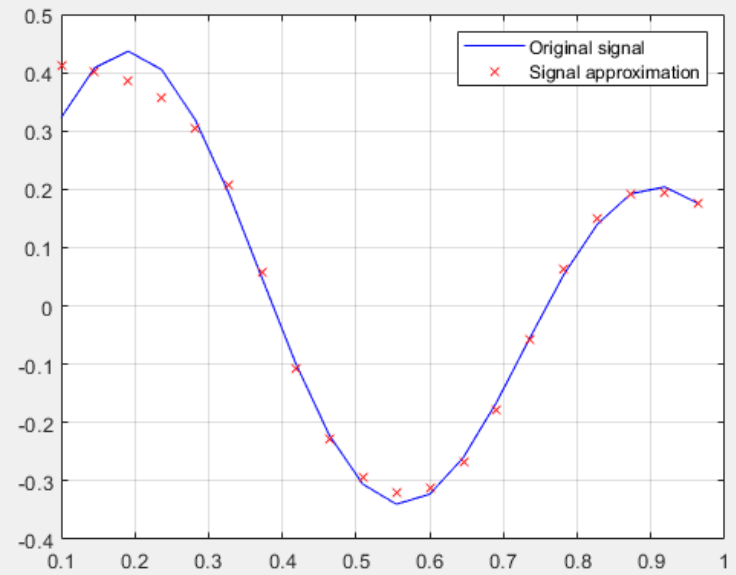
Intelektualiosios sistemos ND 2

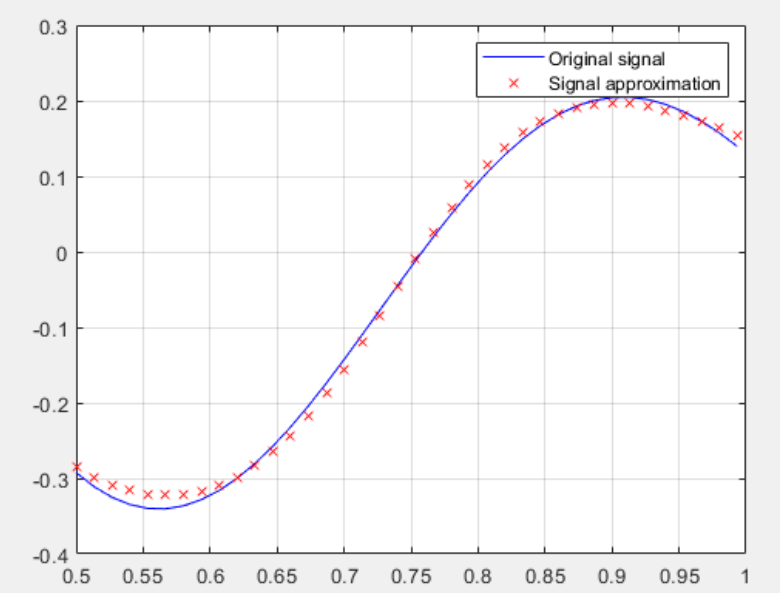
Šis namų darbas yra skirtas sukurti neuroninį tinkle skirtą aproksimuoti tam tikrą sumodeliuotą signalą. Vykstant duomenų kaitai tarp siųstuvo ir imtuvo realaus laiko programoms (RLP) ar sistemoms (RLS) svarbesnis greitis, nei tikslumas, todėl jos nenaudoje saugaus duomenų siuntimo, kuris garantuoja visų duomenų paketų pristatymą imtuvui. Dėl to duomenų praradimo atveju vartotojas pajaus programos trikdžius. Siekiant to išvengti imtuve galima naudoti aproksimacijos įrenginį – nesudėtingą neuronų tinklą, kuris turėdamas dalį duomenų paketų gales greitai sugeneruoti trūkstamų duomenų aproksimacijas, kurios bus naudojamos vietoj nepilnų duomenų paketų. Šiame darbe bus laikoma, kad sėkmingai perduotų duomenų dalis yra funkcijos įėjimo reikšmės (programoje tai kintamasis x), o prarasta duomenų dalis, tai funkcijos išeiga (programoje tai kintamasis Y). Šiam pavyzdžiui veikit reikės programos MATLAB.

Paprastos struktūros neuronų tinklas, sudarytas iš 4 neuronų, galės greitai ir pakankamai kokybiškai sukurti signalo aproksimaciją. Mokyme bus naudojamas backpropagation svorių atnaujinimo metodas, aktyvavimo funkcija – sigmoidė, mokymas bus atliekamas 1 kartą programos pradžioje ir turės 1000000 iteracijų, siekiant užtikrinti didelį tikslumą. Tinklo mokymo rezultatas pateiktas paveiksle numeriu 1.



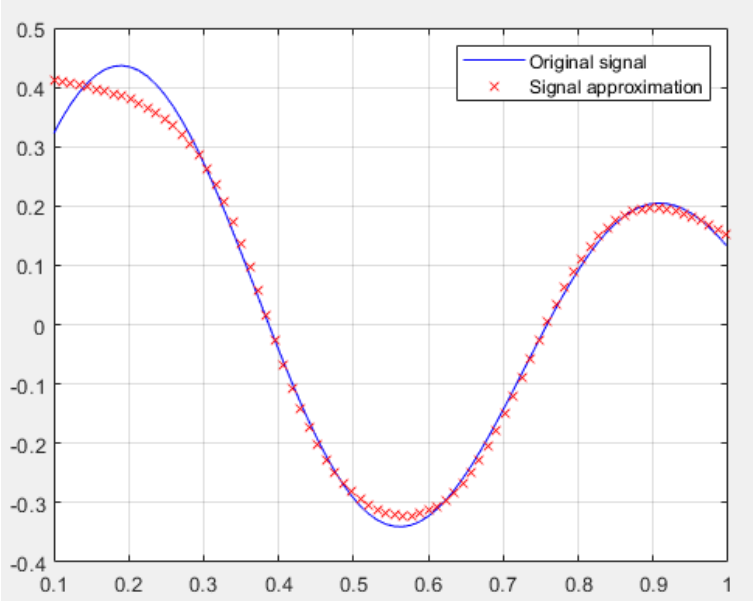
1 pav. Mokymo rezultatas

Po tinklo mokymo jam yra paduodamas pirmas testinis duomenų rinkinys. Pagal gautus duomenis tinklas, naudodamas mokyme sugeneruotus svorius, sukurs jų aproksimaciją. Ji pateikta paveiksle numeriu 2.



2 pav. Testinių duomenų rinkinio 1 aproksimacijos rezultatas

Kaip matyti iš 2 paveikslo neuronų tinklas gan neblogai susidorojo su savo užduotimi, minimalios signalo paklaidos neturės didelės įtakos rezultato kokybei. Šiuo atveju tinklo naudojamų duomenų dydis buvo vidutinis. Variantas su didesniu duomenų kiekiu pateiktas paveiksle numeriu 3.



3 pav. Testinių duomenų rinkinio 2 aproksimacijos rezultatas

Iš 3 paveikslo matyti, kad didėjant duomenų kiekiui didėja ir paklaida. Šiuo atveju didžiausios klaidų sankaupos yra signalo ekstremumai. Jų nukrypimai, nors ir yra padidėję, bet realaus laiko programai didelių nuostolių neatneš. Realaus laiko sistemoms svarbus greitis, o ne

tikslumas, todėl joms negalima naudoti sudėtingos, didelio neuronų kiekiu pagrįsto tinklo aproksimacijos sistemos, kurios skaičiavimai gali pareikalauti gan daug RLS ar RLP resursų ir sutrikdyti jų darbą. Galima didinti aproksimacijų tikslumą didinant mokymo iteracijų skaičių, bandant rinkti kitą aktyvavimo funkciją, plečiant mokymo duomenų rinkinius.

Programos kodas:

```
clear all;
close all;

% Aproximasion using back propogasion
x = 0.1:1/22:1; % input
d = (0.6*sin(2*pi*x/0.7) + 0.3*sin(2*pi*x))/2; %desired response

w11 = randn(1);
b11 = randn(1);
w12 = randn(1);
b12 = randn(1);
w13 = randn(1);
b13 = randn(1);
w14 = randn(1);
b14 = randn(1);

w21 = randn(1);
w22 = randn(1);
w23 = randn(1);
w24 = randn(1);
b21 = randn(1);
e = 0;
e_total =0;
n = 0.015;

for index = 1:length(x)
% Calculate hidden layer values, activation f(x) is a sigmoid
y1 = 1 / ( 1 + exp( -(x(index) * w11 + b11 )));
y2 = 1 / ( 1 + exp( -(x(index) * w12 + b12 )));
y3 = 1 / ( 1 + exp( -(x(index) * w13 + b13 )));
y4 = 1 / ( 1 + exp( -(x(index) * w14 + b14 )));

% Add all of the hidden layer values, activation f(x) is linear
y(index) = y1*w21 + y2*w22 + y3*w23 + y4*w24 + b21;

% Calculate error
e(index) = d(index) - y(index);
end;

% Update the weights
for m = 1:1000000 % executes while the total error is not 0
 % Update weights from the end layer "Backpropogating"

     for index = 1:length(x)
     w21 = w21 + n * e(index) * y1;
     w22 = w22 + n * e(index) * y2;
     w23 = w23 + n * e(index) * y3;
     w24 = w24 + n * e(index) * y4;
     b21 = b21 + n * e(index);
```

```
     end;

  %Update hidden layer weights
     for index = 1:length(x)
     %Gradient calculation
     gradient1 = 1 / (1 + exp( -(x(index) * w11 + b11))) * (1-( 1 / (1 +
exp( -(x(index) * w11 + b11))))) * e(index) * w21;
     w11 = w11 + n * gradient1 * x(index);
     b11 = b11 + n * gradient1;

     gradient2 = 1 / (1 + exp( -(x(index) * w12 + b12))) * (1-( 1 / (1 +
exp( -(x(index) * w12 + b12))))) * e(index) * w22;
     w12 = w12 + n * gradient2 * x(index);
     b12 = b12 + n * gradient2;

     gradient3 = 1 / (1 + exp( -(x(index) * w13 + b13))) * (1-( 1 / (1 +
exp( -(x(index) * w13 + b13))))) * e(index) * w23;
     w13 = w13 + n * gradient3 * x(index);
     b13 = b13 + n * gradient3;

     gradient4 = 1 / (1 + exp( -(x(index) * w14 + b14))) * (1-( 1 / (1 +
exp( -(x(index) * w14 + b14))))) * e(index) * w24;
     w14 = w14 + n * gradient4 * x(index);
     b14 = b14 + n * gradient4;
     end;

        for index = 1:length(x)
        % Calculate hidden layer values, activation f(x) is a sigmoid
        y1 = 1 / ( 1 + exp( -(x(index) * w11 + b11 )));
        y2 = 1 / ( 1 + exp( -(x(index) * w12 + b12 )));
        y3 = 1 / ( 1 + exp( -(x(index) * w13 + b13 )));
        y4 = 1 / ( 1 + exp( -(x(index) * w14 + b14 )));

        % Add all of the hidden layer values, activation f(x) is linear
        y(index) = y1*w21 + y2*w22 + y3*w23 + y4*w24 + b21;

        % Calculate error
        e(index) = d(index) - y(index);
        end;
end;


%learning data set-----------------------------------------------------------
  % Solve equation with new values
        for index = 1:length(x)
        % Calculate hidden layer values, activation f(x) is a sigmoid
        y1 = 1 / ( 1 + exp( -(x(index) * w11 + b11 )));
        y2 = 1 / ( 1 + exp( -(x(index) * w12 + b12 )));
        y3 = 1 / ( 1 + exp( -(x(index) * w13 + b13 )));
        y4 = 1 / ( 1 + exp( -(x(index) * w14 + b14 )));

        % Add all of the hidden layer values, activation f(x) is linear
        Y(index) = y1*w21 + y2*w22 + y3*w23 + y4*w24 + b21;
        end;

figure('Name','Learning data set approximation result');
plot(x,d, 'b', x,Y, 'rx');
grid on;
legend('Original signal','Signal approximation');
```

```
%test data set 1-------------------------------------------------------
clear Y d;
x = 0.5:0.4/30:1; %input values
d = (0.6*sin(2*pi*x/0.7) + 0.3*sin(2*pi*x))/2; %desired response
  % Solve equation with new values
        for index = 1:length(x)
        % Calculate hidden layer values, activation f(x) is a sigmoid
        y1 = 1 / ( 1 + exp( -(x(index) * w11 + b11 )));
        y2 = 1 / ( 1 + exp( -(x(index) * w12 + b12 )));
        y3 = 1 / ( 1 + exp( -(x(index) * w13 + b13 )));
        y4 = 1 / ( 1 + exp( -(x(index) * w14 + b14 )));

        % Add all of the hidden layer values, activation f(x) is linear
        Y(index) = y1*w21 + y2*w22 + y3*w23 + y4*w24 + b21;
        end;

figure('Name','Test data set 1 approximation result');
plot(x,d, 'b', x,Y, 'rx');
grid on;
legend('Original signal','Signal approximation'); %error max 0.05



%test data set 2-------------------------------------------------------
clear Y d;
x = 0.1:0.5/44:1; %input values
d = (0.6*sin(2*pi*x/0.7) + 0.3*sin(2*pi*x))/2; %desired response
  % Solve equation with new values
        for index = 1:length(x)
        % Calculate hidden layer values, activation f(x) is a sigmoid
        y1 = 1 / ( 1 + exp( -(x(index) * w11 + b11 )));
        y2 = 1 / ( 1 + exp( -(x(index) * w12 + b12 )));
        y3 = 1 / ( 1 + exp( -(x(index) * w13 + b13 )));
        y4 = 1 / ( 1 + exp( -(x(index) * w14 + b14 )));

        % Add all of the hidden layer values, activation f(x) is linear
        Y(index) = y1*w21 + y2*w22 + y3*w23 + y4*w24 + b21;
        end;

figure('Name','Test data set 2 approximation result');
plot(x,d, 'b', x,Y, 'rx');
grid on;
legend('Original signal','Signal approximation');
```