Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Sztuczna Inteligencja w Automatyce

Sprawozdanie z projektu 2
zadanie 5

Michał Kwarciński, Bartosz Gałecki

Warszawa, 2022

Spis treści

1. Wstęp 2

2. Zadanie 1 3

2.1. Charakterystyka statyczna 3
2.2. Generacja zbiorów danych uczących i testujących 4

3. Zadanie 2 7

3.1. Opóźnienie 7
3.2. Uczenie sieci neuronowych z wykorzystaniem programu sieci 7
3.3. Błędy ARX i OE w kolejnych iteracjach uczących 7
3.4. Symulacja modelu w trybie OE na zbiorze wartości uczących i testujących 8
3.5. Uczenie modeli algorytmem najszybszego spadku w trybie OE 11
3.6. Uczenie modeli w trybie ARX 12
3.7. Symulacja modelu nauczonego w trybie ARX w trybie OE 13
3.8. Model liniowy wyznaczony metodą najmniejszych kwadratów 15

4. Zadanie 3 19

4.1. Wybór przybornika Matlab 19
4.2. Uczenie sieci za pomocą przybornika 19
4.3. Symulacja w trybie ARX i OE 25
4.4. Porównanie jakości przybornika z programem sieci 26

5. Zadanie 4 27

5.1. Implementacja NPL 27
5.2. Strojenie NPL 27
5.3. Implementacja GPC 29

6. Zadanie dodatkowe 31

6.1. Dodatkowy PID 31
6.2. Dodatkowy NO 31

1. Wstęp

Regulowany proces jest opisanymi równaniami

$$x_1(k) = -\alpha_1 x_1(k-1) + x_2(k-1) + \beta_1 g_1(u(k-4)) \tag{1.1}$$
$$x_2(k) = -\alpha_1(k-1) + \beta_2 g_1(u(k-4)) \tag{1.2}$$
$$y(y) = g_2(x_1(k)) \tag{1.3}$$

gdzie u-sygnał wejściowy, y-sygnał wyjściowy, x_1, x_2-zmienne stanu, α_1=−1,599 028, α_2=0,632 337, β_1=0,010 754, β_2=0,009 231 oraz

$$g_1(u(k-4)) = \frac{\exp(5{,}25u(k-4)) - 1)}{\exp(5{,}25u(k-4)) + 1)} \tag{1.4}$$
$$g_2(x_1(k)) = -1{,}6(1 - \exp(-2x_1(k))) \tag{1.5}$$

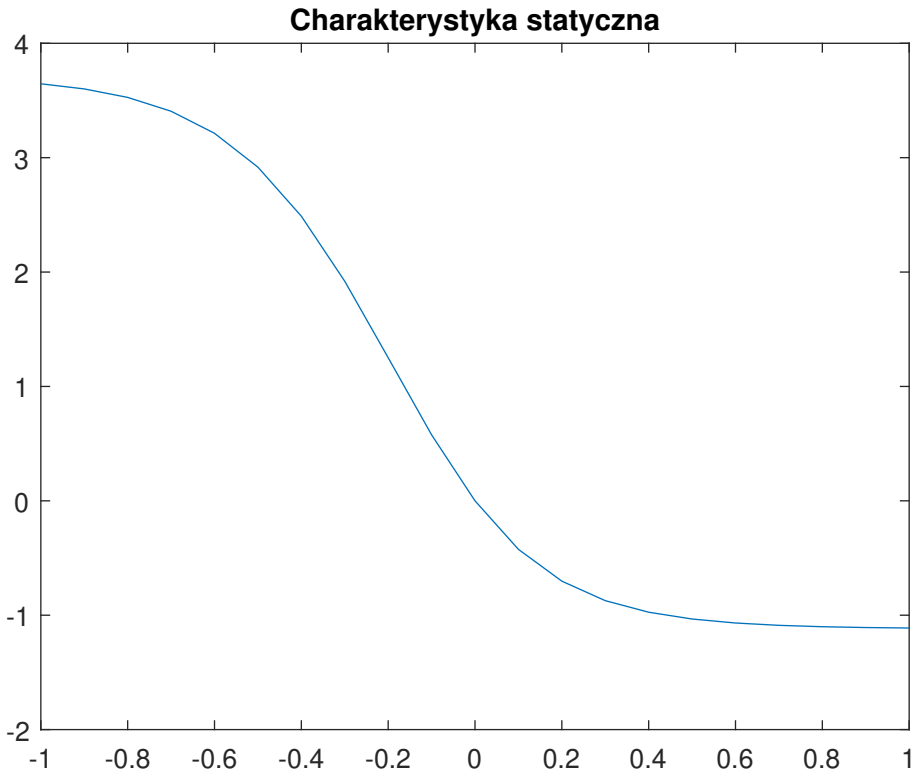
Punktem pracy tego obiektu jest $u = y = x_1 = x_2 = 0$, sygnał wejściowy może się zmieniać w granicach $u \in \langle -1, 1\rangle$

2. Zadanie 1

2.1. Charakterystyka statyczna

Charakterystyka statyczna została wyznaczona metodą eksperymentalną dla sterowań w przedziale $u \in \langle -1, 1 \rangle$

```
U(1:(u_max-u_min)/0.1+1) = [u_min:0.1:u_max];
Y(1:(u_max-u_min)/0.1+1) = 0;
for i=1:length(U)
    u(1:start) = 0;
    u(start:endt) = U(i);
    x1(1:endt) = 0;
    x2(1:endt) = 0;
    y(1:endt) = 0;
    for k=start:endt
        g_1 = g1(u(k-4));
        x1(k) = -alpha1*x1(k-1)+x2(k-1)+betha1*g_1;
        x2(k) = -alpha2*x1(k-1)+betha2*g_1;
        y(k) = g2(x1(k));
    end
    Y(i) = y(endt);
end
```



2.2. Generacja zbiorów danych uczących i testujących

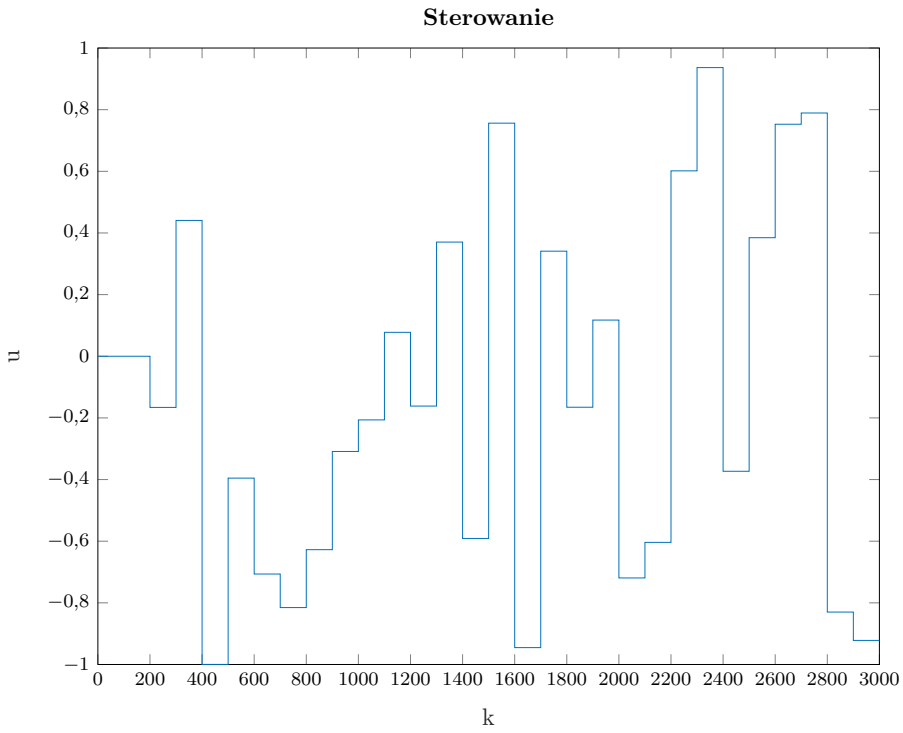
Dane uczące i testujące zostały wygenerowane dla 3000 próbek przy okresie zmian sygnału sterującego wynoszącego 100 kroków.

Sterowanie było generowane przez funkcję

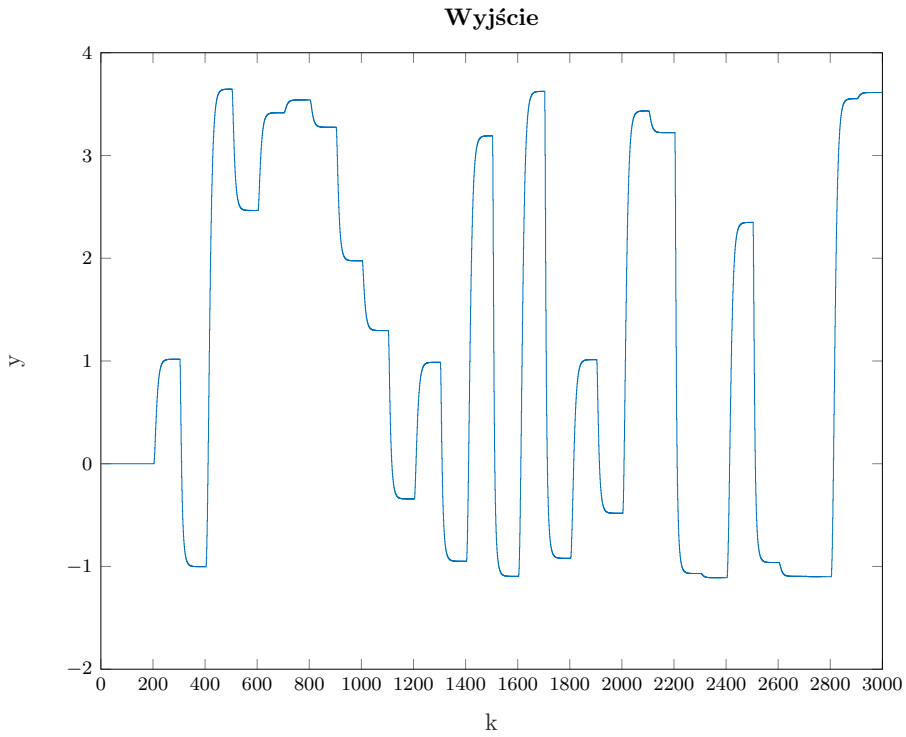
```
function u = losowe_sterowanie(liczbaKrokowDoZmiany,...
    liczbaProbek, u_min, u_max)
    u(1:liczbaProbek) = 0;
    for k=2:(liczbaProbek/liczbaKrokowDoZmiany)-1
        u(k*liczbaKrokowDoZmiany:(k+1)*liczbaKrokowDoZmiany) =...
            (u_max-u_min)*rand(1) + u_min;
    end
end
```

a symulacja przeprowadzona

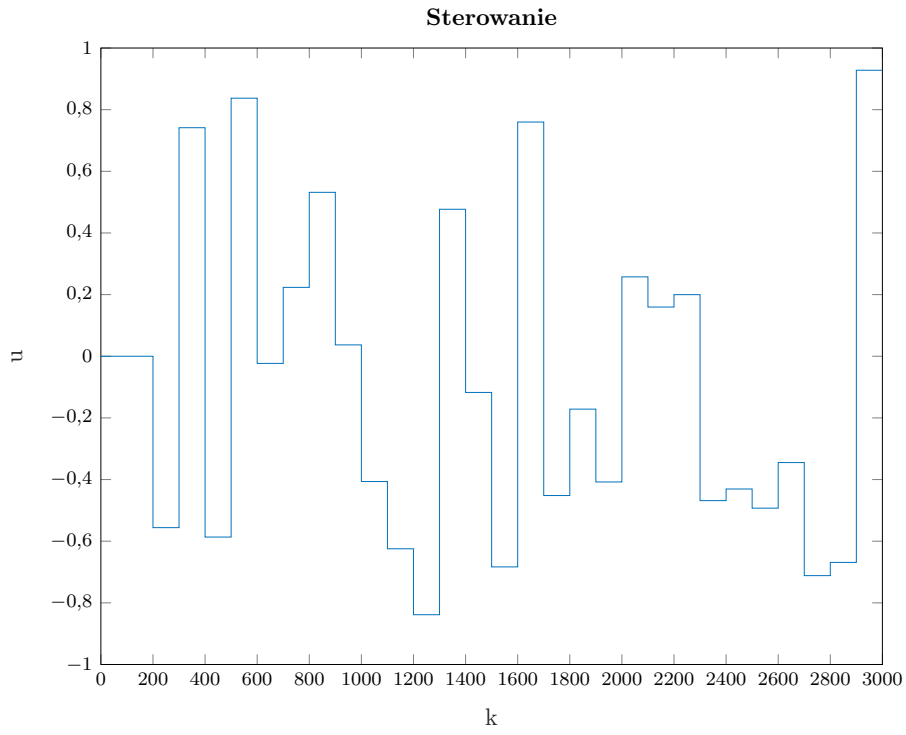
```
u(1:endt) = losowe_sterowanie(100, endt, -1, 1);
x1(1:endt) = 0;
x2(1:endt) = 0;
y(1:endt) = 0;
for k=start:endt
    g_1 = g1(u(k-4));
    x1(k) = -alpha1*x1(k-1)+x2(k-1)+betha1*g_1;
    x2(k) = -alpha2*x1(k-1)+betha2*g_1;
    y(k) = g2(x1(k));
end
```



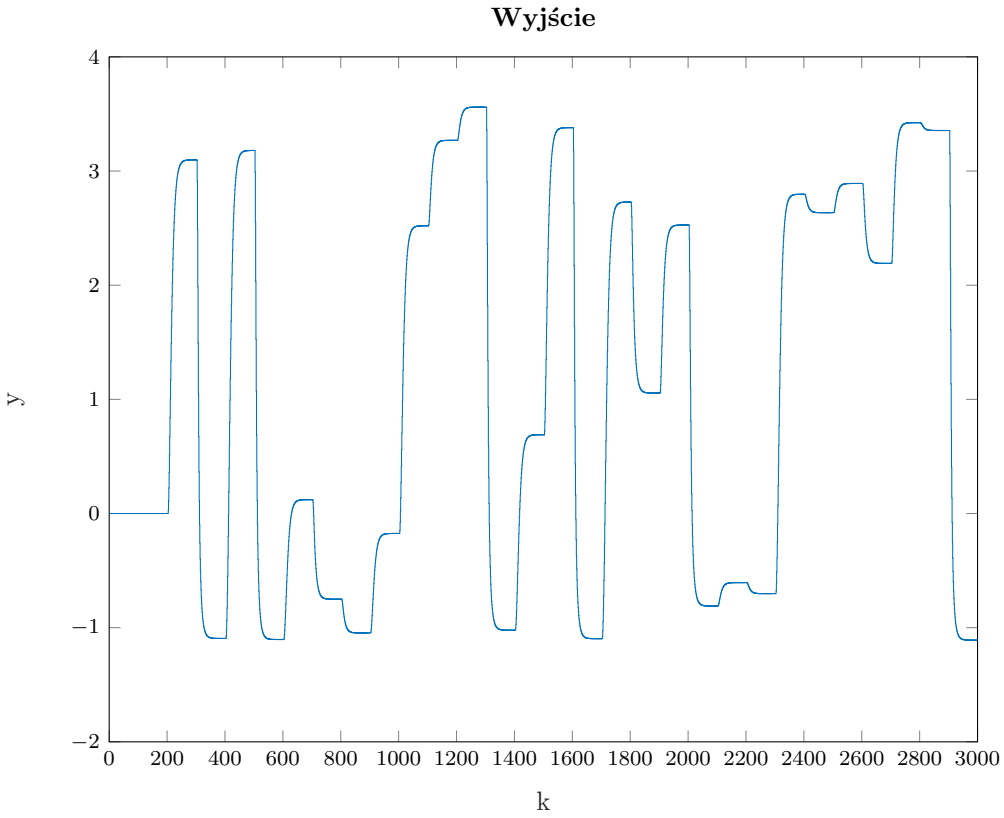
Rys. 2.1: Sterowanie dla zbioru trenującego



Rys. 2.2: Wyjście dla zbioru trenującego



Rys. 2.3: Sterowanie dla zbioru weryfikującego



Rys. 2.4: Wyjście dla zbioru weryfikującego

3. Zadanie 2

3.1. Opóźnienie

Opóźnienie zostało zdeterminowane z równań procesu. Wynos ono $\tau = 4$, gdyż we wzorach 1.1 i 1.2 jest $u(k-4)$ i są to jedyne miejsca w równaniach w którym występuje sterowanie.

3.2. Uczenie sieci neuronowych z wykorzystaniem programu sieci

Zostało wykonane uczenie modelów sieci neuronowych na podstawie wygenerowanych danych uczących. Została przyjęta dynamika drugiego rzędu

$$\hat{y}(k) = f(u(k-4), u(k-5), y(k-1), y(k-2)) \tag{3.1}$$

Każda struktura sieci została trenowana 6 razy i wyniki każdego najlepszego modelu z danej struktury zostały przedstawione w tabelce. Były one uczone w trybie rekurencyjnym, algorytmem BFGS z maksymalną ilością iteracji równą 750.

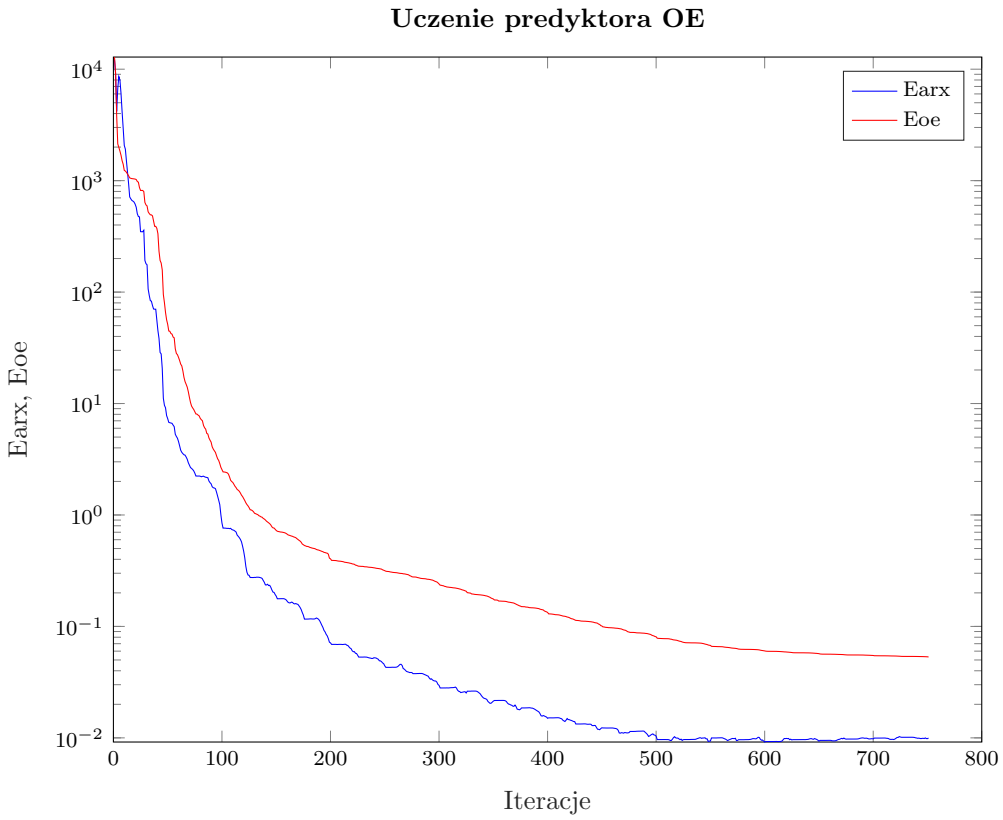
Liczba neuronów	Błąd dla danych testujących	Błąd dla danych uczących
1	115,54	173.45
2	55,41	41.58
3	2,18	1,58
4	1,07	0,95
5	1,03	0,42
6	0,74	0,37
7	0,41	0,19
8	0,28	0,097
9	0,103	0,053
10	0,32	0,10

Tab. 3.1

Jak widać na tabeli 3.1 wraz z wzrostem neuronów ukrytych zmniejszają się wartości błędów uczących i testujących. Dzieje się to tak gdyż z większą ilością neuronów zwiększa się możliwość dopasowania modelu do obiektu.

3.3. Błędy ARX i OE w kolejnych iteracjach uczących

Najlepszym modelem względem tabelki 3.1 jest model z 9 neuronami. Został on wybrany gdyż miał najmniejszy błąd dla danych testujących.

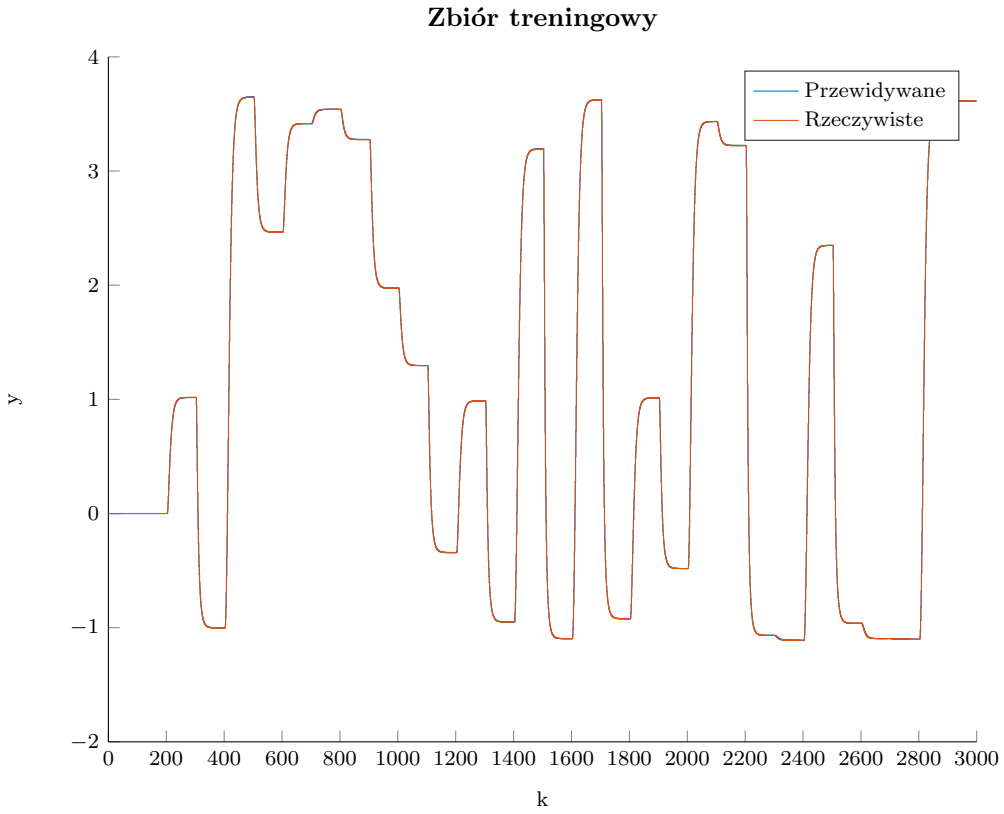


Rys. 3.1

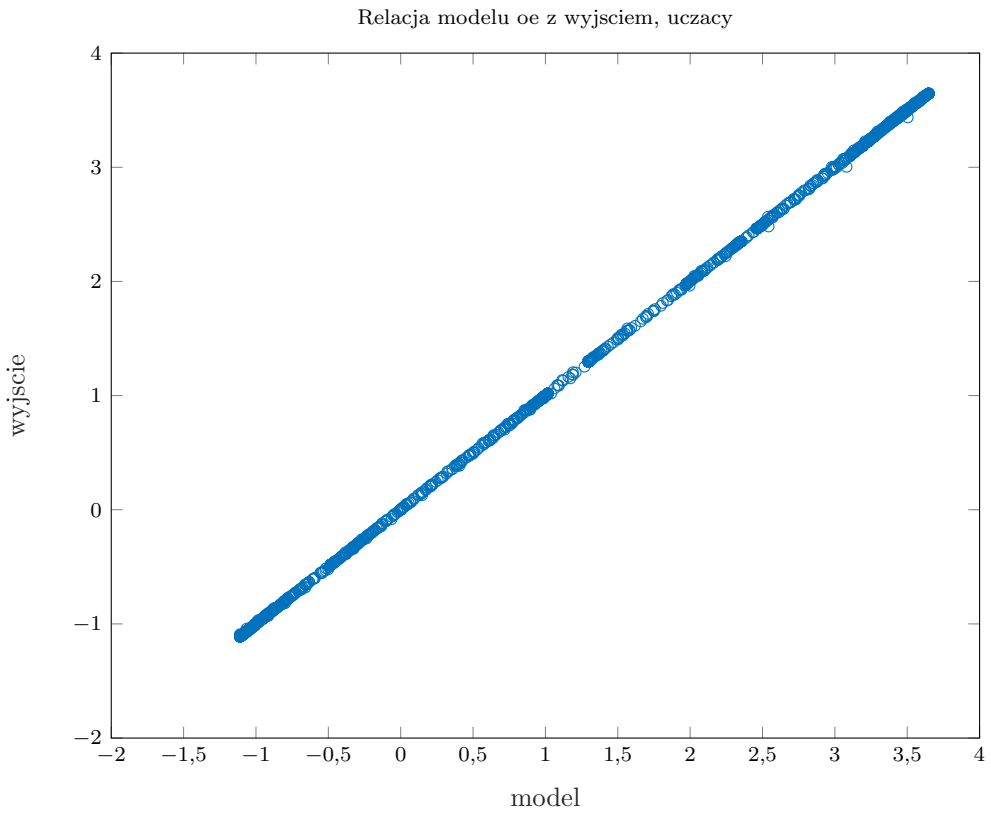
Jak widać na rysunku 3.1 przy optymalizacji predykatora OE zmniejsza się także błąd ARX, co nie powinno dziwić, gdyż jeśli model działa dobrze w trybie rekurencyjnym, też jest dobrym modelem do predykcji na jedną chwilę do przodu.

3.4. Symulacja modelu w trybie OE na zbiorze wartości uczących i testujących

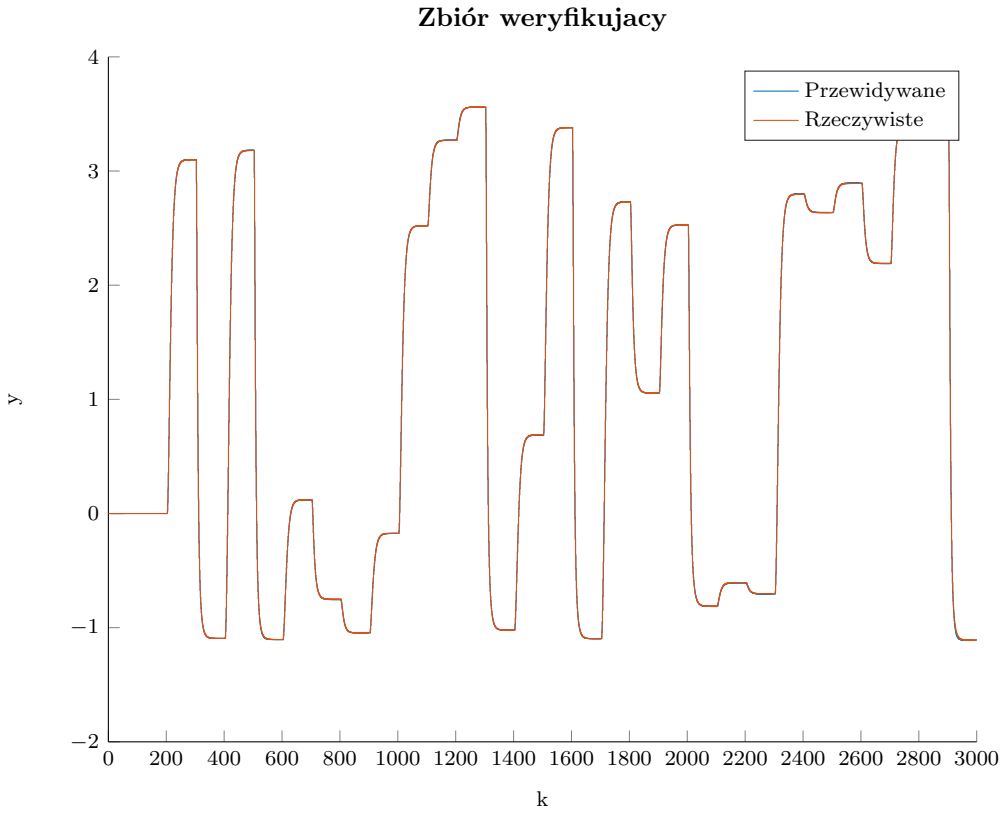
Wybrany model z podpunktu 3.3 został poddany symulacji w trybie OE na zbiorze wartości uczących i testujących


Zbiór treningowy
Przewidywane
Rzeczywiste
y
k
4
3
2
1
0
−1
−2
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000

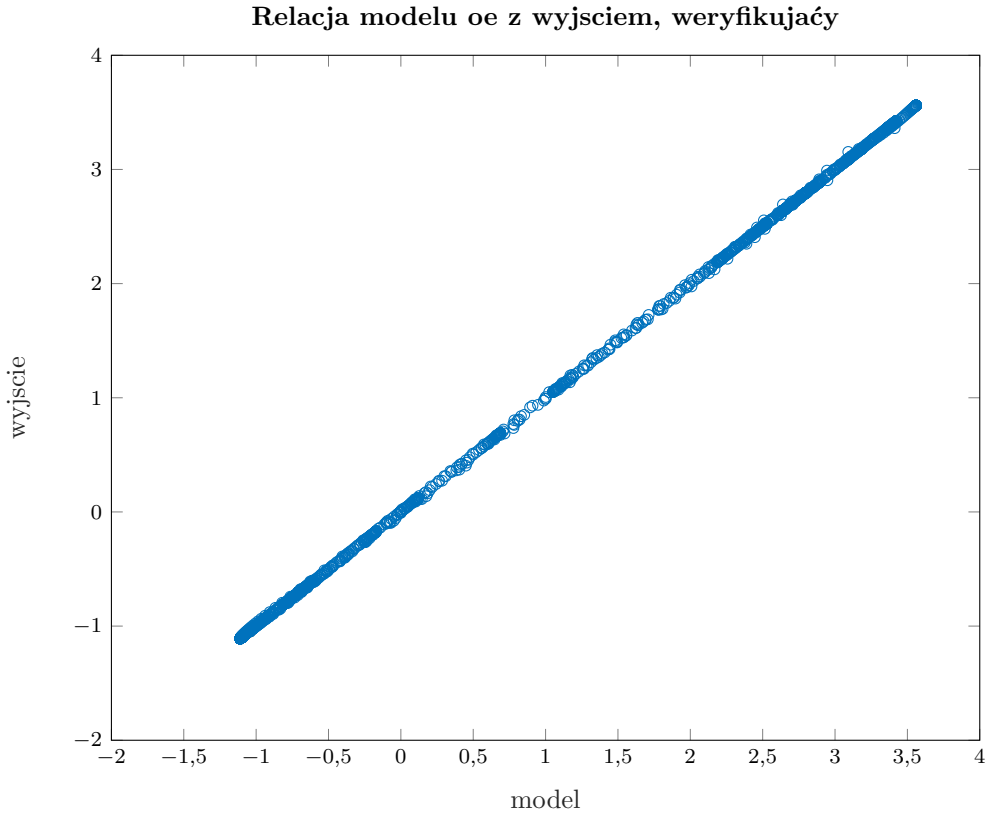
Rys. 3.2


Relacja modelu oe z wyjsciem, uczacy
wyjscie
model
4
3
2
1
0
−1
−2
−2
−1,5
−1
−0,5
0
0,5
1
1,5
2
2,5
3
3,5
4

Rys. 3.3


Zbiór weryfikujacy
Przewidywane
Rzeczywiste
y
k
4
3
2
1
0
−1
−2
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000

Rys. 3.4


Relacja modelu oe z wyjsciem, weryfikujaćy
wyjscie
model
4
3
2
1
0
−1
−2
−2
−1,5
−1
−0,5
0
0,5
1
1,5
2
2,5
3
3,5
4

Rys. 3.5

Jak widać na rysunkach 3.2 i 3.4 przebieg modelu w trybie OE pokrywa się z rzeczywistym przebiegiem obiektu. Relacja modelu z wyjściem dla obu zbiorów (rys. 3.3 i 3.5) się układa w linię prostą co także świadczy o dobrej jakości tego modelu.

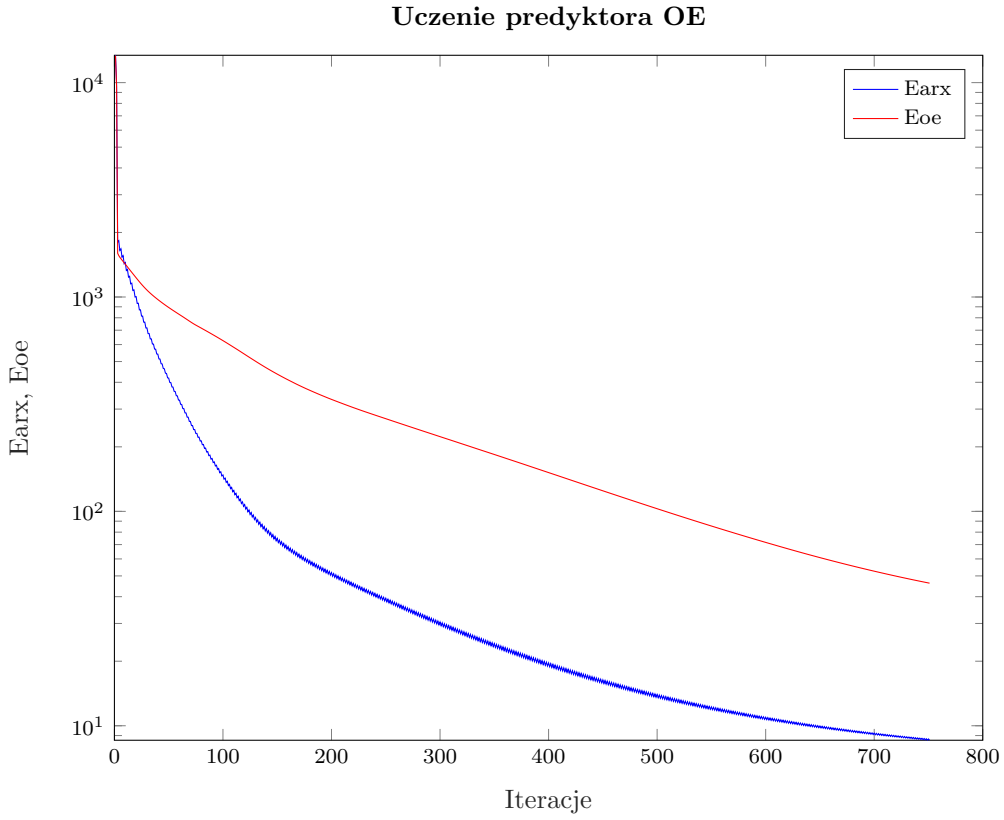
3.5. Uczenie modeli algorytmem najszybszego spadku w trybie OE

Przy uczeniu tych modeli, tak jak przy uczeniu algorytmem BFGS została założona dynamika 2 rzędu z opóźnieniem równym 4, z maksymalną ilością iteracji uczących równej 750.

Liczba neuronów	Błąd dla danych testujących	Błąd dla danych uczących
1	206,62	434,09
2	98,95	186,71
3	34,15	33,81
4	71,22	175,97
5	56,81	96,75
6	81,76	208,13
7	44,49	72,49
8	64,25	104,45
9	46,97	115,72
10	58,45	120,63

Tab. 3.2

Został wybrany model z 7 neuronami ukrytymi, gdyż miał najmniejszy błąd dla danych testujących.



Rys. 3.6

Porównując wartości w w tabelkach 3.1 i 3.2, a także rysunki 3.1 i 3.6 można zauważyć, że model korzystający z algorytmu najszybszego spadku wolniej się uczy i błąd szybciej przestaje spadać.

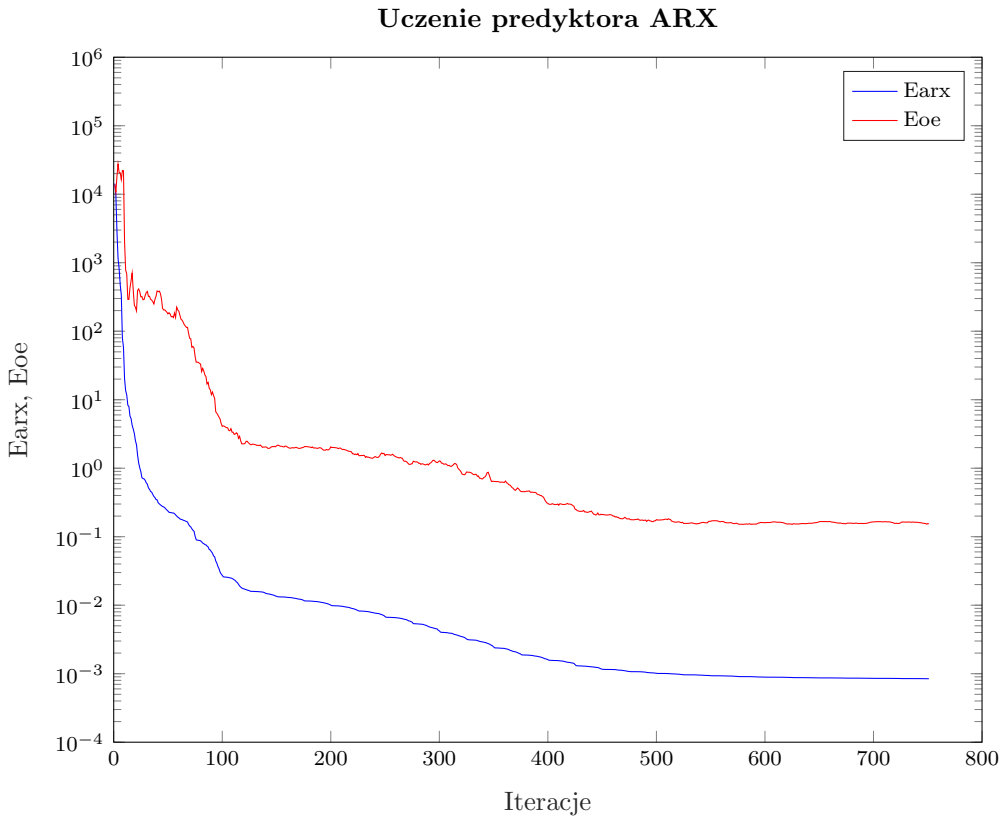
3.6. Uczenie modeli w trybie ARX

Została założona dynamika 2 rzędu z opóźnieniem równym 4. Uczenie zostało przeprowadzone w trybie ARX, algorytmem BFGS z maksymalną ilością iteracji uczących wynoszącą 750.

Liczba neuronów	Błąd dla danych testujących	Błąd dla danych uczących
1	0,54	0,38
2	0,17	0,18
3	0,046	0,023
4	0,02	0,014
5	0,0071	0,0054
6	0,0069	0,0043
7	0,0062	0,0014
8	0,0047	0,0013
9	0,0029	0,00084
10	0,0033	0,00074

Tab. 3.3

Został wybrany model z 9 neuronami, gdyż ma najmniejszy błąd na zbiorze uczącym.

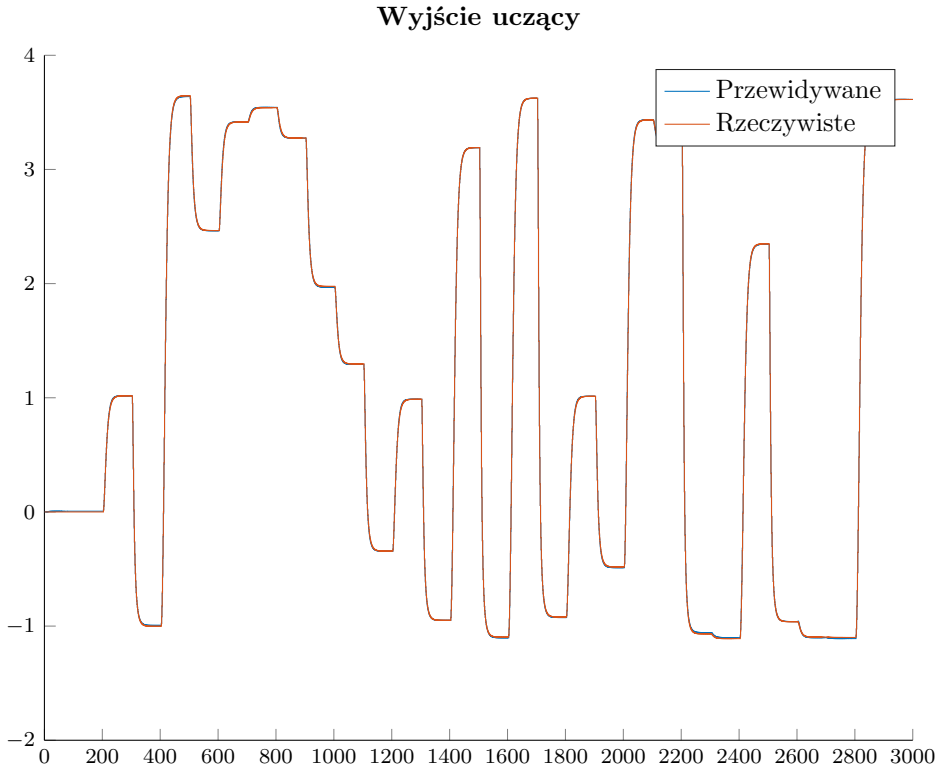


Rys. 3.7

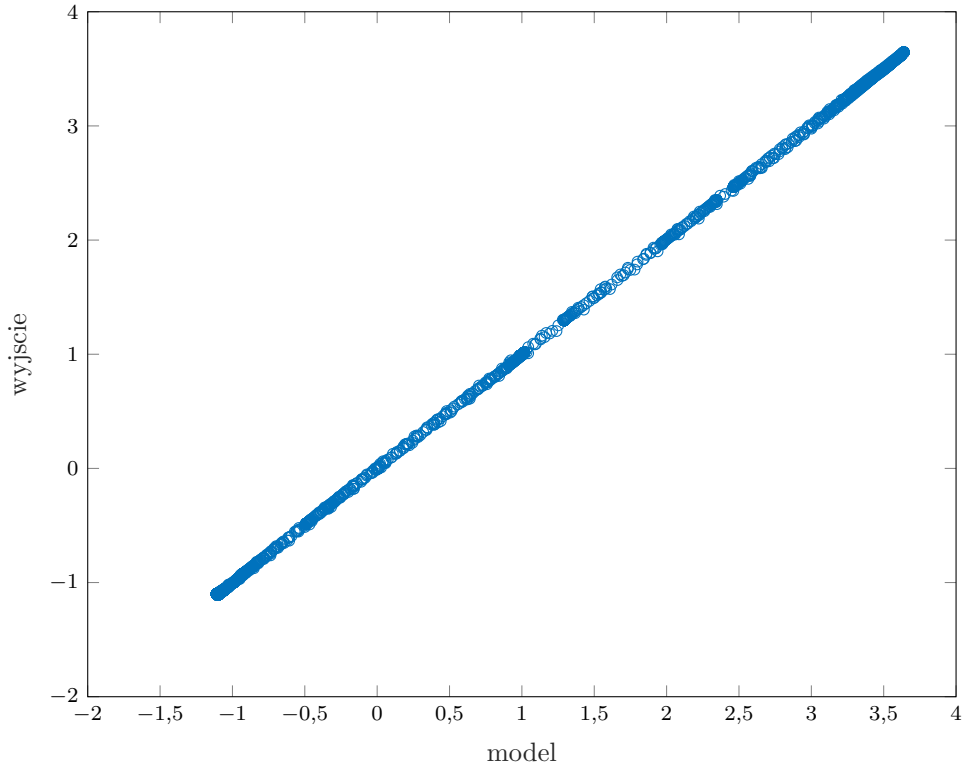
Porównując rysunki 3.7 i 3.1 można zauważyć, że gdy błąd ARX jest mniejszy gdy się optymalizuje względem niego (co nie powinno dziwić), to błąd OE jest większy. Dzieje się to tak, gdyż dobra optymalizacja błędu w trybie ARX nie gwarantuje dobrej optymalizacji modelu w trybie OE.

3.7. Symulacja modelu nauczonego w trybie ARX w trybie OE

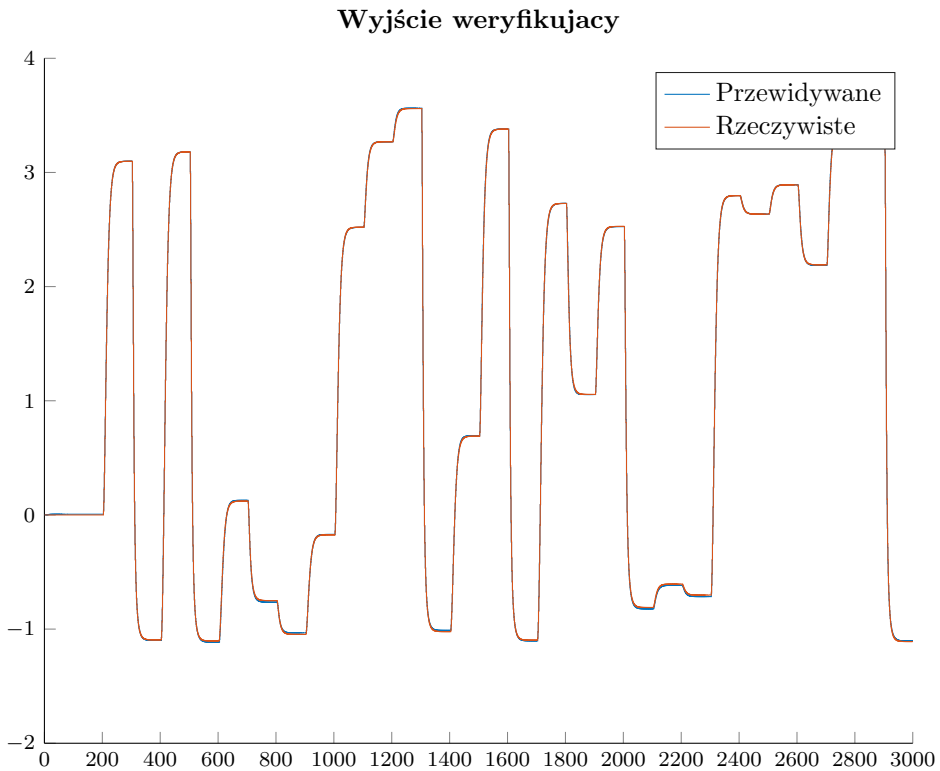
Do przeprowadzenia symulacji został wykorzystany model wybrany w podpunkcie 3.6.



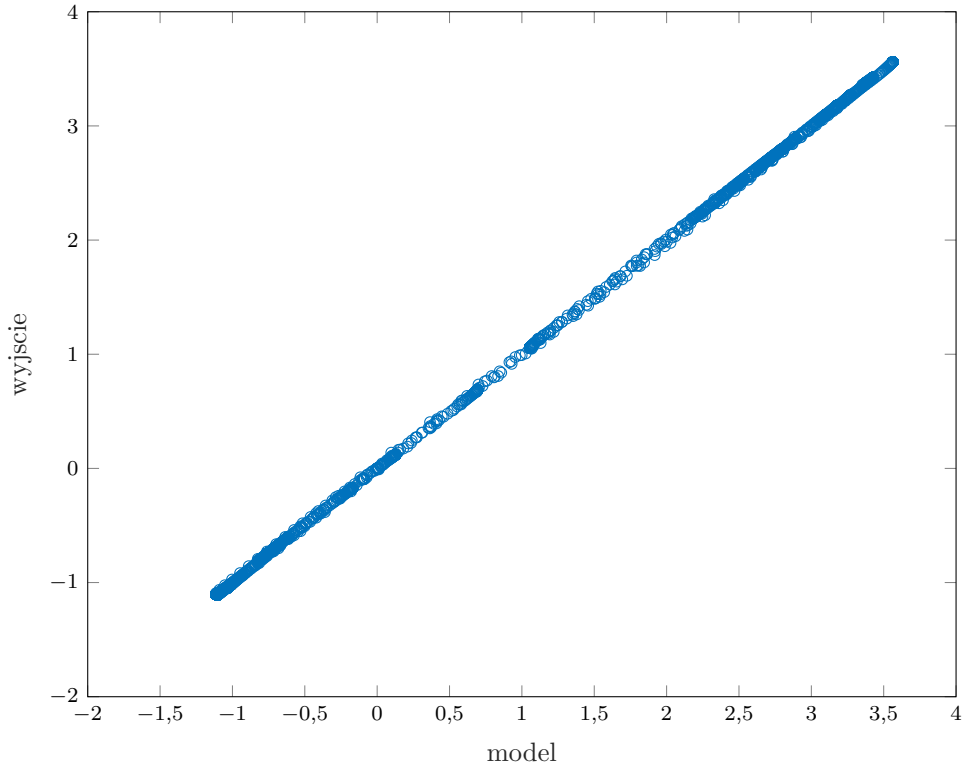
Rys. 3.8



Rys. 3.9



Rys. 3.10



Rys. 3.11

Jak widać na rysunkach 3.8 i 3.10 przebieg modelu w trybie OE pokrywa się z rzeczywistym przebiegiem obiektu. Relacja modelu z wyjściem dla obu zbiorów (rys. 3.9 i 3.9) się układa w linię prostą co także świadczy o dobrej jakości tego modelu. Ten model mimo, że był optymalizowany względem błędu w trybie ARX, okazał się także dobrym modelem w trybie OE, jednak w ogólnym przypadku dobry model w trybie ARX nie oznacza, że model w trybie OE też będzie dobry.

3.8. Model liniowy wyznaczony metodą najmniejszych kwadratów

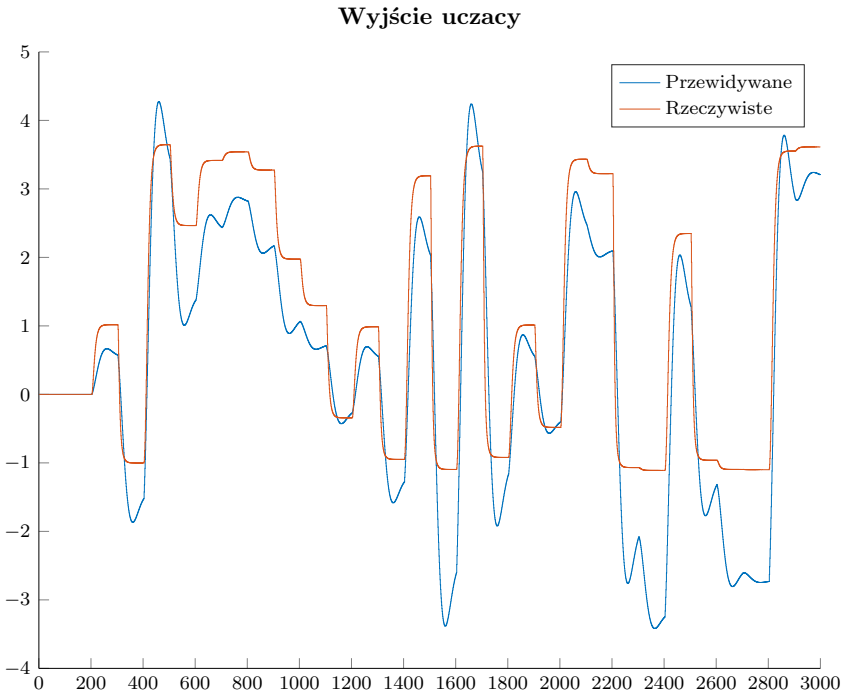
Został założony model liniowy drugiego rzędu z opóźnieniem równym 4

$$\hat{y}(k) = b_4 u(k-4) + b_5 u(k-5) - a_1 y(k-1) - a_2 y(k-2) \tag{3.2}$$

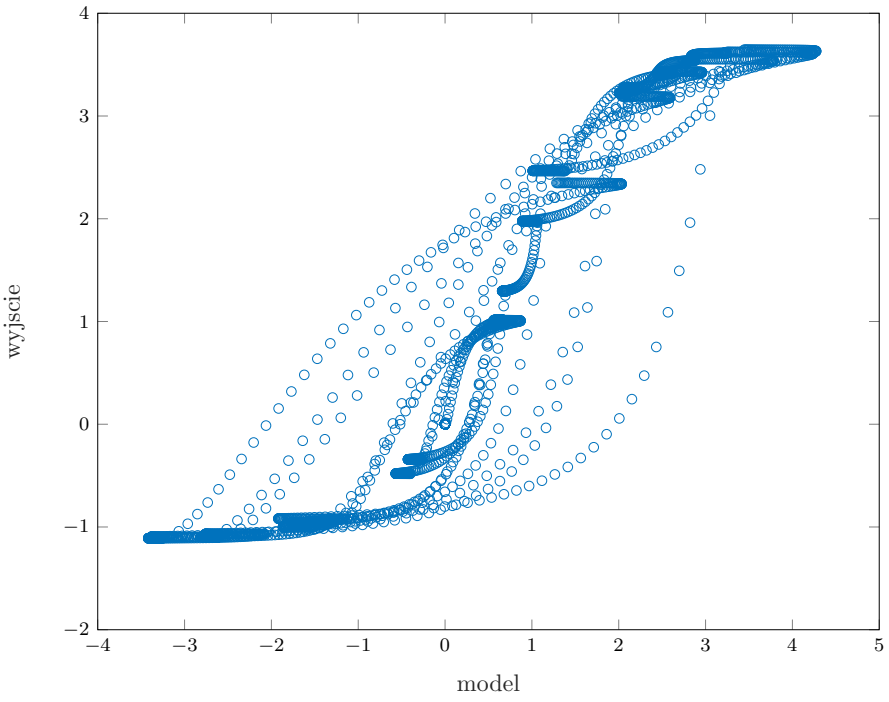
Współczynniki zostały wygenerowane poprzez kod:

```
dane_tr = load("dane.txt");
u_tr = dane_tr(:,1);
y_tr = dane_tr(:,2);

M = [];
for i=6:length(y_tr)
    row = [u_tr(i-4) u_tr(i-5) -y_tr(i-1) -y_tr(i-2)];
    M = [M;row];
end

y_tr = y_tr(6:end);
```

```
b = M\y_tr;
```

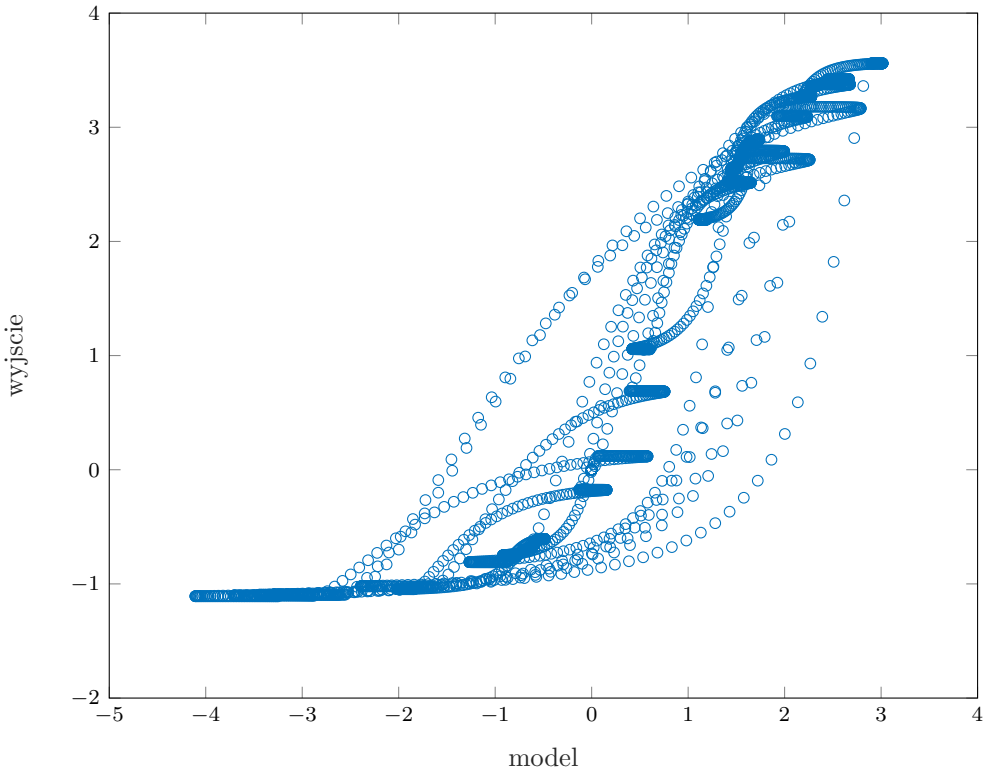


Rys. 3.12



Rys. 3.13

Rys. 3.14



Rys. 3.15

Jak widać na rysunkach 3.12 i 3.14 przebieg modelu w trybie OE bardzo odbiega od przebiegu rzeczywistego wyjście. Relacja modelu z wyjściem dla obu zbiorów (rys. 3.13 i 3.15) nie układa się w linie prostą co także sygnalizuje, że ten model jest zły. Zła jakość tego modelu wynika z tego, że chcieliśmy przybliżyć model nieliniowy modelem liniowym.

4. Zadanie 3

4.1. Wybór przybornika Matlab

Za testowany przybornik wybraliśmy Deep Learning *Toolboox*. Będziemy korzystać z implementacji sieci neuronowej - $FeedForwardNet$, dzięki której w prosty sposób można zaimplementować prostą sieć neuronową. Do inicjalizacji sieci potrzeba napisać prostą jedną linijkę kodu:

```
net = feedforwardnet(X, trainf);
```

gdzie:

X - ilość neuronów ukrytych.
$trainf$ - nazwa funkcji uczącej.

Do rozpoczęcia trenowania sieci należy użyć komendy:

```
net = train(net,LU,LY);
```

gdzie:

LU - macierz danych uczących - wejścia sieci.
LY - macierz danych uczących - wyjścia sieci.

Żeby otrzymać predykcje wyjścia sieci należy użyć komendy:

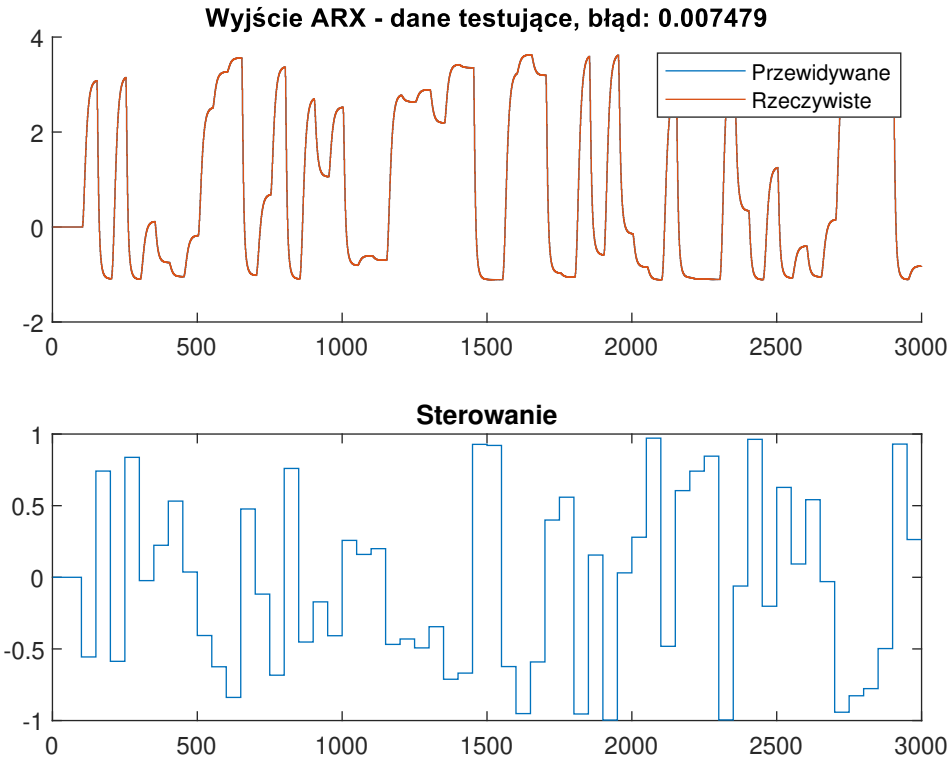
```
Prediction = sim(net, Inputs);
```

gdzie:

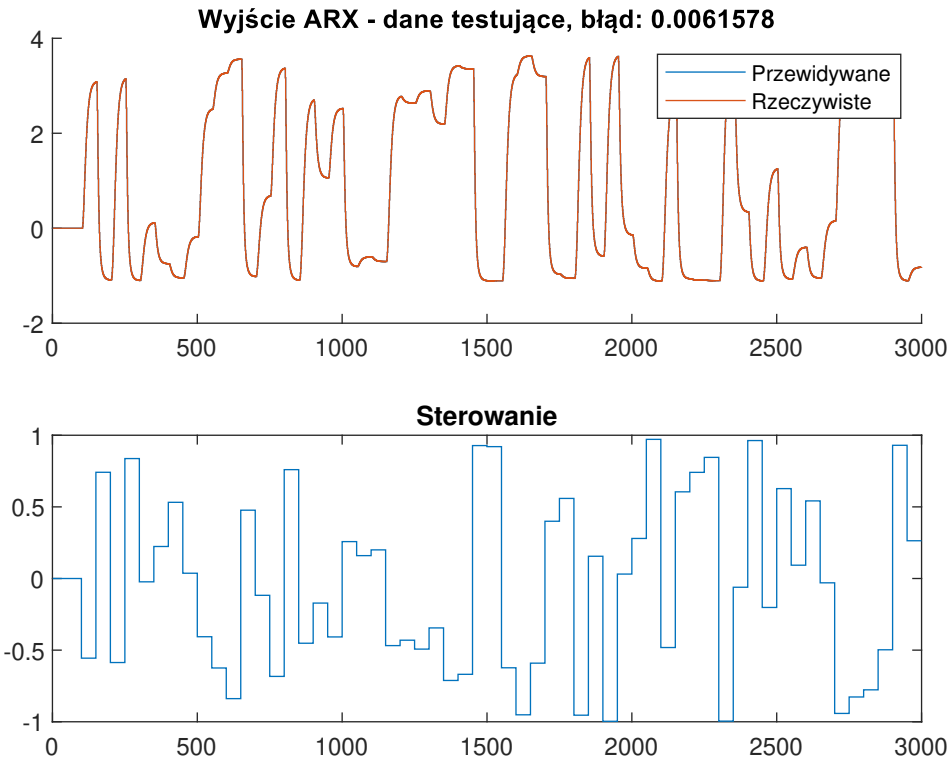
$Prediction$ - predykcja sieci - wyjście.
$Inputs$ - wejścia sieci.

4.2. Uczenie sieci za pomocą przybornika

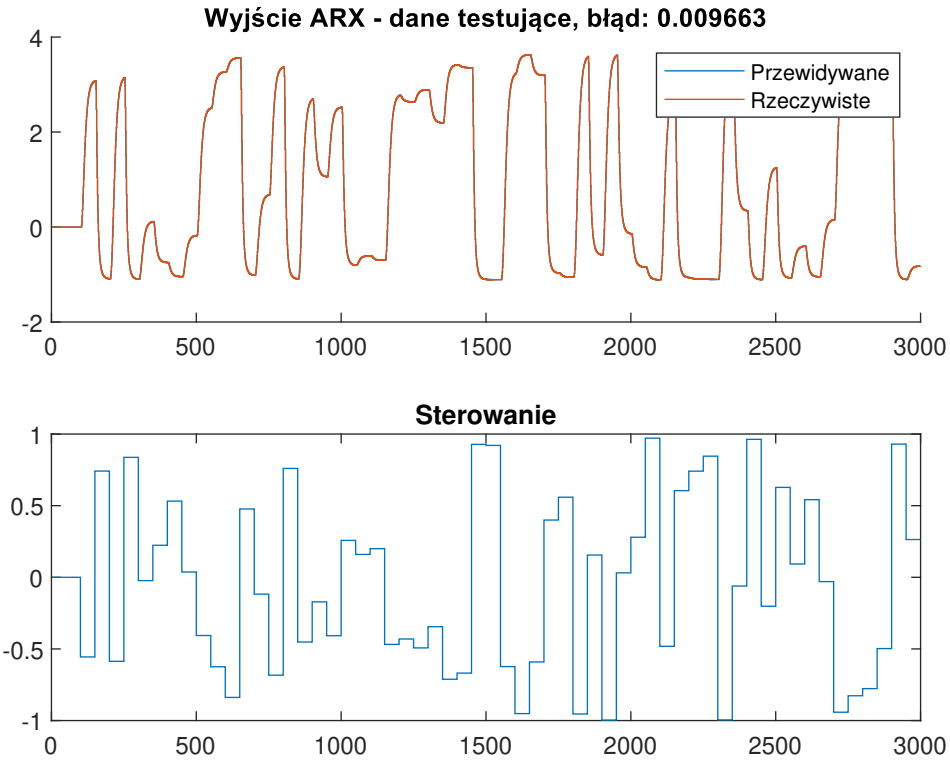
Przeprowadziliśmy symulację uczenia sieci za pomocą przybornika 3 razy. Poprzez zmiany rng(X) (gdzie $X = 1, 2, 3, 4, 5$ zależnie od próby). Rezultaty: Algorytm uczący - Levenberga-Marquardta: Próba 1:



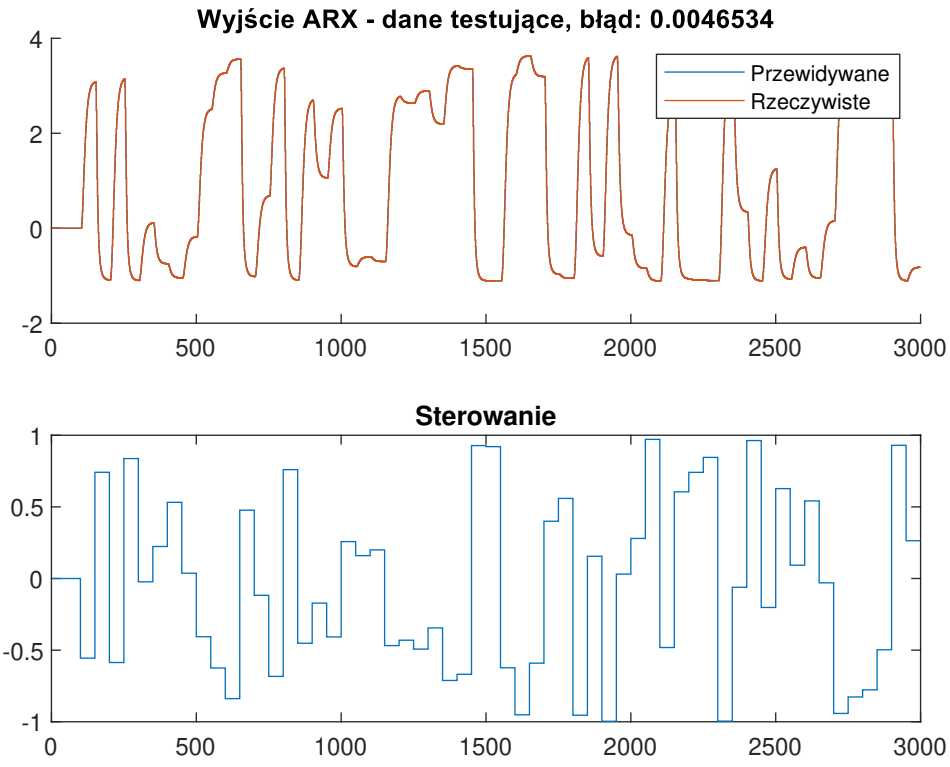
Próba 2:



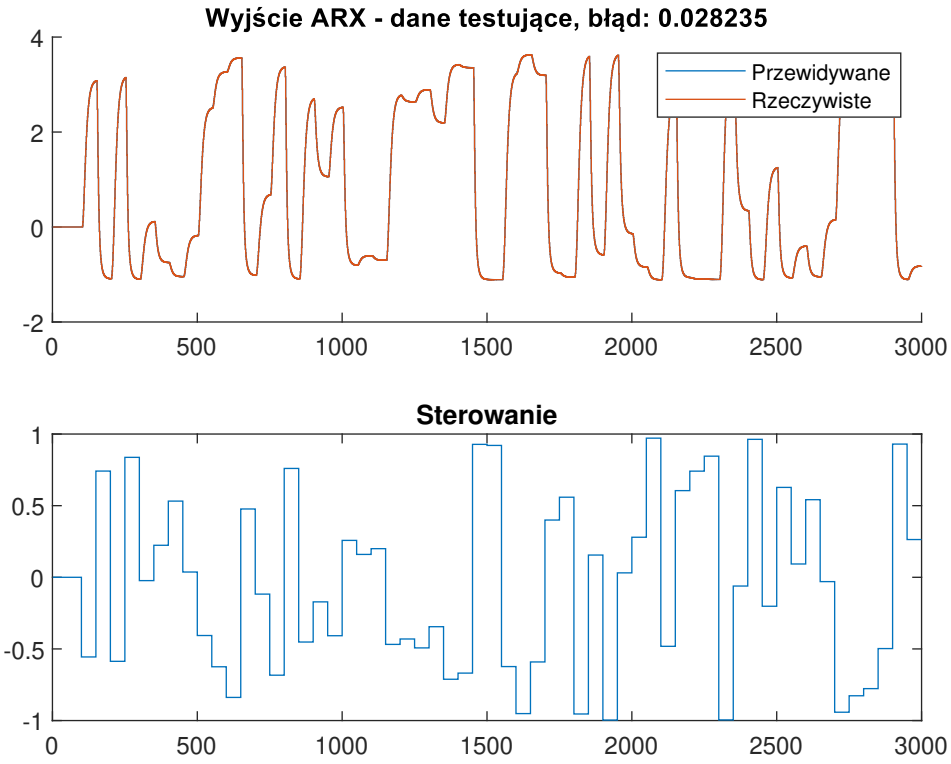
Próba 3:



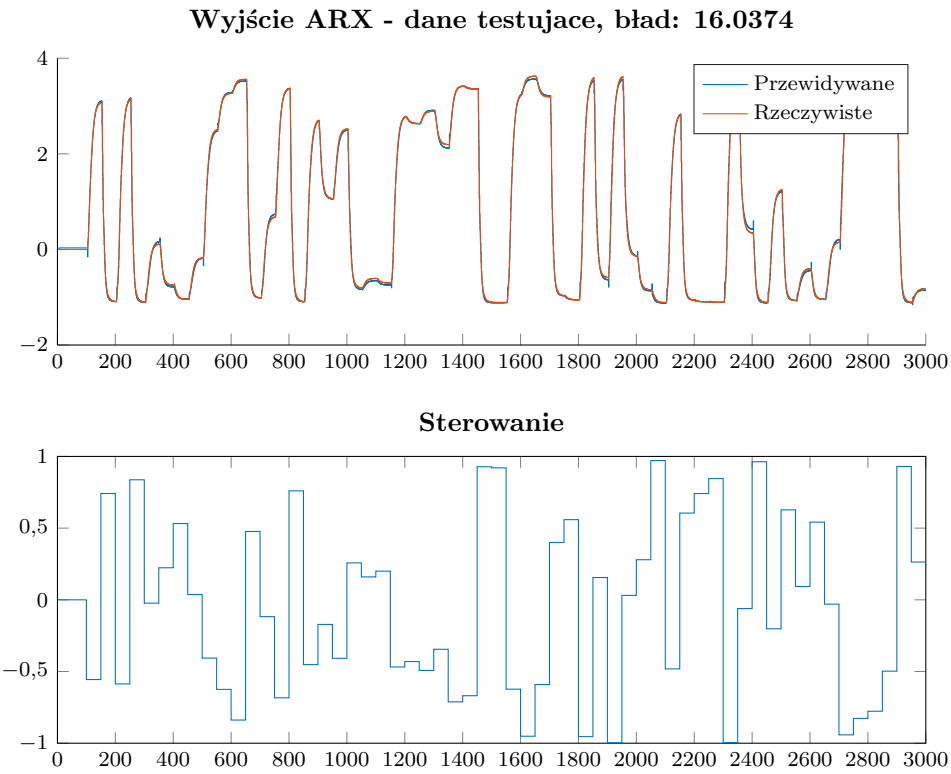
Próba 4:



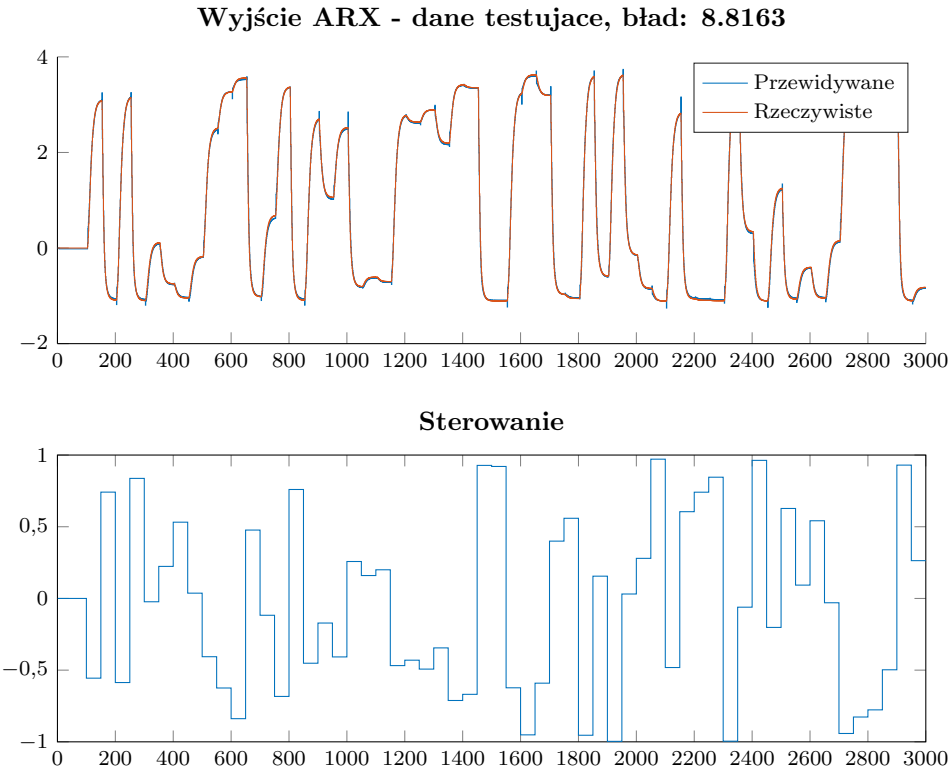
Próba 5:



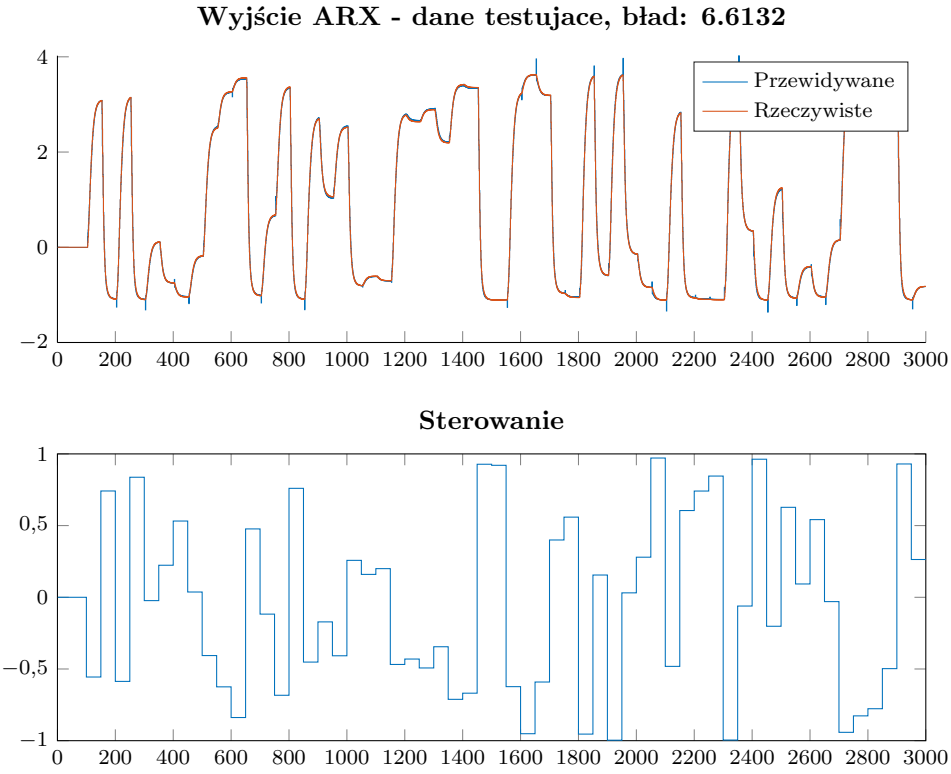
Algorytm uczący - algorytm gradientów sprzężonych: Próba 1:



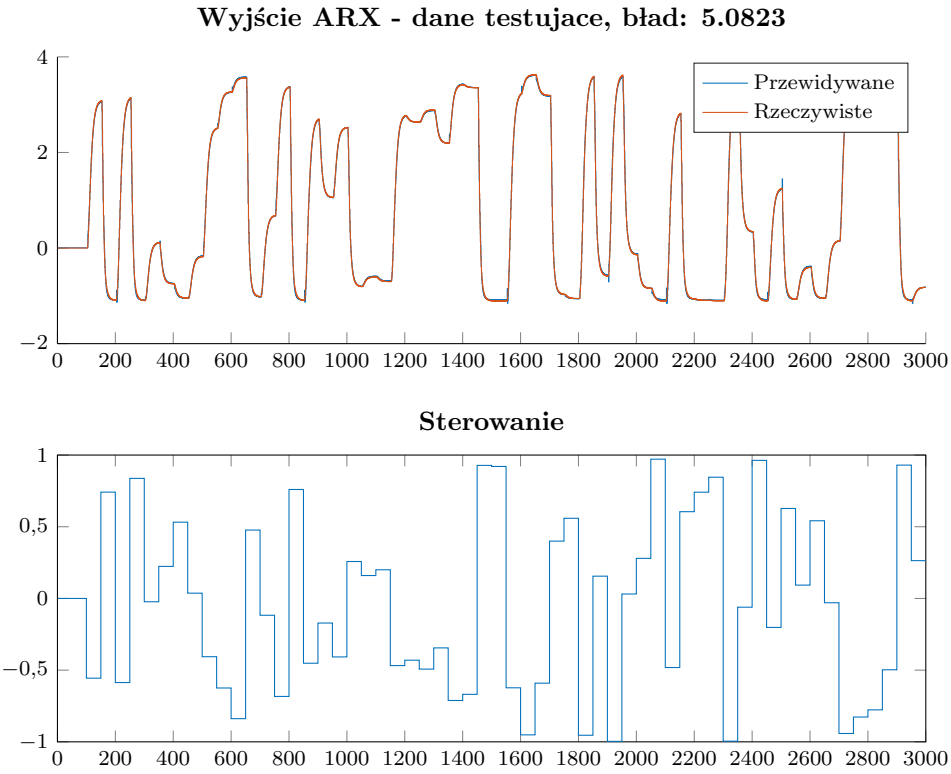
Próba 2:



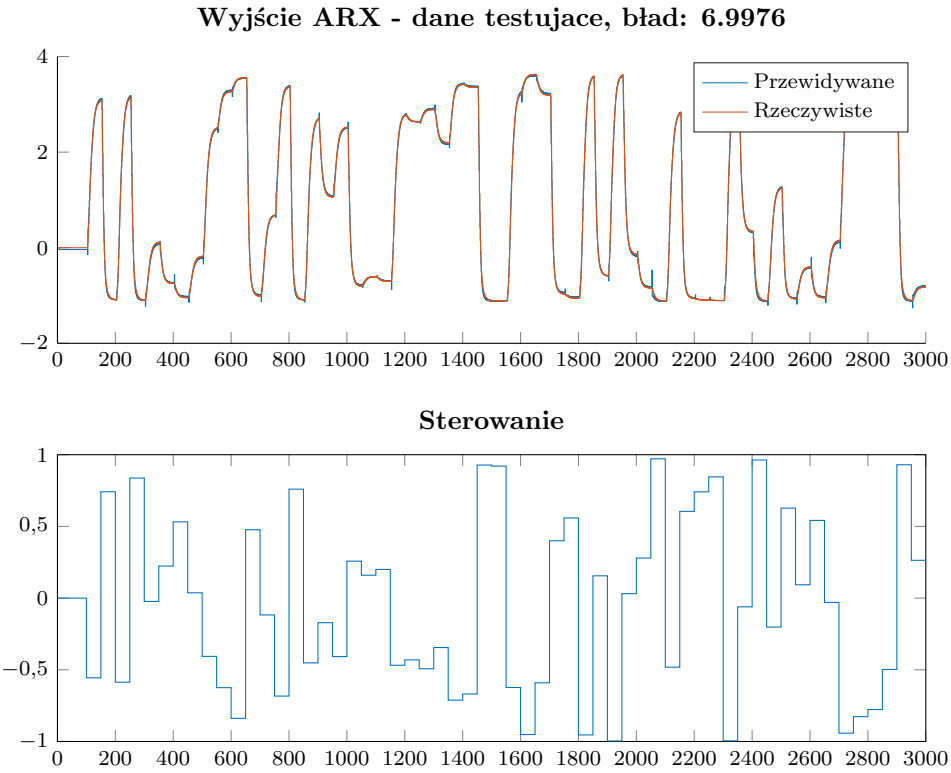
Próba 3:



Próba 4:



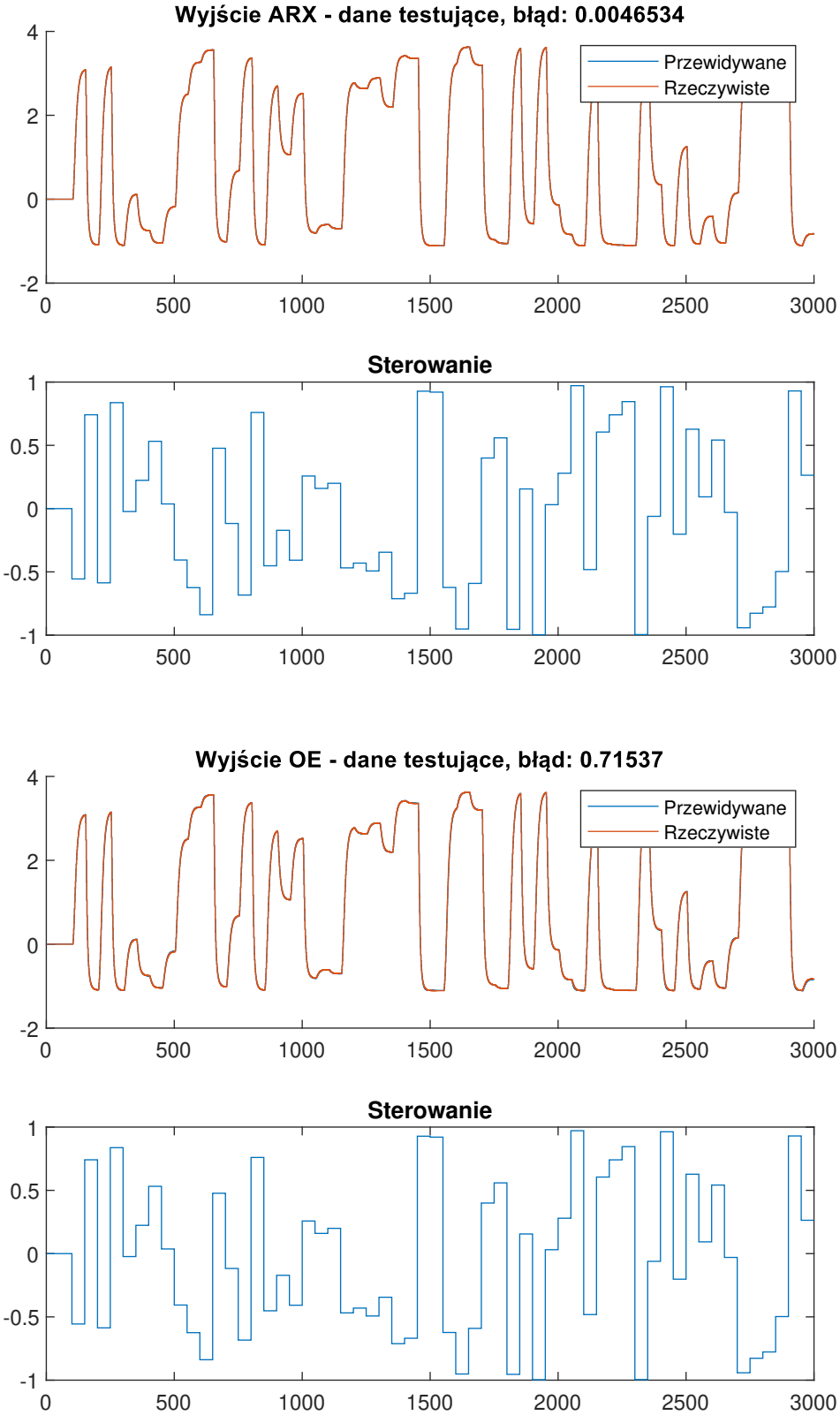
Próba 5:



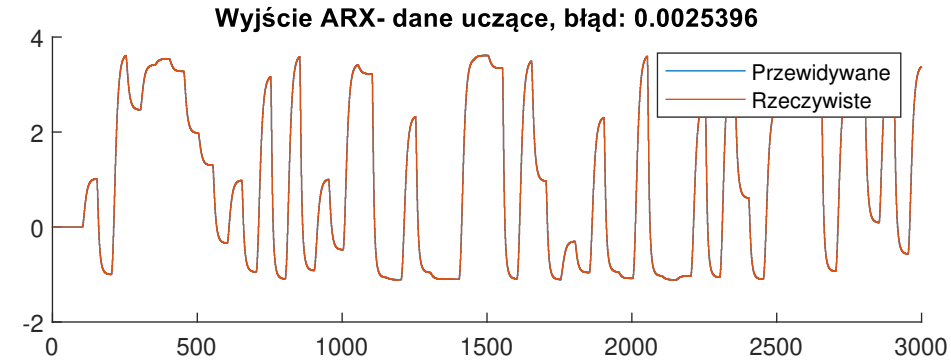
Z powyższych wykresów można zauważyć, że najmniejszy błąd modelu neuronowego jest dla algorytmu uczącego Levenberga-Marquardta, próby 4. Widać również, że algorytm uczący gradientów sprzężonych zdecydowanie gorzej sobie poradził.

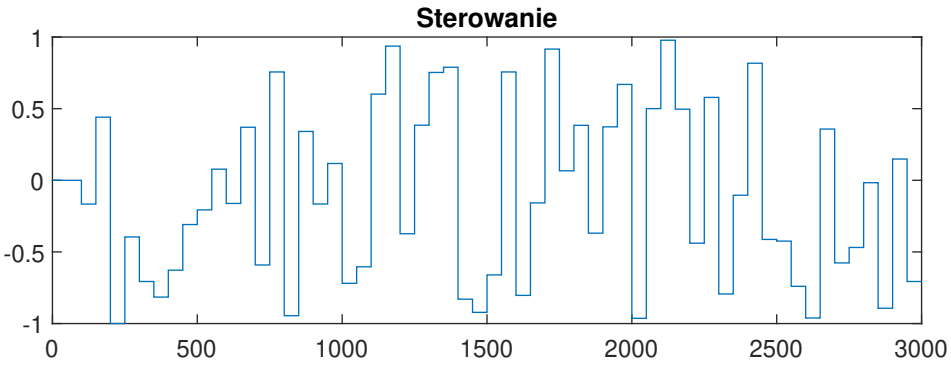
4.3. Symulacja w trybie ARX i OE

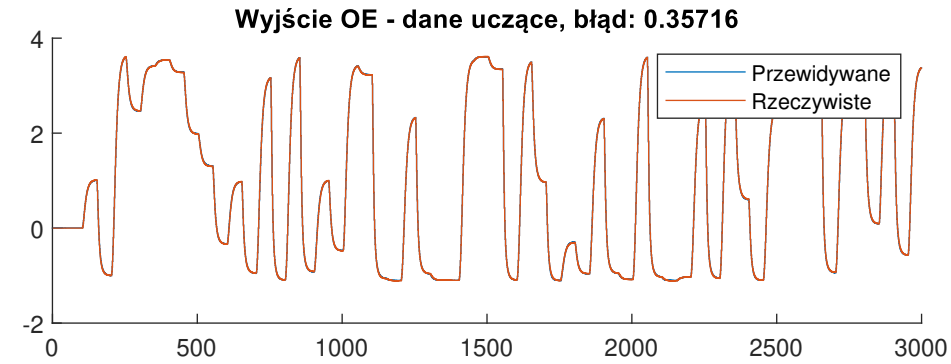
Dla najlepszego modelu wybranego w poprzednim przetestowaliśmy model w trybie dane testujące - Oe i dane uczące - Arx i Oe.

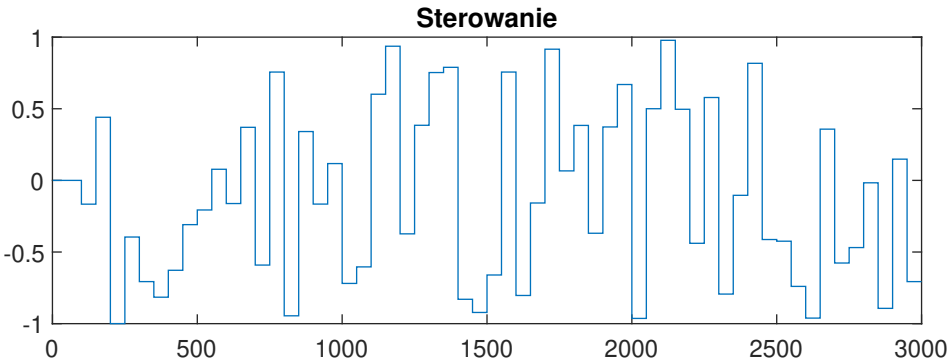


Rys. 4.1









4.4. Porównanie jakości przybornika z programem sieci

Porównując błędy dla danych testujących między najlepszymi modelami otrzymanymi z programu sieci 3.1 (błąd 0,053) i przybornika Matlab 4.1 (błąd 0,715) można zauważyć, że model otrzymany z programu sieci jest lepszy niż model przybornikowy, chodź obydwa bardzo dobrze pokrywają się z rzeczywistym wyjściem.

5. Zadanie 4

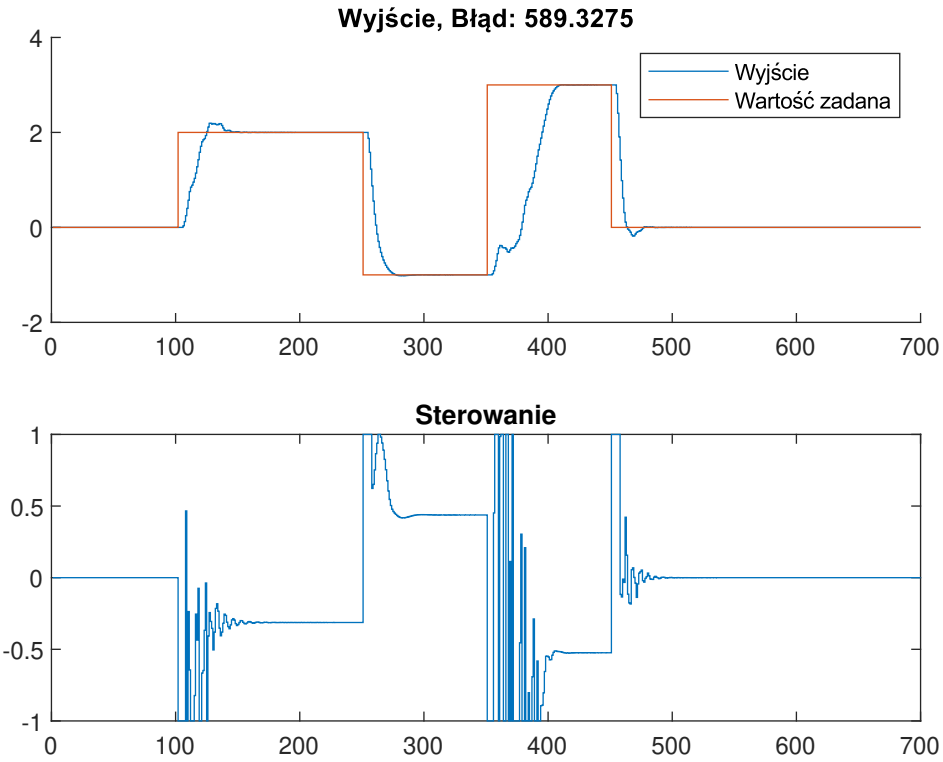
5.1. Implementacja NPL

Zaimplementowaliśmy algorytm regulacji predykcyjnej z Nieliniową Predykcja i Linearyzacją (NPL) bazujący na najlepszym znalezionym modelu neuronowym wyznaczonym w zadaniu II projektu (9 neuronów w warstwie ukrytej) w wersji analitycznej.

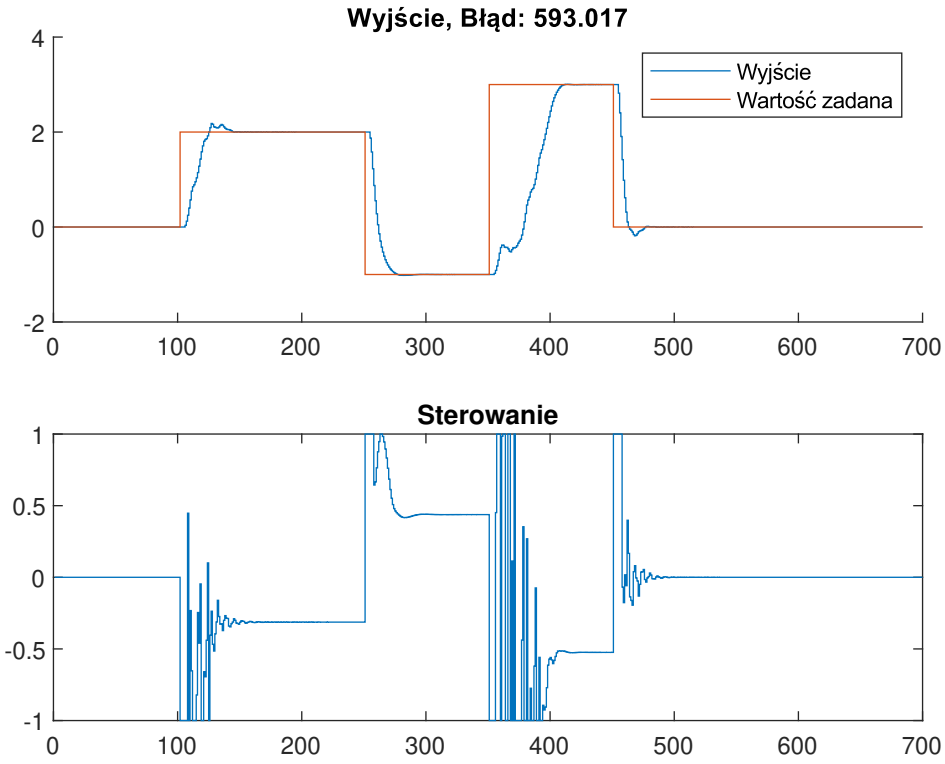
W głównej pętli regulacji po symulacji obiektu obliczane są parametry $a1$, $a2$, $b4$, $b5$ - linearyzacja modelu neuronowego i $f0$ - wyjście z modelu neuronowego w danej chwili. Na ich podstawie otrzymujemy odpowiedź skokową, z której otrzymujemy macierz M, z której obliczamy macierz K. Potem liczony jest wektor odpowiedzi swobodnej - $Y0$, na podstawie predykcji (otrzymywanej z sieci neuronowe) zmiany wyjścia gdyby od poprzedniej chwili nie zmieniało się sterowanie. Na końcu obliczana jest macierz DU, której pierwszy element po przycięciu o ograniczenia jest sterowaniem na daną chwilę.

5.2. Strojenie NPL

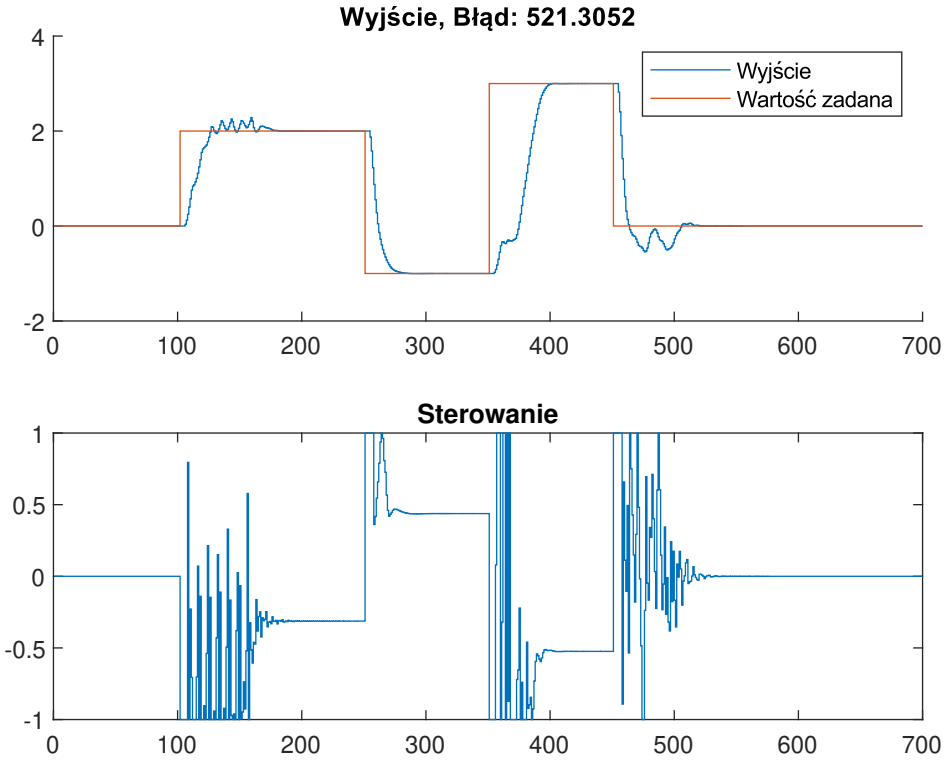
Na podstawie wykresów z zadania 1 podpunktu 2 przyjęliśmy horyzont dynamiki $D = 100$. Test 1 - $N_u = N = 100$, $\lambda = 1$ - błąd wyniósł: 589,33.



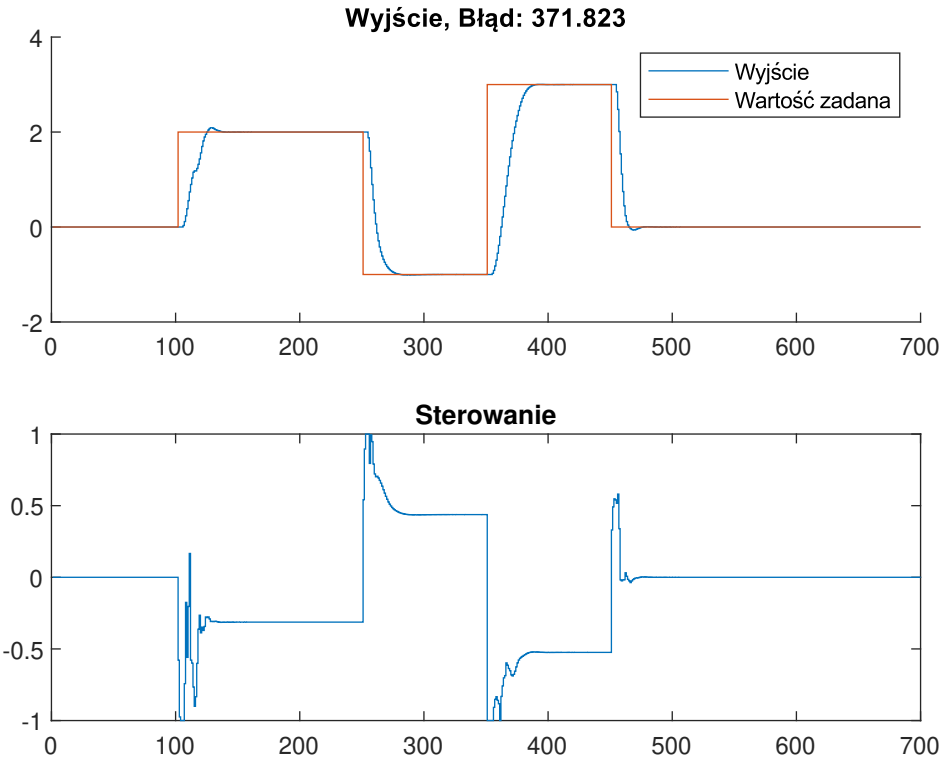
Test 2 - $N = 50$, $N_u = 50$, $\lambda = 1$ - błąd wyniósł: 593,02.



Test 3 - $N = 50$, $N_u = 5$, $\lambda = 1$ - błąd wyniósł: 521,31.



Zwiększyliśmy parametr λ, żeby zmniejszyć skoki sterowania. Test 5 - $\lambda = 10$ - błąd wyniósł: 371,82.

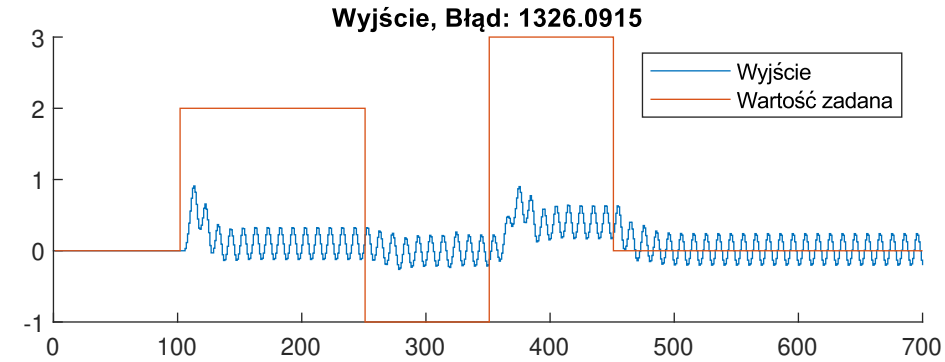


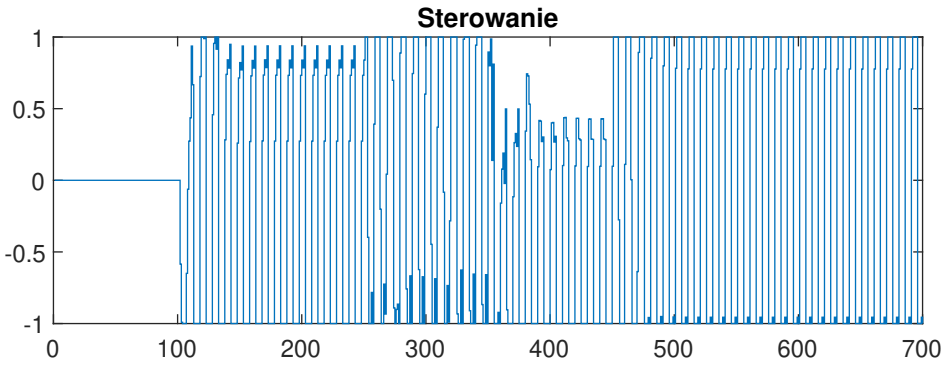
Rys. 5.1

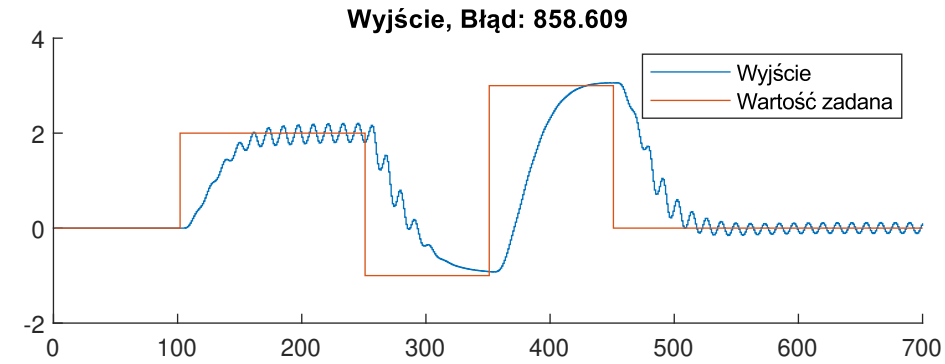
Widać, że algorytm NPL działa bardzo dobrze.

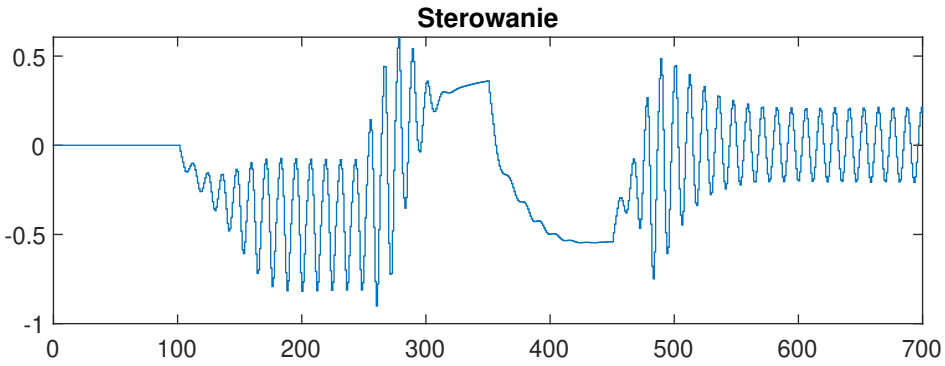
5.3. Implementacja GPC

GPC różni się od NPL implementacją liczenia odpowiedzi swobodnej i sposobem otrzymywania odpowiedzi skokowej. W GPC - w inicjalizacji algorytmu, a w NPL - w każdej iteracji programu. W wykonanym przez nas programie odpowiedź skokowa w GPC liczona jest analitycznie na podstawie współczynników $a1$, $a2$, $b4$ i $b5$ otrzymywanych z zadania 2 - z najmniejszych kwadratów. W NPL otrzymujemy ją z modelu neuronowego trzymanego w zadaniu 2. Obydwie odpowiedzi są wykorzystywane do wyliczenia macierzy M i wektora odpowiedzi swobodnej w dalszych częściach algorytmu.







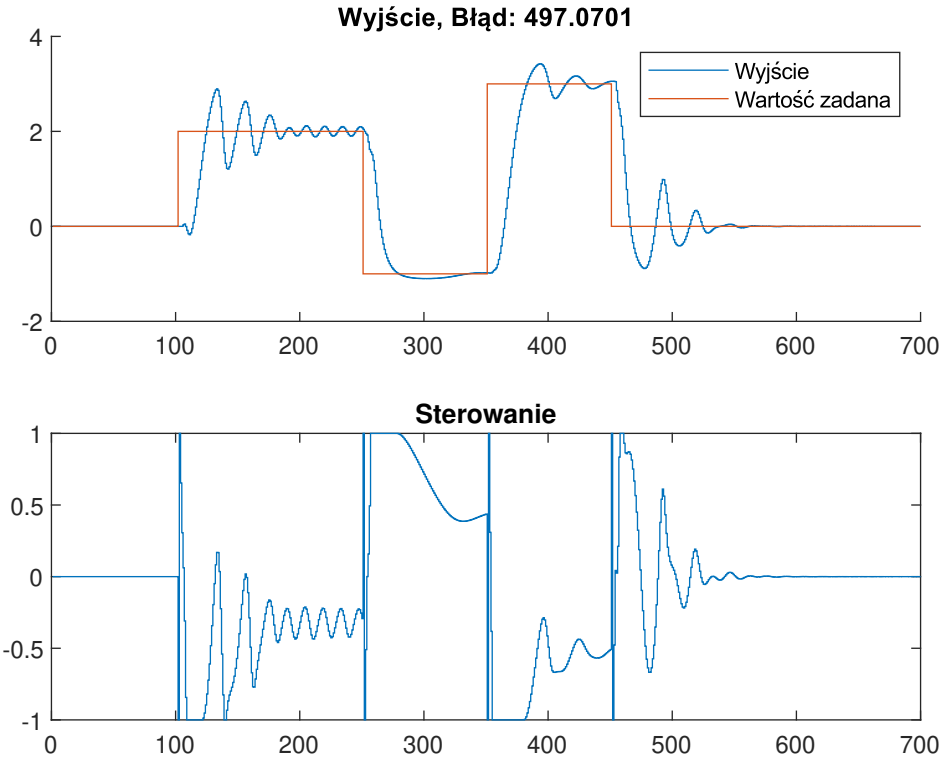


Z wykresów widać, że GPC nie działa dla obiektu nieliniowego. Jedyne wartości, które jest w stanie osiągnąć są w niedalekim sąsiedztwie punktu pracy. Po znacznym zwiększeniu parametru λ (1000 krotnie) algorytm zaczął osiągać zadaną trajektorie.

6. Zadanie dodatkowe

6.1. Dodatkowy PID

Napisaliśmy oddzielny skrypt do algorytmu Pid. Do znalezienia jego parametrów użyliśmy funkcji *fmincon*. Otrzymane parametry przenieśliśmy do skryptu pozwalającego uruchomić wszystkie napisane przez nas algorytmy. Rezultaty okazały się lepsze niż GPC. Może wynikać to z powodu używania przez GPC słabego modelu kwadratowego, który dodatkowo psuje prace regulatora. Pid modelu nie wykorzystuje - co skutkuje, że nie jest obarczony jego błędami.

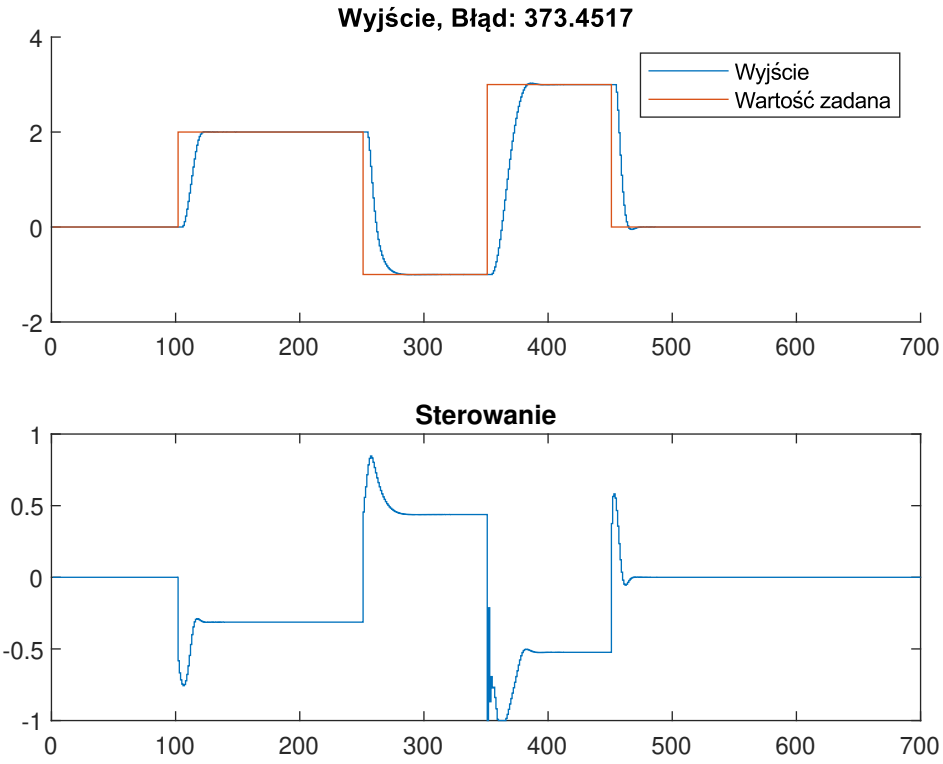


6.2. Dodatkowy NO

Napisaliśmy oddzielną funkcję, w której liczona jest wartość $J(k)$ wykorzystywana w skrypcie przez funkcję *fmincon* do numerycznego znalezienia najlepszego sterowania na daną chwilę.

$$J(k) = \sum_{p=1}^{N}(y^{zad}(k) - \hat{y}(k+p|k))^2 + \lambda \sum_{p=0}^{N_u}(\Delta u(k+p|k))^2 \tag{6.1}$$

Używając tych samych parametrów N, N_u i λ co w najlepszym NPL 5.1 uzyskaliśmy praktycznie taki sam rezultat, tylko w dużo większym czasie.



Nasuwa się wniosek, że przynajmniej dla naszego obiektu zdecydowanie lepszym wyborem jest użycie algorytmu NPL ponieważ czas działania jest dużo lepszy a rezultaty podobne.