Zastosowanie pakietu Matlab i przybornika Neural Networks Toolbox w optymalnym projektowaniu filtrów aktywnych

Karol Józefowicz
Państwowa Wyższa Szkoła Zawodowa w Lesznie
Instytut Politechniczny
64-100 Leszno, ul. Mickiewicza 5, e-mail: k.jozefowicz@pwsz.edu.pl

Summary – In the article we offered using the MATLAB – program to make designing calculations of the high-order active filters , along with simultaneous process of finding the best combination of the designed system. We used all abilities, whitch are given by the MATLAB pakiet - not only in faster construktion and testing the artificial algoritms of the neural networks (ANNs), but also in veryfing statistical properties of our new system. We chose for our examinations – the RC IRS **(*Identical Root Sensitivities*)** high-order active filters as an example of using neural networks in the mentioned optimal designing task.

Streszczenie - W artykule przedstawiono propozycję wykorzystania interakcyjnego środowiska MATLAB do obliczeń projektowych filtrów aktywnych wyższych rzędów wraz z równoczesnym procesem optymalizacji projektowanego układu. Wykorzystano możliwości, jakie daje pakiet MATLAB w szybkiej konstrukcji i testowaniu algorytmów sztucznych sieci neuronowych a także weryfikacji własności statystycznych uzyskanego projektu. Do badań wybrano filtr aktywny RC IRS (*Identical Root Sensitivities*) jako przykład zastosowania SN w omawianym zadaniu optymalnego projektowania, gdzie wybraną funkcją celu była wieloparametrowa funkcja wrażliwości.

I. WSTĘP

W wielu zagadnieniach projektowania, zwłaszcza tam, gdzie wymagana jest optymalizacja parametrów wyjściowych projektowanego układu, a parametry dobierane układu mogą podlegać pewnym fluktuacjom, metody optymalizacji oparte o sieciach neuronowych są coraz to bardziej konkurencyjne w stosunku do konwencjonalnych metod optymalizacji.

W artykule rozważane jest zadanie, polegające na zaprojektowaniu wartości parametrów filtru przez dobór wartości współczynników transmitancji z określoną procentową dokładnością (funkcja $F_1(\boldsymbol{x})$), przy jednoczesnej minimalizacji odpowiedniej, wybranej funkcji celu (funkcja $F_2(\boldsymbol{x})$), która wiąże fluktuacje projektowanych parametrów z wybraną funkcją układową. Zadanie to jest typowym zadaniem optymalizacji wielokryterialnej (MCO – Multi-Criteria Optimization, z wagami w_i, uwzględniającymi rolę poszczególnych kryteriów w globalnej funkcji celu $F(\mathbf{x})$) i może być sformułowane np. w postaci:

$$F(\boldsymbol{x}) = w_1 \cdot F_1(\boldsymbol{x}) + w_2 \cdot F_2(\boldsymbol{x}) \qquad (1)$$

Zakłada się dalej, że rozważane są następujące funkcje celu ($F_2(x)$), kryteria optymalizacji: wieloparametrowa funkcja Schoefflera lub funkcja „najgorszego przypadku", bądź też odpowiadające tym kryteriom wariancje modułu transmitancji filtru.

Wynika stąd, że funkcjonał $F(\boldsymbol{x})$ globalnej optymalizacji (w postaci wzoru (1)) ma „wymienny funkcjonał" $F_2(\boldsymbol{x})$, przy ustalonej postaci funkcjonału $F_1(\boldsymbol{x})$, jak w zależności (2).

Wzór (2) zapewnia realizację transmitancji metodą porównywania współczynników przez minimalizację odpowiedniego funkcjonału (błędu średniokwadratowego), przy czym współczynniki transmitancji a_0, b_0, mają wskaźniki zmienione odpowiednio na: a_1, b_{n+2}, gdzie n – stopień wielomianu licznika, m – stopień wielomianu mianownika transmitancji.

$$F_1(x) = \left\{ \sum_{k=1}^{n+m+2} \left(f_1(x) - a_1 \right)^2 + \left(f_k(x) - b_k \right)^2 \right\}^{\frac{1}{2}} \qquad (2)$$

Z uwagi na to, że kryterium optymalizacyjne - $F_2(\boldsymbol{x})$ ma także postać funkcji opisanej błędem średniokwadratowym, można przyjąć, że także funkcja $F(\mathbf{x})$ jest pośrednio funkcją błędu średniokwadratowego.

II. PRZYKŁAD REALIZACJI OPTYMALNEGO PROJEKTOWANIA Z WYKORZYSTANIEM MATLAB (NNT)

Do badań w niniejszej pracy wykorzystano symulacje i program uczenia dla jednokierunkowej sieci wielowarstwowej napisany w programie MATLAB, który jest szeroko znanym i bardzo często wykorzystywanym pakietem obliczeniowym.

1). GLOBALNE OPTIMUM I OGÓLNE ZASADY OPTYMALIZACJI WIELOPARAMETROWEJ FUNKCJI WRAŻLIWOŚCI

Niech $T(s)$ oznacza transmitancję filtru wyższego rzędu typu *IRS* [14, 15] w postaci:

$$T(s) = f\left[g_1(s), \ldots, g_k(s), \ldots, g_n(s) \right] \qquad (3)$$

gdzie $g_k(s)$ są ogniwami pierwszego lub drugiego rzędu ($k = 1,.., N_f$).

Zakładając, że przedział pulsacji został narzucony i biorąc pod uwagę, że zarówno odchylenie standardowe, jak i wariancja funkcji układowej zależą od pulsacji sygnału, wyznacza się ograniczenie wartości wariancji (odchylenia standardowego) możliwe do uzyskania przy optymalizacji, czyli globalne minimum wariancji:

$$\sigma^2 \geq \frac{1}{3} \cdot \frac{c_s^2}{N} \cdot \left| \mathrm{S}_{\omega}^{T}(\omega) \right|^2 \tag{4}$$

Parametrami dobieranymi w procesie optymalizacji są oczywiście wartości parametrów układowych x_i oraz ich tolerancje względne (procentowe) d_i.

Ogólna idea procesu optymalizacji jest następująca: przy iteracyjnym charakterze całego procesu, powiększane są wartości tolerancji elementów przy jednoczesnym przesuwaniu centrum wektora (wartości nominalnych) dla parametrów układowych (zagadnienie centrowania). W celu sprawnego prowadzenia procesu optymalizacji przy doborze tolerancji parametrów d_i funkcja Schoefflera zostaje przekształcona (w miejsce x_i podstawia się odpowiednio d_i). W tym celu funkcję Schoefflera można zapisać następująco:

$$\left| \mathrm{S}_{x}^{T}(\omega) \right|^2 = \sum_i \left| \mathrm{S}_{x_i}^{T(j\omega)} \right|^2 \tag{5}$$

lub

$$\left| \mathrm{S}_{d}^{T}(\omega) \right|^2 = \sum_i \left| \mathrm{S}_{d_i}^{T(j\omega)} \right|^2, \qquad d_i = w_i \cdot d_{x_i} = \Delta x_i / x_i \tag{6}$$

przy czym w_i jest wagą przypisaną arbitralnie do nominalnej wartości tolerancji $d_{x,i}$.

Podobnie postępuje się w procedurze „centrowania", czyli przesuwania parametrów nominalnych układu.

2). ZASADY REALIZACJI SIECI NEURONOWEJ W ROZWAŻANYM ZADANIU OPTYMALIZACJI

Projektowanie sieci neuronowej zaczyna się na poziomie analizy sformułowanego problemu. Inaczej mówiąc, jakie i ile danych chcemy lub możemy podać na wejścia sieci determinuje wielkość warstwy wejściowej, a to, jaką odpowiedź chcemy uzyskać, ilość wyjść sieci. Dla zilustrowania prezentowanej metody projektowania wybrano filtr dolnoprzepustowy o transmitancji napięciowo-napięciowej (Butterwortha) - wzór (9) [11]. Zastosowany przy tym funkcjonał $F(\boldsymbol{x})$ globalnej optymalizacji - wzór (1), zawierał wagi odpowiednio: w_1=2.0, w_2=1.0.

W celu realizacji filtru wykorzystano strukturę układu zaproponowanego przez Kerwina-Huelsmana-Newcomba. Układ ten przedstawiony został na rys.1. Transmitancja tego układu, w postaci ogólnej, równa jest:

$$T_u(s) = \frac{a_0}{s^2 \cdot b_2 + s \cdot b_1 + b_0} \tag{7}$$

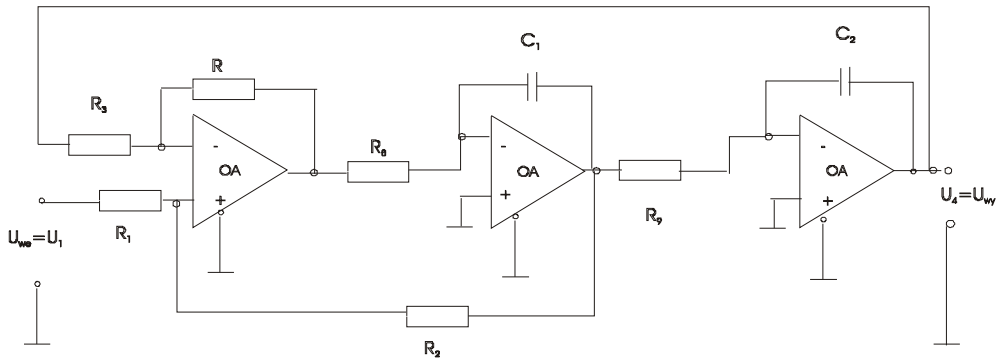
gdzie współczynniki transmitancji podane zostały w postaci układu równań nieliniowych – zależność (10). Równania te wynikają z przyrównania współczynników transmitancji (9) zadanej z odpowiednimi współczynnikami transmitancji wynikającymi z zależności (7). Dla ujednolicenia zapisu przyjęto oznaczenia zmiennych [11] postaci:

$$\begin{aligned} &G_1 = x_1\,,\ G_2 = x_2\,,\ G_3 = x_3\,,\ G = x_4 \\ &G_8 = x_5\,,\ G_9 = x_6\,,\ C_1 = x_7\,,\ C_2 = x_8 \end{aligned} \tag{8}$$

$$T_u(s) = \frac{1}{s^2 + \sqrt{2} \cdot s + 1} \tag{9}$$

$$\begin{aligned} f_1(x) &= x_1 \cdot x_3 \cdot x_5 \cdot x_6 + x_2 \cdot x_3 \cdot x_5 \cdot x_6 = 1 = a_0 \\ f_2(x) &= x_2 \cdot x_8 \cdot x_5 \cdot x_4 + x_2 \cdot x_8 \cdot x_5 \cdot x_3 = b_2 = 1{,}41421 \\ f_3(x) &= x_1 \cdot x_7 \cdot x_8 \cdot x_4 + x_2 \cdot x_7 \cdot x_8 \cdot x_4 = 1 = b_1 \\ f_4(x) &= x_1 \cdot x_4 \cdot x_5 \cdot x_6 + x_1 \cdot x_3 \cdot x_5 \cdot x_6 = 1 = b_0 \end{aligned} \tag{10}$$

Podobnie, jak w pracy [11] dopuszczono 1% błąd realizacji współczynników transmitancji (7, 9) oraz 5% obszar tolerancji dla wszystkich parametrów układu. W niniejszym przypadku przyjęto ponadto, że kryterium optymalizacji (funkcja $F_2(\boldsymbol{x})$), będzie minimum wieloparametrowej funkcji wrażliwości w sensie funkcji Schoefflera.



Rys. 1. Struktura filtru prototypowego opracowanego przez Kerwina-Huelsmana-Newcomba.

Należy teraz dobrać strukturę SN do postawionego zadania optymalnego projektowania filtru. Zazwyczaj przyjmuje się, że liczba neuronów w warstwie wejściowej jest zgodna z liczbą współrzędnych wektora danych, która to liczba w prezentowanym przypadku odpowiada liczbie parametrów projektowanego filtru, tj. 16 (8 parametrów filtru x_i losowanych z przedziału wartości dopuszczalnych i 8 wartości tolerancji względnych (procentowych) d_i tych parametrów). Jako wektor wyjściowy w SN ($d(y)$) przyjęto wektor o 8 współrzędnych, na który składają się 8 parametrów nominalnych (optymalnych) filtru x_{inom}.

W przypadku, gdy zastosowano by warstwę lub warstwy ukryte, to nie ma dobrej recepty na określenie właściwej ilości warstw i liczby neuronów w każdej warstwie. Przyjmuje się, że sieć z jedną warstwą ukrytą powinna nauczyć się rozwiązania większości postawionych problemów, nie ma natomiast dobrej recepty na dobór właściwej ilości neuronów w tej warstwie. Można jednak rozpocząć uczenie z niewielką ich ilością a następnie doświadczalnie zwiększać ilość neuronów. Jak zauważono podczas wstępnych symulacji, w prezentowanym prostym przypadku nie było konieczności, aby zaproponowana elementarna postać SN (16 neuronów na wejściu, 8 na wyjściu) zawierała warstwę ukrytą.

3). ZASADY UCZENIA SIECI NEURONOWEJ W ROZWAŻANYM ZADANIU OPTYMALIZACJI

Aby uzyskać optymalny projekt układu, należy przejść całą procedurę uczenia sieci. Podstawowym celem procesu uczenia jest dobór takich wag, które umożliwią odwzorowanie danych wejściowych w wyjściowe przy założeniu uzyskania jak najniższego błędu. Odpowiednio wytrenowana sieć neuronowa winna mieć zdolności generalizacyjne. W tym celu należy użyć odpowiedniej liczby wzorców uczących. Zgodnie z danymi podanymi w pracy [10], minimalna, krytyczna liczba wzorców uczących wyznaczona jest zależnością:

$$y_n \rangle\rangle n_w \cdot (1 + log(N)) \qquad (11)$$

gdzie: N - jest liczbą neuronów w sieci, y_n – minimalną liczbą wzorców uczących, n_w - jest liczbą wag w sieci.

Wynika stąd, że wymagana minimalna liczba wzorców uczących dla zaproponowanej w poprzednim punkcie pracy architekturze sieci winna być znacząco większa od 305 (gdyż n_w=16·8=128). Powszechnie przyjmuje się, że liczba próbek ciągu danych uczących winna być dziesięciokrotnie większa od liczby *VCdim* [10] dla sieci neuronowej wyznaczonej ze wzoru (11), stąd też w niniejszym przykładzie przyjęto liczbę elementów danych (wektorów) dla ciągu uczącego równą 3000. Ponieważ znamy odpowiedź na dany ciąg danych wejściowych sieci stąd właściwe jest zastosowanie uczenia pod nadzorem (inaczej uczenie z nauczycielem).

Pierwszą czynnością w procesie uczenia jest przygotowanie dwóch ciągów danych: uczącego i weryfikującego. W jego skład wchodzi wektor wejściowy, czyli te dane wejściowe, które podawane są na wejścia sieci i wektor wyjściowy, czyli takie dane oczekiwane, jakie sieć powinna wygenerować na swoich wyjściach.

Tabela 1

WEKTOR **x** WSTĘPNIE USTALONYCH PARAMETRÓW NOMINALNYCH UKŁADU DLA PRZYPADKU OPTYMALNEJ WARTOŚCI FUNKCJI CELU F_1,

Parametr	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
Wartość liczbowa dla F_1	0,662	0,811	0,534	0,654	1,07	1,183	0,758	1,366

W tabeli 1 podano startowe wartości nominalne parametrów filtru wykorzystane w procedurze uczenia sieci.

Po przetworzeniu wektora wejściowego porównywane były wartości otrzymane z wartościami oczekiwanymi i sprawdzenie, czy odpowiedź jest poprawna, a jeżeli nie, to jaki powstał błąd odpowiedzi. Błąd ten był następnie propagowany do sieci, ale w odwrotnej niż wektor wejściowy kolejności (od warstwy wyjściowej do wejściowej) i na jego podstawie następowała taka korekcja wag w każdym neuronie, aby ponowne przetworzenie tego samego wektora wejściowego spowodowało zmniejszenie błędu odpowiedzi. Procedurę taką powtarza się do momentu wygenerowania przez sieć błędu mniejszego niż założony. Wtedy na wejście sieci podaje się kolejny wektor wejściowy i powtarza te czynności. Po przetworzeniu całego ciągu uczącego oblicza się błąd i cały cykl powtarzany jest do momentu, aż błąd ten spadnie poniżej dopuszczalnego. Jak to już było zasygnalizowane wcześniej, SN wykazują tolerancję na nieciągłości, przypadkowe zaburzenia lub wręcz niewielkie braki w zbiorze uczącym. Jest to wynikiem właśnie zdolności do uogólniania wiedzy. Jeżeli mamy już nauczoną sieć, musimy zweryfikować jej działanie. W tym momencie ważne jest podanie na wejście sieci wzorców nie należących do zbioru treningowego w celu zbadania czy sieć może efektywnie generalizować zadanie, którego się nauczyła. Do tego używano ciągu weryfikującego, który ma te same cechy co ciąg uczący tzn. dane dokładnie charakteryzują problem i znamy dokładne odpowiedzi. Ważne jest jednak, aby dane te nie były używane uprzednio do uczenia. Dokonujemy prezentacji ciągu weryfikującego z tą

różnicą, że w tym procesie nie rzutujemy błędów wstecz a jedynie rejestruje się ilość odpowiedzi poprawnych i na tej podstawie orzeka, czy sieć spełnia wymagania, czyli jak została nauczona.

Wagi początkowe, z którymi sieć rozpoczyna naukę z reguły stanowiły liczby wygenerowane przypadkowo z generatora liczb pseudolosowych. Po nauczeniu sieci powtórzono całą procedurę, począwszy od wygenerowania wag początkowych, dla sprawdzenia otrzymanych wyników. Zazwyczaj nie mamy gwarancji, że sieć podczas uczenia nie trafi w minimum lokalne. W przedstawionej metodzie takiego niebezpieczeństwa nie ma, co jest wielką zaletą omawianej metody (punkt 2 niniejszej pracy). Jedynie, w przypadku, gdy postępy podczas uczenia są niezadowalające, na co wskazuje szybkość malenia funkcji celu $F(\boldsymbol{x})$ w stosunku do znanego dla danego filtru minimum absolutnego funkcji wrażliwości konieczne może być wygenerowanie pseudolosowe nowego wektora wag początkowych dla sieci. Do uczenia sieci zastosowano tzw. regułę „delta" uczenia z nauczycielem z modyfikacją Widorowa-Hoffa, dla sieci nadzorowanych o dowolnych funkcjach aktywacji ponieważ minimalizuje ona błąd średni kwadratowy pomiędzy pożądaną odpowiedzią a pobudzeniem.

III. DOBÓR STRUKTURY, PRARAMETRÓW ORAZ OPIS PROCEDUR STOSOWANYCH W PROJEKTOWANIU SIECI NEURONOWYCH W ŚRODOWISKU MATLAB (NEURAL NETWORKS TOOLBOKS) DO WYKONANIA OKREŚLONEGO ZADANIA OPTYMALIZACJI

WEKTORY PARAMETRÓW OPTYMALNYCH

```
T=[0.662; 0.811; 0.534; 0.654; 1.07; 1.183; 0.758; 1.366]; %dla zbioru uczącego
i testującego w trakcie uczenia
TT=[0.637; 0.896; 0.590; 1.30; 1.27; 1.32; 0.558; 1.01]; % dla zbioru testującego
```

MACIERZE DANYCH WYJSCIOWYCH (optymalnych)

```
T1L=powielaj([0.662; 0.811; 0.534; 0.654; 1.07; 1.183; 0.758; 1.366], 1500);
T1V=powielaj([0.662; 0.811; 0.534; 0.654; 1.07; 1.183; 0.758; 1.366], 1000);
```

MACIERZ DANYCH WEJSCIOWYCH UCZACYCH I MACIERZ DANYCH TESTUJACYCH W TRAKCIE UCZENIA

```
P = generator(T,5,3000);
T_proc(1:8,1) = 5;
T_temp = powielaj(T_proc,3000);
P_caly = [P',T_temp']';
PL = wybieraj(P_caly,1,2); %zbór danych wejściowych uczących
PV = wybieraj(P_caly,1,3); %zbiór testujący w trakcie uczenia SN (miara zdolności
zapamiętania danych uczących)
```

MACIERZ DANYCH WEJSCIOWYCH TESTYACYCH

```
PT = generator(TT,5,3000);
TT_proc(1:8,1) = 5;
TT_temp = powielaj(TT_proc,3000);
P_caly_T = [PT',TT_temp']';
Pt = wybieraj(P_caly_T,1,2); %zbiór testujący (miara zdolności uogólniania SN)
```

INICJACJA SIECI NEURONOWEJ

```
%INITFF Inititialize feed-forward network up to 2 layers (inicjalizacja sieci
2 warstwowej [16-26-8 neuronów])
net = newff([-10 10],[16, 26,'tansig',8,'purelin']);
ograniczenia = [-10 10;-10 10;-10 10;-10 10;-10 10;-10 10;-10 10;-10 10;-10
10;-10 10;-10 10;-10 10;-10 10;-10 10;-10 10;]
%przedział zmienności danych wejściowych - rozmiar wektora ograniczeń wynosi 16X2
[W1,b1,W2,b2]=initff(ograniczenia,26,'tansig',8,'purelin');
[W1,b1]=initff(ograniczenia,8,'purelin'); %SN jednowarstwowa
```

WSPÓŁCZYNNIKI TRENINGU DLA RÓŻNYCH METOD OPTYMALIZACJI (parametry uczenia SN)

```
tp=[25 10000 0.02 0.001];
%TP(1) - Epochs between updating display, default = 25. (częstotliwość wyświetlania
wyników uczenia - co 25 epok)
%TP(2) - Maximum number of epochs to train, default = 1000. (max. liczba epok 1000)
%TP(3) - Sum-squared error goal, default = 0.02. (akceptowalny błąd średniokwadratowy
0.02)
%TP(4) - Learning rate, 0.01. (współczynnik uczenia 0.01-0.001 lub mniej)

tpx=[25 10000 0.02 0.01 1.05 10 0.1 1000];
%TP(1) - Epochs between updating display, default = 25.
%TP(2) - Maximum number of epochs to train, default = 1000.
%TP(3) - Sum-squared error goal, default = 0.02.
```

```
%TP(4) - Minimum gradient, default = 0.01
%TP(5) - Learning rate increase, default = 1.05. mnożnik do wzrostu lr (typowo 1.05)
%TP(6) - Learning rate decrease, default = 0.7. mnożnik do zmniejszania lr (typowo
0.7)
%TP(7) - Maximum error ratio, default = 1.04. maksymalny stosunek do starego błędu
bez "rejecting" nowych wartości wag i progów (typowo 1.04)

tpa=[25 10000 0.02 0.001 1.05 0.7 0.95 1.04];
%TP(1) - Epochs between updating display, default = 25.
%TP(2) - Maximum number of epochs to train, default = 1000.
%TP(3) - Sum-squared error goal, default = 0.02.
%TP(4) - Learning rate, 0.01.
%TP(5) - Learning rate increase, default = 1.05. mnożnik do wzrostu lr (typowo 1.05)
%TP(6) - Learning rate decrease, default = 0.7. mnożnik do zmniejszania lr (typowo
0.7)
%TP(7) - wspólczynnik momentum (typowo 0.95)
%TP(8) - Maximum error ratio, default = 1.04. maksymalny stosunek do starego błędu
bez "rejecting" nowych wartosci wag i progów (typowo 1.04)

tpm=[25 10000 0.02 0.01 0.95 1.04];
%TP(1) - Epochs between updating display, default = 25.
%TP(2) - Maximum number of epochs to train, default = 1000.
%TP(3) - Sum-squared error goal, default = 0.02.
%TP(4) - Learning rate, 0.01.
%TP(5) - Momentum constant, default = 0.95. współczynnik momentum (typowo 0.95)
%TP(6) - Maximum error ratio, default = 1.04. maksymalny stosunek do starego błędu
bez "rejecting" nowych wartosci wag i biasu (typowo 1.04)

tplm=[25 10000 0.02 0.00001 0.001 10 0.1 1000];
%TP(1) - Epochs between updating display, default = 25.
%TP(2) - Maximum number of epochs to train, default = 1000.
%TP(3) - Sum-squared error goal, default = 0.02.
%TP(4) - Minimum gradient, default = 1e-6.
%TP(5) - Initial value for MU, default = 0.001.
%TP(6) - Multiplier for increasing MU, default = 10.
%TP(7) - Multiplier for decreasing MU, default = 0.1.
%TP(8) - Maximum value for MU, default = 1e10.
```

METODY UCZENIA SN W ROZWAŻANYM ZADANIU OPTYMALIZACJI

```
net = train(net,p,t);
%TBP2 Train 2-layer feed-forward network w/backpropagation (trenowanie sieci ze
wsteczną propagacją błędu)
[W1,b1,W2,b2,ep,tr]=tbp2(W1,b1,'tansig',W2,b2,'purelin',PL,T1L,tp);
[W1,b1,ep,tr]=tbpx1(W1,b1,'purelin',PL,T1L,tp); %SN jednowarstwowa

%TBPX2 Train 2-layer feed-forward network w/fast backpropagation (trenowanie sieci ze
wsteczną propagacją błędu z metoda momentum i uczeniem adaptacyjnym)
[W1,b1,W2,b2,ep,tr]=tbpx2(W1,b1,'tansig',W2,b2,'purelin',PL,T1L,tp); %ze zbiorem
uczącym
[W1,b1,W2,b2,ep,tr]=tbpx2(W1,b1,'tansig',W2,b2,'purelin',PV,T1V,tp); %ze zbiorem
testujący w trakcie uczenia

%TRAINBPA Train feed-forward network with bp + adaptive learning (trenowanie sieci ze
wsteczną propagacją błędu i uczeniem adaptacyjnym)
[W1,b1,W2,b2,ep,tr]=trainbpa(W1,b1,'tansig',W2,b2,'purelin',PL,T1L,tpa);

%TRAINBPM Train feed-forward networks with bp + momentum (trenowanie sieci ze
wsteczną propagacją błędu z metoda momentum)
[W1,b1,W2,b2,ep,tr]=trainbpm(W1,b1,'tansig',W2,b2,'purelin',PL,T1L,tpm);

%TLM2 Train 2-layer feed-forward network w/Levenberg-Marquardt
[W1,b1,W2,b2,ep,tr]=tlm2(W1,b1,'tansig',W2,b2,'purelin',PL,T1L,tplm);
```

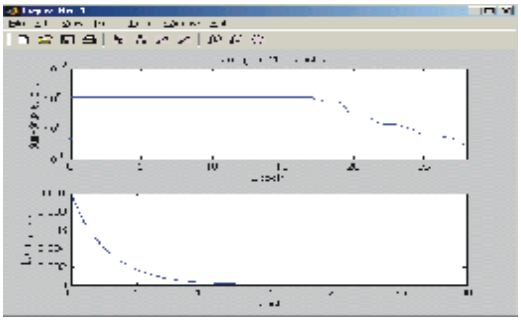
TESTOWANIE SN W ROZPATRYWANYM ZADANIU OPTYMALIZACJI

```
%SIMUFF Simulate feed-forward network
a = simuff(PV,W1,b1,'tansig',W2,b2,'purelin')
a = simuff(Pt,W1,b1,'tansig',W2,b2,'purelin')
```

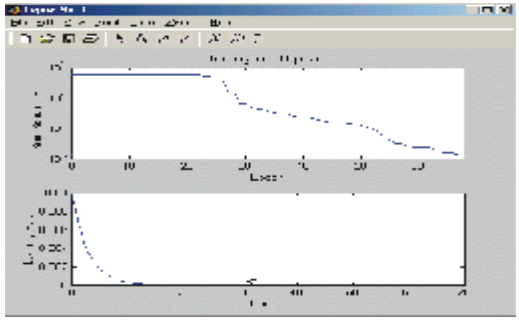
IV. WYBRANE WYNIKI BADAŃ W OPTYMALNYM PROJEKTOWANIU FILTRÓW AKTYWNYCH IRS

Wpływ warstwy ukrytej na zdolność uczenia się sieci neuronowej

Sieć neuronowa jednowarstwowa

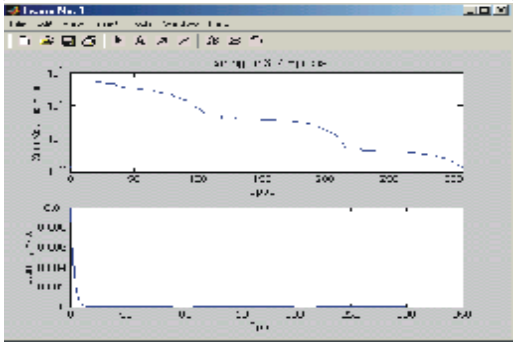


Sieć neuronowa z jedną warstwą ukrytą

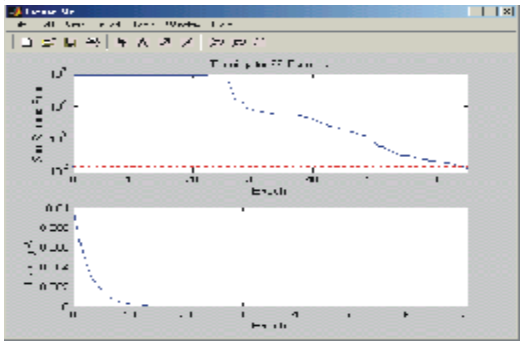


Wpływ liczby neuronów w warstwie ukrytej na zdolność uczenia się SN dwuwarstwowej

6 neuronów w warstwie ukrytej

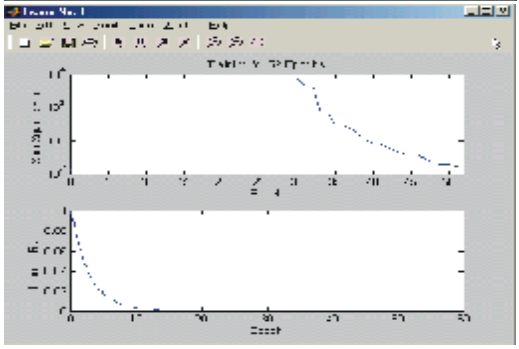


26 neuronów w warstwie ukrytej

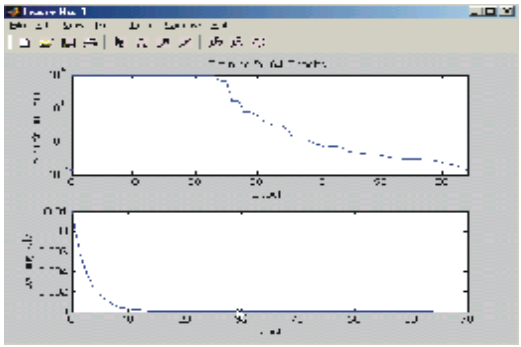


Wpływ współczynnika uczenia na zdolność uczenia się sieci neuronowej dwuwarstwowej z liczbą neuronów ukrytych - 26

Współczynnik uczenia - 0.1

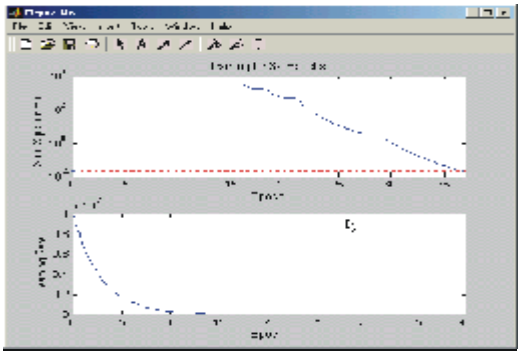


Współczynnik uczenia - 0.001

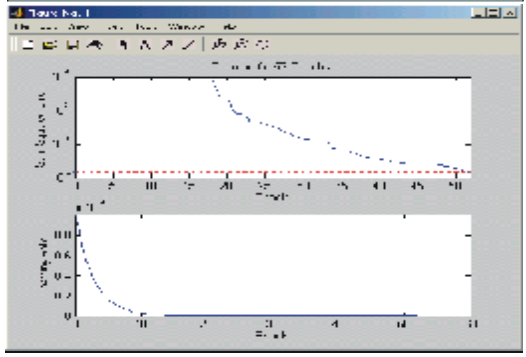


Porównanie efektywności algorytmów uczących (sieć neuronowa o 26 neuronach w warstwie ukrytej i współczynniku uczenia 0.01) na podstawie liczby cykli uczących

Uczenie sieci neuronowej ze wsteczną propagacją błędu i uczeniem adaptacyjnym



Uczenie sieci neuronowej ze wsteczną propagacją błędu z metodą mementum i uczeniem adaptacyjnym



V. PODSUMOWANIE

Do badań w niniejszej pracy wykorzystano symulacje i programy uczenia dla jednokierunkowej sieci wielowarstwowej napisany w programie MATLAB, który jest szeroko znanym i bardzo często wykorzystywanym pakietem obliczeniowym. Użyty w tym celu przybornik Neural Network Toolbox [1, 5, 6, 7, 8] daje duże możliwości pomocy w tworzeniu, implementacji, oraz symulacji wielu rodzajów sieci neuronowych. Wspomniany pakiet MATLAB dzięki możliwości szybkiego i stosunkowo prostego konstruowania SN, bardzo dobrze nadaje się również do rozwiązywania zagadnień „centrowania" (tzn. „lokowania" wektora wartości nominalnych parametrów na hiperpłaszczyźnie rozwiązań układowych) oraz do projektowania przypisanych do tego wektora np. optymalnych tolerancji układowych. Jak powszechnie wiadomo SN doskonale filtruje zaburzone wartości sygnałów, a także fluktuacje parametrów zawarte w wektorze wejściowym sieci zwłaszcza o charakterze stochastycznym. Połączenie tej zalety SN z ich inną podstawową cechą, jaką jest łatwość analizy danych i kojarzenia najlepszych spośród możliwych rozwiązań, czyni z nich obiecujące narzędzie dla projektanta, zwłaszcza w sytuacji uwzględniania na etapie projektowania możliwości fluktuacji w praktyce przemysłowej zaprojektowanych parametrów układu. Nauczona (prezentowana w pracy) sieć wyjątkowo sprawnie rozwiązywała postawione jej zadanie optymalizacji, jednak tylko w przypadku niewielkich zmian wartości parametrów. Zdolności uogólniania osiągnięte przez sieć dają możliwości optymalizacji także w przypadku wektora wejściowego spoza zakresu wartości użytych do nauczenia sieci, co wykazały uzyskane wyniki. Jednak, w przypadku większych zmian parametrów, wydaje się, że niezbędne jest powiększenie sieci neuronowej przez wprowadzenie nowej dodatkowej warstwy ukrytej.

LITERATURA

[1] Brzózka J., Dorobczyński L.: Programowanie w Matlab, Wydawnictwo MIKOM, Warszawa 1998.
[2] Haykin S.: Neural Networks, a Comprehensive Foundation, Macmillan College Publishing Company, New York, 1994.
[3] Hecht-Nielsen R.: Neurocomputing, Addison Wesley Publishing Company, New York, 1990.
[4] Hertz J., Krogh A., Palmer R. G. I.: Wstęp do teorii obliczeń neuronowych, Wyd. II, WNT, Warszawa, 1995.
[5] Mrozek B., Mrozek Z., MATLAB Uniwersalne środowisko do obliczeń naukowo-technicznych, PLJ, Warszawa 1996.
[6] Mrozek B., Mrozek Z., MATLAB 5.x, SIMULINK 2.x, poradnik użytkownika, PLJ, Warszawa 1998.
[7] MATLAB Notebook User's Guide, version 5, The Math. Works, Inc., 1997.
[8] MATLAB Products Catalog, The Math. Works, Inc., 1997.
[9] Osowski S.: Signal flow graphs and neural networks, Biological Cybernetics, 1994, Vol. 70, s. 387-395.
[10] Osowski S.: Sieci neuronowe w ujęciu algorytmicznym, WNT, Warszawa, 1996.
[11] Rybarczyk A., Szulc M.: Wykorzystanie pakietu obliczeniowego MATLAB do minimalizacji wariancji parametrów wyjściowych układu scalonego CMOS, Mat. VII Konf. ZkwE'2002, Poznań-Kiekrz, 22-24 kwietnia 2002, s. 389-392.
[12] Nowakowski B., Rybarczyk A.: Sztuczne sieci neuronowe – realizacja fizyczna w oparciu o układy FPGA, Mat. VII Konf. ZkwE'2002, Poznań-Kiekrz, 22-24 kwietnia 2002, s. 357-360.
[13] Rybarczyk A., Nowakowski B.: Neural Network – Hardware Implementation Using FPGA Konf. MIXDES 2002, 20-22 June 2002, Wrocław, Poland.
[14] Rybarczyk A.: Yield Optimization of Analog Filters Using Statistical Design Approach and Multiparameter Sensitivity Measure, Mat. 12th European Conference on Circuit Theory and Design (ECCTD'95), Istanbul (Turkey), 27-31 sierpień 1995, tom I s. 287-290.
[15] Rybarczyk A.: Optymalizacja uzysku produkcyjnego układu CMOS oparta na optimum wrażliwościowym jako punkcie startowym, Mat. IV Konf. ZkwE'99, Poznań-Kiekrz, 12-14 kwietnia 1999, s. 83-86.
[16] Dłubis E.: Praktyczne informacje o sieciach neuronowych Mat. I Konf. Entuzjastów Informatyki, Chełm, 24-25 maj 2002, s. 47-57.