Sieci neuronowe

Paweł Głogowski

1. Cel badań

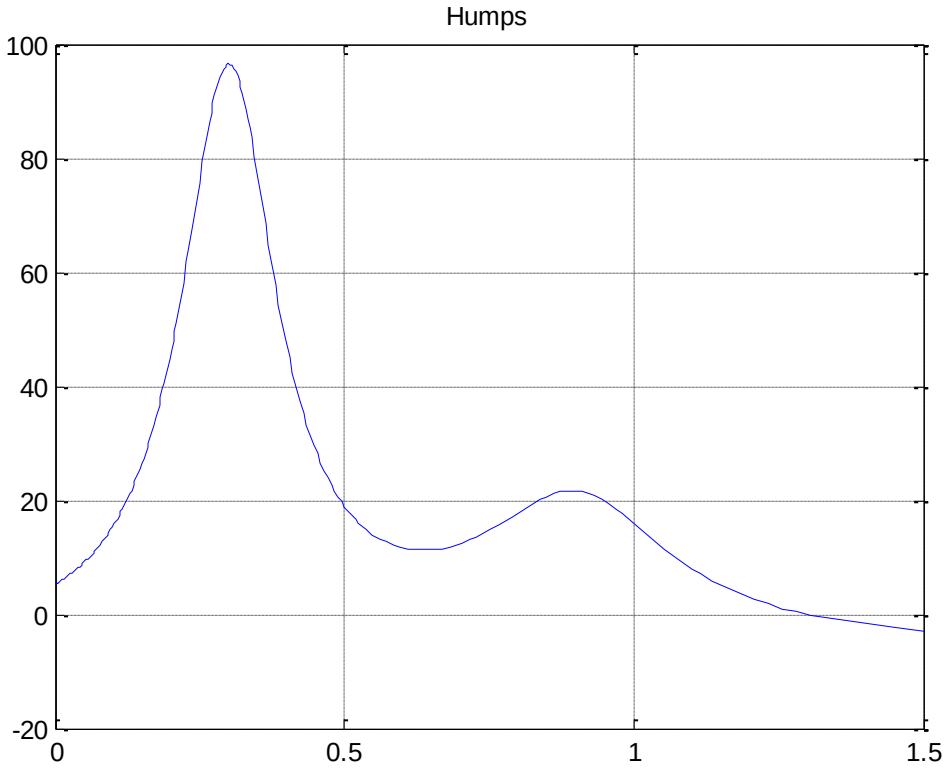
Celem badań jest wygenerowanie zbioru danych uczących [P,T] tak, aby:

- Aproksymować funkcję Humps

  Wzór funkcji:

$$fplot('1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6',[0,1.5]),grid$$

  Wykres funkcji:

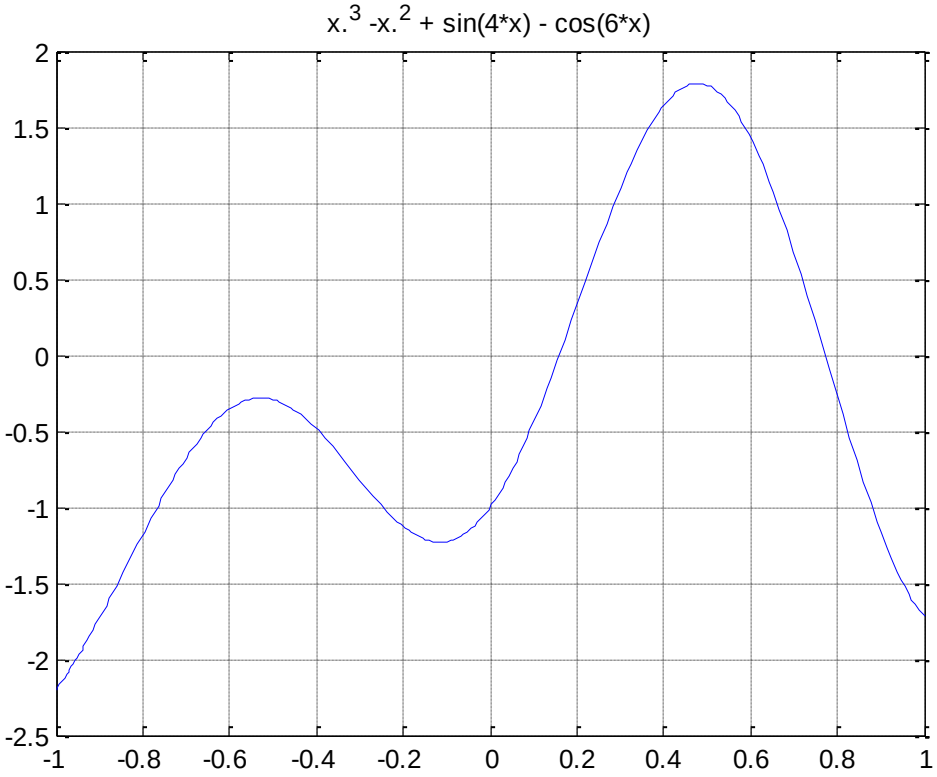


- Aproksymować funkcję opisaną poniżej

  Wzór funkcji:

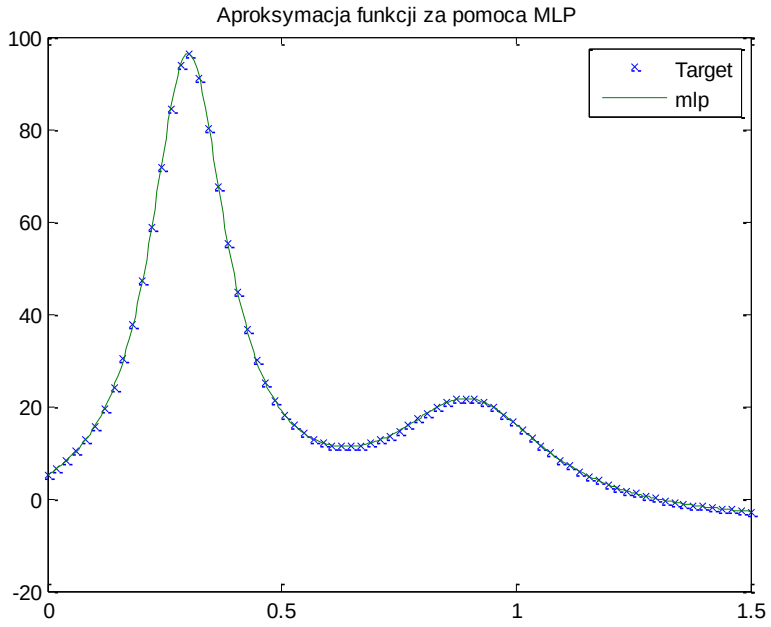
$$fplot('x.^3-x.^2+\sin(4*x)-\cos(6*x)',[-1,1]),grid\ \%8$$

  Wykres funkcji:



2. Aproksymacja funkcji Humps za pomocą MLP

Funkcja obliczająca: humps_MLP.m

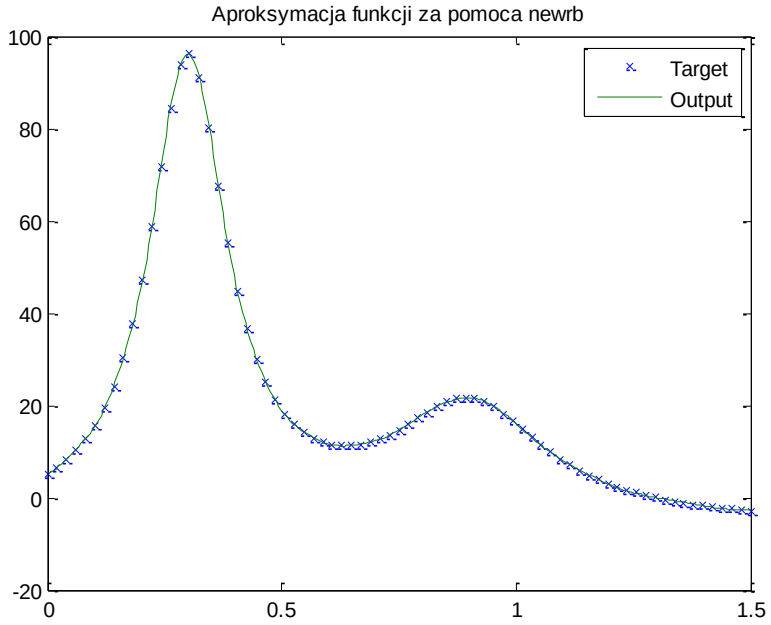



```
%obliczanie wartosci funkcji (dane treningowe)
P = linspace(0, 1.5, 75);
T = humps(P);


%parametry sieci
net = newff(P,T);
hiddenSizes = 20;
net = fitnet(hiddenSizes);  %dopasowanie sieci
net.trainFcn = 'trainbr';   %Bayesian regulation backpropagation
net.trainParam.show = 50;
net.trainParam.lr = 0.05;   %Learning rate
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;


%uczenie sieci
[net,tr] = train(net,P,T);
%dane testowe (wiecej punktow niz przy danych treningowych)
x = linspace(0, 1.5, 250);
%aproksymacja funkcji przy pomocy sieci
y = sim(net,x);
%rysowanie wykresu dla funkcji rzeczywistej i aproksymowanej
plotperform(tr)
figure, plot(P,T,'x',x,y)
title('Aproksymacja funkcji za pomoca MLP');
legend({'Target','mlp'});
```

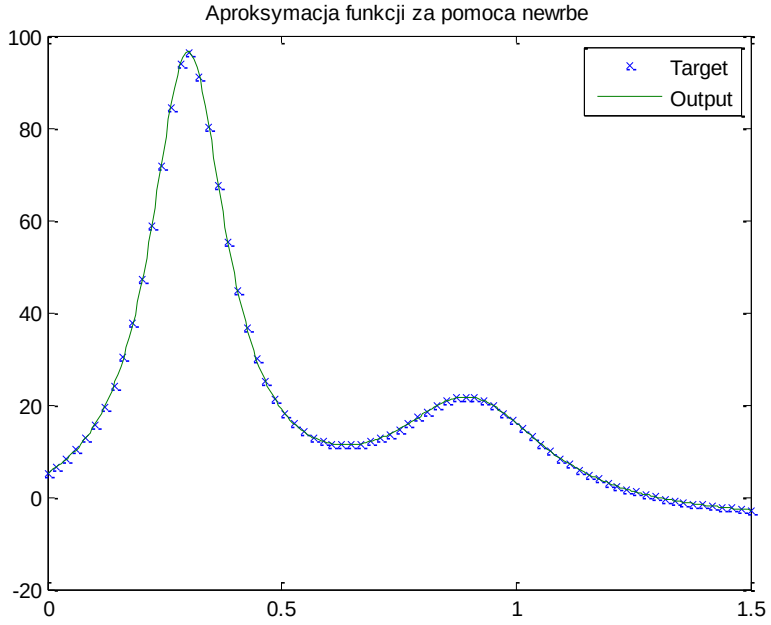
- Aproksymację wykonano przy pomocy funkcji uczenia **trainbr** oraz 20 wartsw ukrytych

3. Aproksymacja funkcji Humps za pomocą RBF

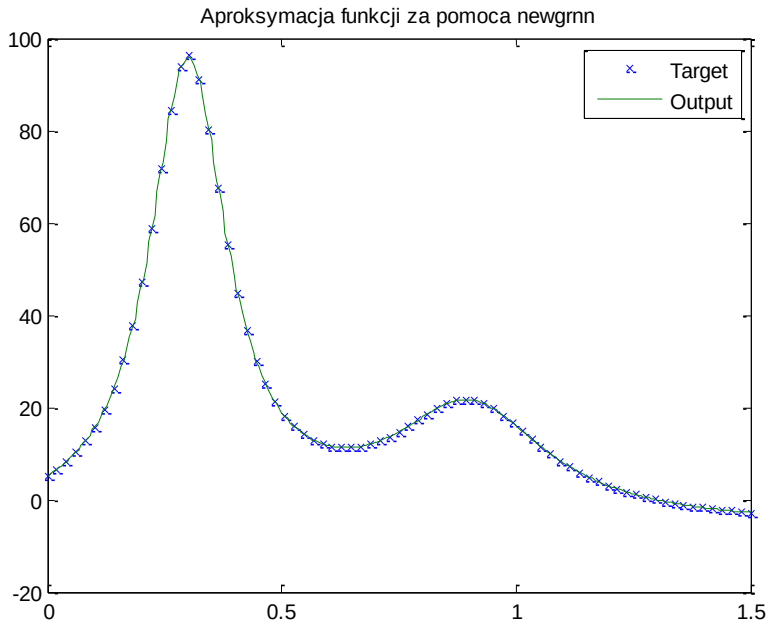
Funkcja obliczająca: humps_RBF.m




```
%obliczanie wartosci funkcji (dane treningowe)
P = linspace(0, 1.5, 75);
T = humps(P);
%parametry sieci RBF
eg = 0.02;  % sum-squared error goal (błąd
sredniokwadratowy)
sc = 0.1;   % spread constant (stała rozpietosc
funkcji Gaussa)


%newrb - aproksymacja za pomocą newrb
net = newrb(P,T,eg,sc);
x = linspace(0, 1.5, 250);
y = sim(net,x);
hold on;
figure, plot(P,T,'x',x,y);
title('Aproksymacja funkcji za pomoca newrb');
legend({'Target','Output'});
hold off;
```




```
% newrbe - aproksymacja za pomocą newrbe
sc = 0.05;
net1 = newrbe(P,T, sc);
y1 = sim(net1,x);
hold on;
figure, plot(P,T,'x',x,y1);
title('Aproksymacja funkcji za pomoca newrbe');
legend({'Target','Output'});
hold off;
```

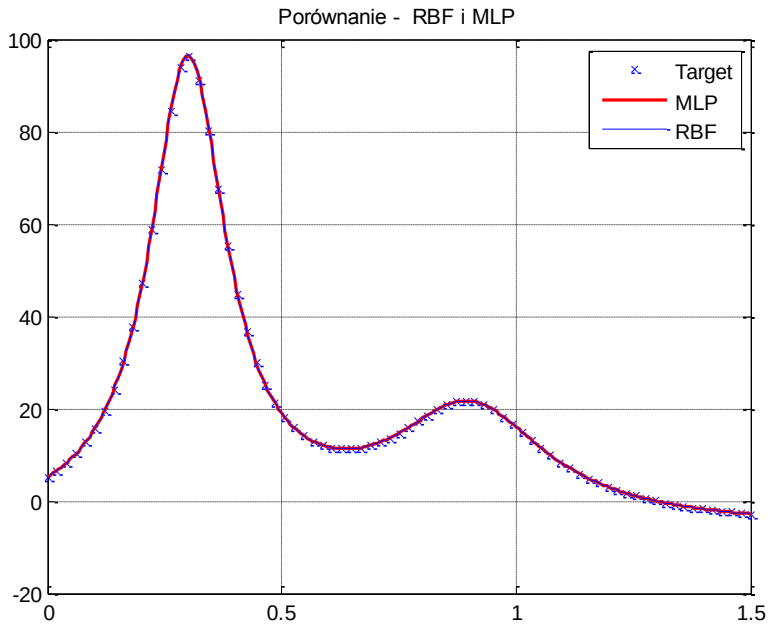



```
% newgrnn - aproksymcja za pomocą newgrnn
sc = 0.01;
net2 = newgrnn(P,T,sc);
y2 = sim(net2,x);
hold on;
figure, plot(P,T,'x',x,y2);
title('Aproksymacja funkcji za pomoca
newgrnn');
legend({'Target','Output'});
hold off;
```

- Najlepsze wyniki aproksymacji uzyskano przy użyciu funkcji newrb
- Dla każdej funkcji należało oddzielnie dopasować wartość stałej rozpiętości funkcji Gaussa

4. Porównanie aproksymacji funkcji Humps na jednym wykresie dla sieci MLP i RBF

Funkcja obliczająca: humps_porownanie.m



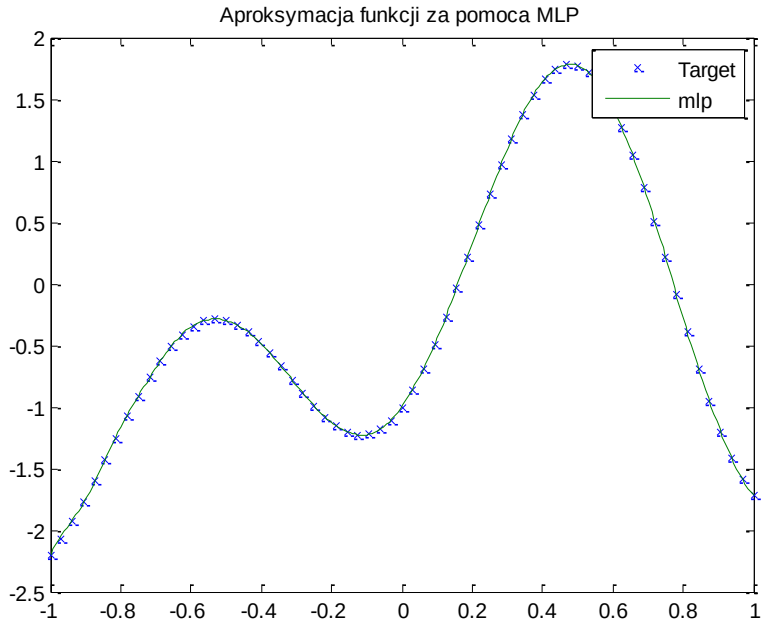

```
%obliczanie wartosci funkcji (dane
treningowe)
P = linspace(0, 1.5, 75);
T = humps(P);


%parametry sieci
net = newff(P,T);
hiddenSizes = 20;
net = fitnet(hiddenSizes);
net.trainFcn = 'trainbr';   %Bayesian
regulation backpropagation
net.trainParam.show = 50;
net.trainParam.lr = 0.05;   %Learning rate
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
%uczenie sieci
[net,tr] = train(net,P,T);
%dane testowe (wiecej punktow niz przy
danych treningowych)
x = linspace(0, 1.5, 250);
%aproksymacja funkcji przy pomocy sieci
y = sim(net,x);


%parametry sieci RBF
eg = 0.00;
sc = 0.05;


%newrb - aproksymacja za pomocą newrb
net1 = newrb(P,T,eg,sc);
x1 = linspace(0, 1.5, 250);
y1 = sim(net1,x1);


figure;
plot(P,T,'x');
hold on;
plot(x,y,'r','LineWidth', 2);
hold on;
plot(x1,y1,'b','LineWidth', 1);
hold on;
legend({'Target','MLP','RBF'});
hold on;
```

```
title('Porównanie -  RBF i MLP');
grid;
hold off;
```

- Do uczenia sieci neuronowej zostało użyte 75 liniowo rozłożonych punktów wyliczonych przez funkcję Humps. Przy pomocy utworzonych sieci MLP oraz RBF zostało wygenerowane 250 aproksymowanych punktów odzwierciedlających funkcję.
- Wyniki aproksymacji dla sieci MLP i RBF są bardzo zbliżone.

5. Aproksymacja funkcji y = $x.\hat{}3 - x.\hat{}2 + \sin(4 * x) - \cos(6 * x)$ za pomocą MLP

Funkcja obliczająca: fun_MLP.m

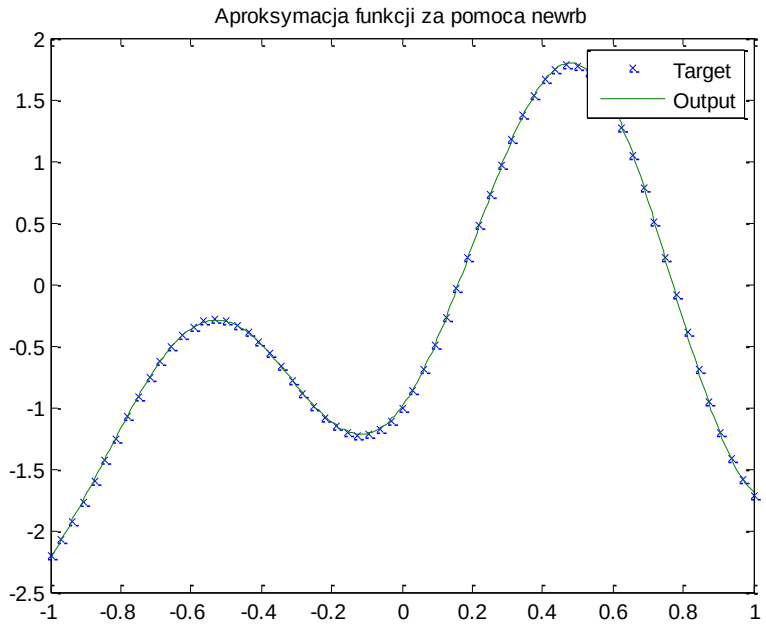



```
%obliczanie wartosci funkcji (dane
treningowe)
P = linspace(-1, 1, 65);
T = fun(P);


%parametry sieci
net = newff(P,T);
hiddenSizes = 10;
net = fitnet(hiddenSizes);  %dopasowanie
sieci
net.trainFcn = 'trainlm';   %Levenberg-
Marquardt backpropagation
net.trainParam.show = 50;
net.trainParam.lr = 0.1;    %Learning rate
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;


%uczenie sieci
[net,tr] = train(net,P,T);
%dane testowe (wiecej punktow niz przy
danych treningowych)
x = linspace(-1, 1, 250);
%aproksymacja funkcji przy pomocy sieci
y = sim(net,x);
%rysowanie wykresu dla funkcji
rzeczywistej i aproksymowanej
plotperform(tr)
figure, plot(P,T,'x',x,y)
title('Aproksymacja funkcji za pomoca
MLP');
legend({'Target','mlp'});
```

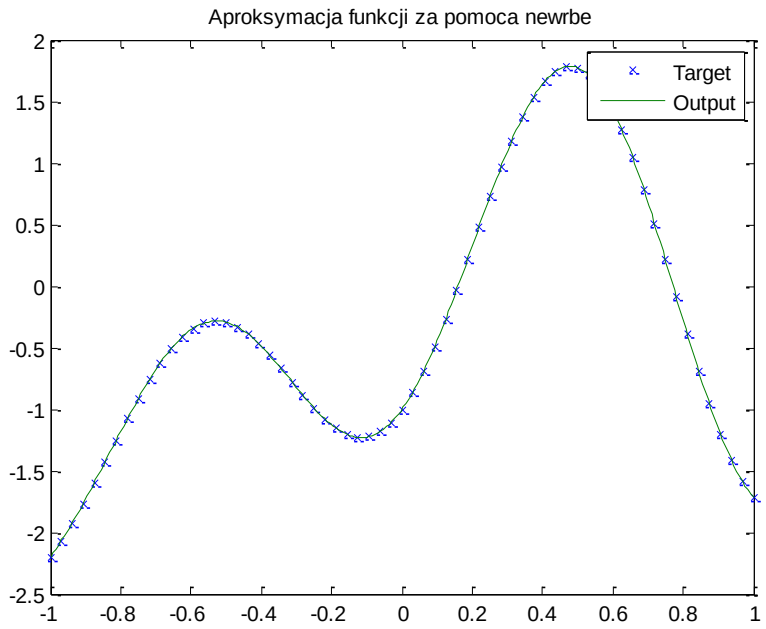
- Aproksymację wykonano przy użyciu funkcji uczelnia **trainlm** oraz 10 warstw ukrytych

6. Aproksymacja funkcji y = $x.\hat{}3 - x.\hat{}2 + \sin(4 * x) - \cos(6 * x)$ za pomocą RBF

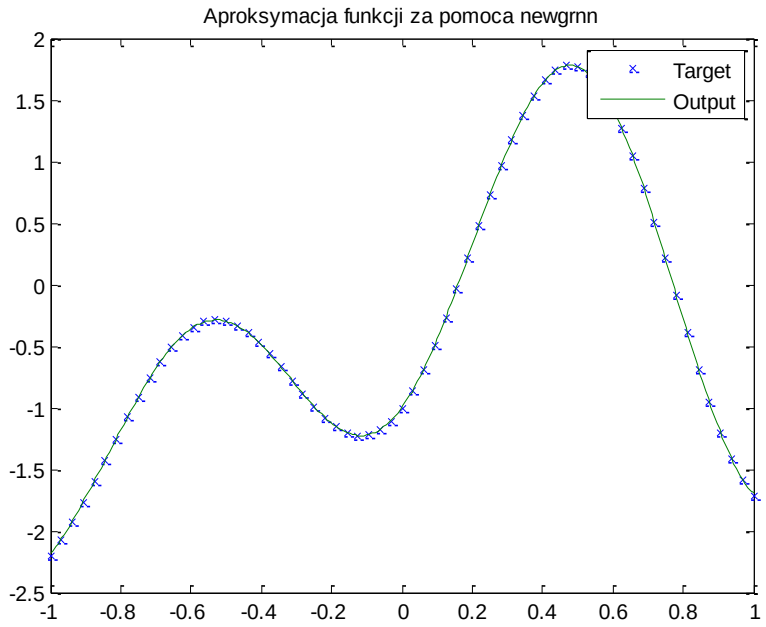
Funkcja obliczająca: fun_RBF.m




```
%obliczanie wartosci funkcji (dane
treningowe)
P = linspace(-1, 1, 65);
T = fun(P);
%parametry sieci RBF
eg = 0.001;  % sum-squared error goal (błąd
sredniokwadratowy)
sc = 1;    % spread constant (stała
rozpietosc funkcji Gaussa)


%newrb - aproksymacja za pomocą newrb
net = newrb(P,T,eg,sc);
x = linspace(-1, 1, 250);
y = sim(net,x);
hold on;
figure, plot(P,T,'x',x,y);
title('Aproksymacja funkcji za pomoca
newrb');
legend({'Target','Output'});
hold off;
```




```
% newrbe - aproksymacja za pomocą newrbe
sc = 0.1;
net1 = newrbe(P,T, sc);
y1 = sim(net1,x);
hold on;
figure, plot(P,T,'x',x,y1);
title('Aproksymacja funkcji za pomoca
newrbe');
legend({'Target','Output'});
hold off;
```

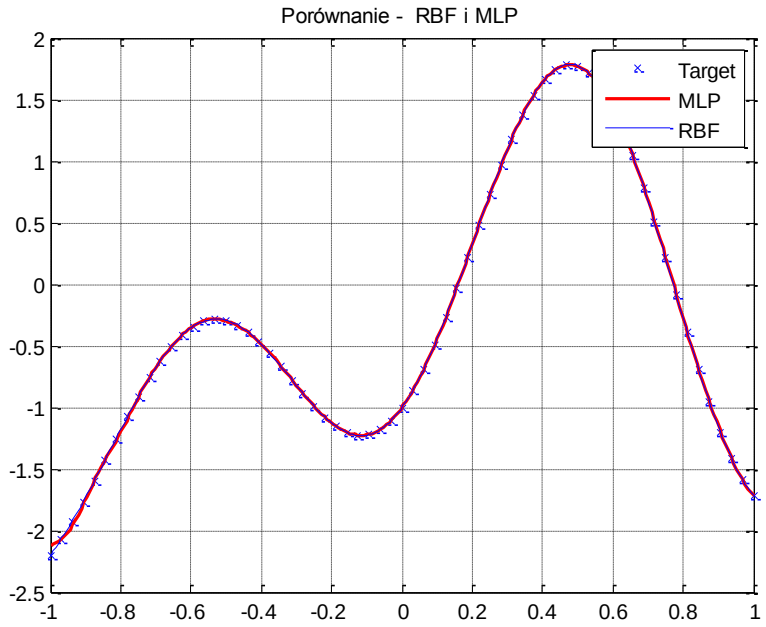



```
% newgrnn - aproksymcja za pomocą newgrnn
sc = 0.02;
net2 = newgrnn(P,T,sc);
y2 = sim(net2,x);
hold on;
figure, plot(P,T,'x',x,y2);
title('Aproksymacja funkcji za pomoca
newgrnn');
legend({'Target','Output'});
hold off;
```

- Najlepsze wyniki aproksymacji uzyskano przy użyciu funkcji newrbe
- Dla każdej funkcji należało oddzielnie dopasować wartość stałej rozpiętości funkcji Gaussa

7. Porównanie aproksymacji funkcji y = $x.\hat{}3 - x.\hat{}2 + \sin(4 * x) - \cos(6 * x)$

Funkcja obliczająca: fun_porownanie.m




```
%obliczanie wartosci funkcji (dane
treningowe)
P = linspace(-1, 1, 65);
T = fun(P);


%parametry sieci
net = newff(P,T);
hiddenSizes = 10;
net = fitnet(hiddenSizes);
net.trainFcn = 'trainlm';   %Levenberg-
Marquardt backpropagation
net.trainParam.show = 50;
net.trainParam.lr = 0.1;    %Learning rate
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
%uczenie sieci
[net,tr] = train(net,P,T);
%dane testowe (wiecej punktow niz przy
danych treningowych)
x = linspace(-1, 1, 250);
%aproksymacja funkcji przy pomocy sieci
y = sim(net,x);


%parametry sieci RBF
sc = 0.1;


% newrbe - aproksymacja za pomocą newrbe
net1 = newrbe(P,T, sc);
x1 = linspace(-1, 1, 250);
y1 = sim(net1,x1);


figure;
plot(P,T,'x');
hold on;
plot(x,y,'r','LineWidth', 2);
hold on;
plot(x1,y1,'b','LineWidth', 1);
hold on;
legend({'Target','MLP','RBF'});
```

```
hold on;
title('Porównanie -  RBF i MLP');
grid;
hold off;
```

‘

- Do uczenia sieci neuronowej zostało użyte 65 liniowo rozłożonych punktów wyliczonych przez badaną funkcję. Przy pomocy utworzonych sieci MLP oraz RBF zostało wygenerowane 250 aproksymowanych punktów odzwierciedlających funkcję.
- Na powyższym wykresie widzimy, że tak jaki w przypadku funkcji Humps, tak i dla indywidualnie przydzielonej funkcji wyniki aproksymacji obiema sieciami (MLP i RBF) dają bardzo zbliżone wyniki.

Wnioski:

Aproksymacja obu funkcji przy użyciu sieci MLP oraz RBF dała satysfakcjonujące wyniki. Przy odpowiednio dobranych parametrach sieci zostały prawidłowo nauczone. Ciężko określić, która z sieci jest lepsza do aproksymacji tego rodzaju funkcji. Na korzyść sieci RBF przemawia szybsze otrzymywanie mniejszych wartości błędów oraz łatwiejsza konfiguracja parametrów.

Pytania:

- Czy w przypadku danych uczących, które nie są liniowo separowane, perceptron jest w stanie nauczyć się odpowiednio klasyfikować takie dane?
    - Nie jest w stanie - pojedynczy perceptron klasyfikuje tylko liniowo separowane zbiory. Perceptron potrzebuje jasno określonej granicy między danymi, reakcja może być tylko pozytywna lub negatywna. Do klasyfikacji zbiorów separowanych nieliniowo potrzebowalibyśmy stworzyć sieć perceptronów.

- Czy zawsze ze wzrostem liczby neuronów warstwy ukrytej otrzymujemy mniejszy błąd uczenia przy tej samej liczbie iteracji(epok)?
    - Nie. W zależności od dobranych parametrów możemy sprawić iż sieć zacznie się przeuczać (jeśli iteracji będzie za wiele), albo pozostawimy ją niedouczoną (gdy iteracji będzie za mało). W pierwszym przypadku doprowadzimy do tego, iż będzie zbyt mocno reagować nawet na drobne anomalia we wprowadzonych danych, w drugim źle zaproksymuje naszą funkcję. Należy dobrać parametry z umiarem, po odpowiedniej obserwacji.

- Jaki wpływ ma zmiana funkcji aktywacji w sieci MLP, na uzyskane wyniki aproksymacji obu funkcji?

    - Gdy dokonamy zmiany funkcji aktywacji otrzymujemy wynik który rożni się od poprawnego wyniku. Przy wyborze nieodpowiedniej funkcji aproksymacja naszej funkcji staje się nie możliwa lub będzie obarczona dużym błędem.

- Jaki algorytm uczenia sieci MLP, zaimplementowany jako opcja funkcji train, wybrano do aproksymacji obu zadanych funkcji?

    - Do aproksymacji funkcji , jako opcja funkcji train wybrałem algorytm trainbr (Levenberga-Marquardta).Najlepiej z możliwych dawał wynik najbliższy prawidłowemu.

- Jaki wpływ na efekt uczenia sieci RBF ma zmiana liczby neuronów radialnych w warstwie ukrytej?

    - Gdy zwiększamy liczbę neuronów radialnych w warstwie ukrytej błąd uczenia staje się coraz mniejszy. Po pewnym czasie jednak zmiany są bardzo małe i prawie nie znaczące.

- Jaki algorytm uczenia sieci jest zaimplementowany w M-pliku rbfndm1.m oraz w funkcjach newrb, newrbe, newgrnn z Neural Network Toolbox?

    - Algorytm wstecznej propagacji błędów.