МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Кафедра вычислительной математики и программирования

Лабораторная работа №3
по спецкурсу «Нейроинформатика»

Многослойные сети.
Алгоритм обратного распространения ошибки

Выполнил: Днепров И.С.
Группа: М8О-407Б, вариант 10
Преподаватели: Тюменцев Ю.В.

Москва, 2020

Цель работы

Целью работы является исследование свойств многослойной нейронной сети прямого распространения и алгоритмов ее обучения, применение сети в задачах классификации и аппроксимации функции.

Основные этапы работы:

1. Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.

2. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.

3. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов второго порядка.

Для обучения многослойных нейронных сетей прямого распространения используются методы поиска экстремума функций многих переменных.

Оборудование

Процессор: 2,4 GHz Intel Core 2 Duo

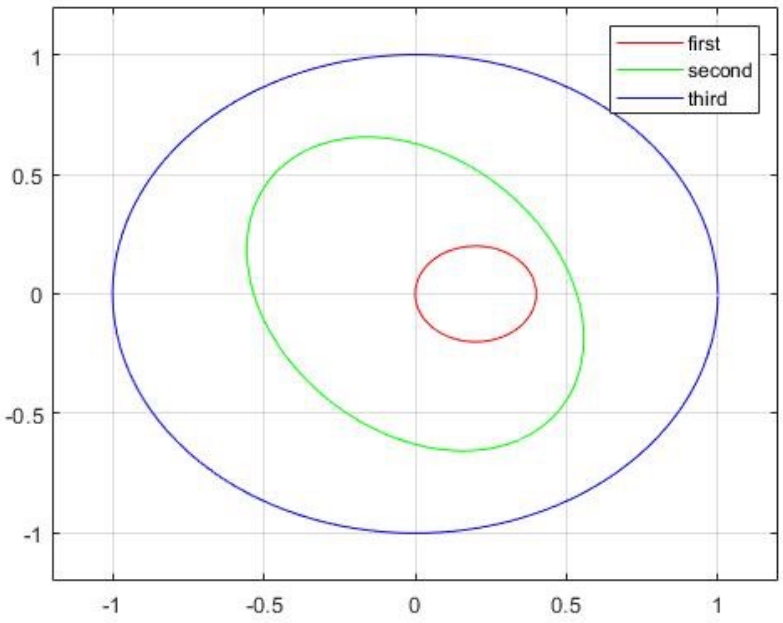
Оперативная память: 8 ГБ 1067 MHz DDR3

Программное обеспечение

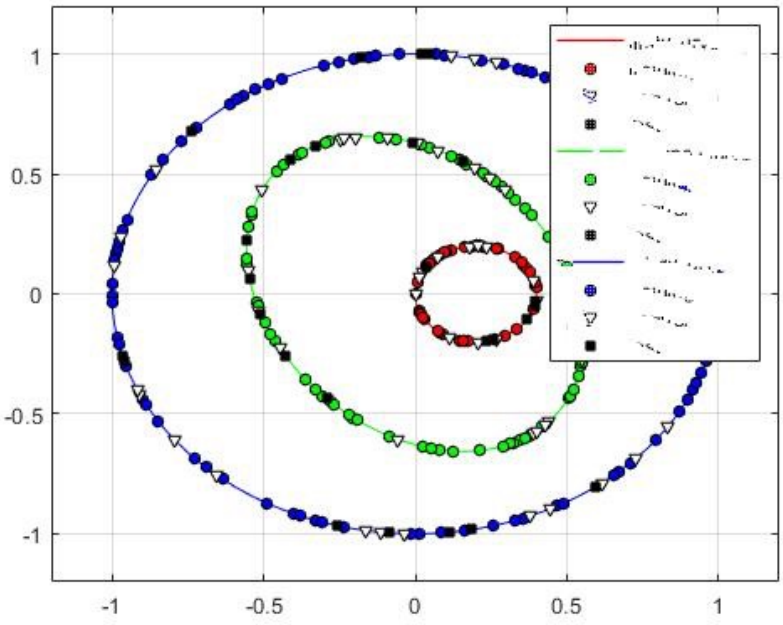
Matlab R2020b, 64-bit.

Задание 1.

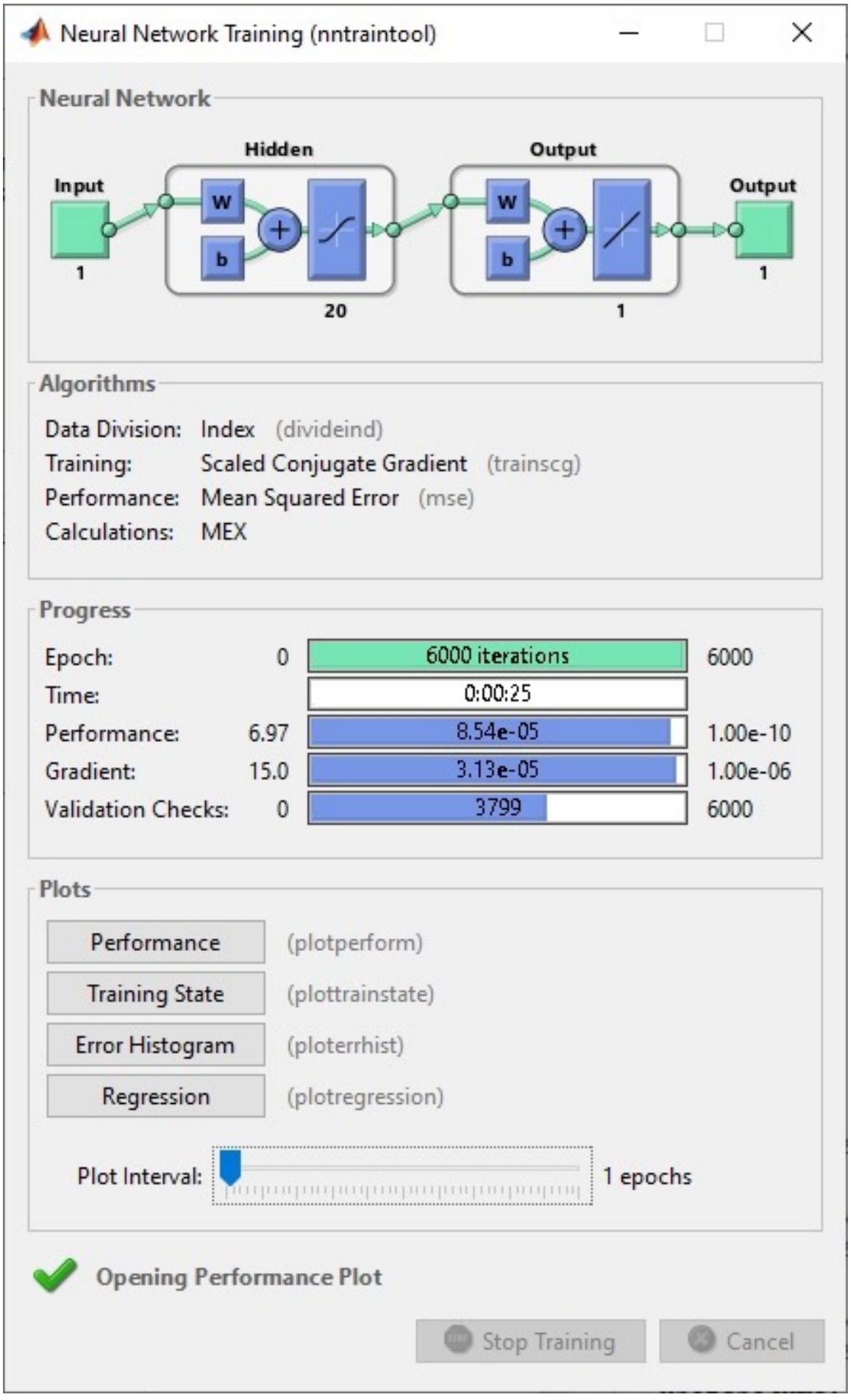
Алгебраические линии:



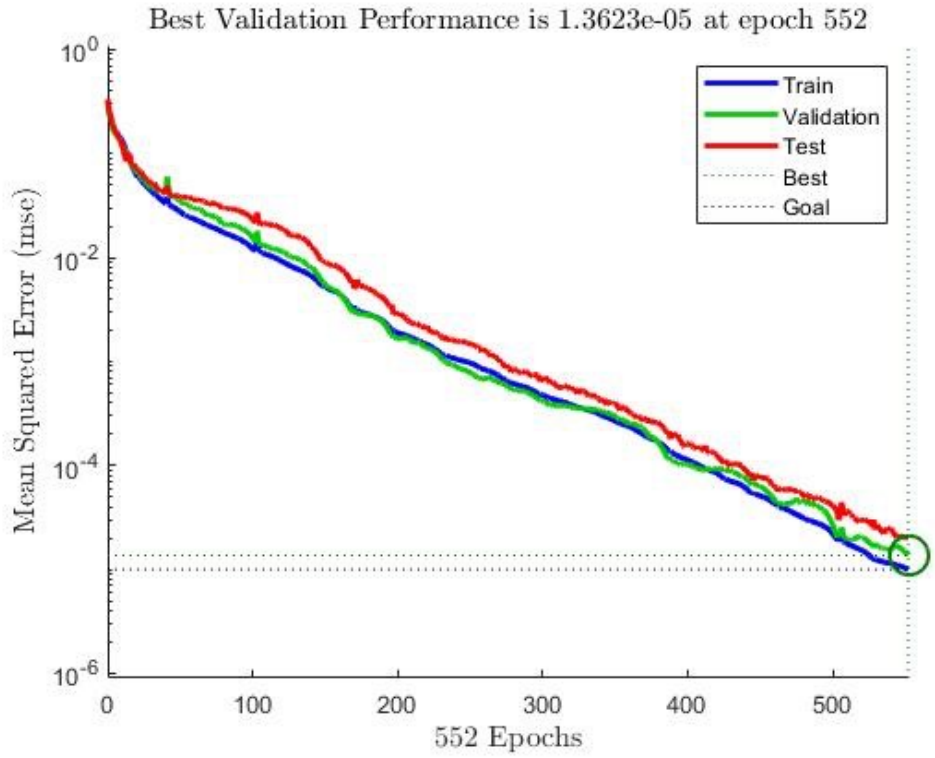
Множество точек:



Обучение сети:



Сходимость погрешность:

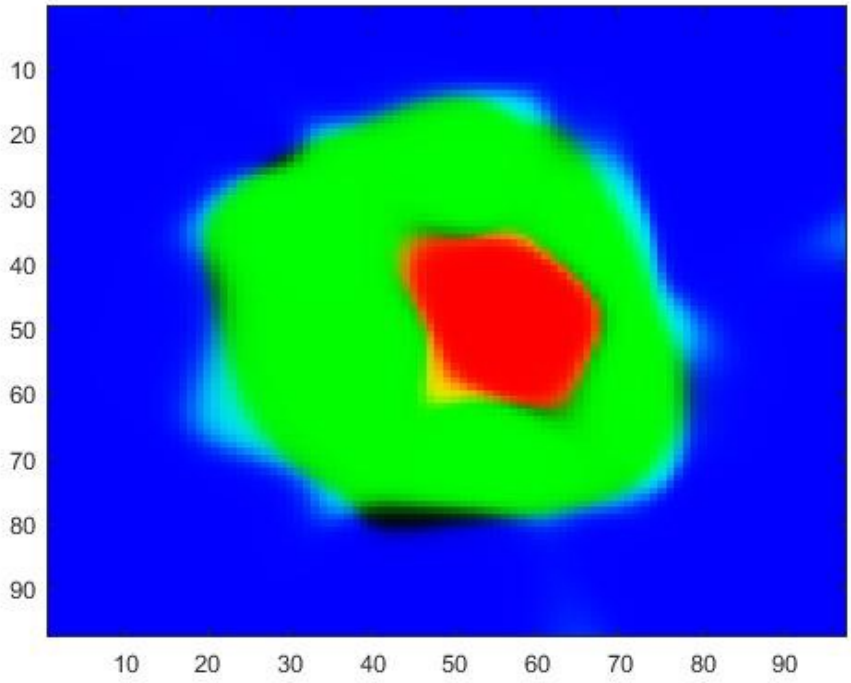


Результат обучения:

Обучающие: 196/196

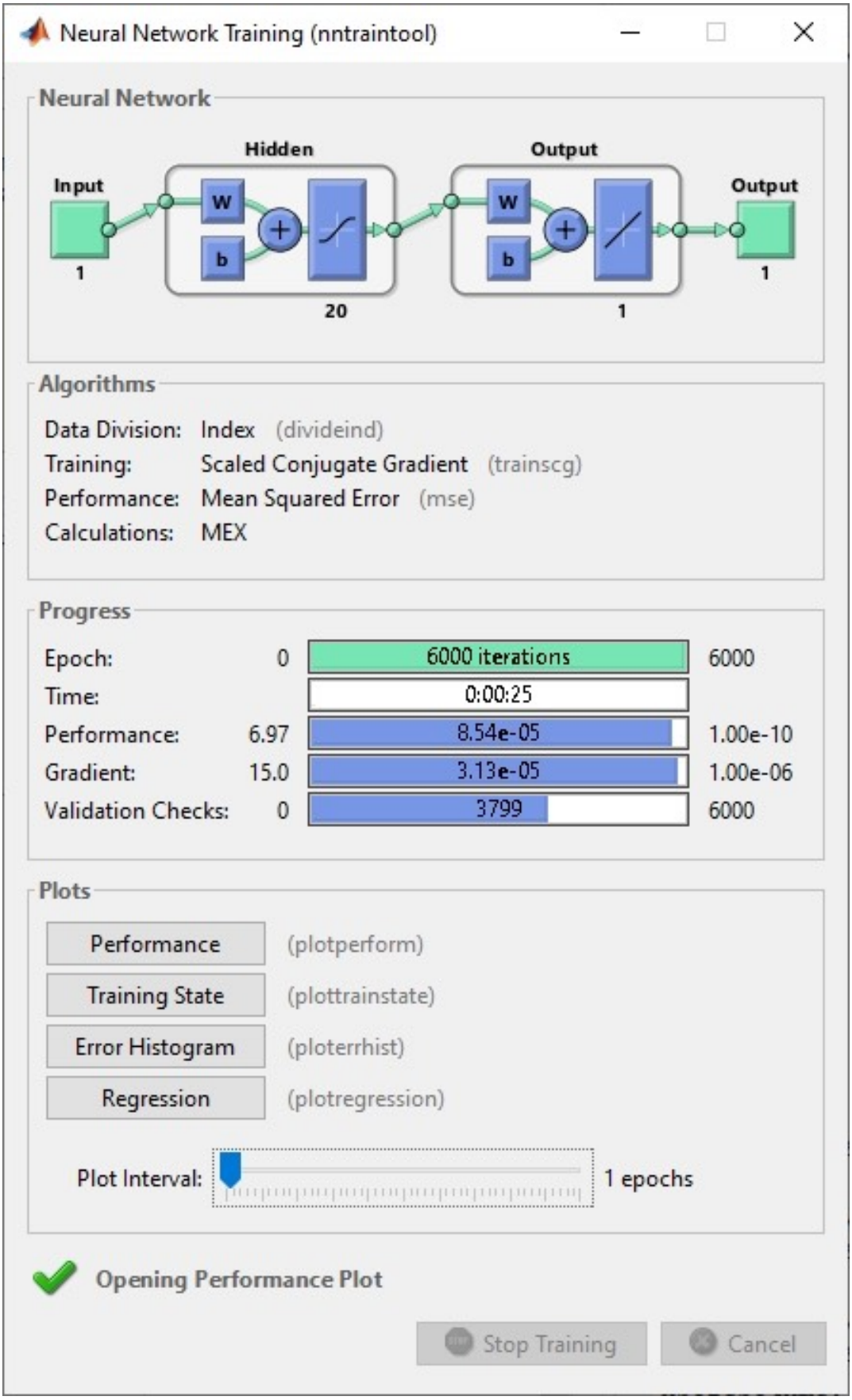
Проверочные: 56/56

Тестовые: 28/28

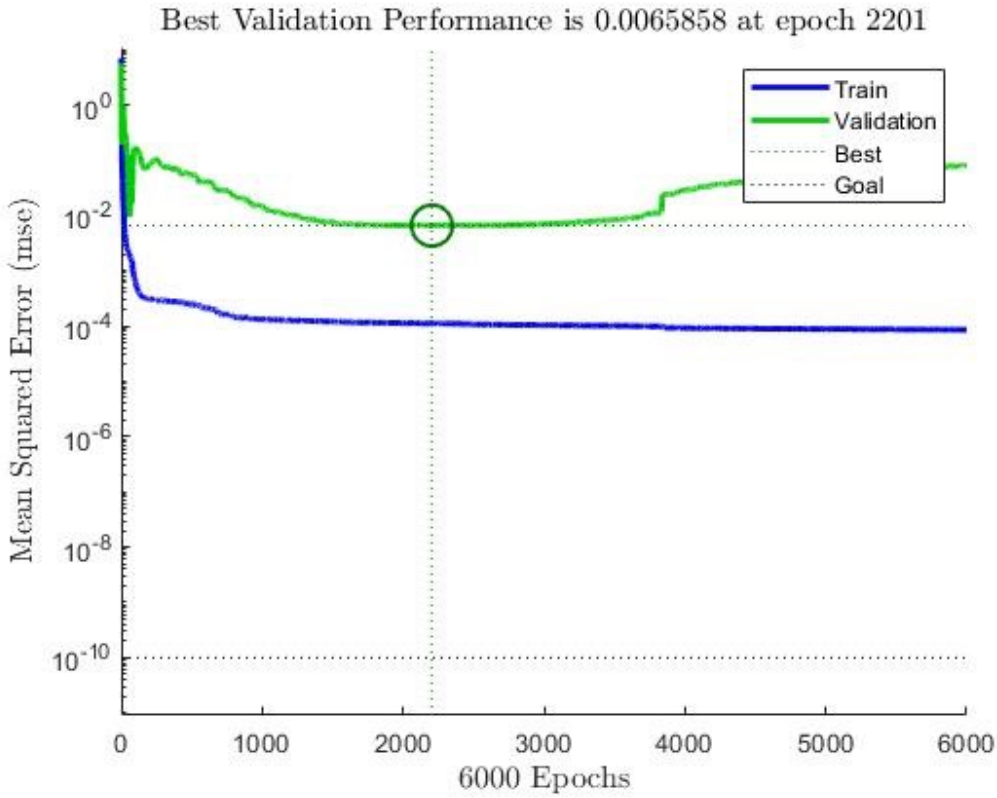


Задание 2.

Обучение сети:



Сходимость ошибки:



R^2: 0.999794

MSE: 0.000080

RMSE: 0.008940

Относительная СКО: 0.452129%

MAE: 0.007389

min abs err: 0.000030

max abs err: 0.022907

MAPE: 2.570994

Доля с ошибкой менее 5%: 92.817680%

Доля с ошибкой от 5% до 10%: 3.314917%

Доля с ошибкой от 10% до 20%: 1.657459%

Доля с ошибкой от 20% до 30%: 1.104972%

Доля с ошибкой более 30%: 1.104972%

R^2: 0.823065

MSE: 0.001906

RMSE: 0.043659

Относительная СКО: 12.503080%

MAE: 0.040229

min abs err: 0.004383

max abs err: 0.072249

MAPE: 4.885656

Доля с ошибкой менее 5%: 60.000000%

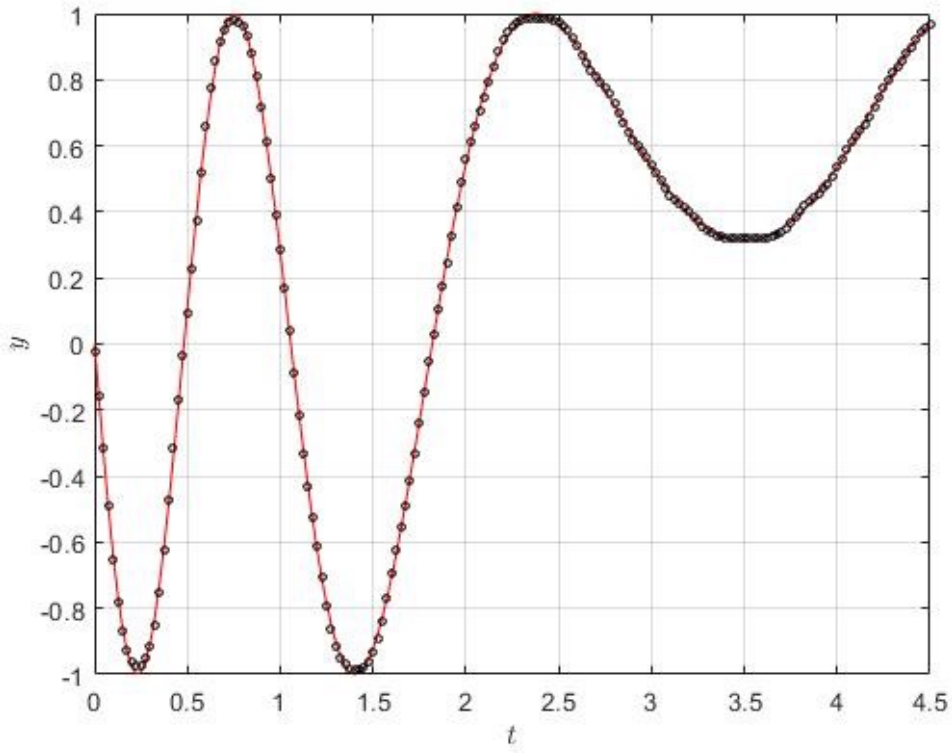
Доля с ошибкой от 5% до 10%: 30.000000%

Доля с ошибкой от 10% до 20%: 10.000000%

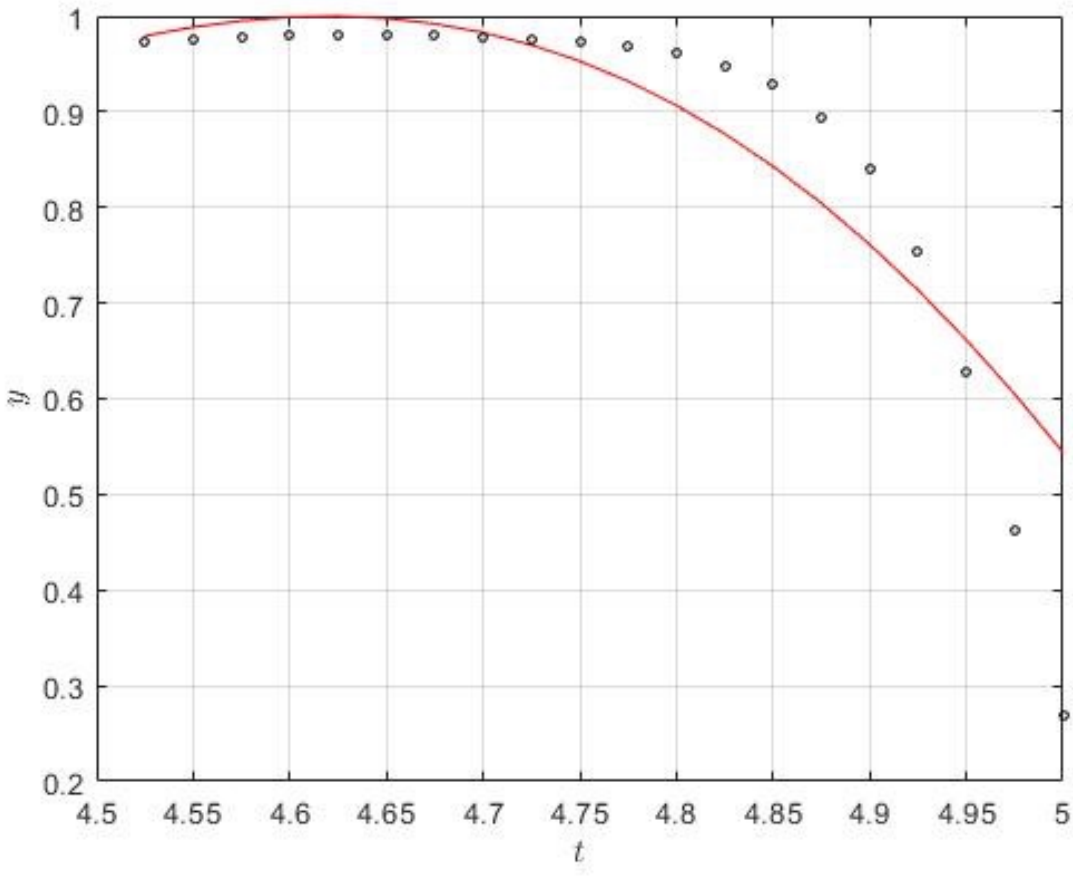
Доля с ошибкой от 20% до 30%: 0.000000%

Доля с ошибкой более 30%: 0.000000%

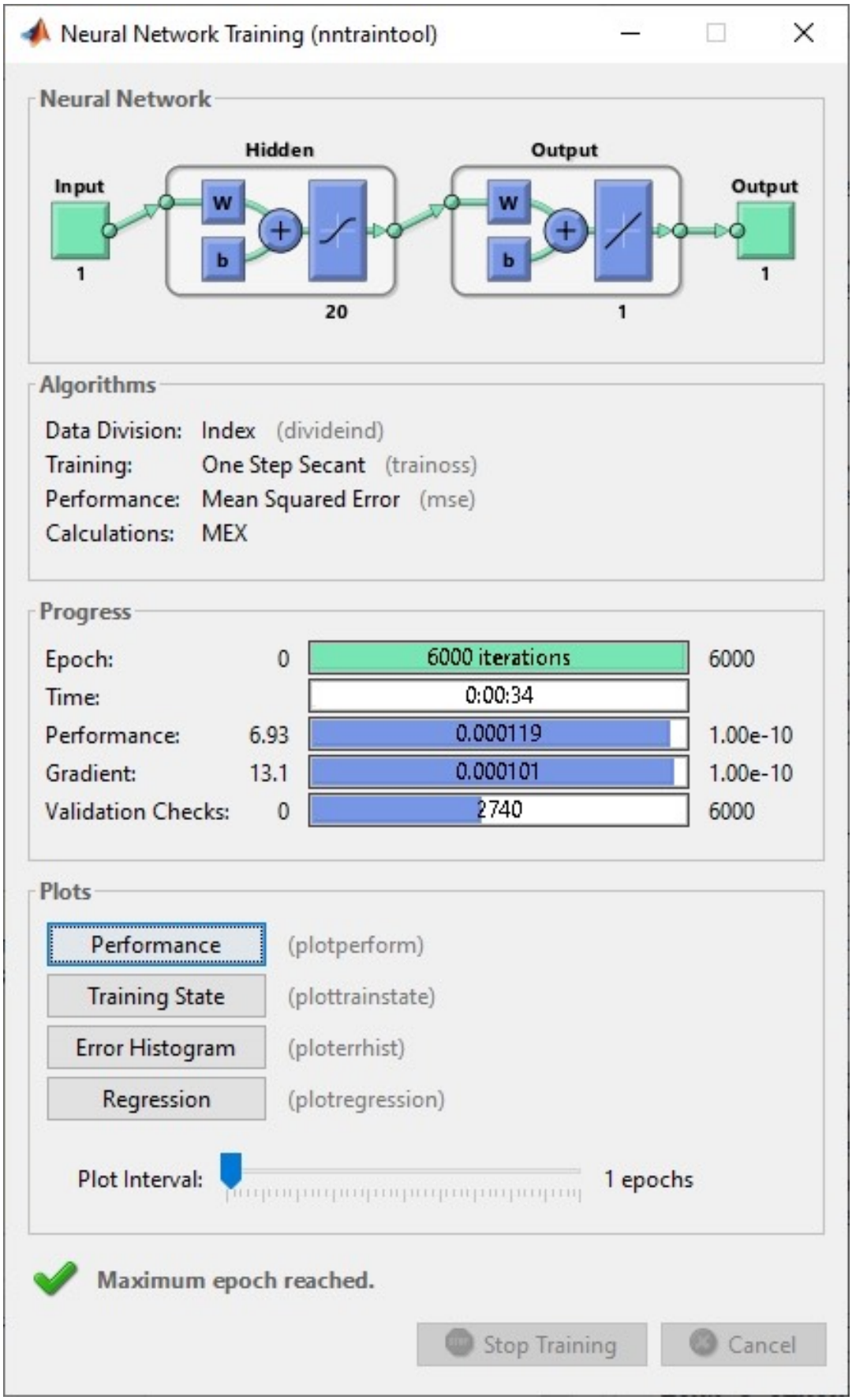
Результат на обучающем множестве:



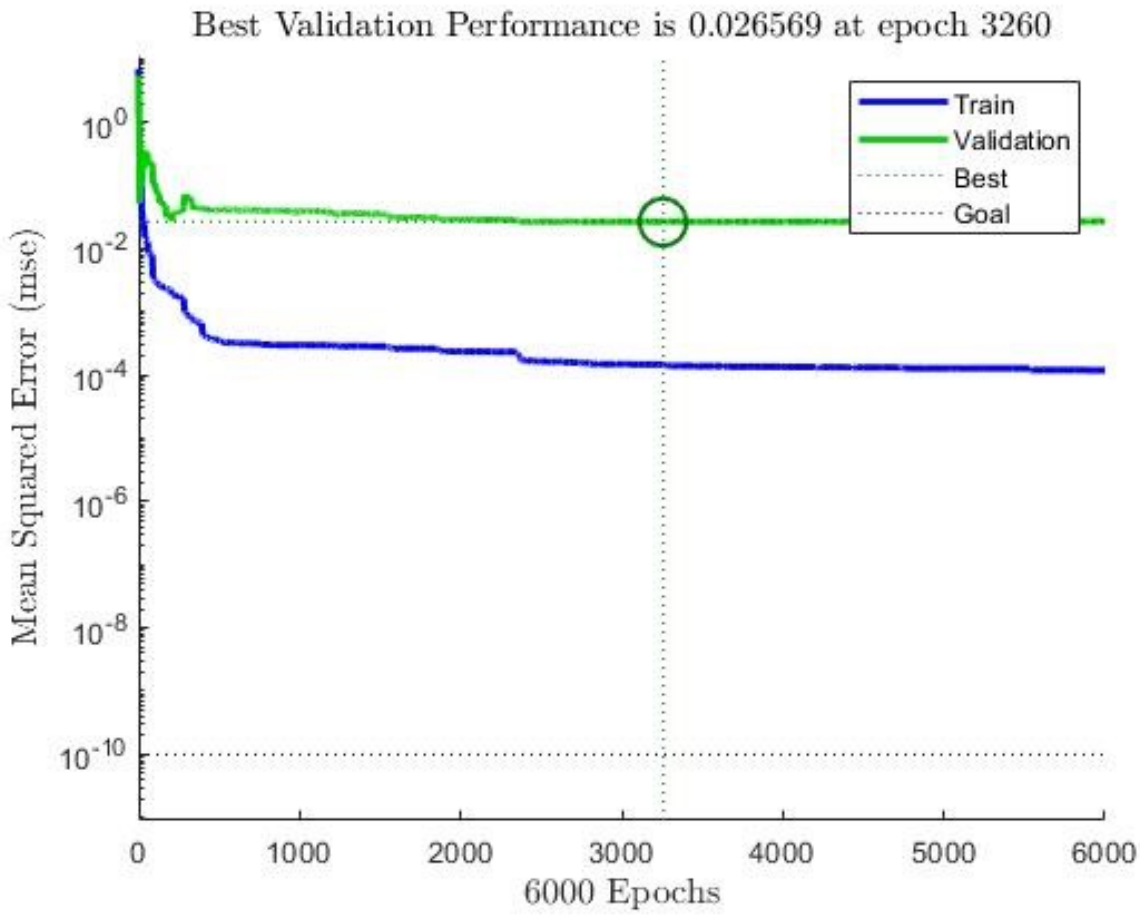
Результат на тестовом множестве:



Задание 3.
Обучение сети:



Сходимость погрешности:



R^2: 0.995518

MSE: 0.001728

RMSE: 0.041567

Относительная СКО: 2.058610%

MAE: 0.029674

min abs err: 0.000330

max abs err: 0.146281

MAPE: 68.557808

Доля с ошибкой менее 5%: 65.193370%

Доля с ошибкой от 5% до 10%: 16.574586%

Доля с ошибкой от 10% до 20%: 9.392265%

Доля с ошибкой от 20% до 30%: 4.419890%

Доля с ошибкой более 30%: 4.419890%

R^2: 0.752854

MSE: 0.013456

RMSE: 0.116002

Относительная СКО: 16.983764%

MAE: 0.108081

min abs err: 0.026509

max abs err: 0.171058

MAPE: 12.929543

Доля с ошибкой менее 5%: 10.000000%

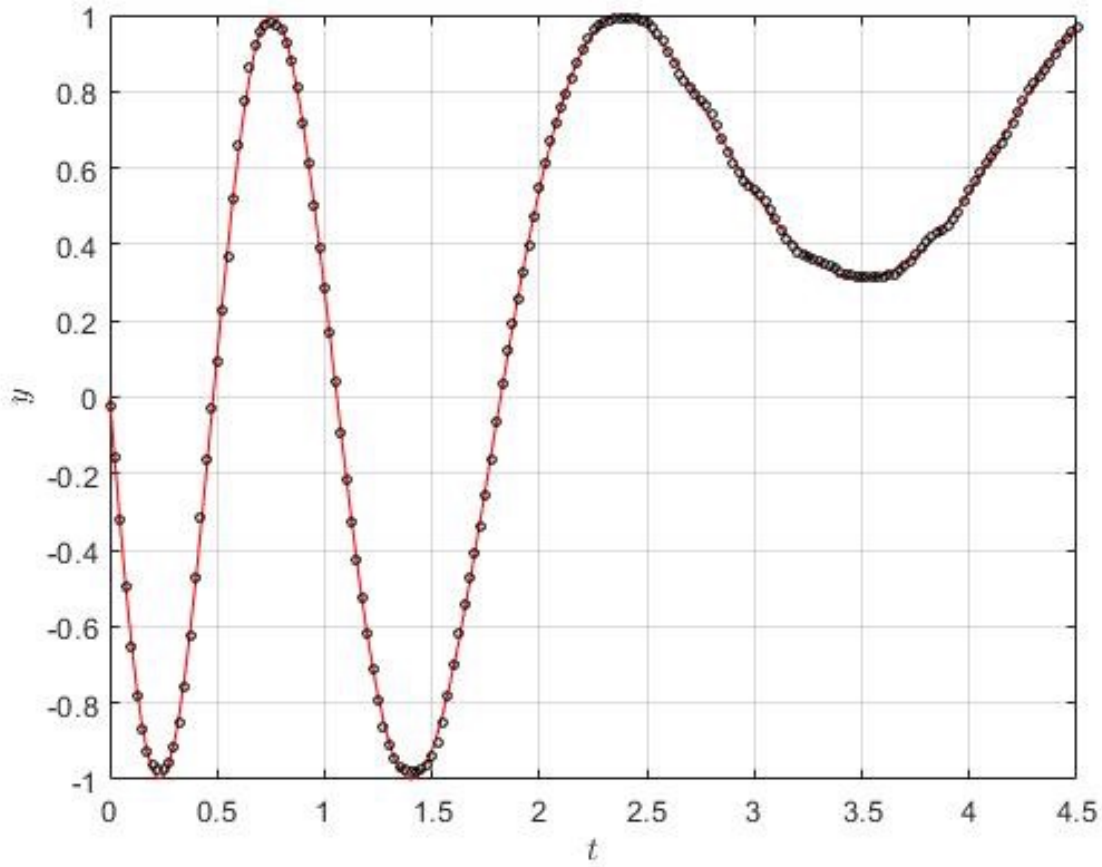
Доля с ошибкой от 5% до 10%: 15.000000%

Доля с ошибкой от 10% до 20%: 60.000000%

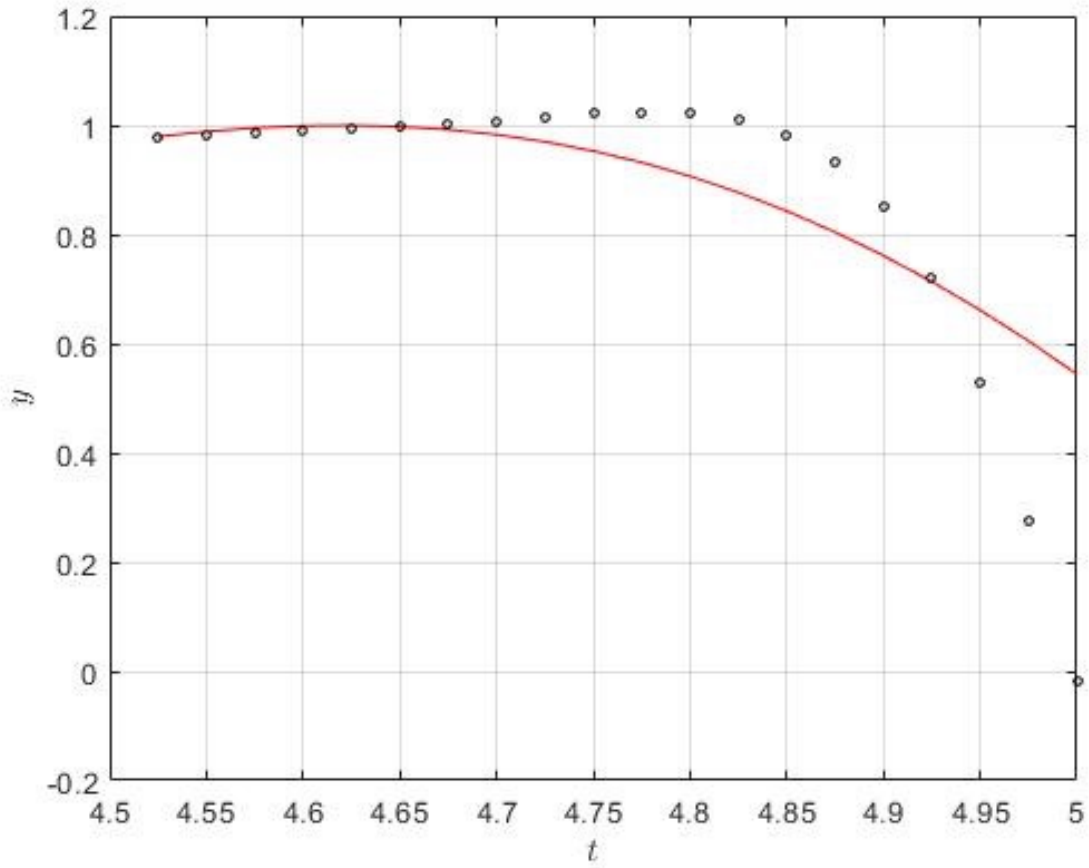
Доля с ошибкой от 20% до 30%: 15.000000%

Доля с ошибкой более 30%: 0.000000%

Результат на обучающем множестве:



Результат на тестовом множестве:



Код программы

Lab3.m

```
set(0, 'DefaultTextInterpreter', 'latex');

% Этап 1
% Задаем три элипса

t = 0:0.025:2*pi;

a = 0.2;
b = 0.2;
alpha = 0;
x0 = 0.2;
y0 = 0;
firstCurve = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [a * cos(t); b *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

a = 0.7;
b = 0.5;
alpha = -pi/3;
x0 = 0;
y0 = 0;
secondCurve = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [a * cos(t); b *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

a = 1;
b = 1;
alpha = 0;
x0 = 0;
y0 = 0;
thirdCurve = [cos(alpha), -sin(alpha); sin(alpha),  cos(alpha)] * [a * cos(t); b *
sin(t)] + [x0 * ones(1, length(t)); y0 * ones(1, length(t))];

% Показываем эти элипсы
plot(firstCurve(1, :), firstCurve(2, :), 'r', secondCurve(1, :), secondCurve(2, :),
'g', thirdCurve(1, :), thirdCurve(2, :), 'b');
legend('first', 'second', 'third');
axis([-1.2 1.2 -1.2 1.2]);
grid on;

%% Создаем множество точек и разделяем его
% Генерируем точки, принадлежашие элипсам
firstDots = firstCurve(:, randperm(end, 60));
secondDots = secondCurve(:, randperm(end, 100));
thirdDots = thirdCurve(:, randperm(end, 120));

% Делим точки
[firstTraining, firstControl, firstTest] = dividerand(firstDots, 0.7, 0.2, 0.1);
[secondTraining, secondControl, secondTest] = dividerand(secondDots, 0.7, 0.2, 0.1);
[thirdTraining, thirdControl, thirdTest] = dividerand(thirdDots, 0.7, 0.2, 0.1);

% Визуализация разделения точек
```

```
p = plot(firstCurve(1, :), firstCurve(2, :), '-r', firstTraining(1, :), firstTraining(2,
:), 'or',  firstControl(1, :),   firstControl(2, :),   'rV', firstTest(1, :),
firstTest(2, :),  'rs', ...
         secondCurve(1, :), secondCurve(2, :), '-g', secondTraining(1, :),
secondTraining(2, :), 'og',  secondControl(1, :),   secondControl(2, :),   'gV',
secondTest(1, :),  secondTest(2, :),  'gs', ...
         thirdCurve(1, :), thirdCurve(2, :), '-b', thirdTraining(1, :),
thirdTraining(2, :), 'ob',  thirdControl(1, :),   thirdControl(2, :),   'bV',
thirdTest(1, :),   thirdTest(2, :), 'bs');

mSize = 5;
lWidth = 1;

p(1).LineWidth = lWidth;
p(2).MarkerEdgeColor = 'black';
p(2).MarkerFaceColor = 'r';
p(2).MarkerSize = mSize;
p(3).MarkerEdgeColor = 'black';
p(3).MarkerFaceColor = 'white';
p(3).MarkerSize = mSize;
p(4).MarkerEdgeColor = 'black';
p(4).MarkerFaceColor = 'black';
p(4).MarkerSize = mSize;

p(5).LineWidth = lWidth;
p(6).MarkerEdgeColor = 'black';
p(6).MarkerFaceColor = 'g';
p(6).MarkerSize = mSize;
p(7).MarkerEdgeColor = 'black';
p(7).MarkerFaceColor = 'white';
p(7).MarkerSize = mSize;
p(8).MarkerEdgeColor = 'black';
p(8).MarkerFaceColor = 'black';
p(8).MarkerSize = mSize;

p(9).LineWidth = lWidth;
p(10).MarkerEdgeColor = 'black';
p(10).MarkerFaceColor = 'b';
p(10).MarkerSize = mSize;
p(11).MarkerEdgeColor = 'black';
p(11).MarkerFaceColor = 'white';
p(11).MarkerSize = mSize;
p(12).MarkerEdgeColor = 'black';
p(12).MarkerFaceColor = 'black';
p(12).MarkerSize = mSize;

axis([-1.2 1.2 -1.2 1.2]);
legend('first:Curve', 'training', 'control', 'test', ...
       'second:Curve', 'training', 'control', 'test', ...
       'third:Curve', 'training', 'control', 'test');
grid on;

%% Готовим данные для обучения
X = [firstTraining secondTraining thirdTraining firstControl secondControl
thirdControl firstTest secondTest thirdTest];
```

```
y = [[1; 0; 0] * ones(1, length(firstTraining)) [0; 1; 0] * ones(1,
length(secondTraining)) [0; 0; 1] * ones(1, length(thirdTraining)) ...
     [1; 0; 0] * ones(1, length(firstControl)) [0; 1; 0] * ones(1,
length(secondControl)) [0; 0; 1] * ones(1, length(thirdControl)) ...
     [1; 0; 0] * ones(1, length(firstTest)) [0; 1; 0] * ones(1, length(secondTest))
[0; 0; 1] * ones(1, length(thirdTest))];

% Создаем сеть и обучаем ее
network = feedforwardnet(20);
network = configure(network, X, y);
network.inputs{1}.range = [-1.2 1.2; -1.2 1.2];
network.outputs{2}.range = [0 1; 0 1; 0 1];
network.layers{1}.transferFcn = 'tansig';
network.layers{2}.transferFcn = 'tansig';
network.trainFcn = 'trainrp';

trainingSetSize = length(firstTraining) + length(secondTraining) +
length(thirdTraining);
controlSetSize = length(firstControl) + length(secondControl) + length(thirdControl);
testSetSize = length(firstTest) + length(secondTest) + length(thirdTest);

network.divideFcn = 'divideind';
network.divideParam.trainInd = 1:trainingSetSize;
network.divideParam.valInd = trainingSetSize+1:trainingSetSize+controlSetSize;
network.divideParam.testInd =
trainingSetSize+controlSetSize+1:trainingSetSize+controlSetSize+testSetSize;

% Инициализируем веса и параментры обучениря
network = init(network);
network.trainParam.epochs = 1500;
network.trainParam.max_fail = 1500;
network.trainParam.goal = 1e-5;

% Обучаем сеть
network = train(network, X, y);

% Результаты обучения сети
traintingDotsProperClassified = sum(sum((sim(network, X(:, 1:trainingSetSize)) >=
0.5) == logical(y(:, 1:trainingSetSize)), 1) == 3);
controlDotsProperClassified = sum(sum((sim(network, X(:,
trainingSetSize+1:trainingSetSize+controlSetSize)) >= 0.5) == logical(y(:,
trainingSetSize+1:trainingSetSize+controlSetSize)), 1) == 3);
testDotsProperClassified = sum(sum((sim(network, X(:,
trainingSetSize+controlSetSize+1:trainingSetSize+controlSetSize+testSetSize)) >=
0.5) ==
logical(y(:,trainingSetSize+controlSetSize+1:trainingSetSize+controlSetSize+testSetS
ize)), 1) == 3);
fprintf('Обучающие: %d/%d\nПроверочные: %d/%d\nТестовые: %d/%d\n',
traintingDotsProperClassified, trainingSetSize, controlDotsProperClassified,
controlSetSize, testDotsProperClassified, testSetSize);

% Разделение по классам на картине
h = 0.025;
n = int32((1.2 + 1.2) / h) + 1;
```

```
x = zeros(2, n * n);
for i = 1:n
    for j = 1:n
        x(:, (i-1)*n + j) = [-1.2 + (double(i)-1)*h; 1.2 - (double(j)-1)*h];
    end
end
image(permute(reshape(sim(network, x), [3 n n]), [2 3 1]));

%% Этапы 2 и 3
f = @(t) sin(t .^ 2 - 7 * t);
t = 0:0.025:5;

% Обучающий набор
X = t;
y = f(t);

%Строим нейросеть с обучающей функцией trainscg
network = feedforwardnet(20);

network = configure(network, X, y);
network.layers{1}.transferFcn = 'tansig';
network.layers{2}.transferFcn = 'purelin';

network.trainFcn = 'trainscg';

trainingSetSize = ceil(length(X) * 0.9);
network.divideFcn = 'divideind';
network.divideParam.trainInd = 1:trainingSetSize;
network.divideParam.valInd = trainingSetSize+1:length(X);
network.divideParam.testInd = [];

% Инициализируем веса и параментры обучениря
network = init(network);
network.trainParam.epochs = 6000;
network.trainParam.max_fail = 6000;
network.trainParam.goal = 1e-10;

% Обучаем ее
network = train(network, X, y);

%% Результаты для обучающего подмножества
disp(dataForTable(sim(network, X(1:trainingSetSize)), y(1:trainingSetSize)));
p = plot(X(1:trainingSetSize), y(1:trainingSetSize), X(1:trainingSetSize),
sim(network, X(1:trainingSetSize)), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
xlabel('$t$');
ylabel('$y$');
grid on;

%% Результаты для контрольного подмножества
disp(dataForTable(sim(network, X(trainingSetSize+1:length(X))),
y(trainingSetSize+1:length(X))));
```

```
p = plot(X(trainingSetSize+1:length(X)), y(trainingSetSize+1:length(X)),
X(trainingSetSize+1:length(X)), sim(network, X(trainingSetSize+1:length(X))), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
xlabel('$t$');
ylabel('$y$');
grid on;

%% Строим нейросеть с обучающей функцией trainoss
network = feedforwardnet(20);

network = configure(network, X, y);
network.layers{1}.transferFcn = 'tansig';
network.layers{2}.transferFcn = 'purelin';

network.trainFcn = 'trainoss';

trainingSetSize = ceil(length(X) * 0.9);
network.divideFcn = 'divideind';
network.divideParam.trainInd = 1:trainingSetSize;
network.divideParam.valInd = trainingSetSize+1:length(X);
network.divideParam.testInd = [];

% Инициализируем веса и параментры обучениря
network = init(network);

network.trainParam.epochs = 6000;
network.trainParam.max_fail = 6000;
network.trainParam.goal = 1e-10;

% Обучаем ее
network = train(network, X, y);

%% Результаты для обучающего подмножества
disp(dataForTable(sim(network, X(1:trainingSetSize)), y(1:trainingSetSize)));
p = plot(X(1:trainingSetSize), y(1:trainingSetSize), X(1:trainingSetSize),
sim(network, X(1:trainingSetSize)), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
xlabel('$t$');
ylabel('$y$');
grid on;

%% Результаты для контрольного подмножества
disp(dataForTable(sim(network, X(trainingSetSize+1:length(X))),
y(trainingSetSize+1:length(X))));
p = plot(X(trainingSetSize+1:length(X)), y(trainingSetSize+1:length(X)),
X(trainingSetSize+1:length(X)), sim(network, X(trainingSetSize+1:length(X))), 'o');
p(1).Color = [1 0 0];
p(2).MarkerSize = 3;
p(2).Color = [0 0 0];
xlabel('$t$');
```

```
ylabel('$y$');
grid on;
```

dataForTable.m

```
function res =  dataForTable(y, yp)
    R2 = 1 - sum((y - yp) .^ 2)/sum((y - mean(y)) .^ 2);
    MSE = mse(y - yp);
    RMSE = sqrt(MSE);
    СКО = RMSE / (max(y) - min(y)) * 100;
    MAE = mae(y - yp);
    MinAbsErr = min(abs(y - yp));
    MaxAbsErr =  max(abs(y - yp));
    MAPE = mean(abs((y - yp) ./ y)) * 100;
    errors = abs((y - yp) ./ y) * 100;
    Under5PersentPortion = sum(errors < 5) / length(y) * 100;
    Under10PersentPortion = sum(5 <= errors & errors < 10) / length(y) * 100;
    Under20PersentPortion = sum(10 <= errors & errors < 20) / length(y) * 100;
    Under30PersentPortion = sum(20 <= errors & errors < 30) / length(y) * 100;
    Over30PersentPortion = sum(errors >= 30) / length(y) * 100;

    res = sprintf(['R^2: %f\n' ...
       'MSE: %f\n' ...
       'RMSE: %f\n' ...
       'Относительная СКО: %f%%\n' ...
       'MAE: %f\n'...
       'min abs err: %f\n' ...
       'max abs err: %f\n' ...
       'MAPE: %f\n' ...
       'Доля с ошибкой менее 5%%: %f%%\n' ...
       'Доля с ошибкой от 5%% до 10%%: %f%%\n'...
       'Доля с ошибкой от 10%% до 20%%: %f%%\n' ...
       'Доля с ошибкой от 20%% до 30%%: %f%%\n' ...
       'Доля с ошибкой более 30%%: %f%%\n'], ...
      R2, MSE, RMSE, СКО, MAE, MinAbsErr, MaxAbsErr, MAPE,  Under5PersentPortion,
Under10PersentPortion, Under20PersentPortion, Under30PersentPortion,
Over30PersentPortion);

end
```

Вывод

В этой лабораторной работе исследовал свойства многослойных нейронных сетей. Использовал их для классификации и для прокламации функций. Я решил все сети создавать с 20 нейронами на скрытом слое, чтобы была возможность объективно оценить скорость достижения приемлемого результата. В итоге, как и ожидалось, для задачи разделения линейнонеразделимых множеств многослойная нейросеть работает гораздо лучше, чем для аппроксимации функции и поле первого же этапа делит все точки из всех множеств верно. Для задачи аппроксимации функции я использовал метод Миллера и одношаговый метод секущих. Метод Миллера достиг приемлемой точности только после третьего этапа обучения, а одношаговый метод секущих — за один этап. Да, качество обучения у метода второго порядка получилось ниже, но это только потому, что обучение было запущено только один раз. После трех этапов обучения этот алгоритм показал точность выше, чем метод Миллера.