**Лабораторная работа №1**

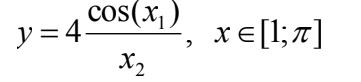
**Задача:** Создать, используя nntool НС для аппроксимации функции.

Значения для входа и цели вычислить в заданном диапазоне.

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.

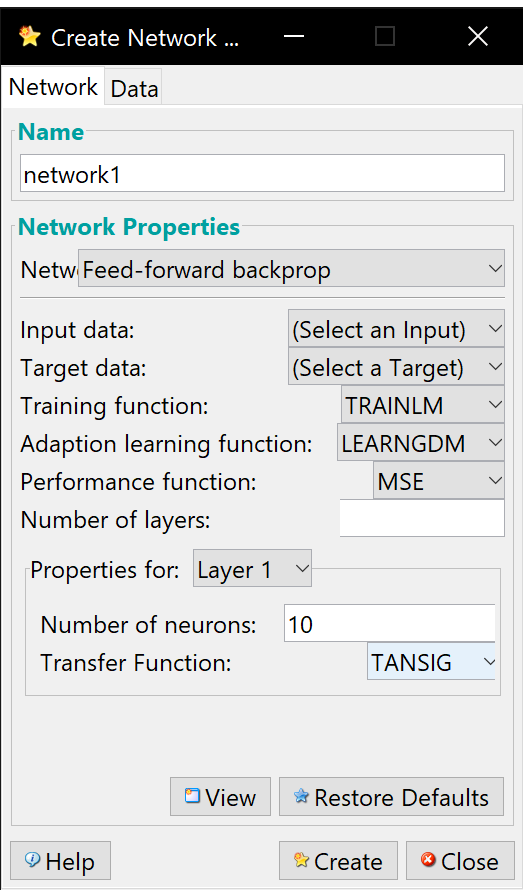
Вариант



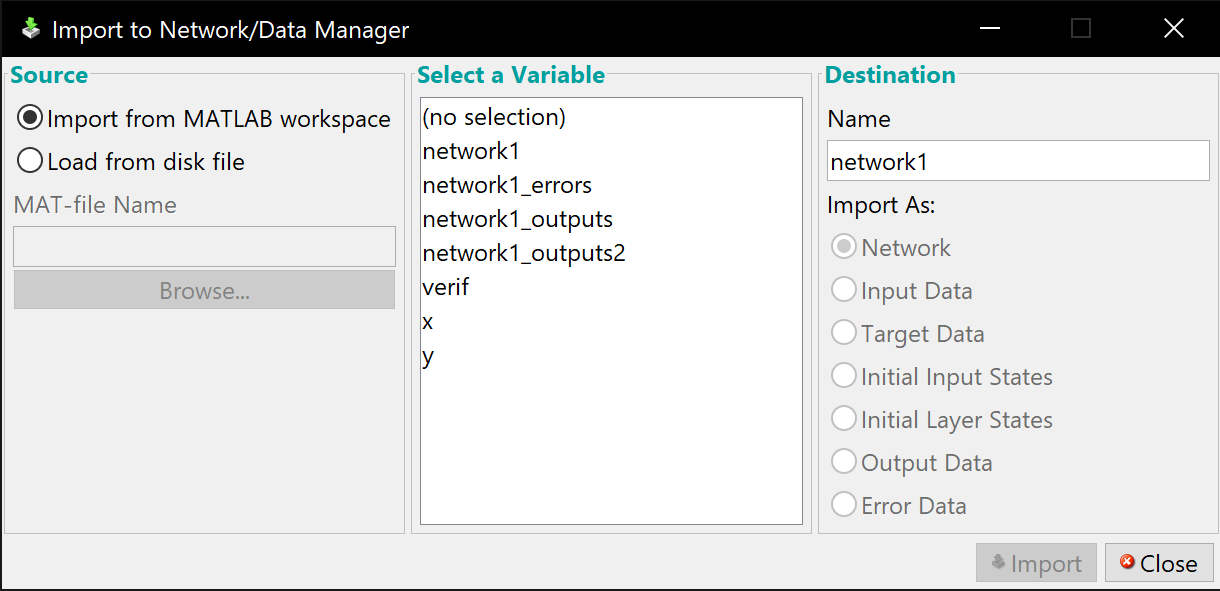
**Ход работы:**

Создадим входные данные x1, x2 и данные цели

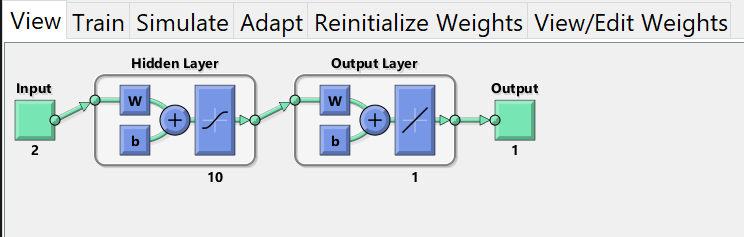
import random  
import math  
  
x1 = [round(random.uniform(-1, 1), 2) for \_ in range(100)]  
x2 = [round(random.uniform(-1, 1), 2) for \_ in range(100)]  
y = [round(4\*math.cos(x1\_)/x2\_, 2) for x1\_, x2\_ in zip(x1,x2)]  
  
x1\_str = ''  
x2\_str = ''  
y\_str = ''  
for i in range(100):  
 x1\_str += ' ' + str(x1[i])  
 x2\_str += ' ' + str(x2[i])  
 y\_str += ' ' + str(y[i])  
x12\_input = '[' + x1\_str + ';' + x2\_str + ']'  
y\_input = '[' + y\_str + ']'

****

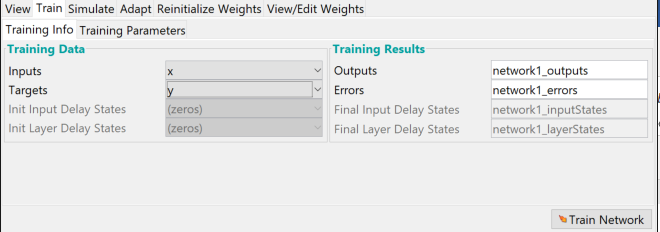
Создадим нейронную сеть

****

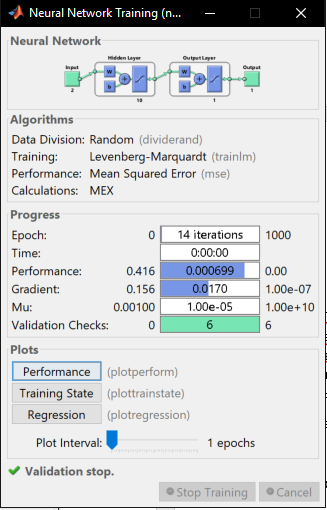
Импортируем её в окружение

****

Выведем топологию сети на экран



Зададим параметры на обучение



Обучим сеть

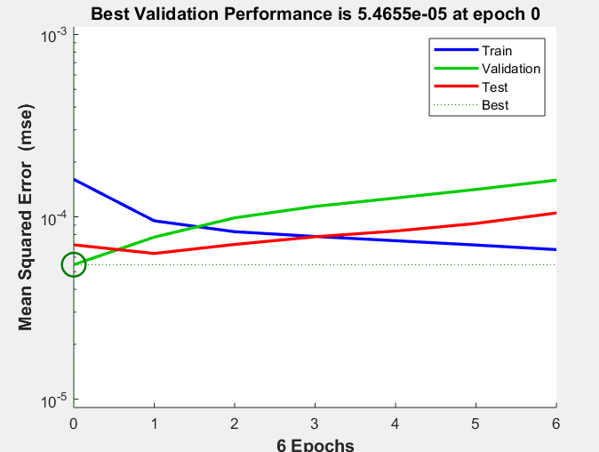


График производительности сети

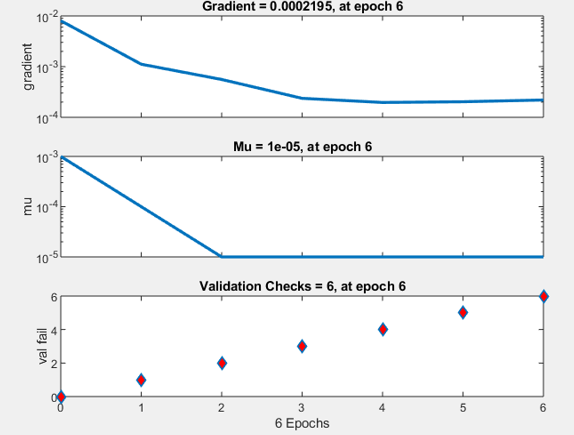
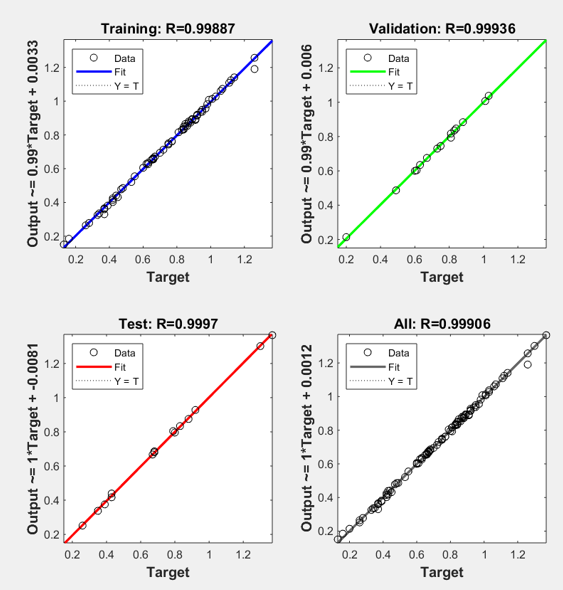


График процесса обучения сети



Процесс регрессии

Проведём тестирование сети

Значения на вход: [1.0,0.1],[0.1,1.0],[ 1.0,1.0],[ 0.1,0.1]

Ожидаемые значения на выход(цели): 21.6, 3.98, 2.16, 39.8

Значения предсказанные сетью: 21.61209223472, 3.9800166611, 2.161209223, 39.8001666

**Вывод:**

Познакомился с основными методами работы GUI Matlab. Создал нейронную сеть апроксимирующую заданную функцию.

**Лабораторная работа №2**

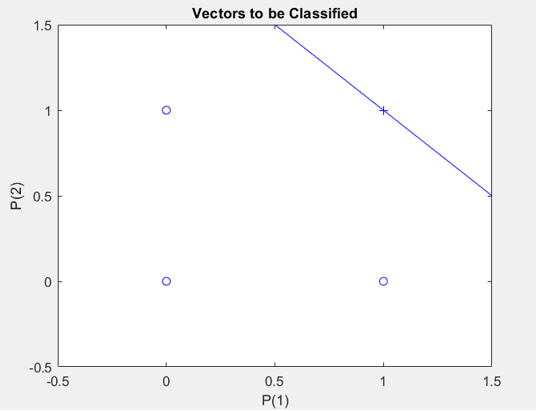
**Задача:** Реализовать логические функции с помощью персептрона (функция perceptron), изобразить входные векторы и разделяющую прямую на плоскости, сделать выводы о реализуемости логических функций

**Ход работы:**

Зададим код работы перцептрона

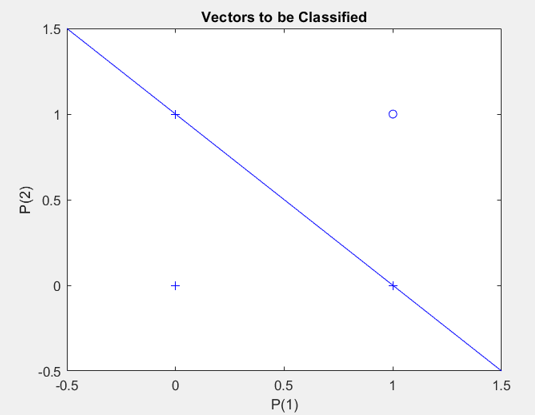
T = [{Insert your function here}]  
P = [0 0 1 1; 0 1 0 1];  
net = perceptron;  
net.adaptParam.passes = 20;  
net = configure(net,P,T);  
plotpv(P, T);  
linehandle = plotpc(net.IW{1}, net.b{1});  
E = 1;  
net = init(net);  
linehandle = plotpc(net.IW{1}, net.b{1});  
while(mse (E));  
[net, Y, E] = adapt(net,P,T);  
linehandle = plotpc(net.IW{1}, net.b{1}, linehandle);  
drawnow;

Зададим перцептрону различные логические функции для анализа



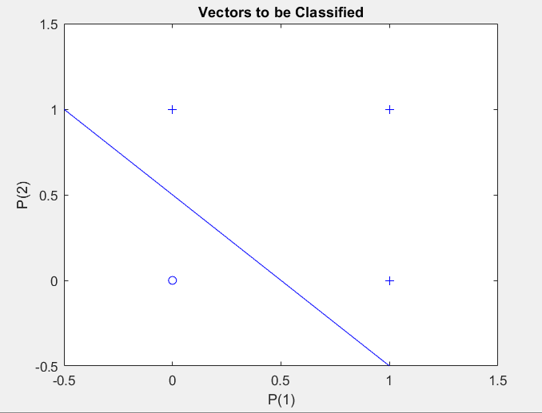
Функция: And

Результат: Модель работает. Разбиение верное.



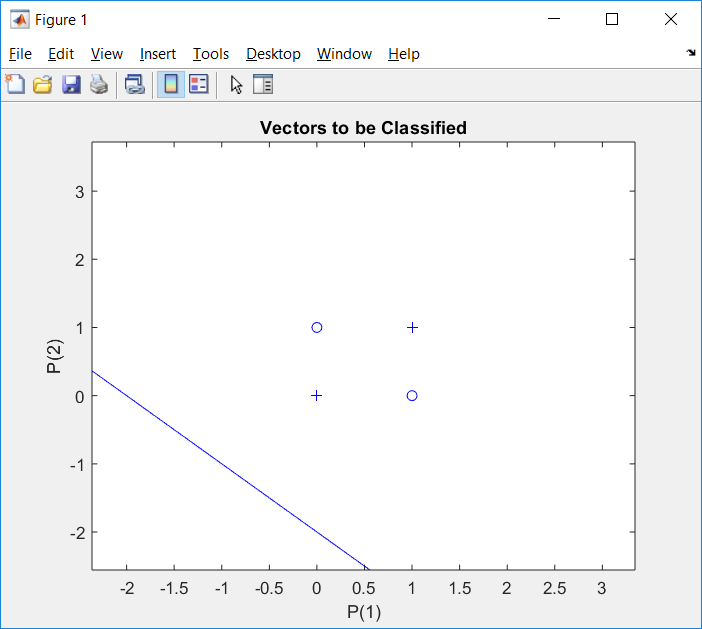
Функция: Nand

Результат: Модель работает. Разбиение верное.



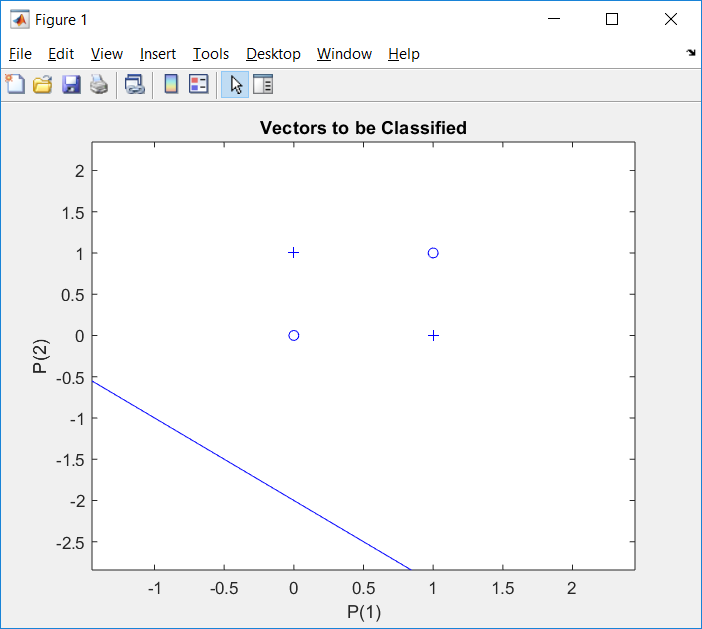
Функция: or

Результат: Модель работает. Разбиение верное.



Функция: xor

Результат: Модель не работает. Разбиение **неверное**.



Функция: Nxor

Результат: Модель не работает. Разбиение **неверное**.

**Вывод:**

Обучил простейшую нейронную сеть модели перцептрона определять и разделять простейшие логические функции. Определил возможности модели простейшего нейрона

**Лабораторная работа №3**

**Задача:** Произвести исследования следующих алгоритмов обучения НС: adapt, trainlm, learngd, traingd, learngdm, traingdm, traingda, traingdx, trainrp на примере обучения сети feedforwardnet. 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:**

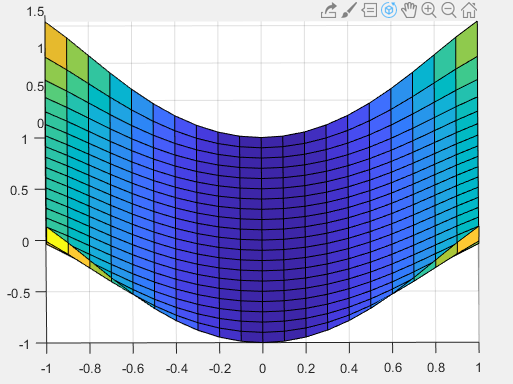
Сгенерируем данные:

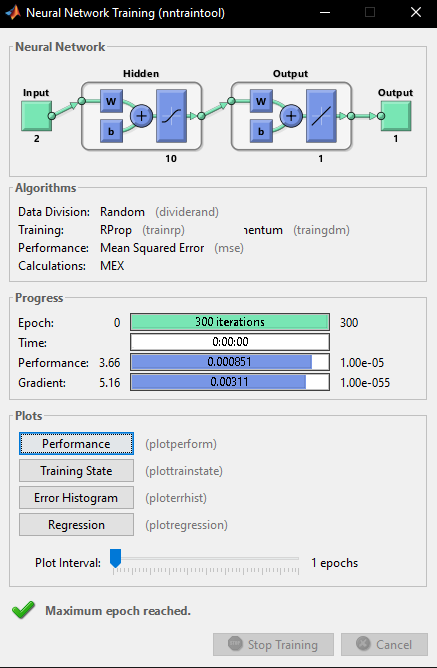
См. код ниже

x=-1:0.1:1;  
y=x;   
p=[x;y];   
p=num2cell(p,1);   
t=sin(x.^2)./cos(y.^2);   
t=num2cell(t,1);

Построим график поверхности функции для аппроксимации:

[x,y]=meshgrid(x,y);   
z=sin(x.^2)./cos(y.^2);   
surf(x,y,z)

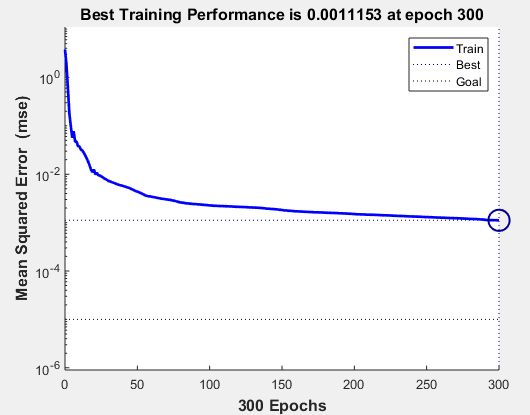




Результат обучения алгоритмом GD

Код

net = feedforwardnet(10, 'traingd');  
net.biases{1,1}.learnFcn='learngd';  
net.biases{2,1}.learnFcn='learngd';  
net.layerWeights{2,1}.learnFcn='learngd';  
net.inputWeights{1,1}.learnFcn='learngd';  
net.layerWeights{2,1}.learnParam.lr = 0.2;  
net.adaptParam.passes=300;  
tic, net=train(net,p,t); toc



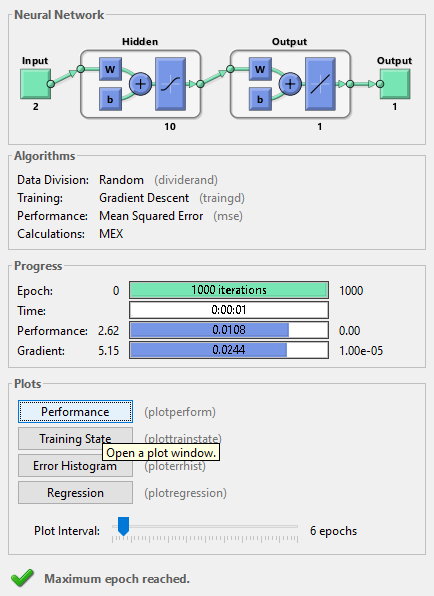
Кривая обучения

Результат: MSE = 0.0193

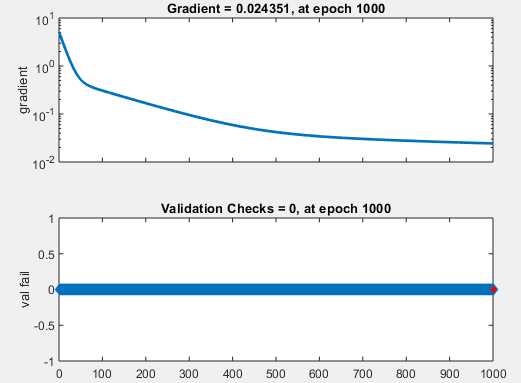
Рассмотрим групповой вариант алгоритма GD

Код

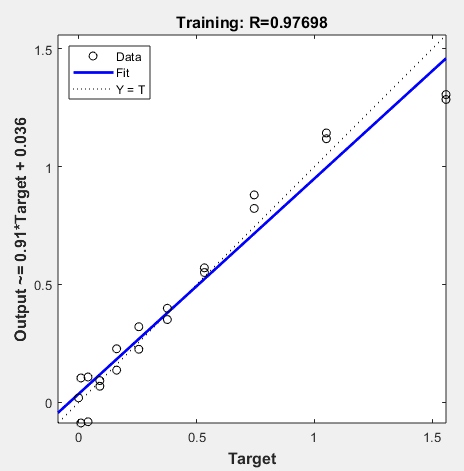
net = feedforwardnet(10, 'traingd');  
net.trainParam.show = 50;  
net.trainParam.lr = 0.05;   
net.trainParam.goal = 1e-005;  
tic, net=train(net,p,t); toc



Процесс обучения



Процесс обучения

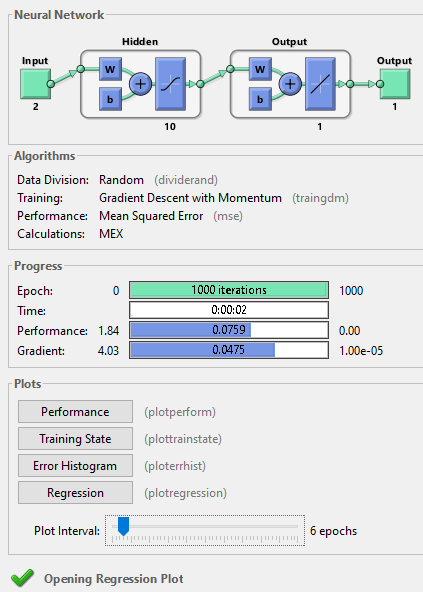


Регрессия

Результат Mse = 0.0058.

Рассмотрим алгоритм градиентного спуска с возмущением GDM:

net = feedforwardnet(10, 'traingdm');  
net.biases{1,1}.learnFcn='learngdm';  
net.biases{2,1}.learnFcn='learngdm';   
net.layerWeights{2,1}.learnFcn='learngdm';   
net.inputWeights{1,1}.learnFcn='learngdm';  
net.layerWeights{2,1}.learnParam.lr=0.2;  
net.adaptParam.passes=300;  
tic, net=train(net,p,t); toc



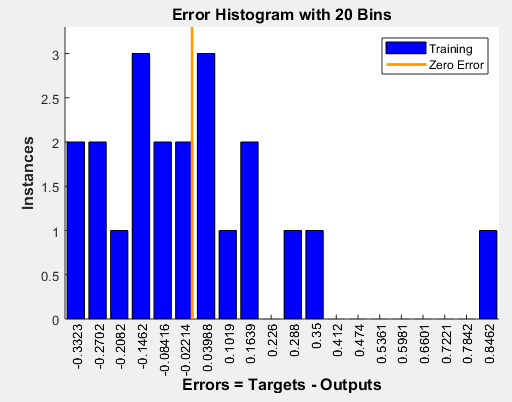
Процесс обучения



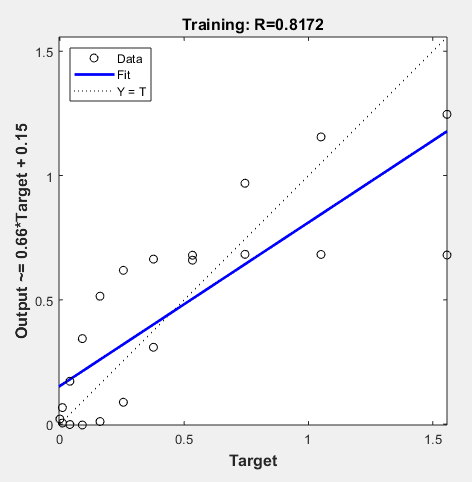
Значение градиента



Процесс обучения



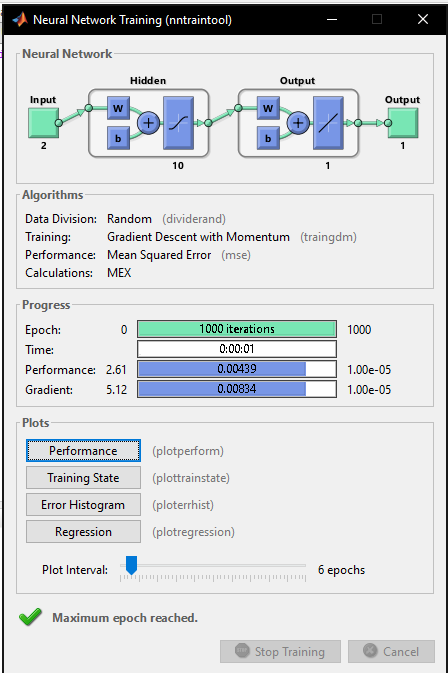
Гистограмма ошибок



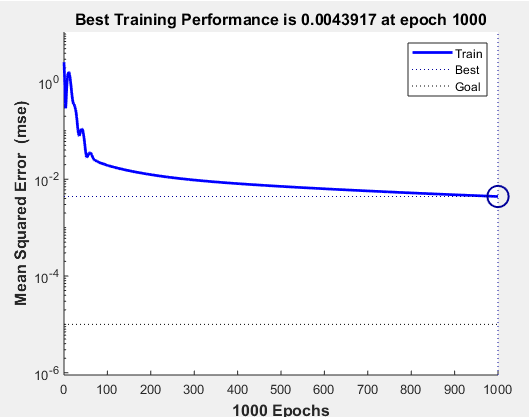
Регрессия

MSE = 0.02.

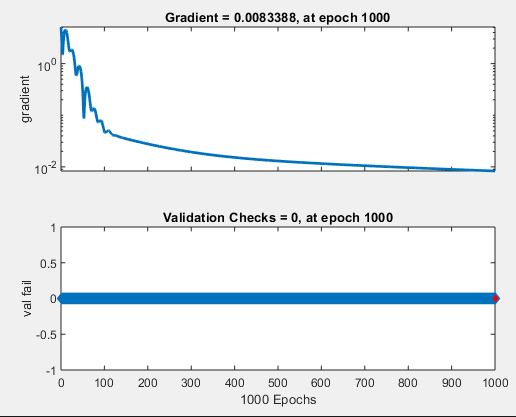
Рассмотрим вариант алгоритма GDM с групповым обучением



Процесс обучения



Процесс обучения



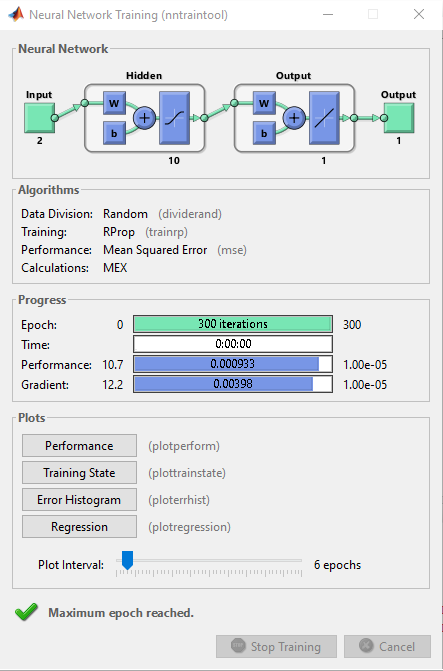
Градиент



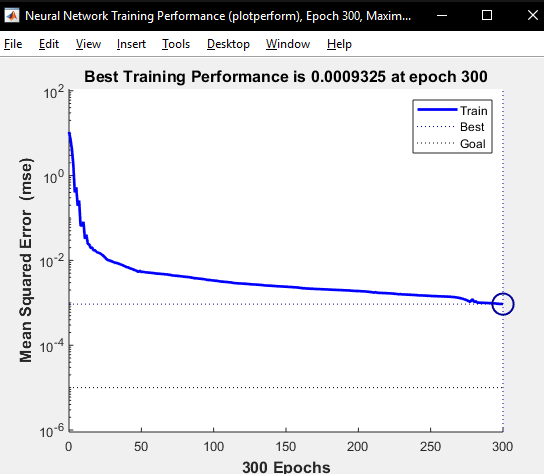
Результат регрессии

MSE = 0.0057

Рассмотрим алгоритм градиентного спуска с выбором параметра скорости настройки:



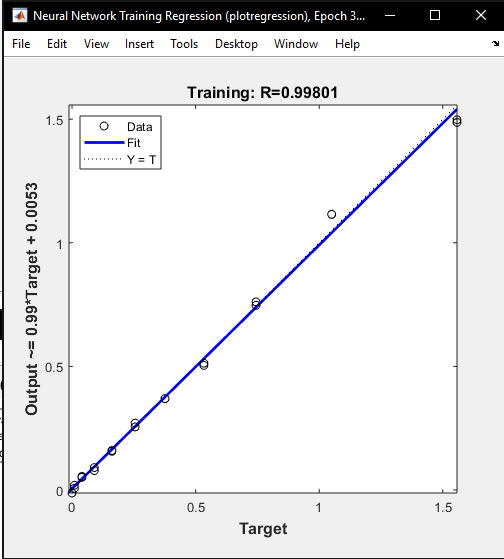
Процесс обучения



Процесс обучения



Градиент



Результат регрессии

MSE = 0.001

Наилучший алгоритм GDA.

Вопросы:

В: Каким алгоритмом обучают 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, изменяя 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:** Изучил 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 2)**

Построить модель нейронной сечи, аппроксимирующей полином

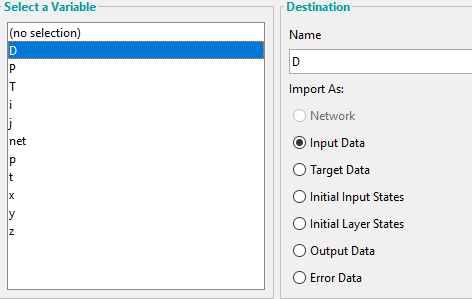


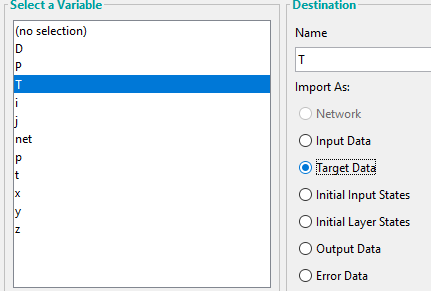
На интервале [-1,1]

Коэффициенты полинома [-1, 3, -1, 2, -1, -1]

Сгенерируем входные и выходные данные в рабочее окружение.

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 + 3\*j\*j - 1\*i\*j +2\*i - 1\*j - 1  
x = x+1;  
end;  
end;

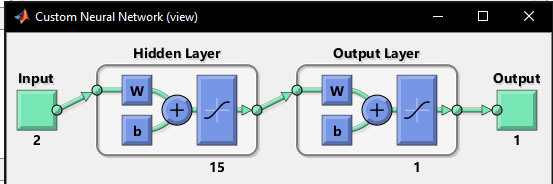
****

****

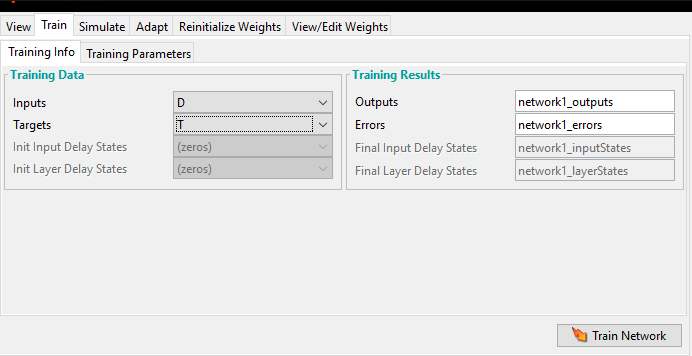
Импортируем переменные в nntool

Создадим нейронную сеть с двумя слоями:

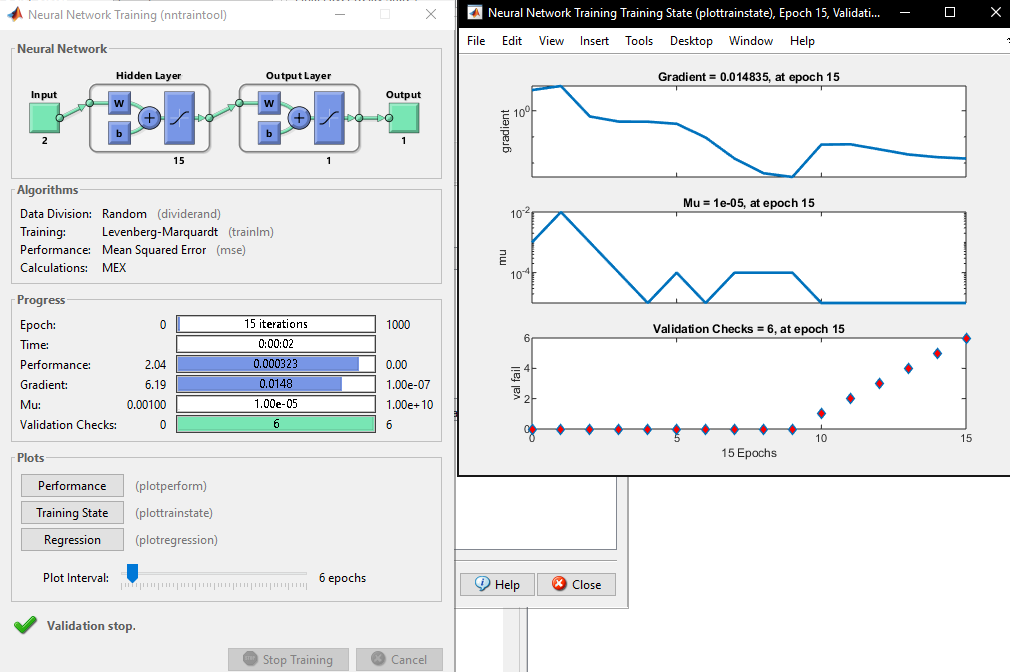
1. 15 нелинейных нейронов
2. 1 с линейной функцией активации



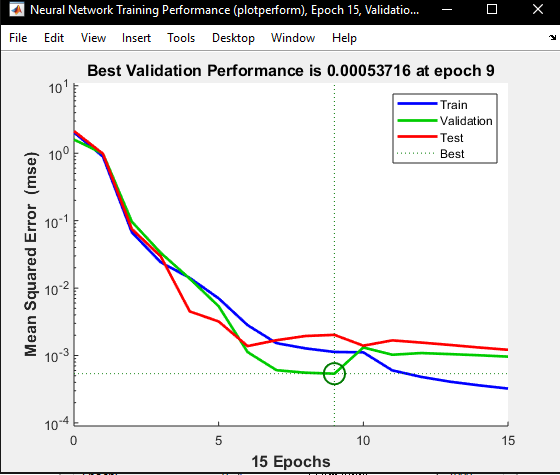
Топология сети



Начнём обучение

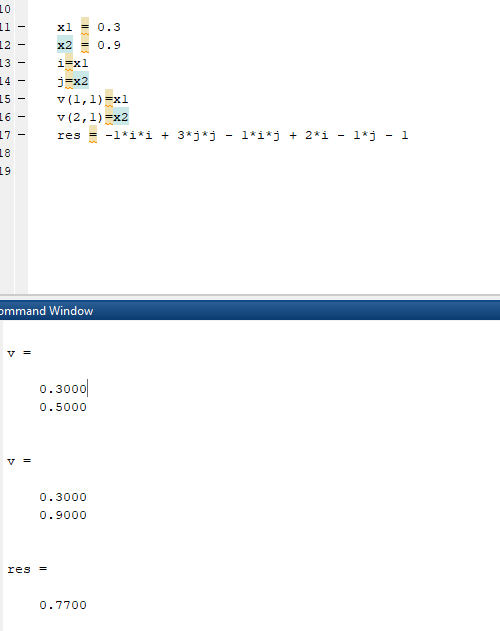


Процесс обучения



Производительность сети. Найден минимум

Проверим модель



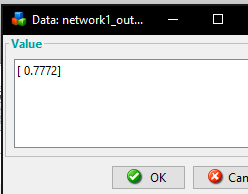
Возьмём

х1=0.3

x2=0.9

Правильный ответ 0.77

Ответ модели



**Задание 2(Вариант 2)**

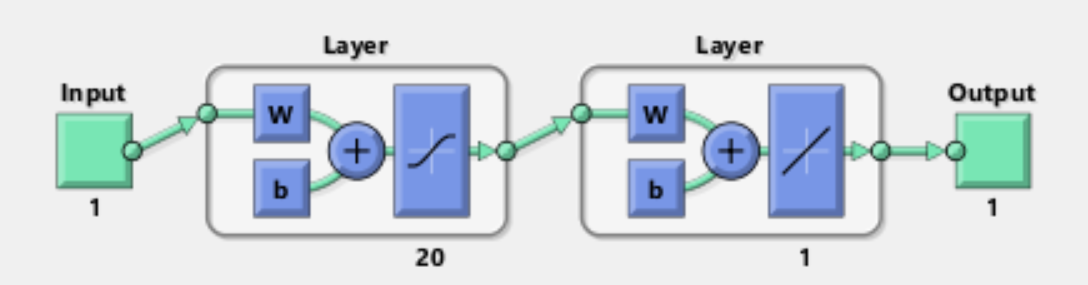
Построить модель нейронной сети для аппроксимации функции f(t) на промежутке [0,10]:

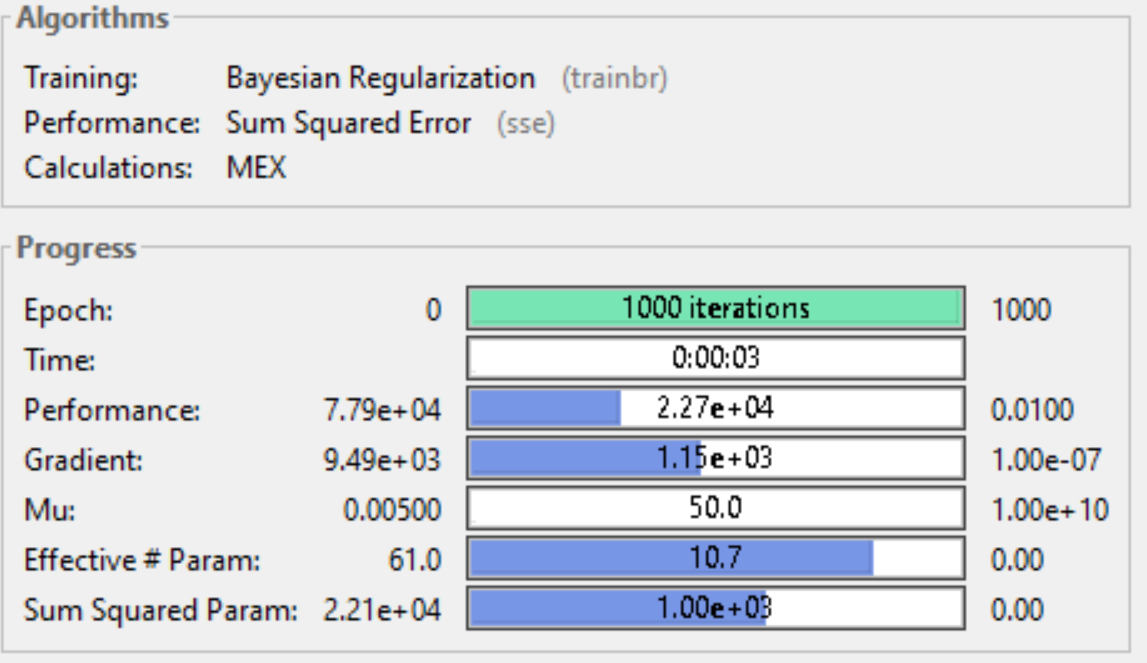
****

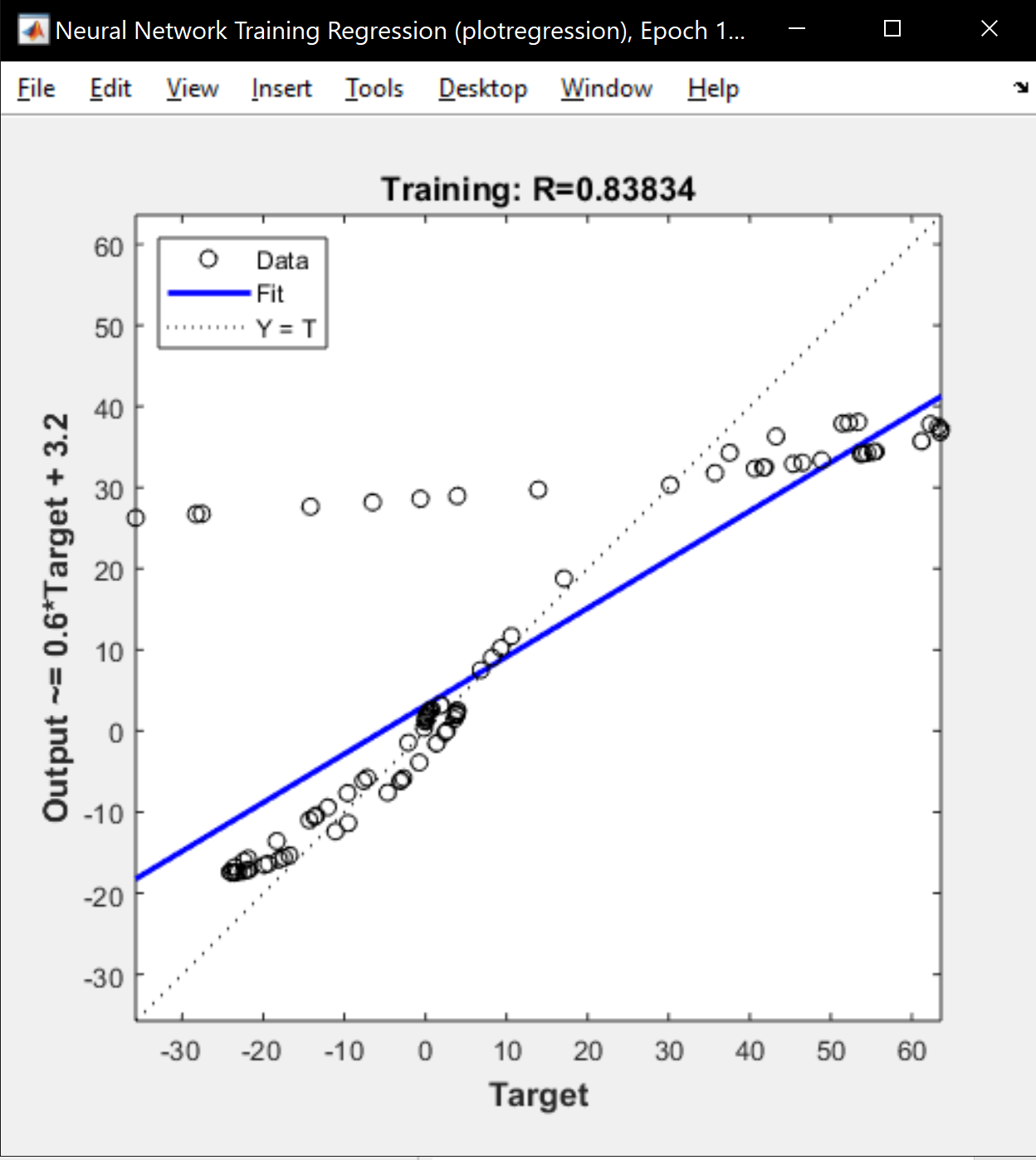
**Генерируем данные**

a = 0;  
b = 10;  
for i = 1:100  
 P(i) = (b-a).\*rand(1,1) + a;  
 T(i) = P(i)\*P(i)\*sin(P(i));  
end;

**Построим сеть**

****

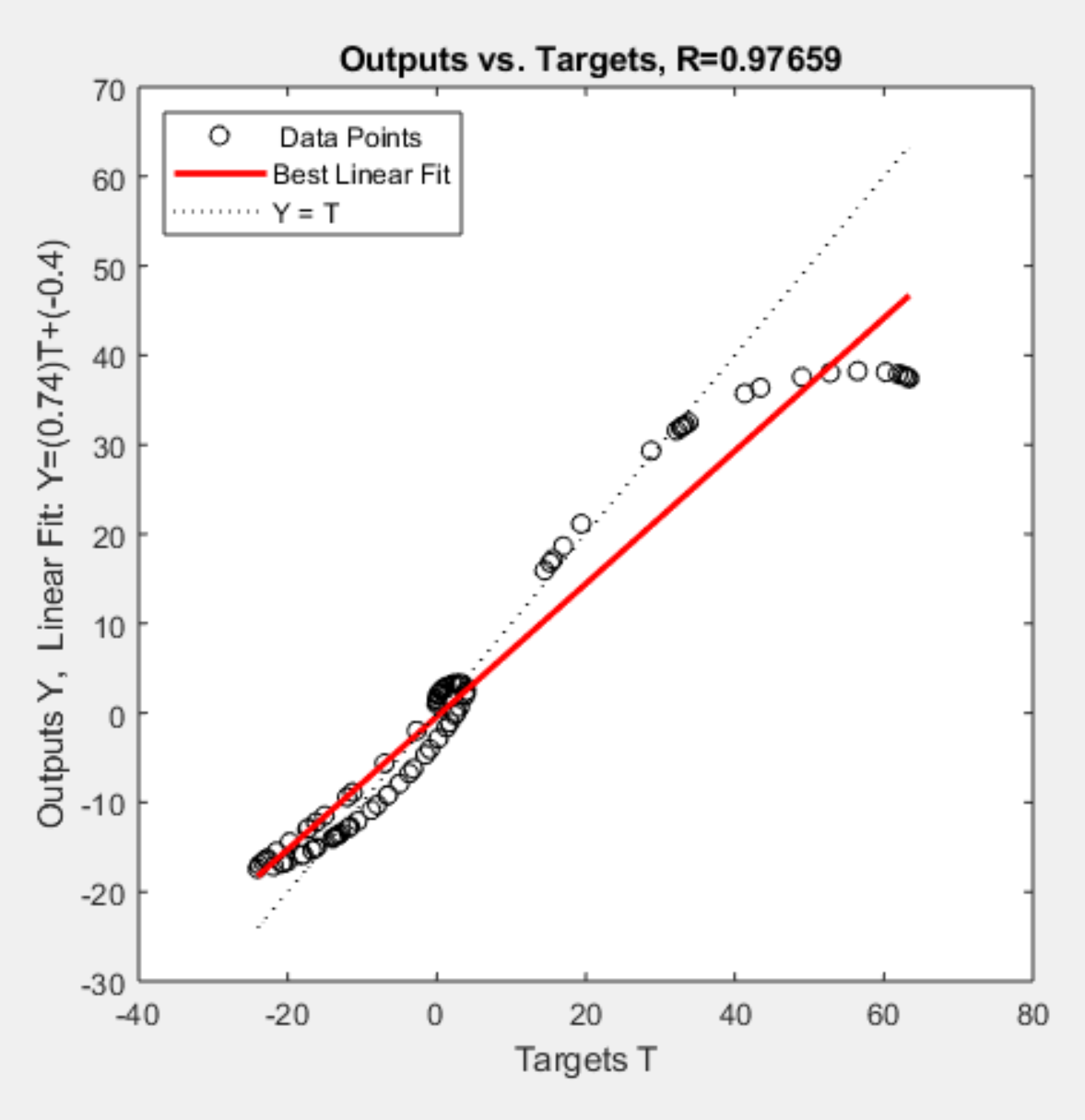
****

****

Регрессия

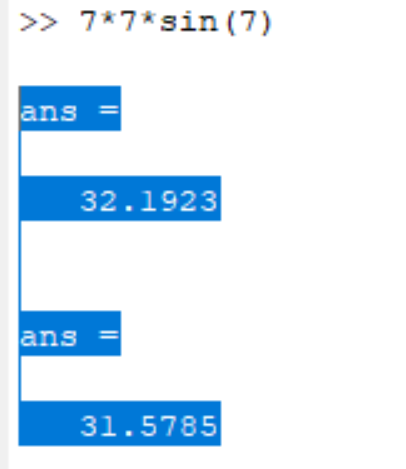
Сгенерируем данные для проверки

for i = 1:100  
P(i) = (b-a).\*rand(1,1) + a;  
T(i) = P(i)\*P(i)\*sin(P(i));  
end;  
  
y = sim(net, P);  
[m,b,r] = postreg(y(1,:),T(1,:));



Сверим данные

Сверим данные с конкретным примером



Первое значение – реальная функция

Второе значение – предсказанные данные

**Вывод:** Получил практические навыки использования nntool на примерах для аппроксимации функции

**Лабораторная работа №5**

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

**Ход работы:**

Сгенерируем данные для входа и выхода

Код

[alphabet, targets] = prprob;  
i = 10;  
ti = alphabet(:, i);  
letter{i} = reshape(ti, 5, 7);  
letter{i}  
  
[R,Q] = size(alphabet);  
[S2,Q] = size(targets);

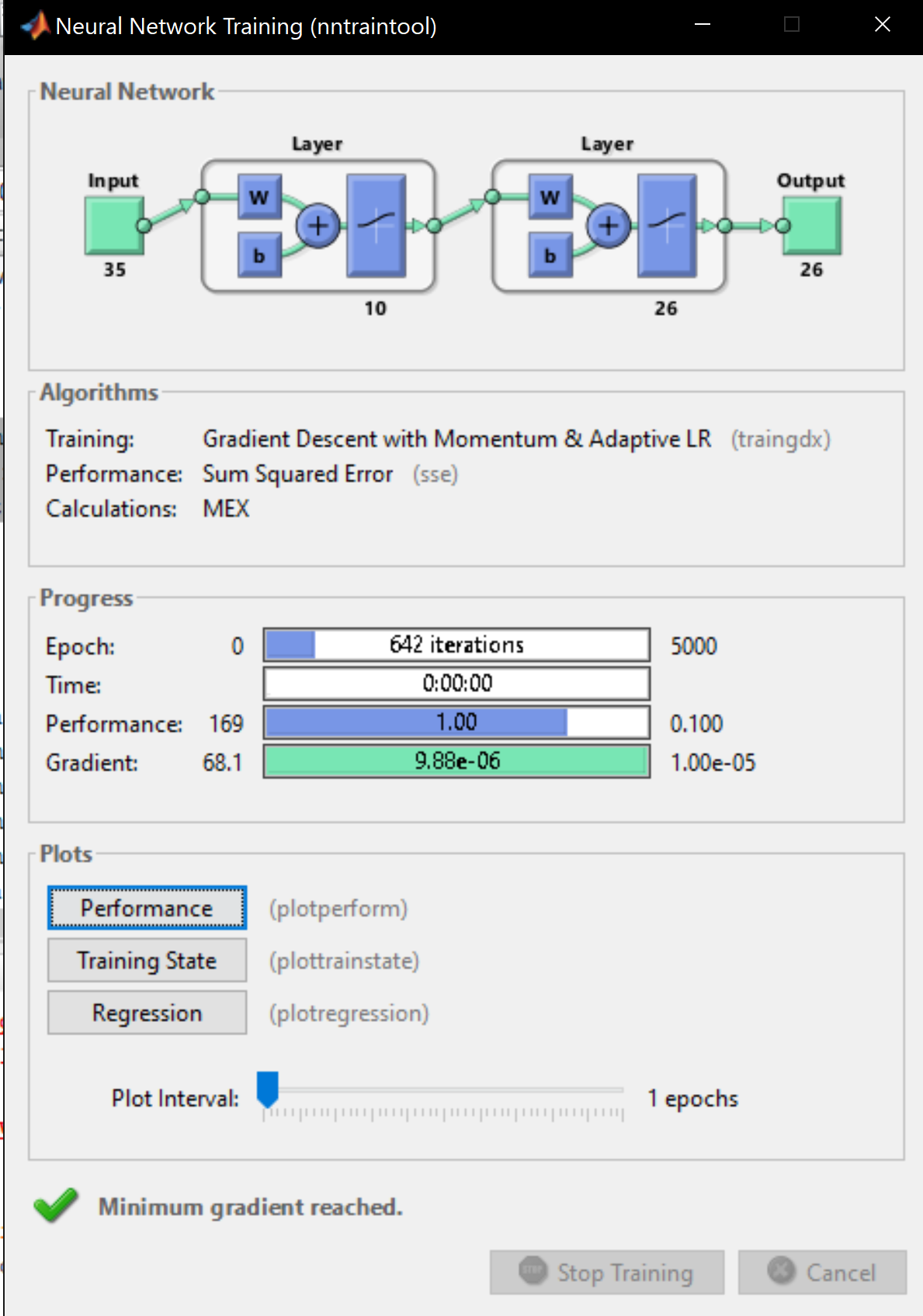
Подготовим нейронную сеть для с функцией обучения traingdx (2 слоя)

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;

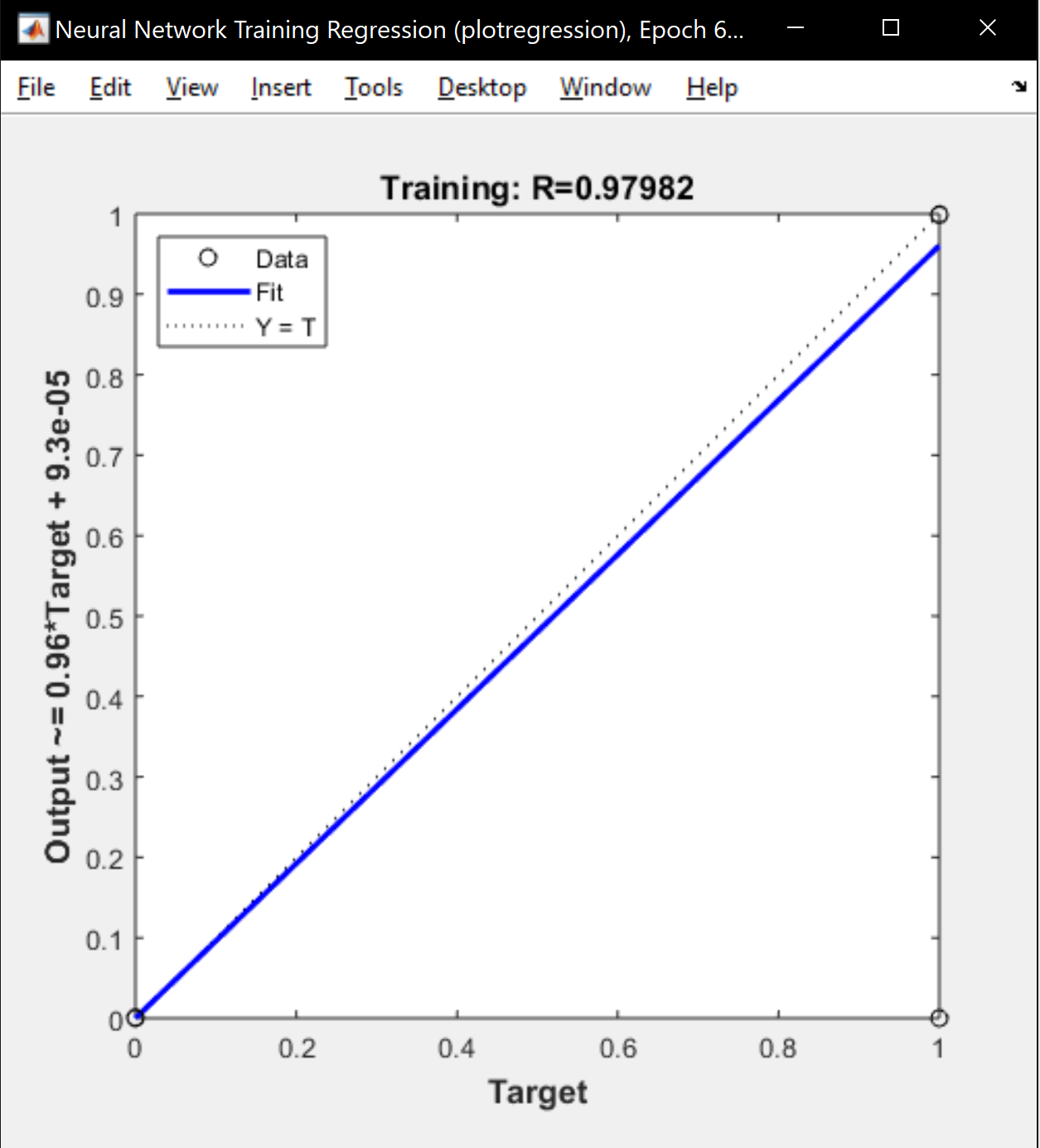
Обучим сеть на чистых данных

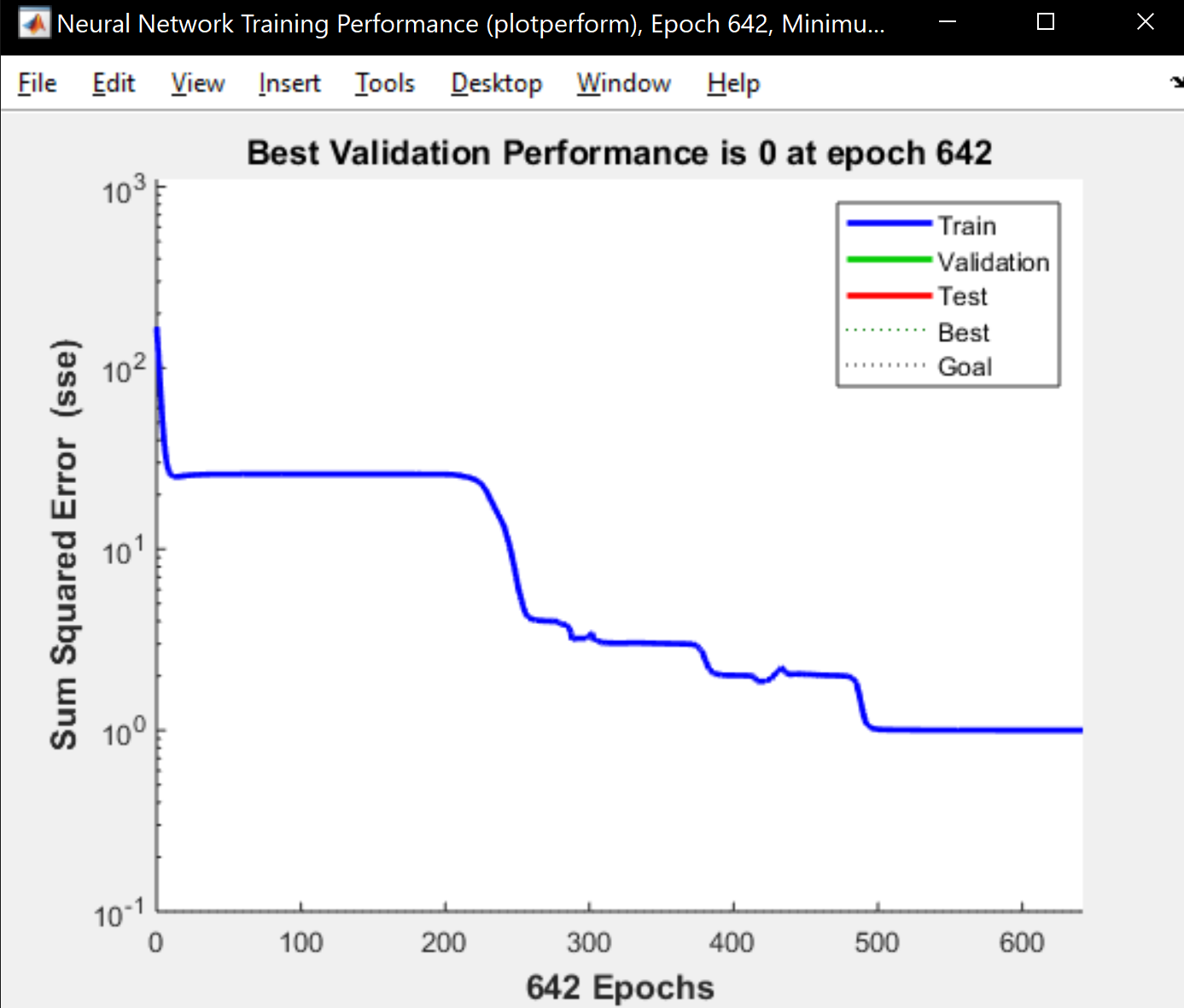
P = alphabet;  
T = targets;  
  
net.performFcn = 'sse'; % Sum-Squared Error performance function  
net.trainParam.goal = 0.1; % Sum-squared error goal.  
net.trainParam.show = 20; % Frequency of progress displays (in epochs).  
net.trainParam.epochs = 5000; % Maximum number of epochs to train.  
net.trainParam.mc = 0.95;  
[net,tr] = train(net, P, T);

Процесс обучения



Регрессия





Обучение



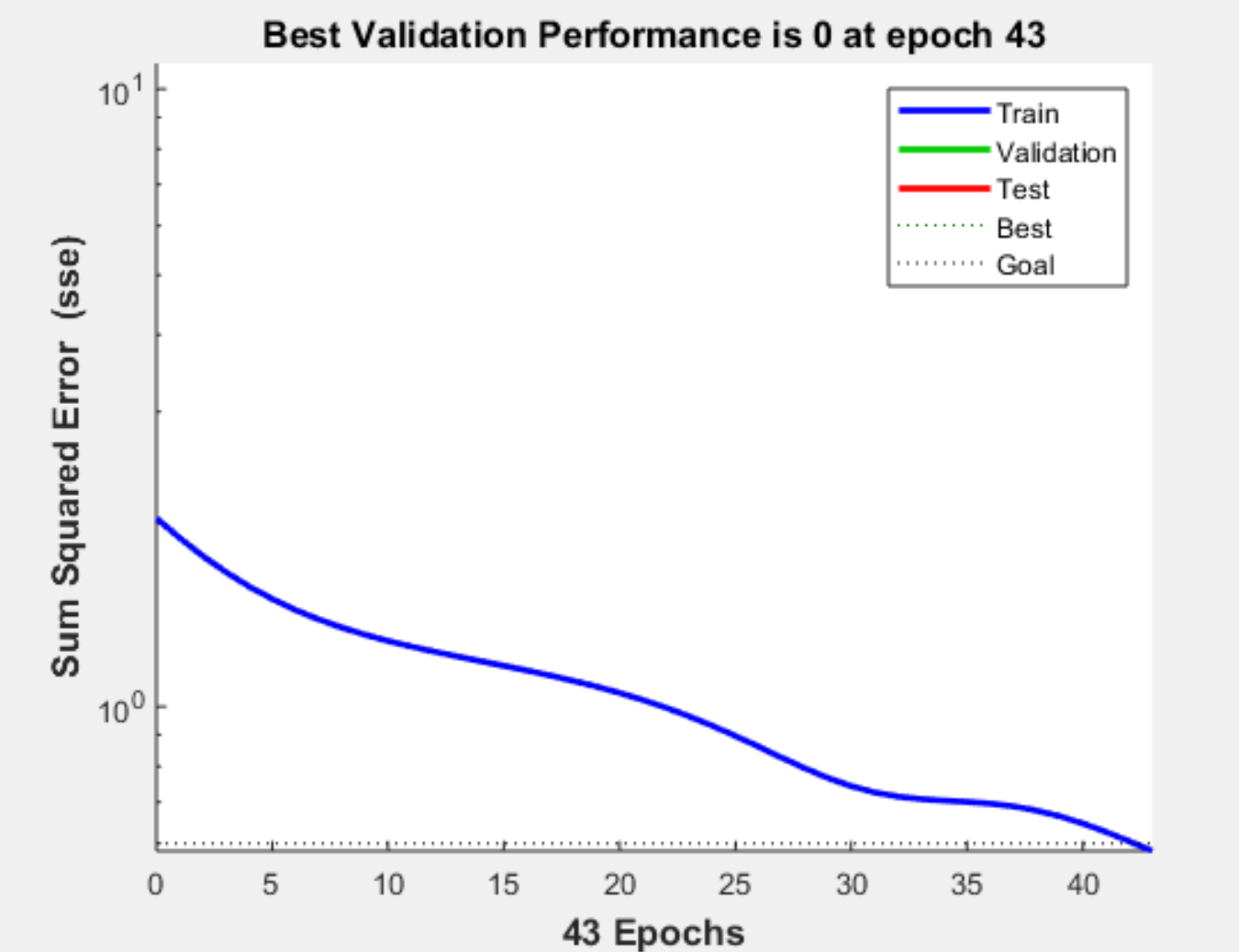
Состояние обучения

Обучим сеть на двух алфавитах. С шумом и без. Обучаем последовательно 10 раз

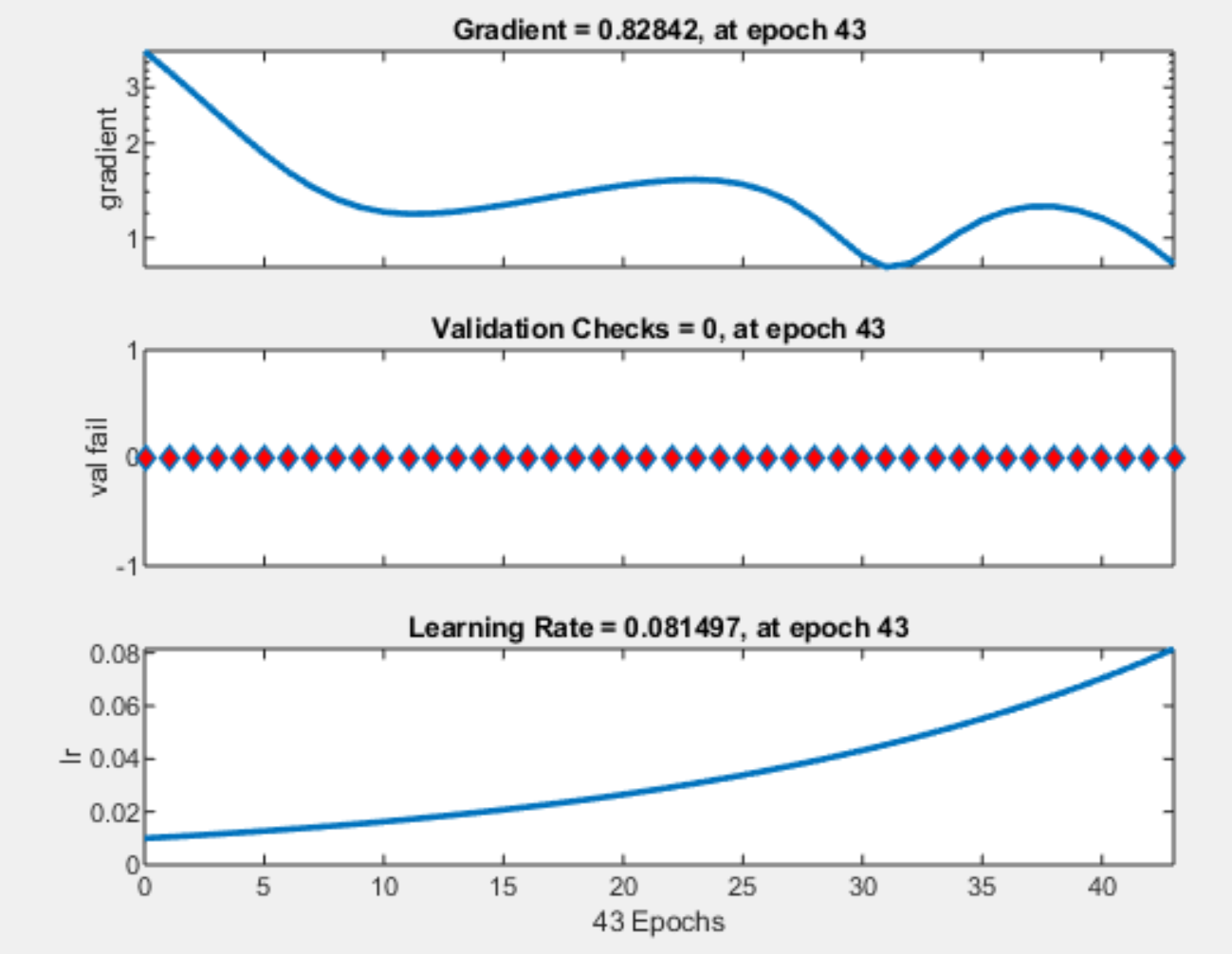
netn.trainParam.goal = 0.6;   
netn.trainParam.epochs = 300;   
  
T = [targets targets targets targets];  
for pass = 1:10  
P = [alphabet, alphabet, ...  
(alphabet + randn(R,Q)\*0.1), (alphabet + randn(R,Q)\*0.2)];  
[netn,tr] = train(netn,P,T);  
End



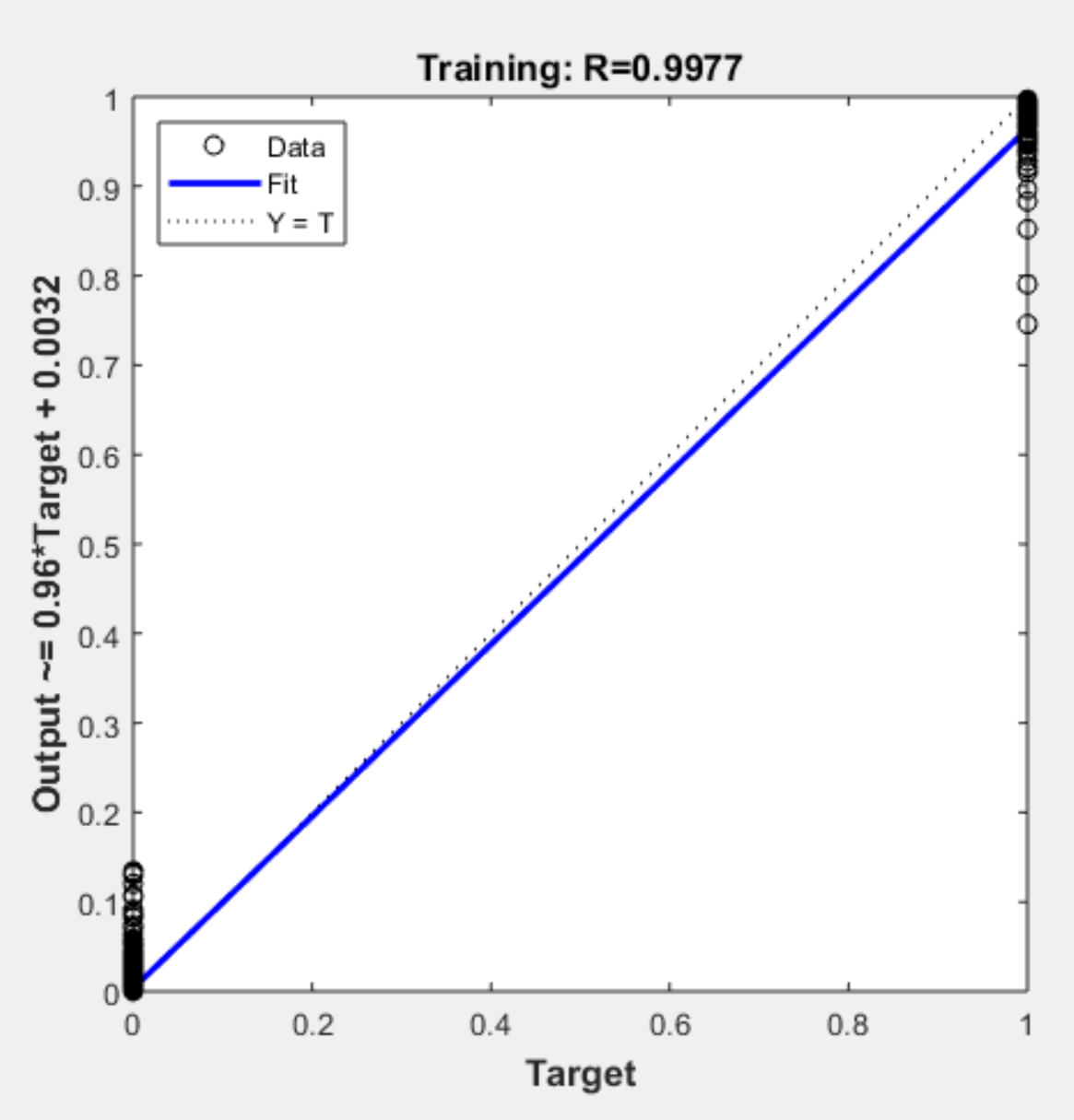
Процесс последнего обучения



Понижение ошибки



Прогресс обучения



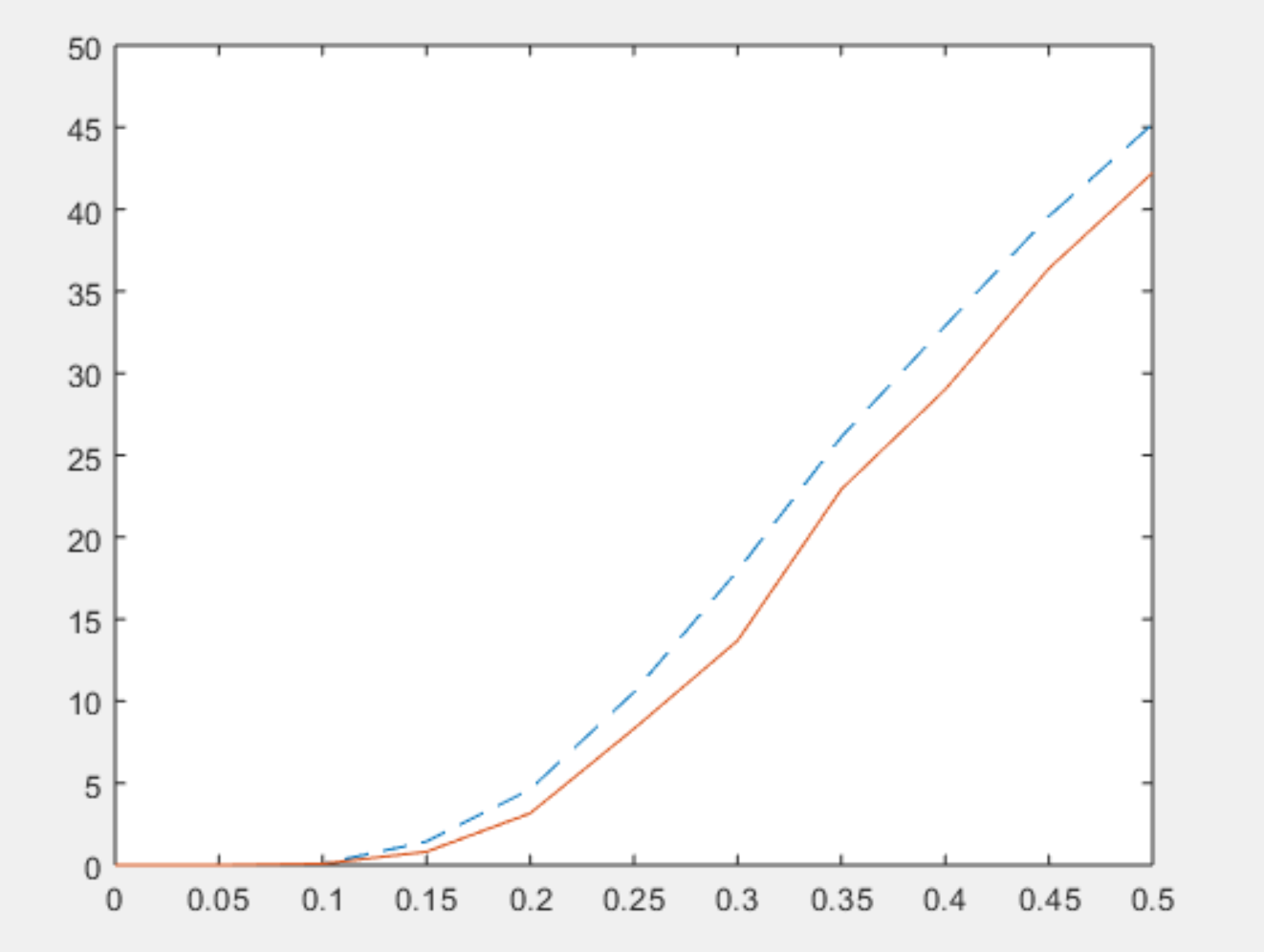
Регрессия

Проверим эффективность функционирования сети:

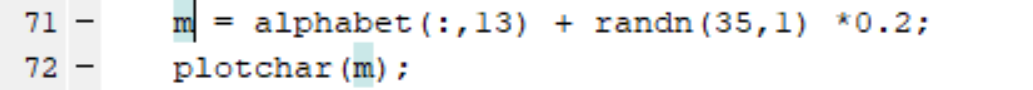
noise\_range = 0:.05:.5;  
max\_test = 100;  
network1 = [];  
network2 = [];  
T = targets;  
for noiselevel = noise\_range  
fprintf('Уровень шума %.2f.\n',noiselevel);  
errors1 = 0;  
errors2 = 0;  
for i = 1:max\_test  
P = alphabet + randn(35,26)\*noiselevel;  
% Тест для сети 1  
A = sim(net,P);  
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]; % среднее значения ошибок (100тпоследовательностей из 26 векторов)  
network2 = [network2 errors2/26/100]; % среднее значения ошибок (100 последовательностей из 26 векторов)  
end  
  
plot(noise\_range,network1\*100,'--',noise\_range,network2\*100);

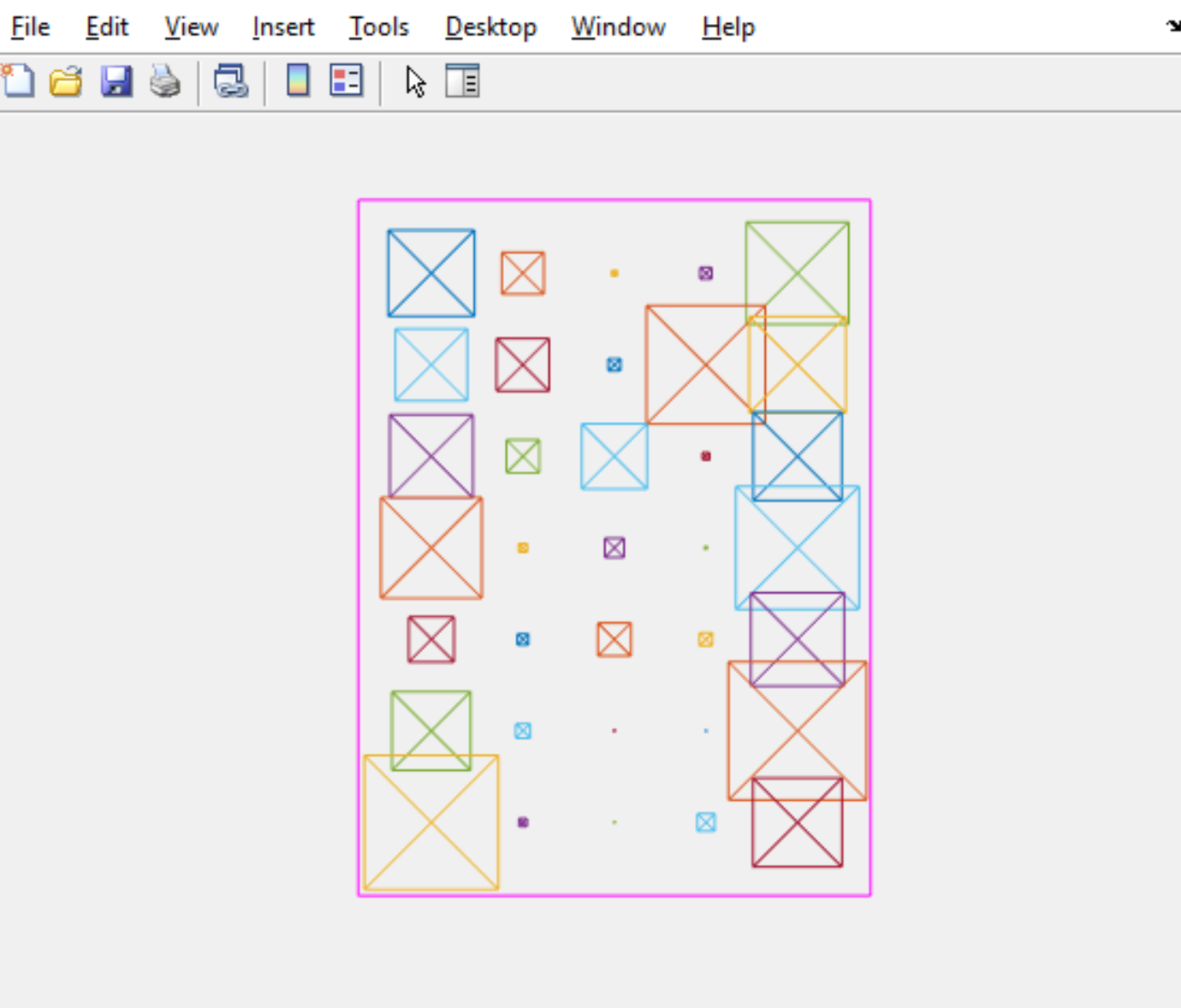
Код

Выведем сравнительный график для подсчёта количества ошибок от уровня шума



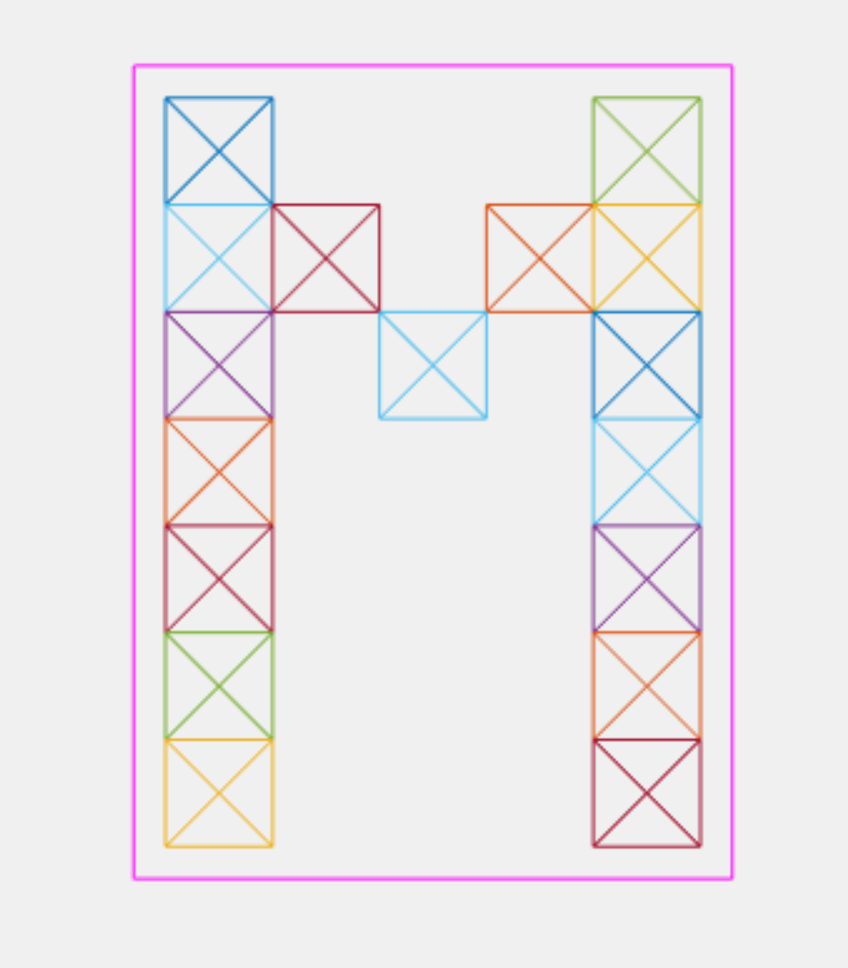
Выведем букву M





Попробуем угадать букву моделью

A2 = sim(net, noisyM);  
A2 = compet(A2);  
answer = find(compet(A2) == 1)  
plotchar(alphabet (:,answer));



Модель угадала

**Вывод:** Построил и обучил нейронную сеть для классификации символов. Получил навыки использования нейронных сетей в практических задачах

**Лабораторная работа №6**

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

**Задача 1 (Вариант 2)**

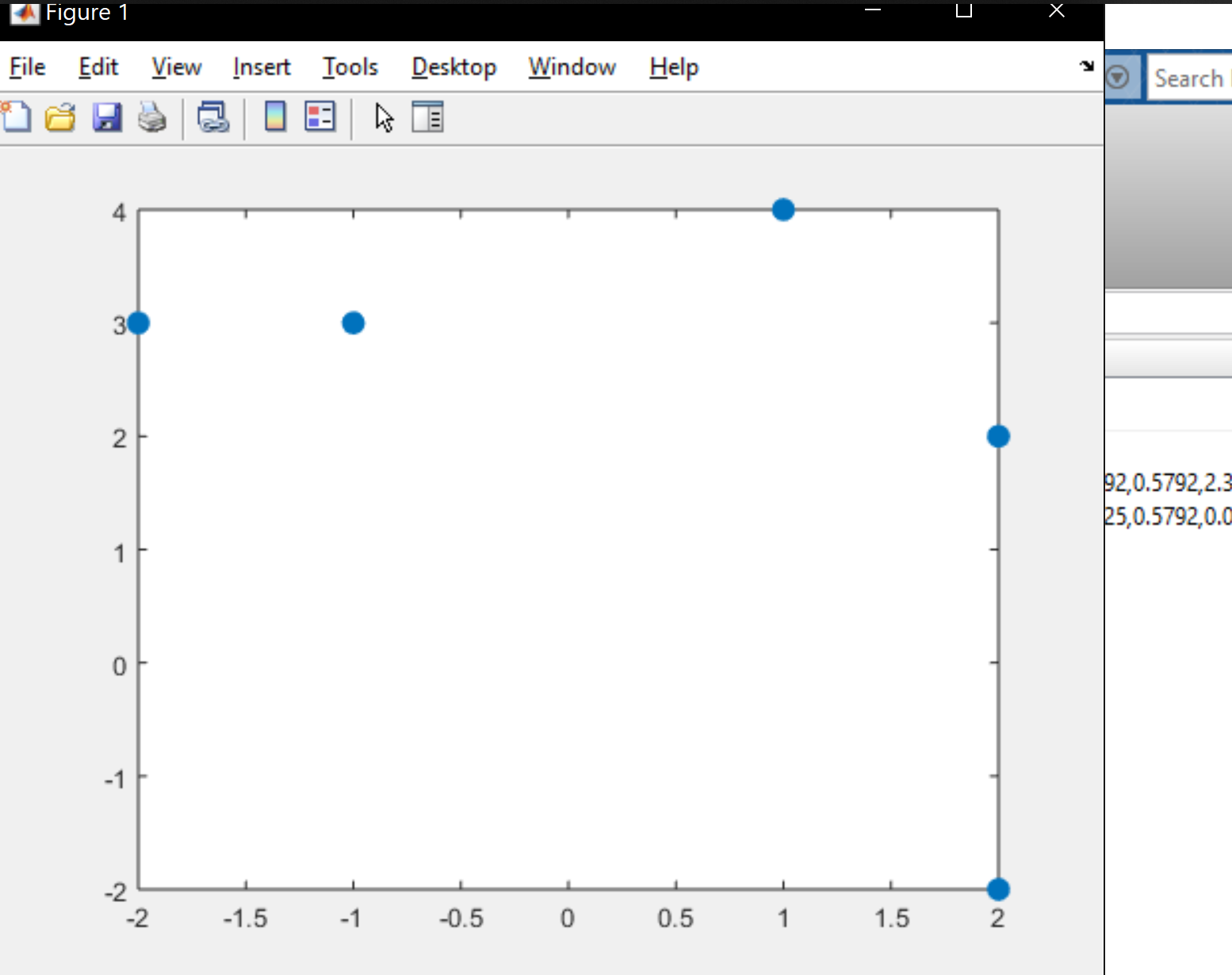
Создание и исследование моделей сетей GRNN: подготовить массивы входных векторов P и целей T для нейронной сети GRNN, создать и обучить сеть GRNN для аппроксимации функции.

Значения вектора входа [-1 -2 2 2 1]

Значение целевого вектора [3 3 2 -2 4]

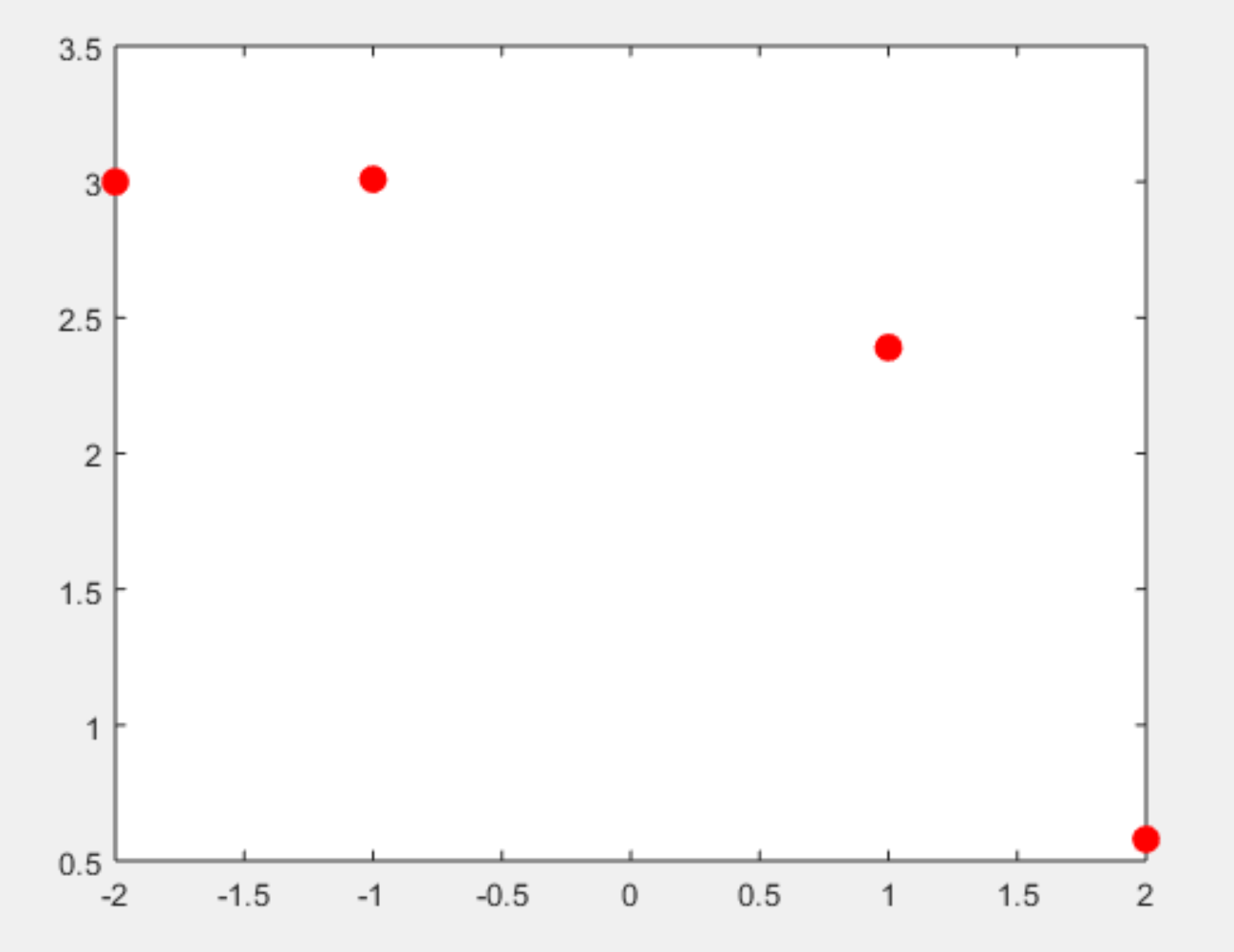
Создадим сеть и изобразим последовательности входов и выходов на графике

spread = 0.8; % установка разброса радиальных базисных функций  
net = newgrnn(P, T, spread); % создание НС регрессии  
A = sim (net, P); % имитация работы НС регрессии  
plot(P, T, '.', 'markersize', 30); % изображение аппроксимируемой функции

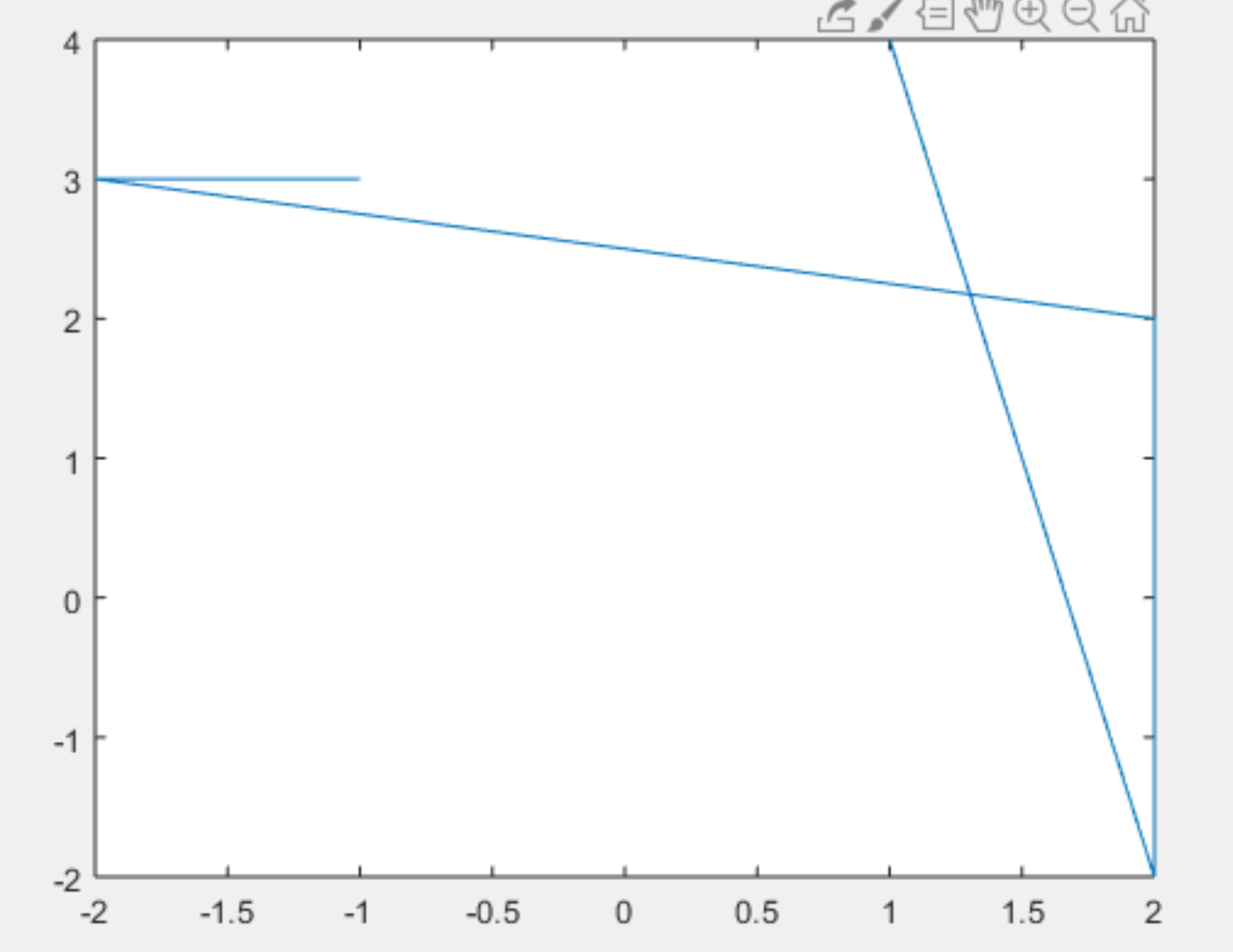


изображение аппроксимируемой функции

plot(P, A, '.', 'markersize', 30, 'color', [1 0 0]); % изображение работы необученной НС регрессии

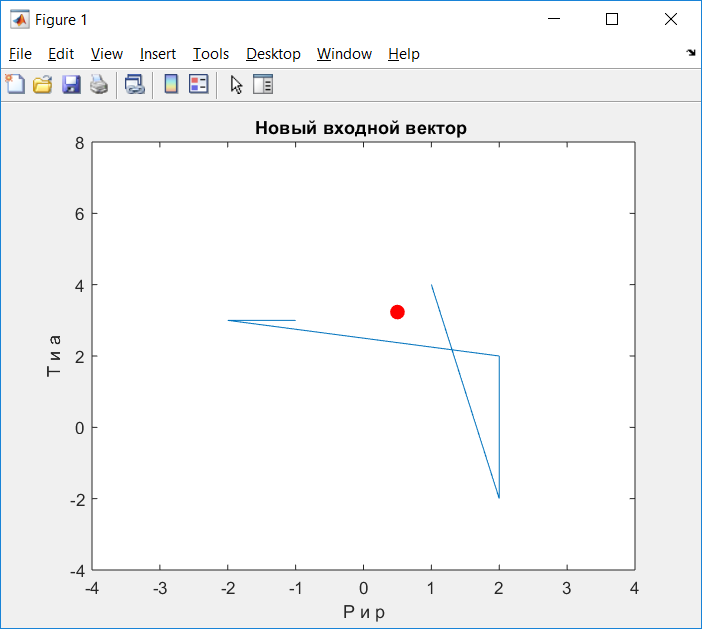


cla reset; % очистка координатных осей  
p = 0.5; % установка нового входа НС регрессии  
a = sim(net, p); % получение отклика НС регрессии  
plot(P, T, 'markersize', 30); % изображение аппроксимируемой функции

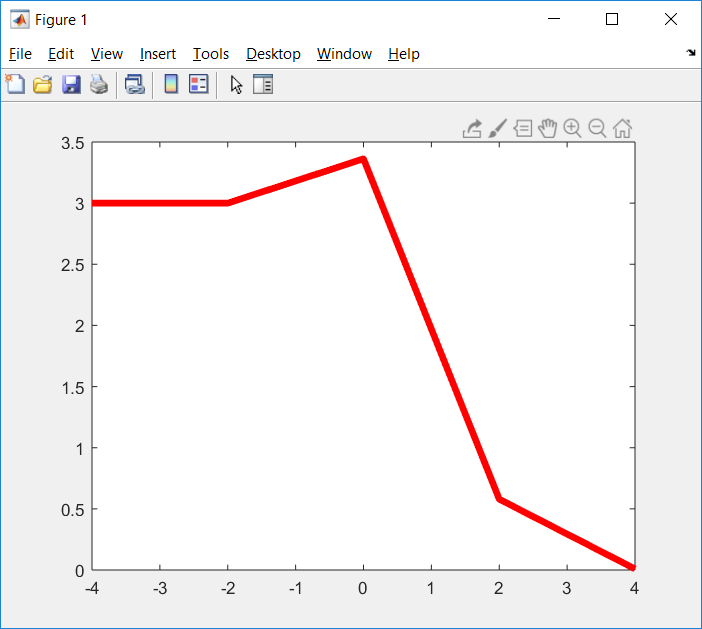


axis([-4 4 -4 8]); % установка диапазонов осей X и Y  
hold on; % включение режима добавления графиков  
plot(p, a, '.', 'markersize', 30, 'color', [1 0 0]); % изображение отклика НС регрессии на вход р  
title( 'Новый входной вектор '); % написать заголовок графика  
xlabel('P и р'); % пометить ось X  
ylabel('T и а'); % пометить ось Y  
cla reset; % очистить координатную сетку

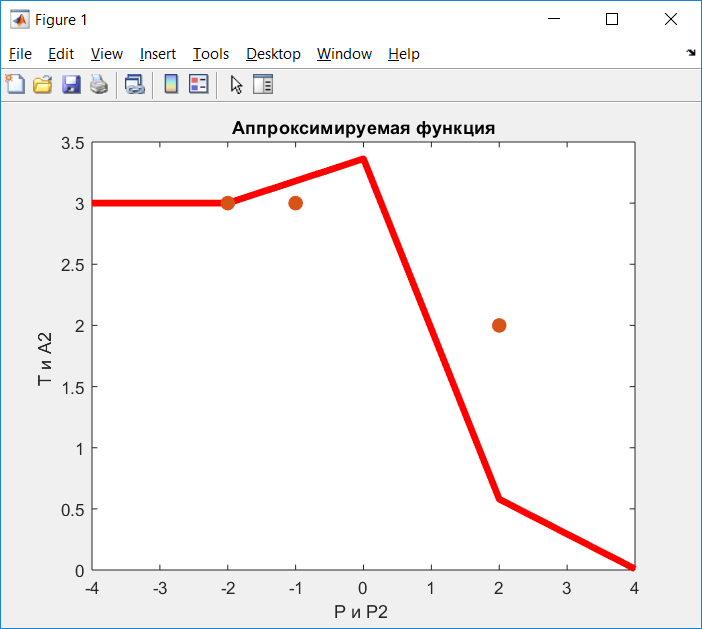
Изображения функции и новой точки



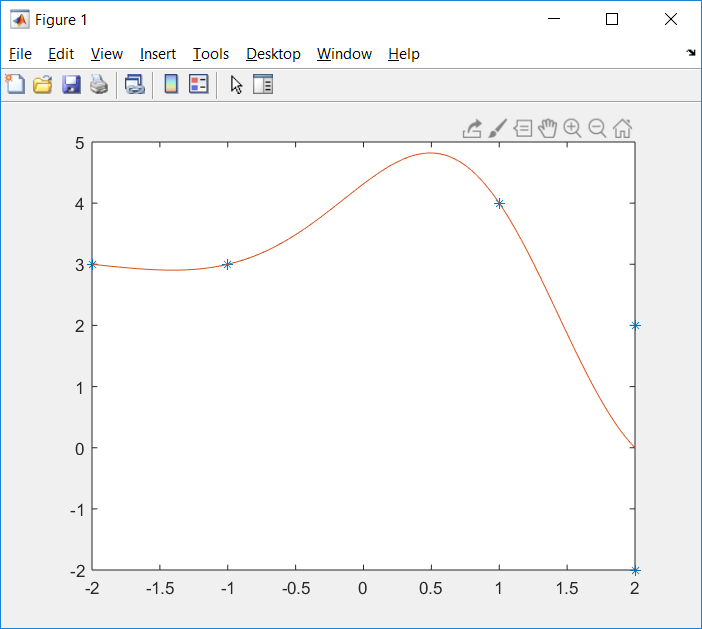
P2 = -4:2:4; % определить последовательность входов Р2  
A2 = sim(net, P2); % получить отклик НС регрессии на последовательность входов Р2  
plot(P2, A2, 'linewidth', 4, 'color', [1 0 0]); % отклик НС регрессии



hold on; % включить режим добавления графиков  
plot(P, T, '.', 'markersize', 30); % изобразить аппроксимируемую функцию  
axis([-4 4 0 3.5]); % установить диапазон осей  
title('Аппроксимируемая функция'); % озаглавить график  
xlabel('P и Р2 '); % пометить ось X  
ylabel('T и А2 '); % пометить ось Y

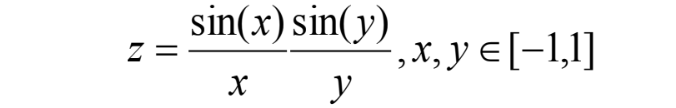


cla reset;  
plot(P, T, '\*');  
e = 0,02; % целевой среднеквадратичной ошибки  
sp = 1; % величина разброса радиальной базисной нейронной сети  
net = newrb(P, T, e, sp); % создание радиальной базисной сети  
X = -2 : .01 : 2;  
Y = sim (net, X); % формирование отклика Y  
hold on; % включение режима добавления графика  
plot(X, Y); % изображение результатов аппроксимации  
hold off; % отключение режима добавления графика



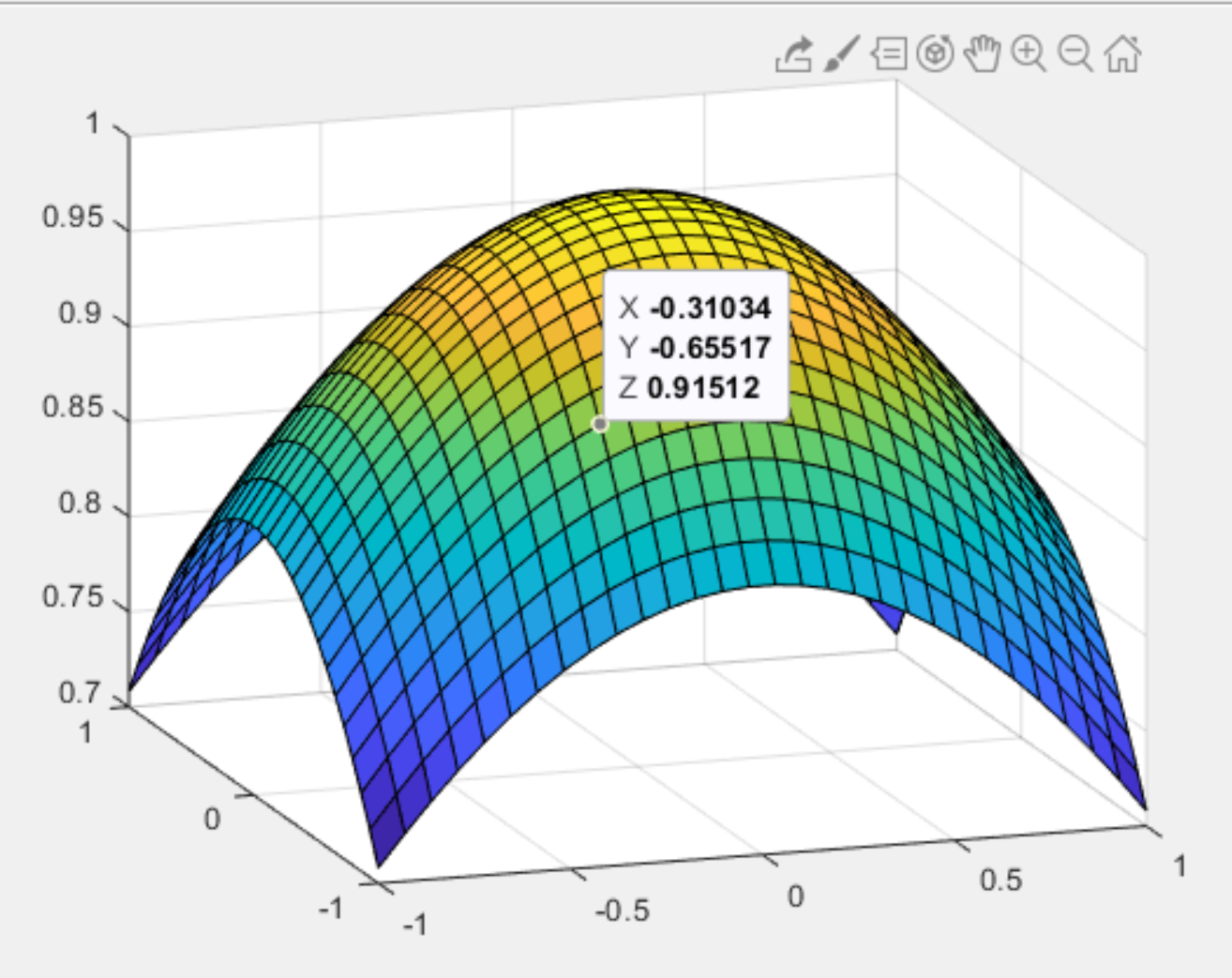
**Задача 2**

Построить модель радиальной базисной сети, аппроксимирующей функцию:



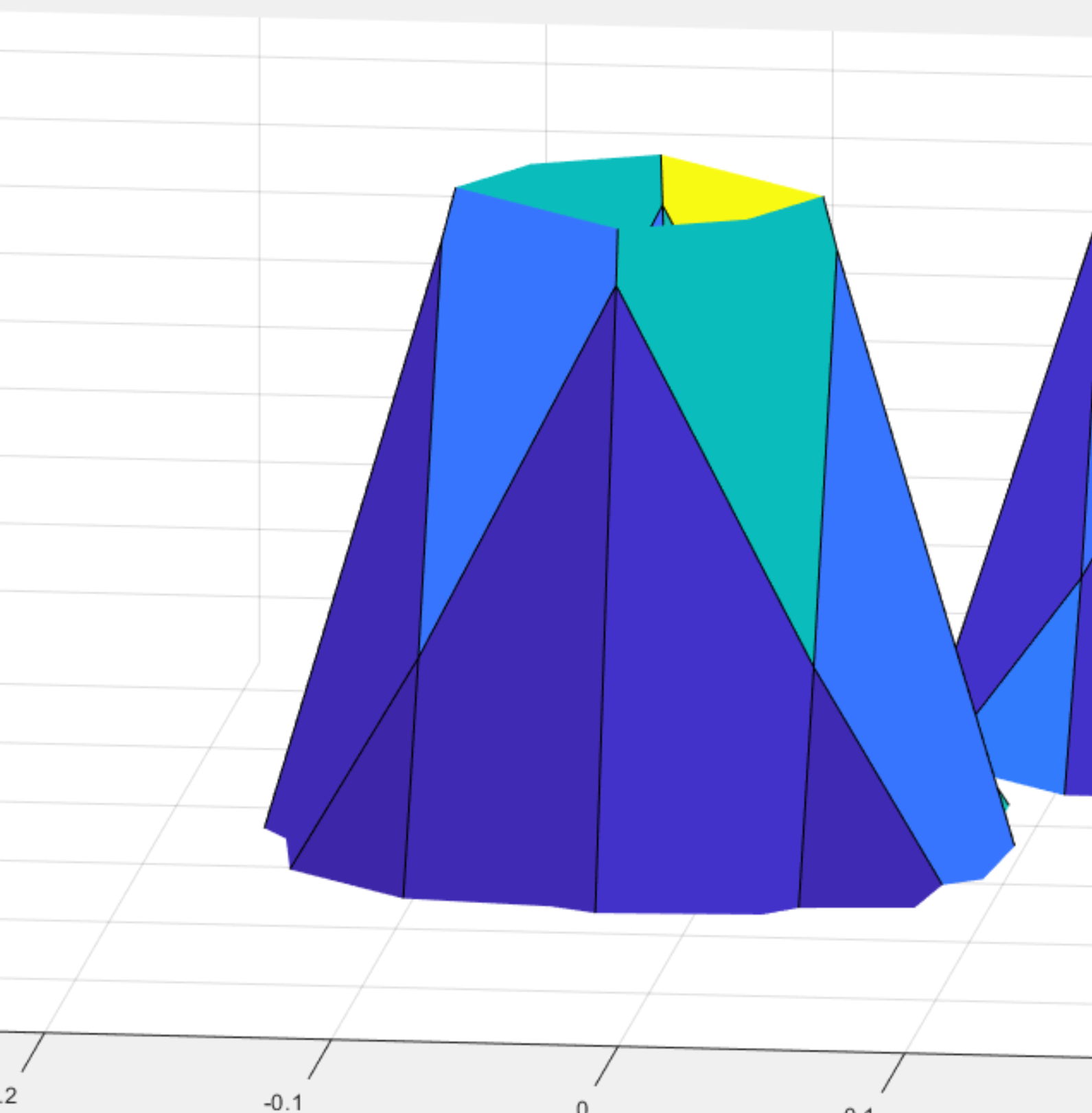
Построим искомую функцию

x1 = -1.0; x2 = +1.0; y1 = -1.0; y2 = +1.0;  
nx = 30; ny =30 ;  
  
step\_x = (x2-x1)/(nx-1); step\_y = (y2-y1)/(ny-1);  
step\_min = min(step\_x,step\_y);  
[x,y] = meshgrid([x1:step\_x:x2], [y1:step\_y:y2]);  
z = (sin(x)./x).\*(sin(y)./y);  
surf(x,y,z)



Создаем сеть и апроксимируем функцию:

xx = reshape(x,1,nx\*ny);  
yy = reshape(y,1,nx\*ny);  
zz = (sin(xx)./xx).\*(sin(yy)./yy);  
p = [xx; yy];  
t = zz;  
goal = 0.05;  
spread = 1.0\*step\_min;  
net = newrb(p,t, goal,spread);  
net.layers{1}.size  
b = sim(net,p);  
[zz' b']  
c = reshape(b,ny,nx);  
surf(x,y,c)



Результат апроксимации

**Ход работы:**

**Ответы на вопросы:**

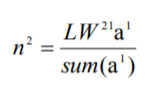
В: Какую функцию называют радиальной базисной функцией?

O: Это функция активации вида radbas(n) = exp(-n^2), где



В: Из каких слоев состоит радиально–базисная НС, НС регрессии, вероятностная НС?

O: Радиально-базисная НС состоит и двух слоев: скрытого радиального базисного слоя и выходного линейного слоя. НС регрессии состоит из тех же слоев, что и радиально-базисная НС с разницей в структуре второго слоя: в нем используется блок normprod для вычисления нормированного скалярного произведения строки массива весов LW21 и вектора входа a1 в соответствии со следующим соотношением (полученная n2 идет в pureline):



Вероятностная НС так же базируется на архитектуре радиально базисной сети с изменением во втором слое: второй слой является конкурирующим, подсчитывая вероятность принадлежности входного вектора к классу.

В: Какие виды НС предназначены для решения задачи аппроксимации функций, а какие – для классификации объектов?

О: Радиально-базисные и регнессионные сети предназначены для апроксимации функций, а вероятностные НС – для классификации объектов.

**Вывод:**

Изучил радиально-базисные, вероятностные нейронные сети, а также нейронные сети регрессии. Изучил их особенности и варианты применения.

**Лабораторная работа №7**

**Задача:** Исследование сети кохонена и алгоритма обучения без учителя

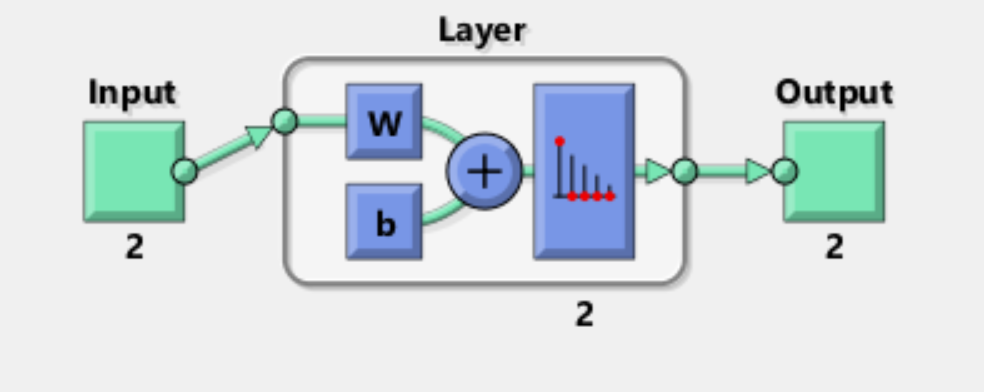
**Ход работы:**

**Задание 1**

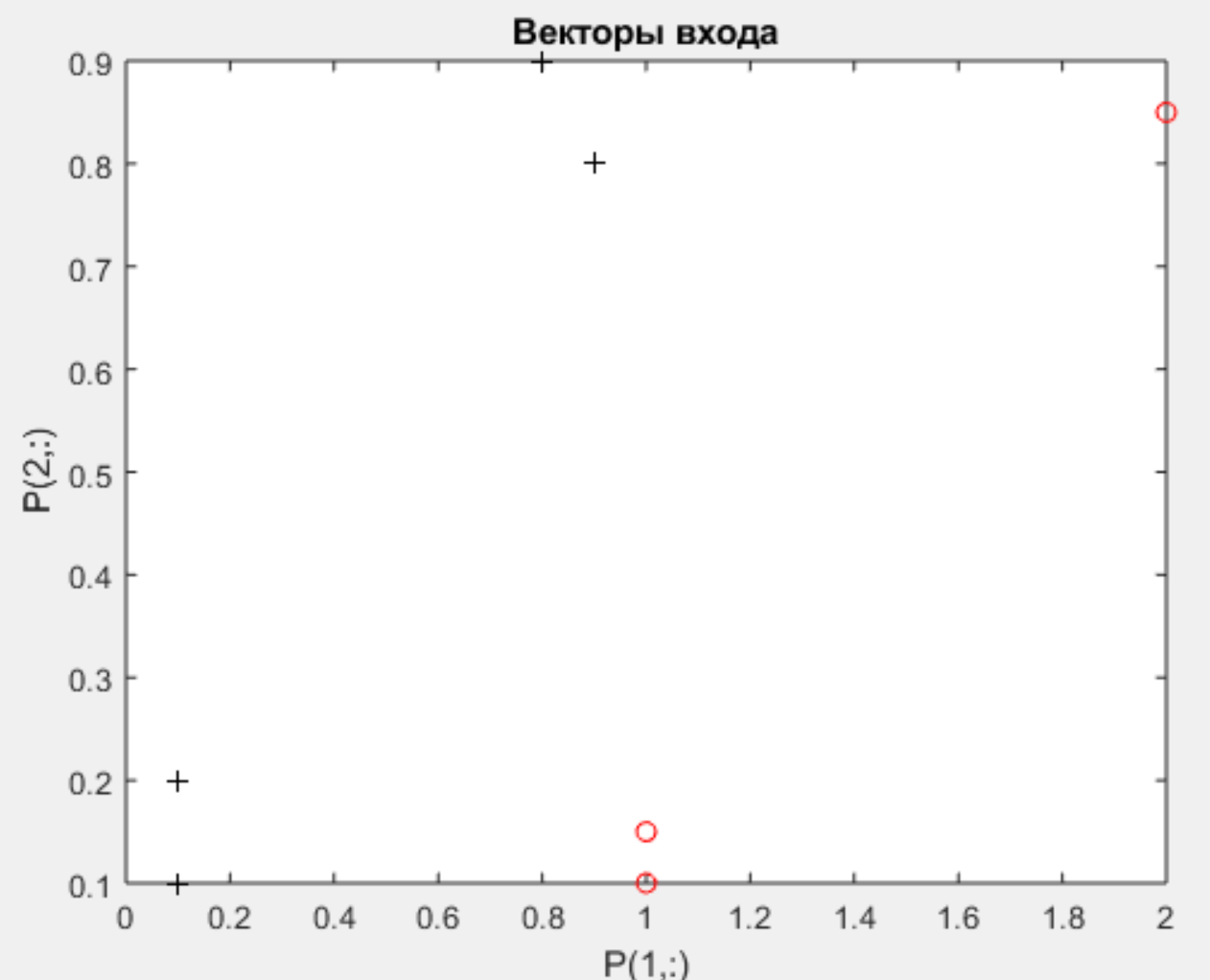
Создать слой Кохонена для двух векторов входа, проанализировать его структурную схему и параметры вычислительной модели, произвести обучение сети и моделирование

Создадим и обучим сеть

P = [.1 .8 .1 .9; .2 .9 .1 .8]; % для обучения слоя;  
net = newc([01;01],2); % создание слоя;  
gensim (net); % структура слоя;  
net = train(net,P); % обучение слоя;  
w = net.Iw{1,1}; % веса после обучения;  
b =net.b{1}; % смещение после обучения;  
plot(P(1,:),P(2,:),'+k')  
title ('Векторы входа'),xlabel('P(1,:)'), ylabel('P(2,:)')  
hold on  
plot (w, 'or')  
P1= [0.2:0.1:0.7; 0.2:0.1:0.7];  
y = sim(net,P1)  
yc = vec2ind(y)



Топология сети



Симулируем работу сети на новых данных:

Симуляция:

P1= [0.2:0.1:0.7; 0.2:0.1:0.7];  
y = sim(net,P1)  
yc = vec2ind(y)

Ответ: yc = 1 1 1 2 2 2

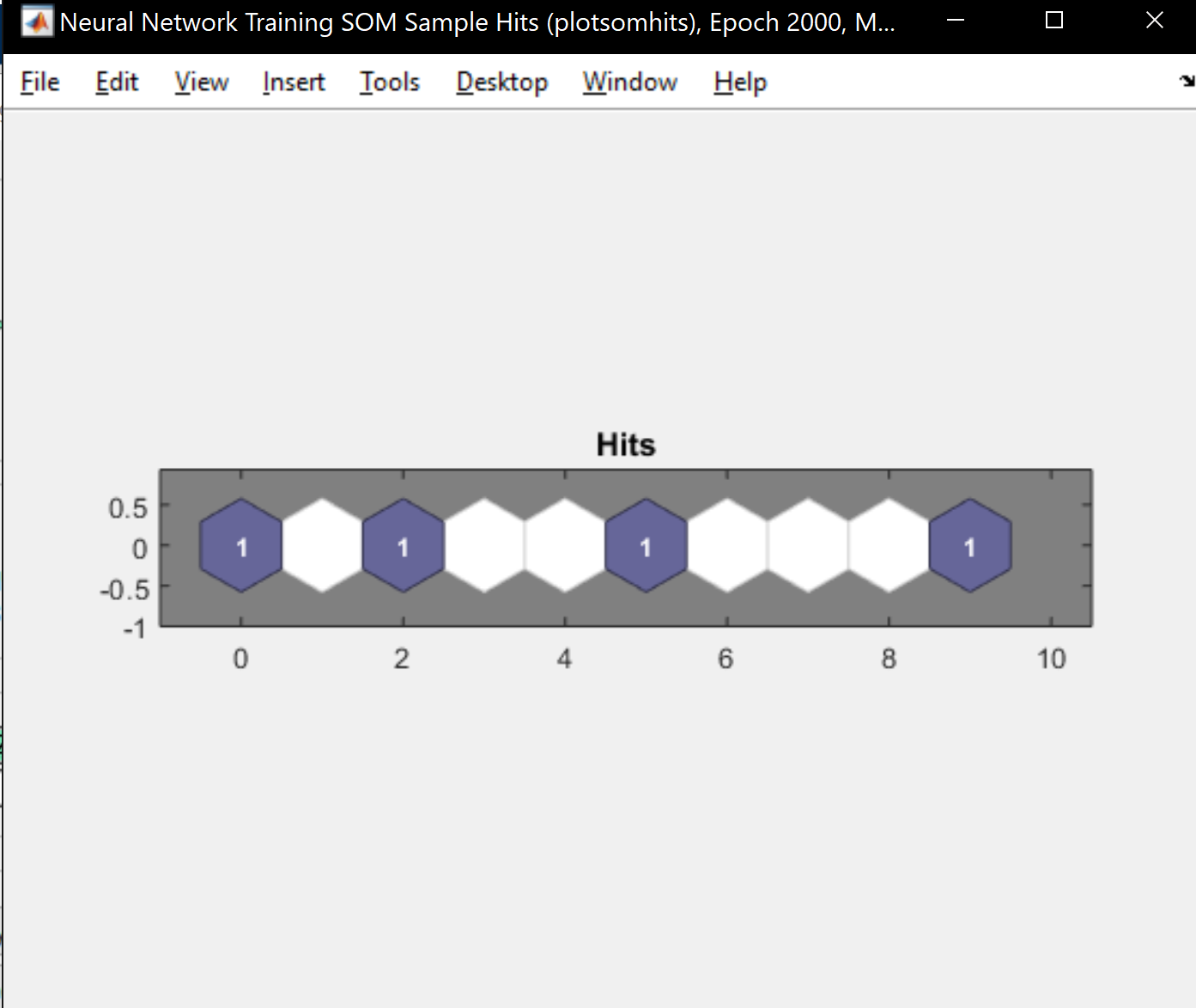
**Задание 2**

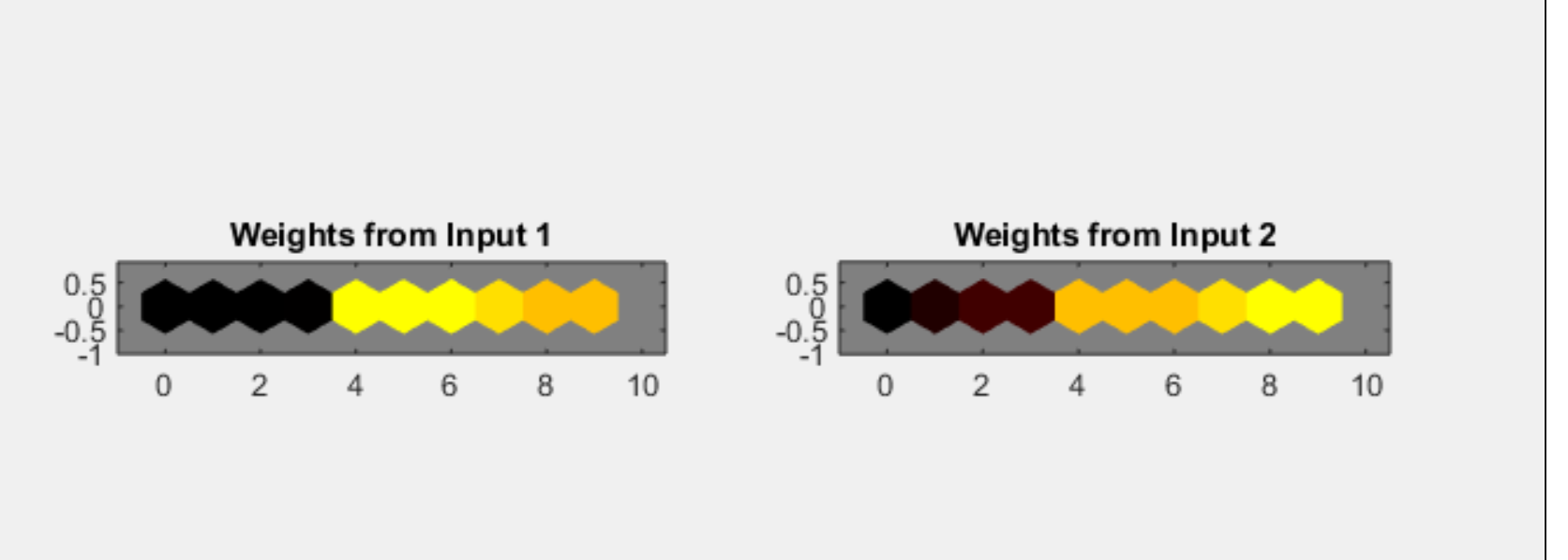
Создать одномерную карту Кохонена из 10 нейронов, обучить её на последовательности из 100 двухэлементных векторов единичной длины, распределенных равномерно в пределах от 0 до 90º, построить график распределения векторов по кластерам и выполнить моделирование сети для одного вектора входа.

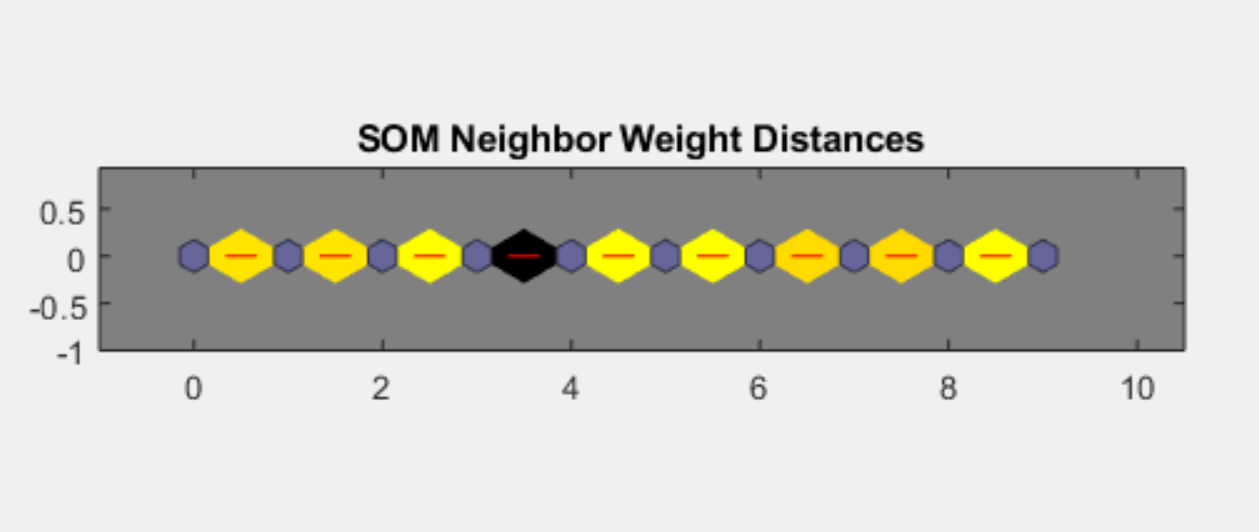
Создадим и обучим сеть

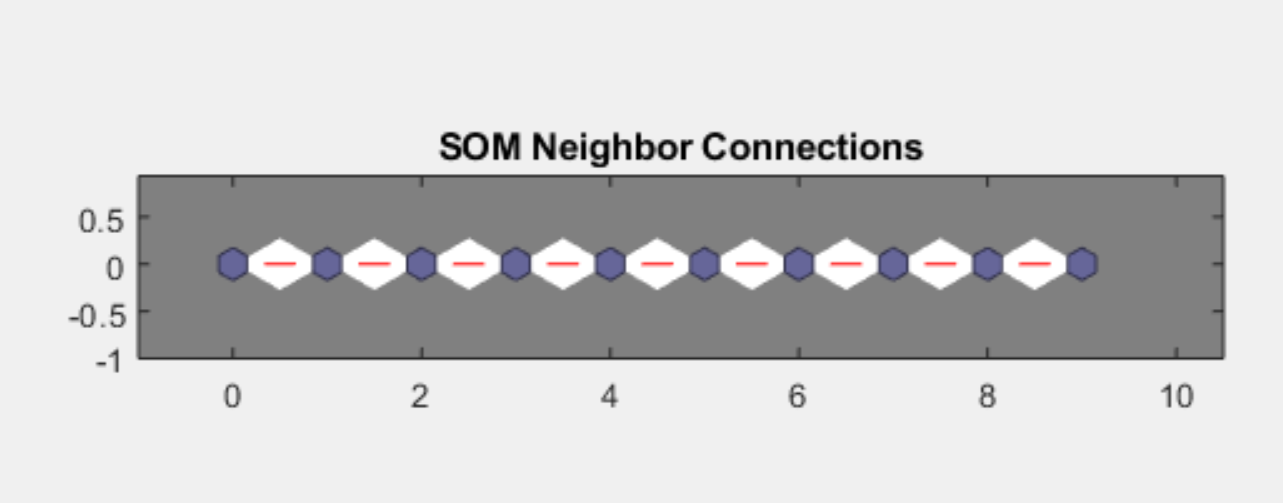
P = [.1 .8 .1 .9; .2 .9 .1 .8]; % для обучения слоя;  
angels=0 : 0.5 +pi/99 : 0.5\*pi;  
p=[sin(angels); cos(angels)];  
plot(P(1, 1:10:end), P(2, 1:10:end), '-r')  
hold on  
net=newsom([0 1 ;0 1], [10]);  
net.trainparam.epochs=2000;  
net.trainparam.show=100;  
[net,tr]=train(net,P);

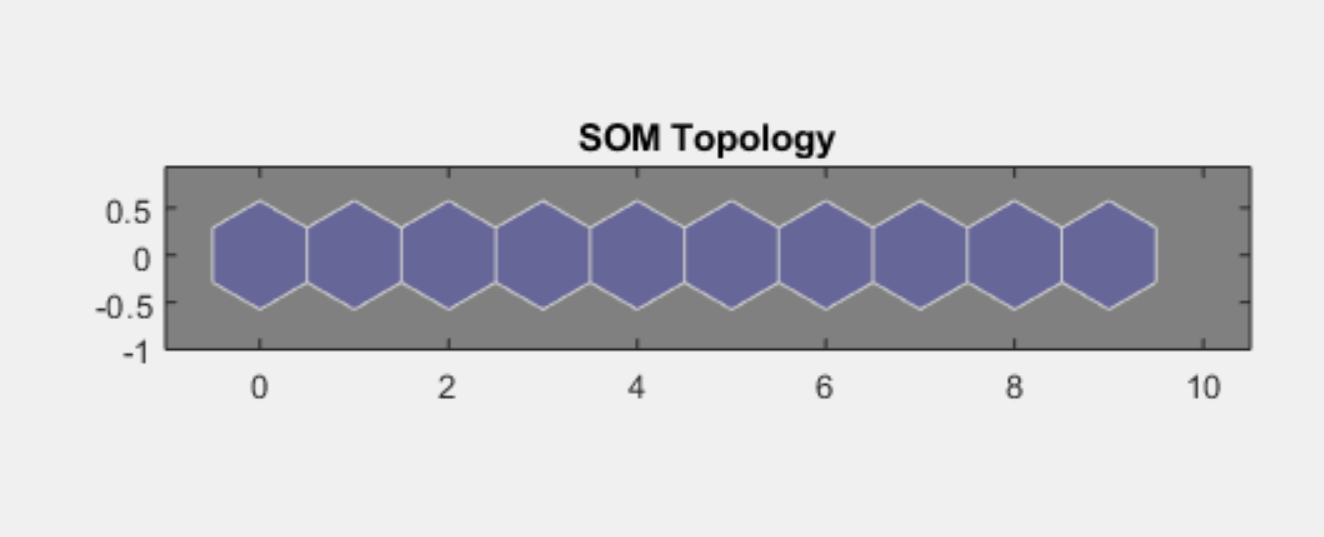




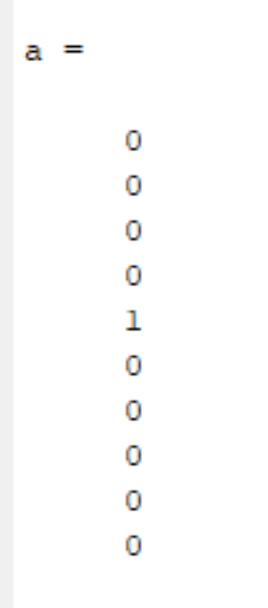






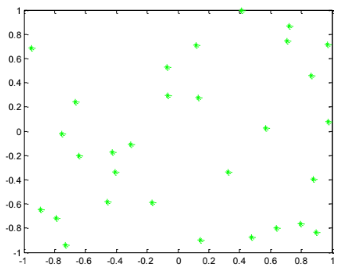


Моделируем на обучающей выборке



**Задание 3 Вариант 2**

Оцифровать и провести кластеризацию данных, используя слой Кохонена и самоорганизующуюся карту SOM.



Оцифруем данный график с помощью программы GetData Graph Designer:

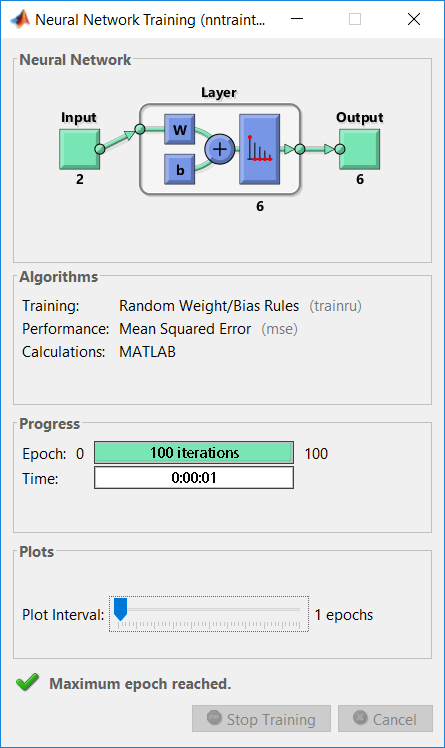
Введем данные в рабочу область Matlab:

X = [-0.954545454545455 -0.753246753246753 -0.662337662337662 -0.889610389610390 -0.792207792207792 -0.727272727272727 -0.642857142857143 -0.461038961038961 -0.409090909090909 -0.428571428571429 -0.311688311688312 -0.168831168831169 -0.071428571428571 -0.071428571428571 0.116883116883117 ]  
];  
P = zeros(2, 30)  
for i = 1:30  
P(1, i) = X(i)  
P(2, i) = Y(i)  
end;

Проведем кластеризацию, используя сеть Кохена:

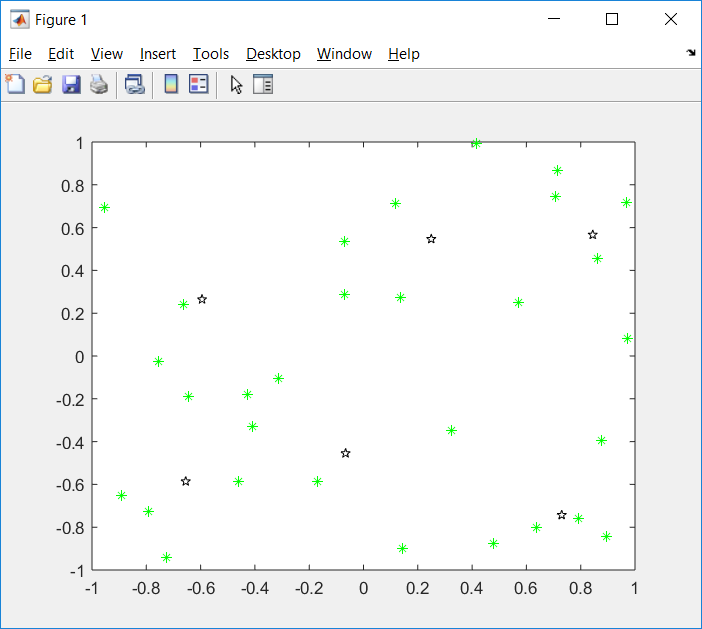
Создадим и обучим сеть

plot(P(1,:), P(2,:), '\*g');  
hold on;  
nclusters = 6;  
a1 = -1;  
a2 = 1;   
b1 = -1;   
b2 = 1;  
net = newc([a1 a2; b1 b2], nclusters, 0.1, 0.0005);  
wo = net.IW{1};  
bo = net.b{1};  
net.trainParam.epochs=100;  
net.trainParam.show=20;  
net = train(net,P);



И добавим к графику найденных «победителей» кластеров:

w = net.IW{1};  
bn = net.b{1};  
plot(w(:,1),w(:,2),'kp');

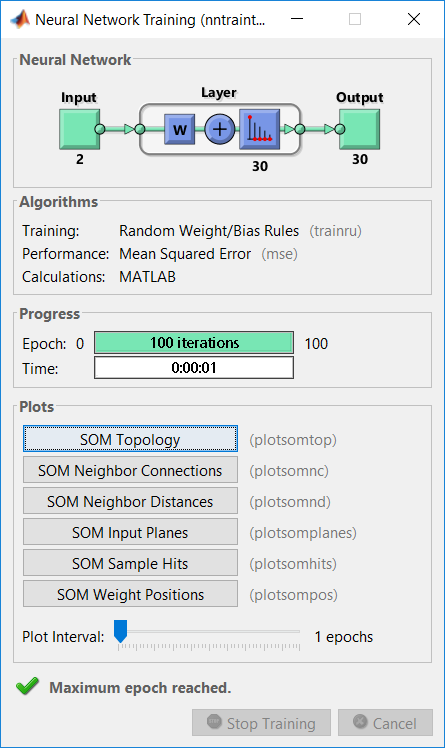


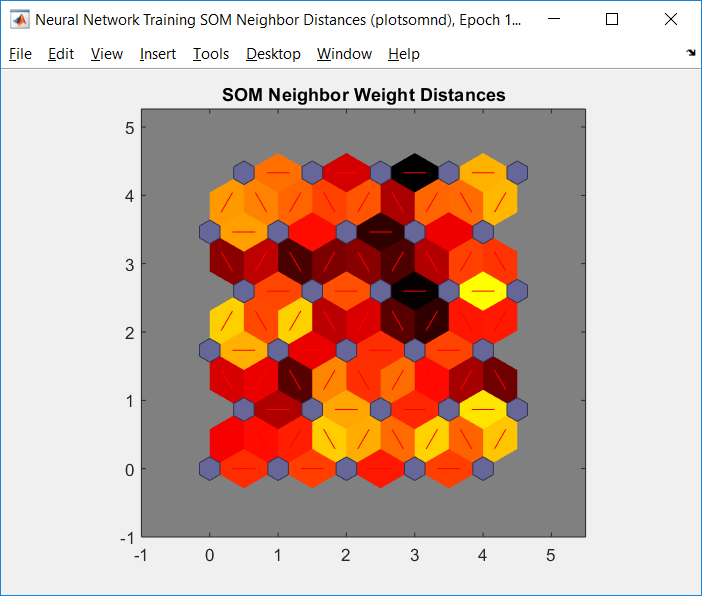
Проведем кластеризацию, использую самоорганизующуюся карту SOM:

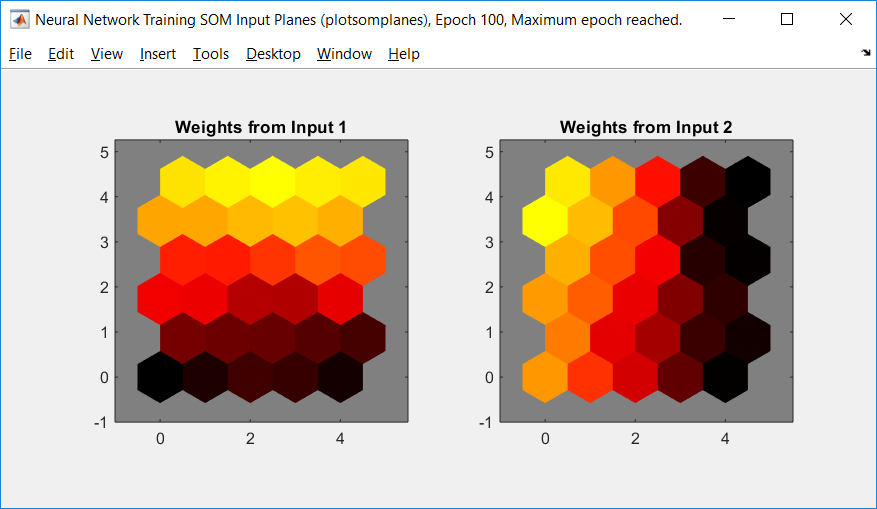
Создадим карту и обучим ее на наших данных:

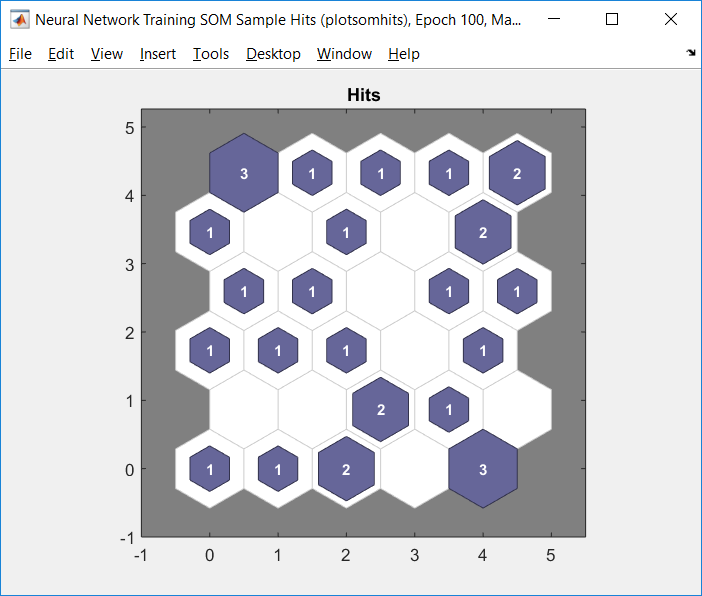
net = newsom ([-1 1; -1 1], [5 6]);  
net.trainParam.epochs = 100;  
net = train(net, P)

Изучим результаты обучения:











Посмотрим, к какому классу сеть отнесет новые данные:

p = [0; 0];  
a = sim(net, p)

В моём случае сеть отнесла новые данные к 13 кластеру.

Ответы на вопросы

В: В чем заключается задача кластеризации?

O: Задача кластеризации заключается в поиске различных классов образов при заранее неизвестном количестве этих классов.

В: Какую структуру имеет НС Кохонена?

O: НС Кохонена имеет вид двухслойной сети. Первый слой – слой входных нейронов, второй – слой Кохонена. Слой Кохонена может быть одномерным, двумерным и трехмерным.

В: Каким алгоритмом обучается НС Кохонена?

O: Алгоритм самообучения Кохонена: задается мера, например, евклидова. Согласно ей каждый новый вход относится к кластеру, к победителю которого он ближе. Далее веса победителя и всех остальных активных нейронов Кохена корректируется, в результате чего n-мерное входное пространство проецируется на m-мерную сетку Кохена.

**Вывод:** Изучил сеть Кохена, карту SOM, построил примеры сети и карты для кластеризации тестовых данных.

**Лабораторная работа №8**

**Задача:**  Исследование сети хопфилда. исследовать устойчивость сети Хопфилда и её сходимость к

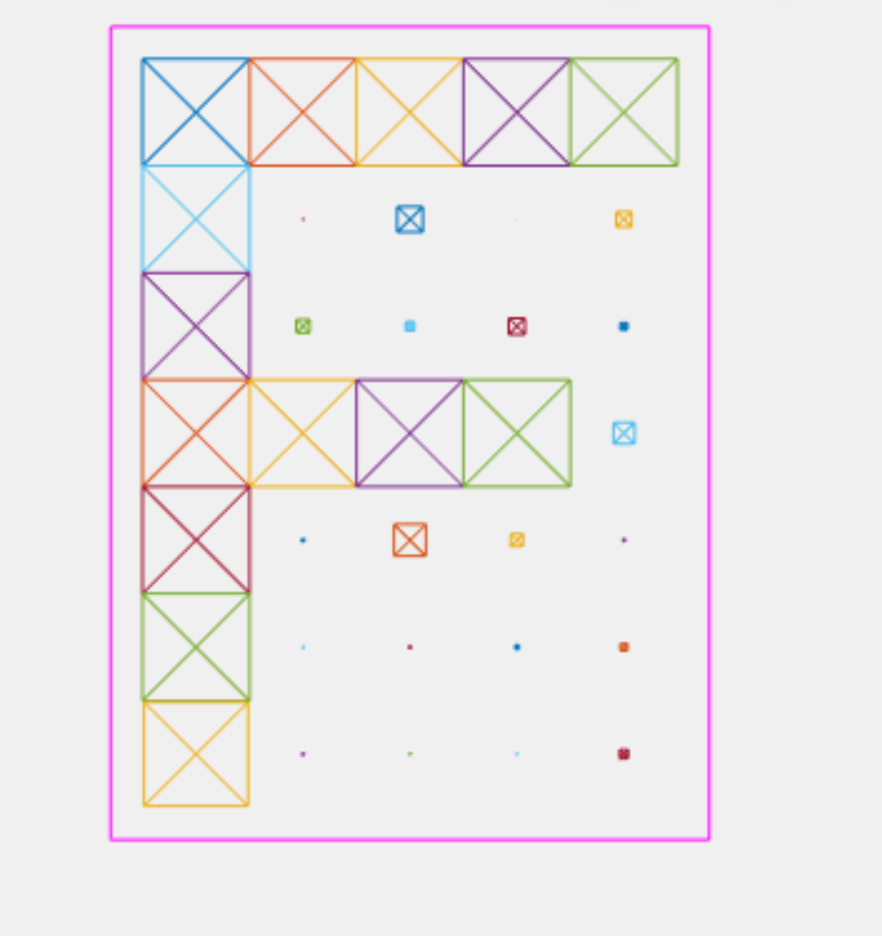
аттракторам, применение сети Хопфилда для распознавания образов.

**Ход работы:**

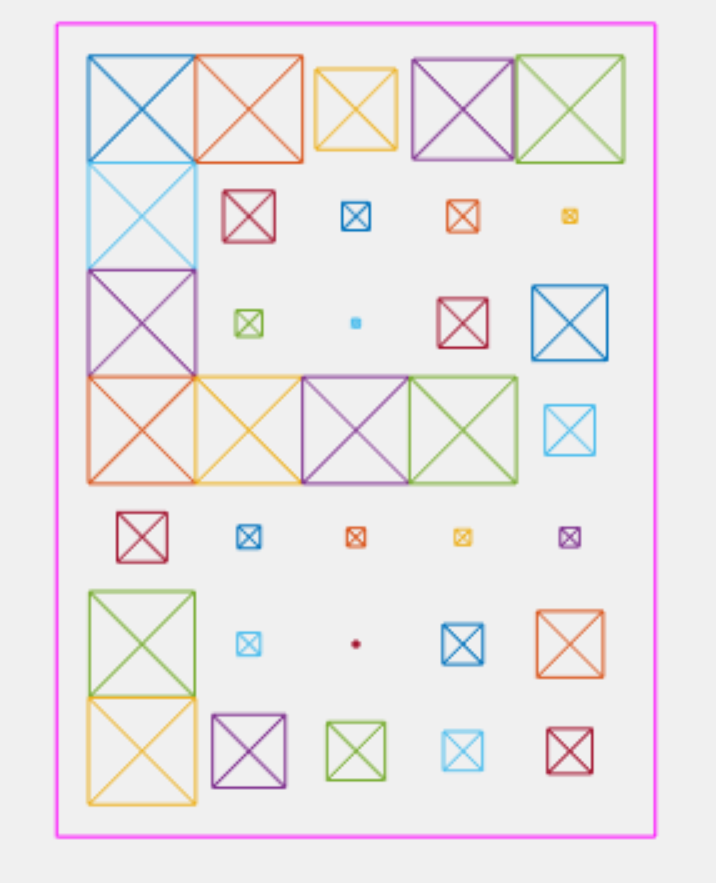
Создадим и обучим Хопфилда: обучение без учителя. Матрицы весов симметричны

[alphabet,targets] = prprob;  
for i = 1:26  
letter{i} = reshape(alphabet(:, i), 5, 7)';  
end;  
net = newhop(letter);  
[Y,Pf,Af] = sim(net, letter);

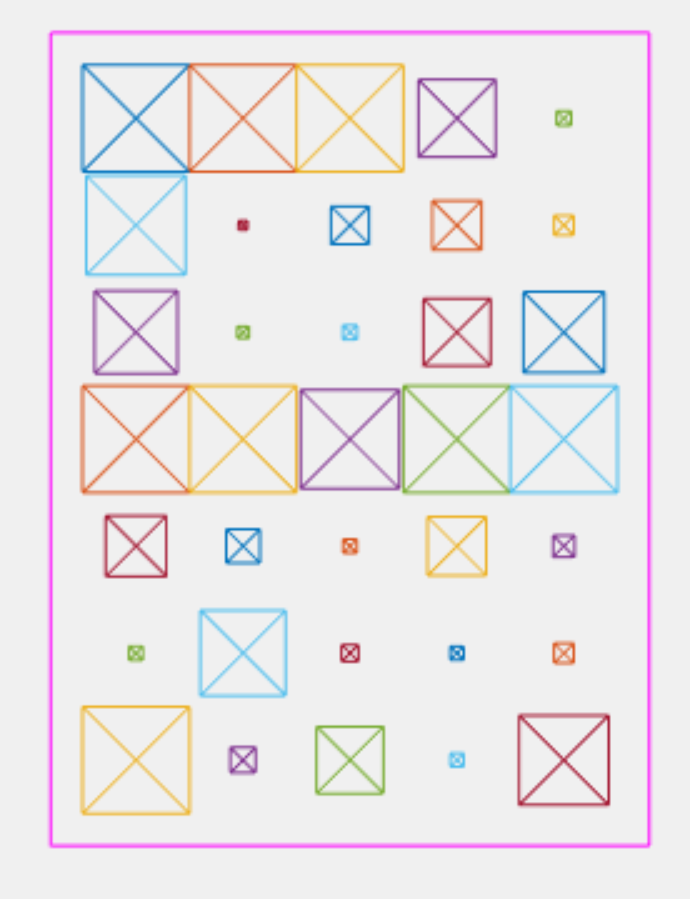
Теперь попробуем разпознать, например, букву F с шумом 0.1:



Пример символа с шумом 0.1



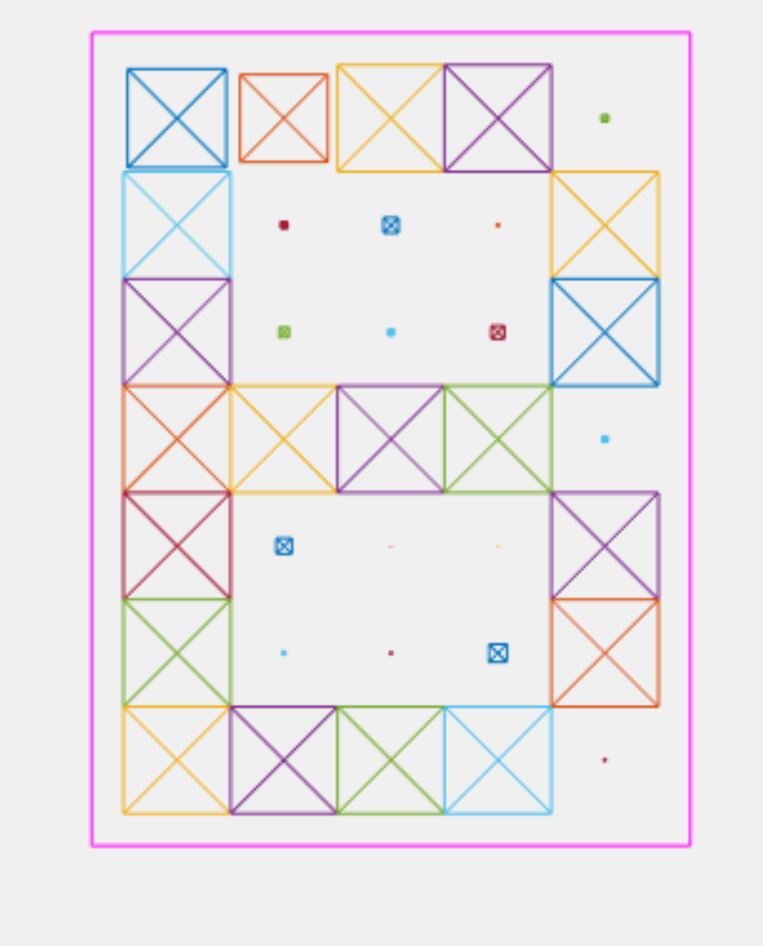
Шум 0.3



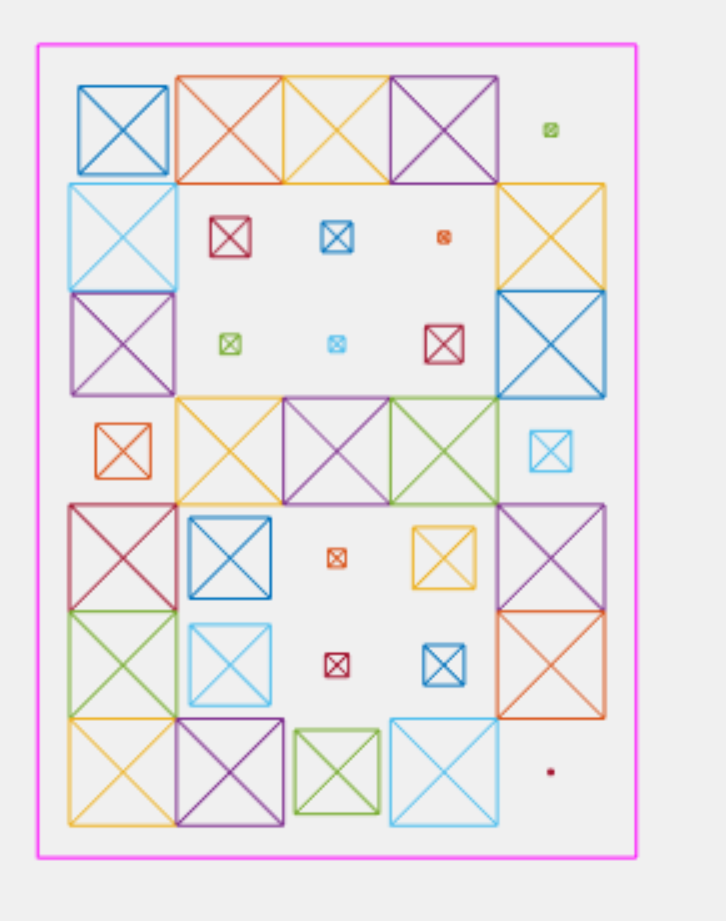
Шум 0.4

Другие буквы пропущеные через сеть

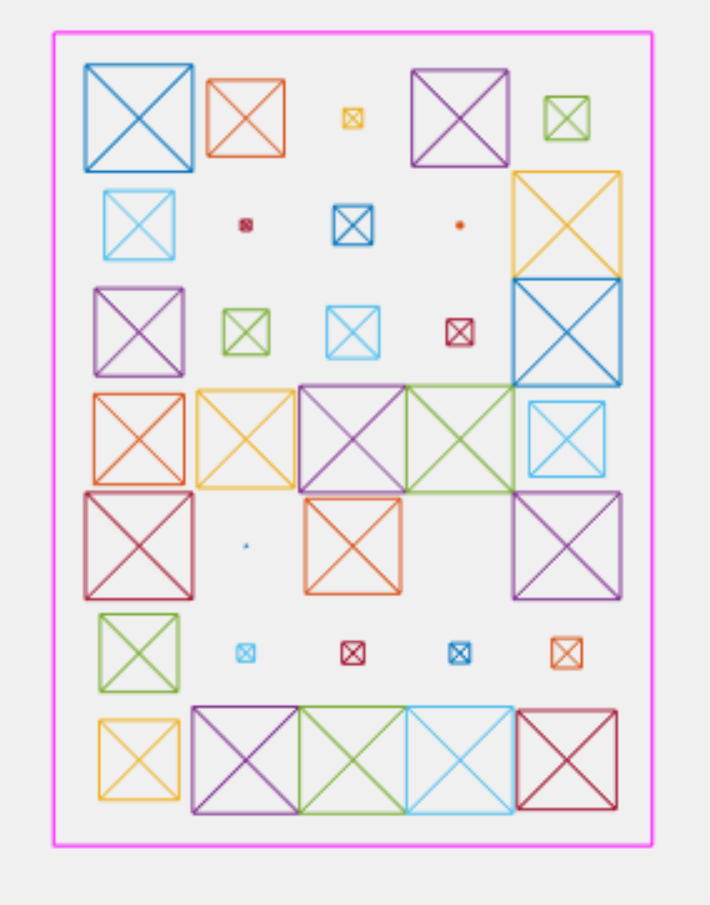
Noisy = alphabet(:,26) + randn(35,1) \*0.1;  
[Y,Pf,Af] = sim(net,5,[],reshape(noisy, 5, 7)');  
plotchar(reshape(Y', 1, 35)');

****

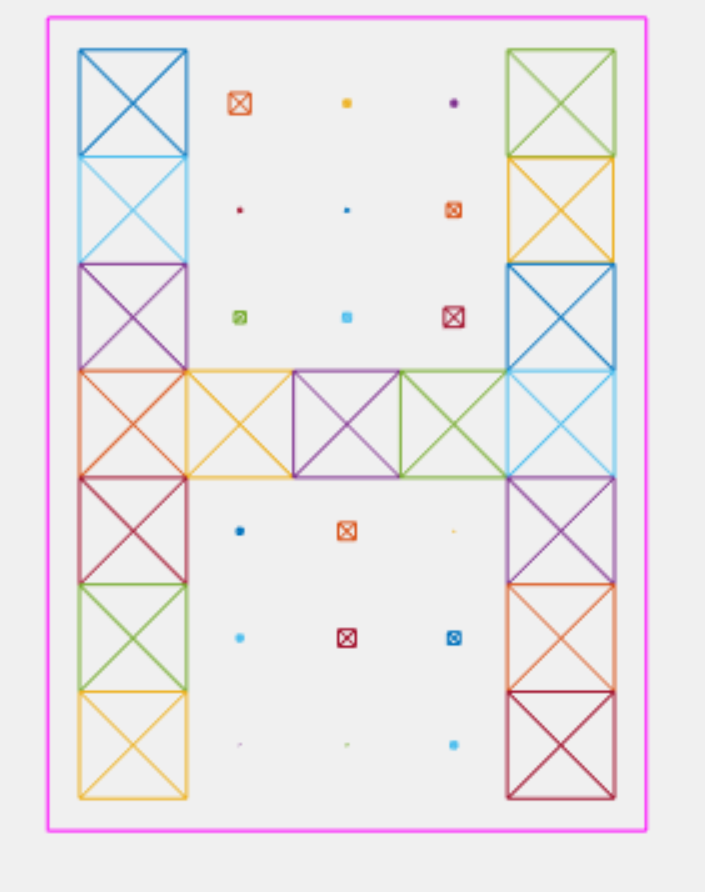
**0.1**

****

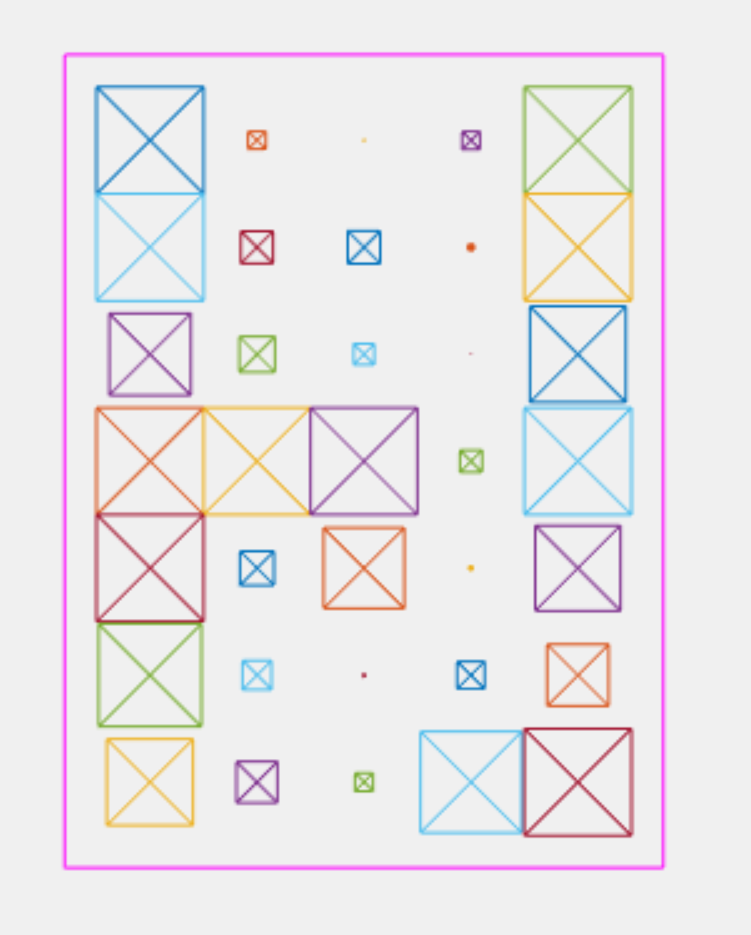
**0.3**

****

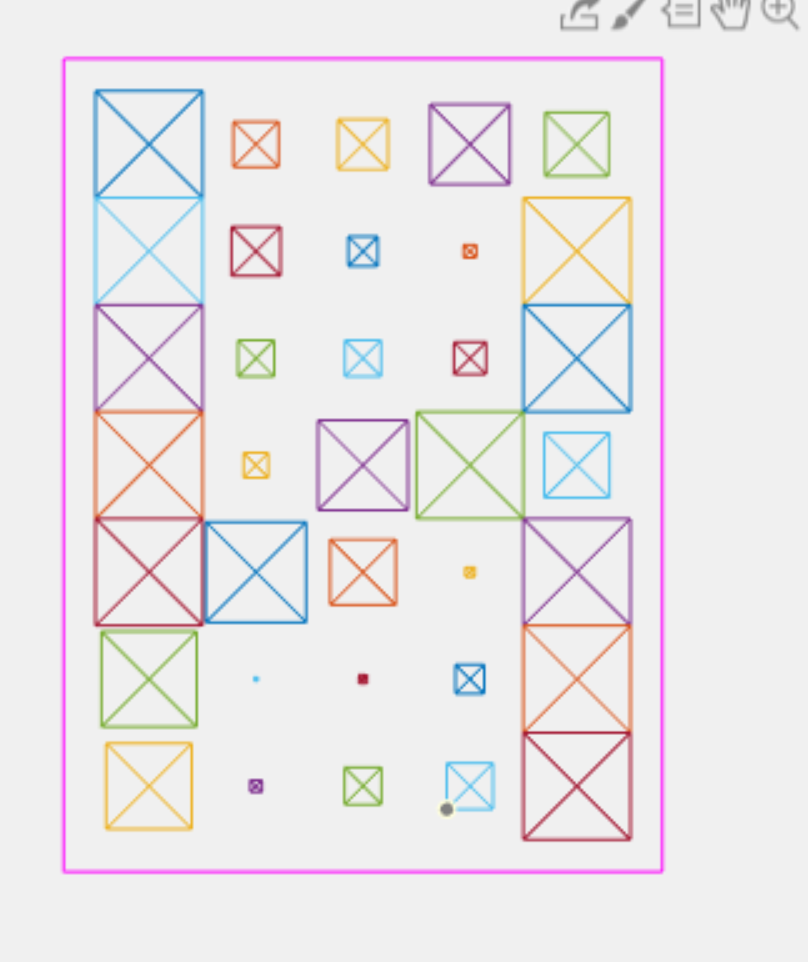
**0.4**

****

**0.1**

****

**0.3**

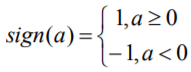
****

**0.4**

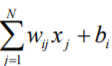
**Ответы на вопросы**

В: Какова активационная характеристика нейронов сети Хопфилда?

O: Функция активации нейронов сети Хопфилда – сигнум с значениями +-1:



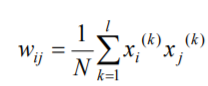
Где a:



Для i-ого выхода

В: Чему равно начальное состояние нейронов сети Хопфилда?

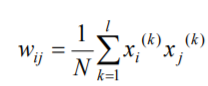
O: Начальное состояние yj(0) = xi, начальные значения весов для выборки из k N-мерных элементов:



bi часто равна 0.

В: Как рассчитывается матрица синаптических коэффициентов сети Хопфилда? Какими свойствами она обладает?

O: По формуле



Синаптические коэффициенты обладают следующим свойством: wii = 0, а вся матрица является симмитричной wij=wji.

В: Как приближенно оценивается объем памяти сети Хопфилда?

O: По следующей формуле:



В: На какие типы можно разделить множество аттракторов сети Хофилда? Что такое «ложная память»?

O: Аттракторы бывают динамические и статические. Статический аттрактор – ситуация, когда значения t и t+1 в конце-концов совпали. Динамический – состояния t+1 и t-1, t и t+2 совпадают. В большинстве случаев, такая ситуация нежелательна, потому что определенного ответа сети не сущесвует. Такая ситуация и называется ложной памятью.

**Вывод:** Изучил сеть Хопфилда, создал сеть Хопфилда для распознавания букв латинского алфавита, исследовал её устойчивость к шуму.