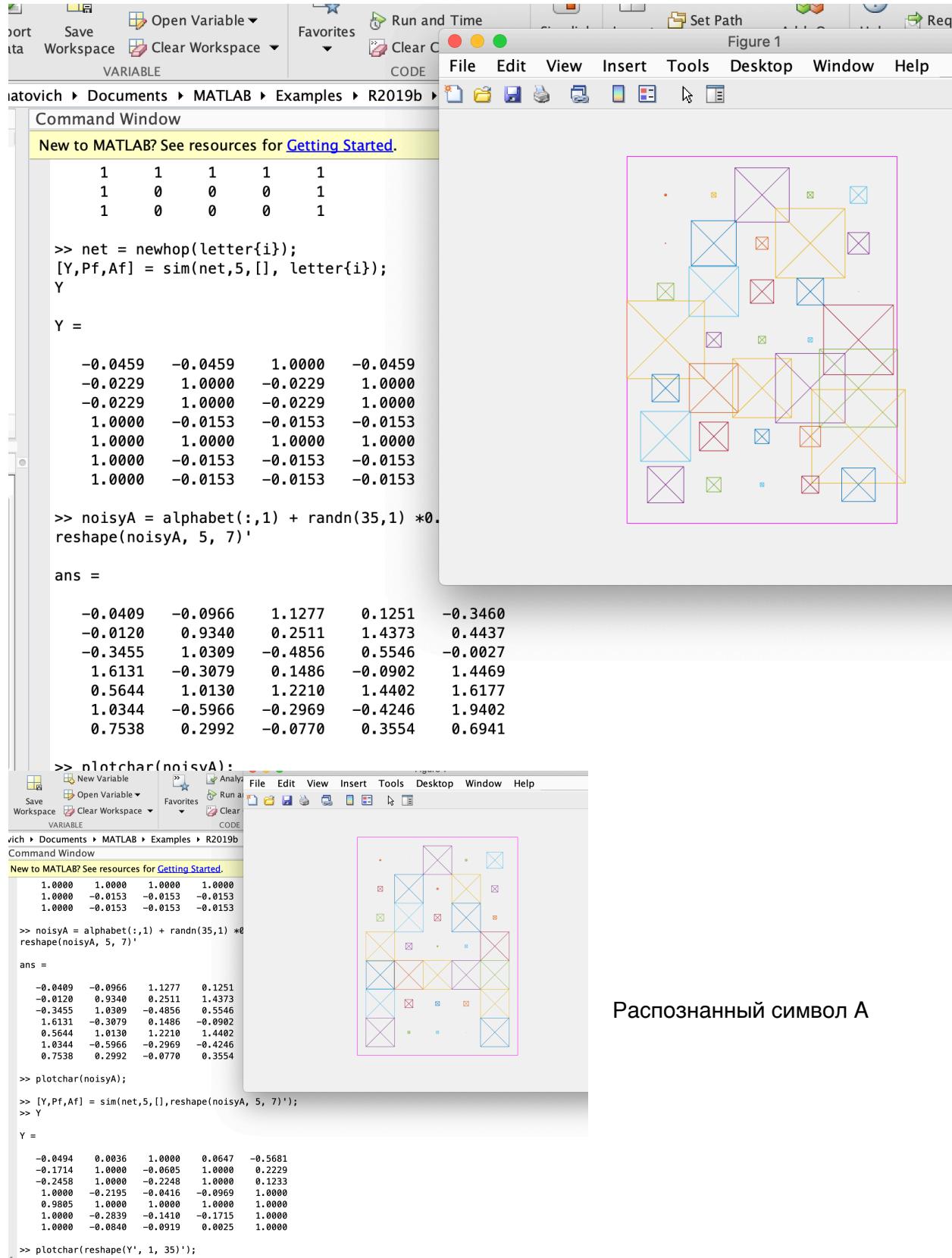
```

>>
>> Ai = {[ -0.8; -0.7; 0.6] };
[Y,Pf,Af] = sim(net,{1 5},{},Ai);
Y{1}

ans =
-0.9295
-1.0000
0.9884

```

Создадим зашумленную букву А



Распознанный символ А