Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Sztuczna Inteligencja w Automatyce

Projekt 2

Kacper Marchlewicz, Adam Wróblewski

Warszawa, 2023

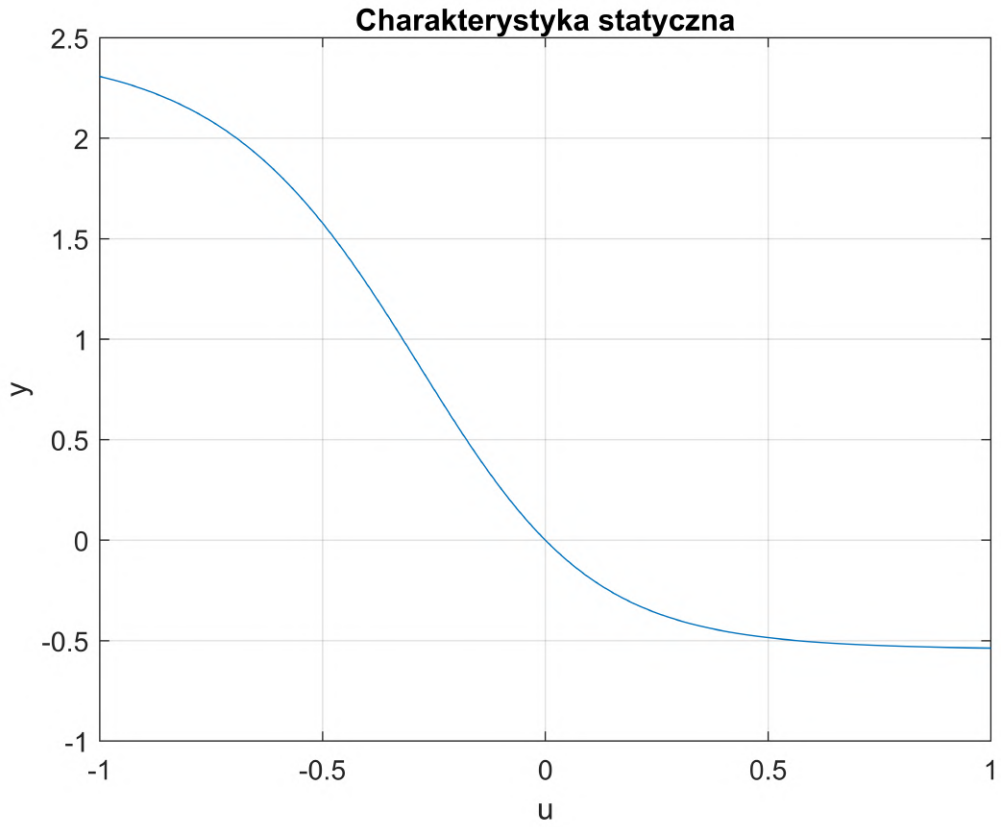
Spis treści

1. Zadanie 1 - Symulacja procesu

1.1. Podpunkt 1 - Wyznaczenie charakterystyki statycznej

Metodą analityczną wyznaczyliśmy charakterystykę statyczną podanego procesu:

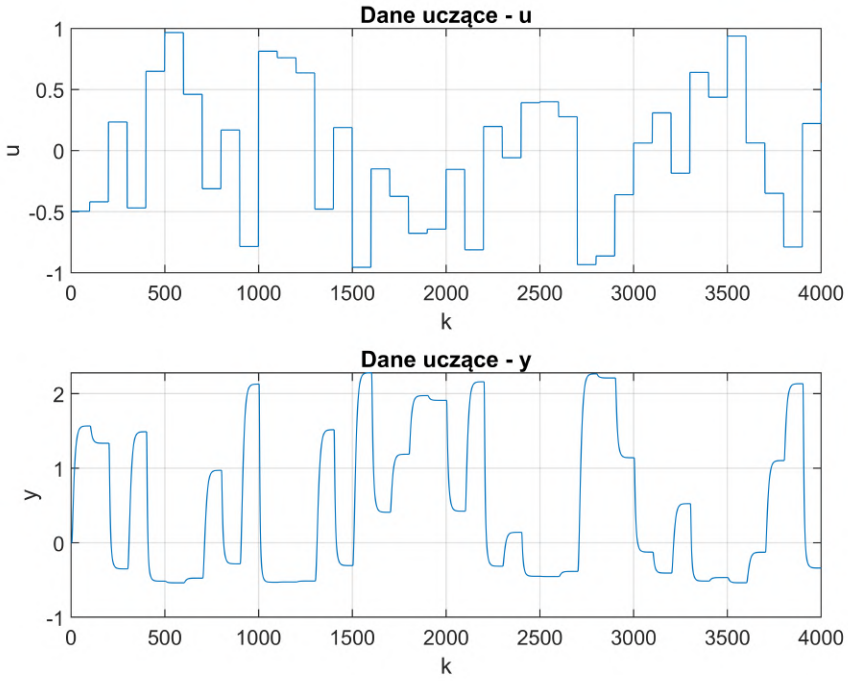
$$y(u) = g_2(\frac{(\beta_1 + \beta_2) * g_1(u)}{1 + \alpha_1 + \alpha_2}) \tag{1.1}$$



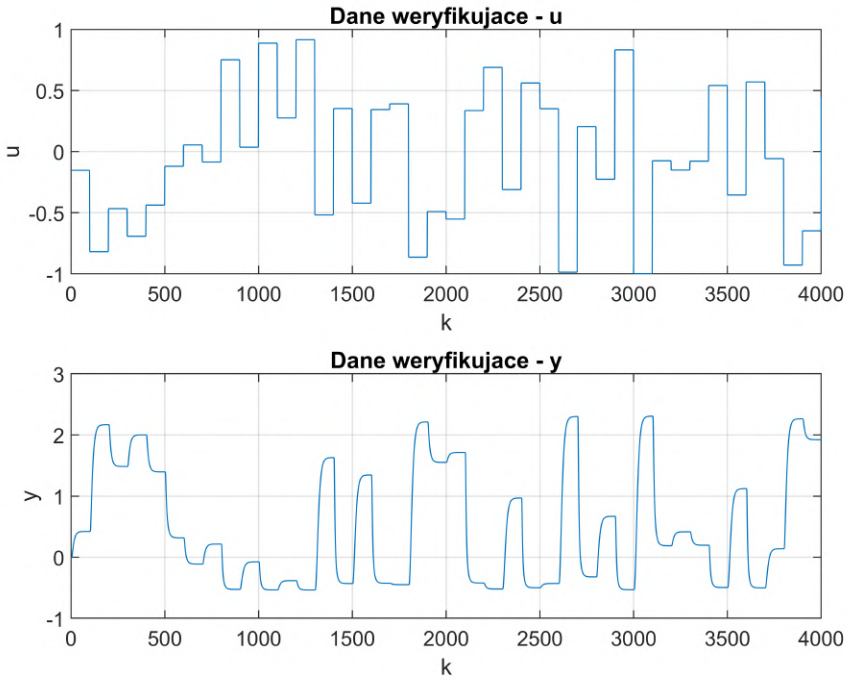
Rys. 1.1: Charakterystyka statyczna procesu

1.2. Podpunkt 2 - Symulacja procesu

Przeprowadziliśmy symulację procesu dla sekwencji zmian sygnału sterującego $u \in < -1; 1 >$. Wygenerowaliśmy dwa zbiory danych: zbiór uczący i zbiór weryfikujący. Każdy ze zbiorów składa się z 4000 próbek, a okres zmian sygnału sterującego wynosi 100 próbek.

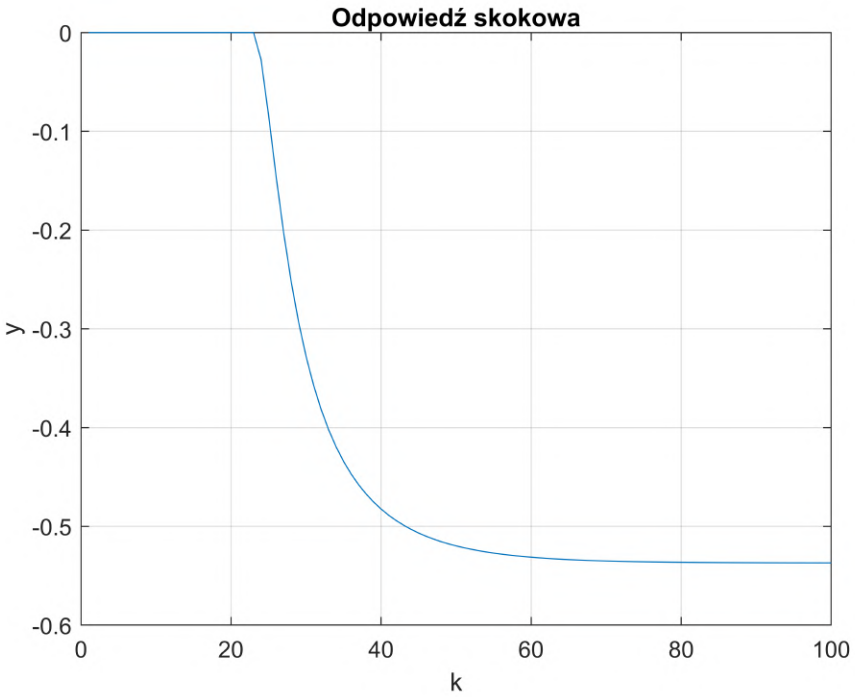


Rys. 1.2: Dane uczące



Rys. 1.3: Dane weryfikujące

Odpowiedź skokowa procesu



Rys. 1.4: Odpowiedź skokowa procesu

2. Zadanie 2 - Modelowanie Procesu

2.1. Podpunkt 1 - Opóźnienie procesu

Opóźnienie procesu odczytaliśmy z jego równiań - wynosi ono 4.

2.2. Podpunkt 2 - Uczenie serii modeli neuronowych

Przeprowadziliśmy uczenie modeli neuronowych w trybie rekurencyjnym (predyktor OE) dla liczby neuronów K = 1...10. Liczba iteracji uczących wynosiła 800. Dla każdej liczby neuronów uczenie zostało powtórzone 5 razy. W poniższych tabelach prezentujemy błędy dla obu zbiorów danych wraz z zaznaczonymi modelami dla których błąd dla zbioru weryfikującego był najmniejszy.

Tab. 2.1: Błąd modelu ARX na zbiorze uczącym

	Blad ARX ucz				
K/n	1	2	3	4	5
1	63,1897	63,1897	63,1897	63,1897	63,1897
2	21,7499	1,0657	1,0679	1,0735	17,4664
3	0,3796	0,3796	0,3796	0,9169	10,0975
4	0,1867	0,2270	0,2761	0,1991	0,2300
5	0,1579	0,1264	0,1319	0,0719	0,1852
6	0,0665	0,1552	0,0585	0,0500	0,0521
7	0,0525	0,0873	0,0897	0,0579	0,0241
8	0,0683	0,0557	0,0094	0,0207	0,0433
9	0,0224	0,0242	0,0411	0,0231	0,0241
10	0,0271	0,0331	0,0173	0,0067	0,0188

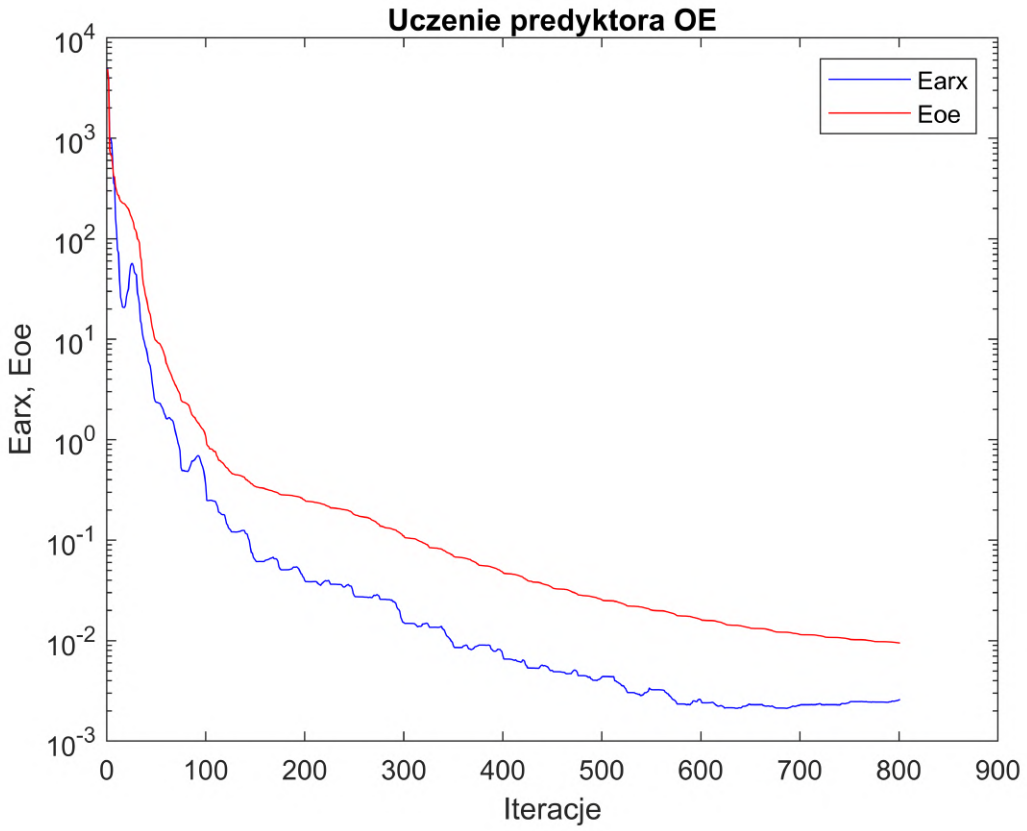
Tab. 2.2: Błąd modelu ARX na zbiorze weryfikującym

	Blad ARX wer				
K/n	1	2	3	4	5
1	82,3721	82,3721	82,3721	82,3721	82,3721
2	80,8514	1,1273	1,1701	1,1268	27,5667
3	0,6330	0,6332	0,6330	1,0766	96,5959
4	0,3495	0,3328	0,4432	0,6692	0,3573
5	0,7817	0,1901	2,2605	0,1629	0,5844
6	0,4196	0,3613	0,2125	0,1107	0,3403
7	0,0978	0,2041	0,1585	0,2843	0,3325
8	0,2887	0,3303	0,0164	0,0386	0,1255
9	0,1382	0,0777	0,0747	0,0459	0,0643
10	0,2371	0,0464	0,0420	0,1099	0,6889

Wraz ze zwiększeniem liczby neuronów maleje błąd uczący jak i weryfikujący. Wynika to z faktu że zwiększając liczbę neuronów model może lepiej dopasować się do funkcji/obiektu.

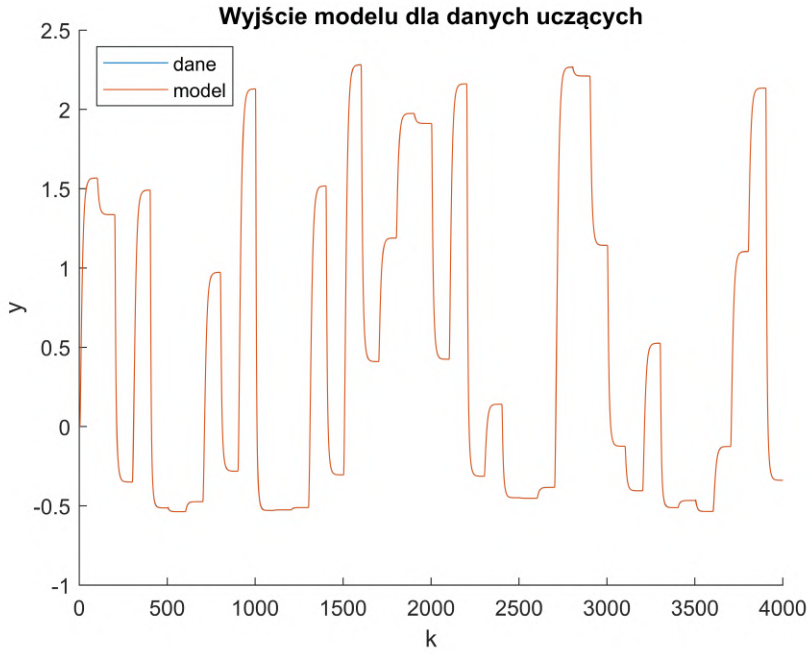
2.3. Podpunkt 3 - Wybór najlepszego modelu neuronowego

Najlepszym modelem neuronowym jest model o 8 neuronach w warstwie ukrytych - dał najmniejszy błąd spośród wszystkich modeli.

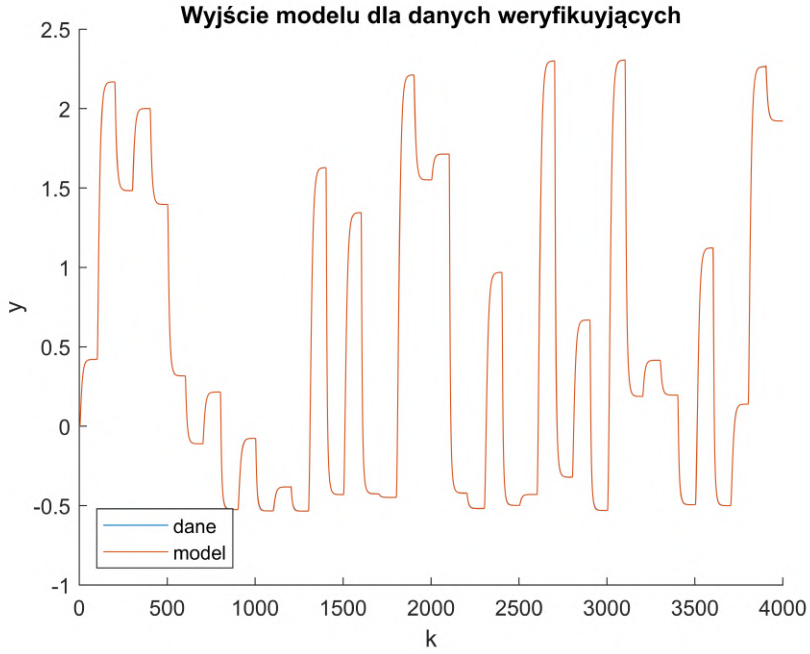


Rys. 2.1: Zmiana błędu predyktora ARX i OE w kolejnych iteracjach uczących.

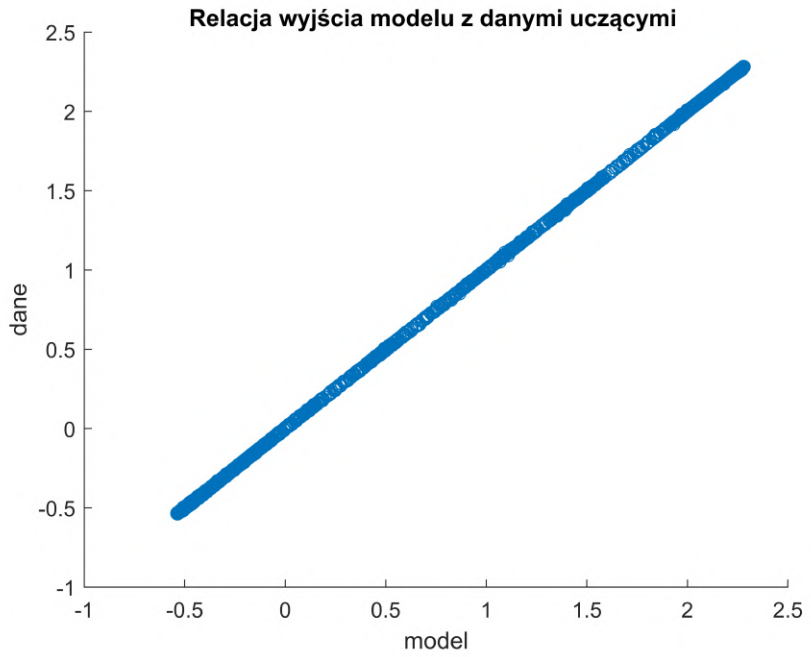
2.4. Podpunkt 4 - Symulacja w trybie rekurencyjnym



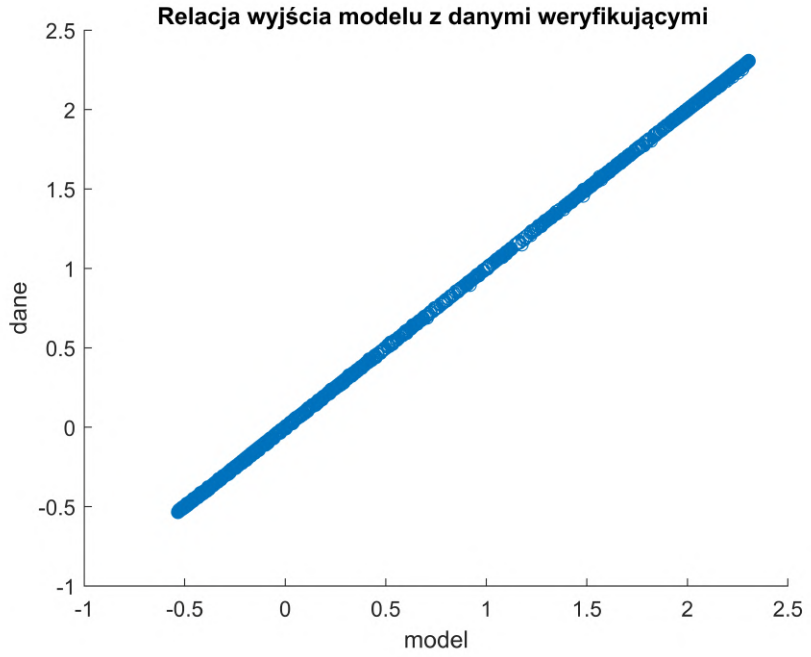
Rys. 2.2: Porównanie modelu z danymi uczącymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



Rys. 2.3: Porównanie modelu z danymi weryfikującymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



Rys. 2.4: Relacja wyjścia modelu z danymi uczącymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



Rys. 2.5: Relacja wyjścia modelu z danymi weryfikującymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.

Błąd modelu na zbiorze uczącym = 0,005. Błąd modelu na zbiorze weryfikującym = 0,0165.

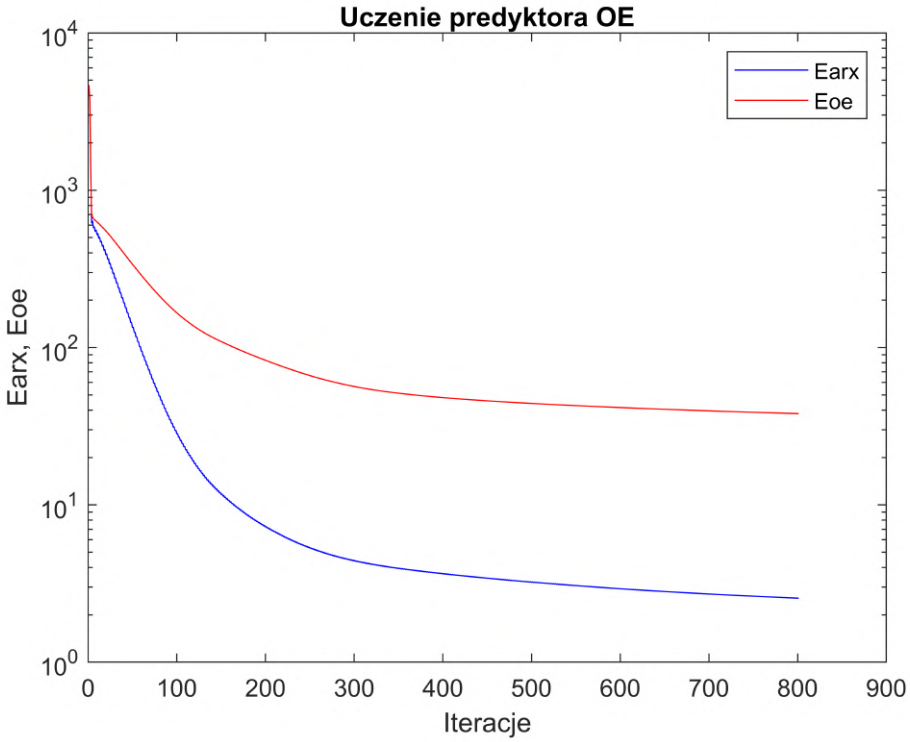
Model neuronowy o liczbie neuronów ukrytych równej 8 dobrze odwzorowuje zarówno dane uczące jak i weryfikujące, błędy na tych zbiorach są niewielkie.

2.5. Podpunkt 5 - Uczenie modelu neuronowego w trybie rekurencyjnym

W programie *sieci.exe* przeprowadziliśmy uczenie modeli neuronowych w trybie rekurencyjnych z wykorzystaniem algorytmu najszybszego spadku.

Tab. 2.3: Błąd modelu na zbiorze weryfikującym

1	2	3	4	5
69,8957	59,2772	63,2477	69.,2893	66,5400

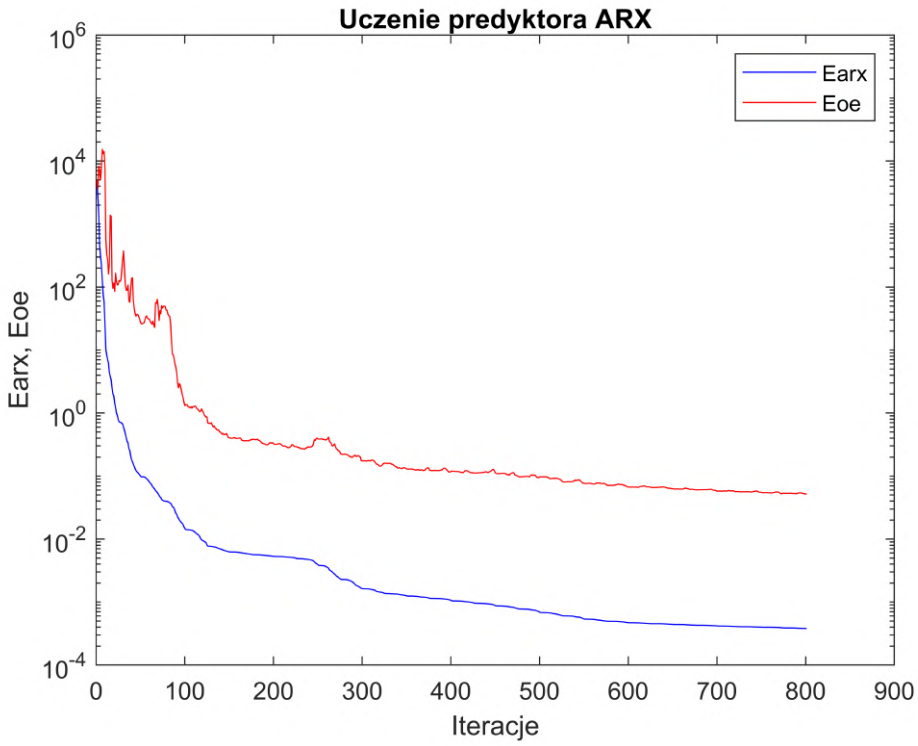


Rys. 2.6: wykres zmiany błędów predyktora ARX i OE w kolejnych iteracjach uczących.

2.6. Podpunkt 6 - Uczenie modelu neuronowego w trybie bez rekurencji

Tab. 2.4: Błąd modelu na zbiorze weryfikującym

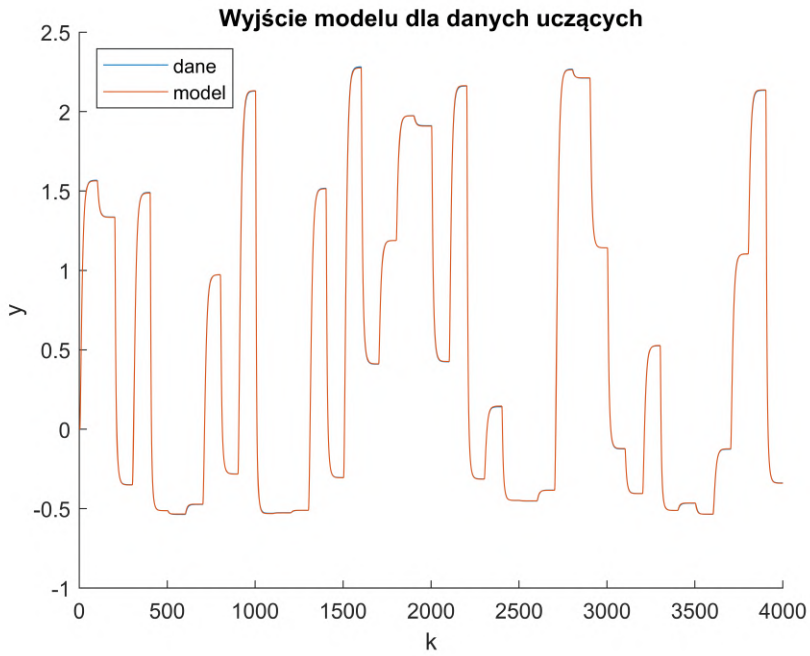
1	2	3	4	5
0,1695	0,1677	0,1863	0,4878	0,1700



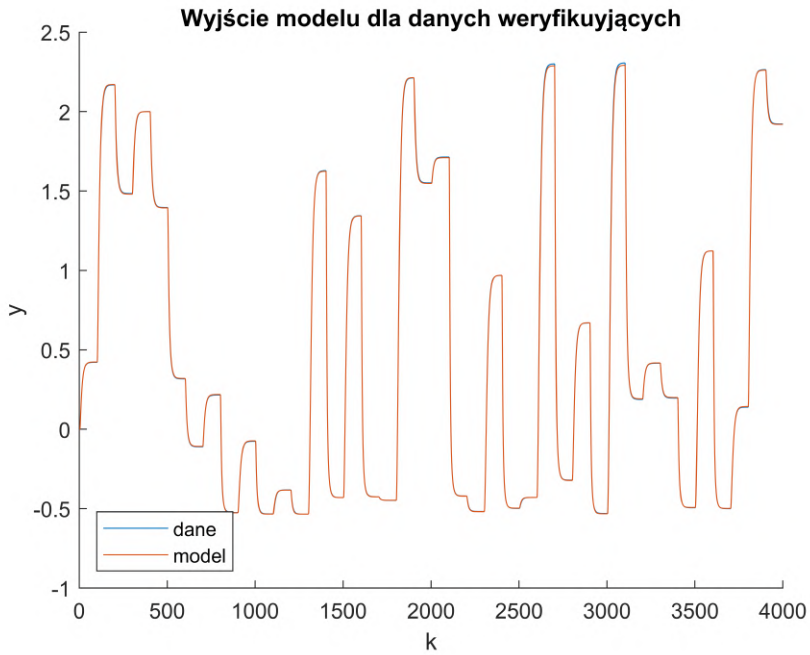
Rys. 2.7: wykres zmiany błędów predyktora ARX i OE w kolejnych iteracjach uczących.

2.7. Podpunkt 7 - Symulacja w trybie rekurencyjnym

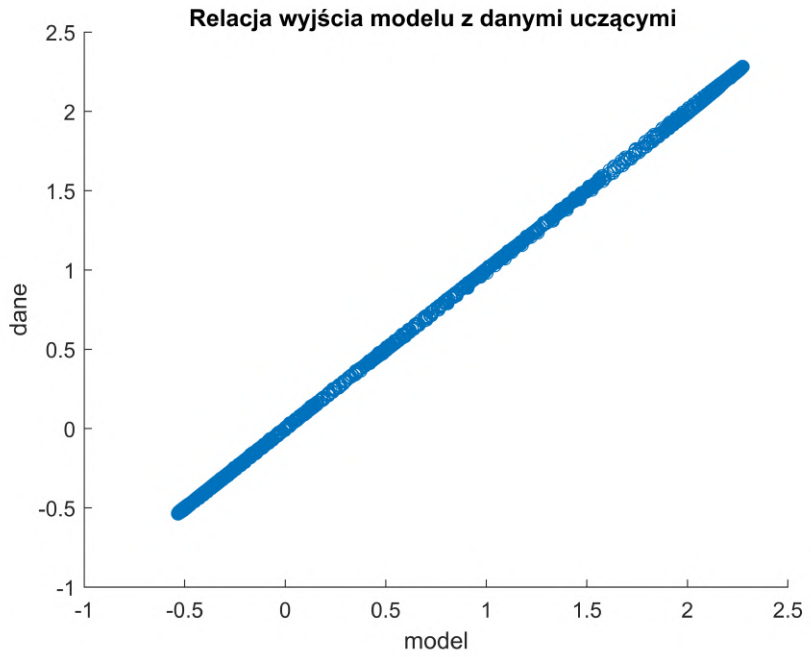
Symulacja wybranego model neuronowy uczonego w trybie ARX z wykorzystaniem algorytmu BSGS.



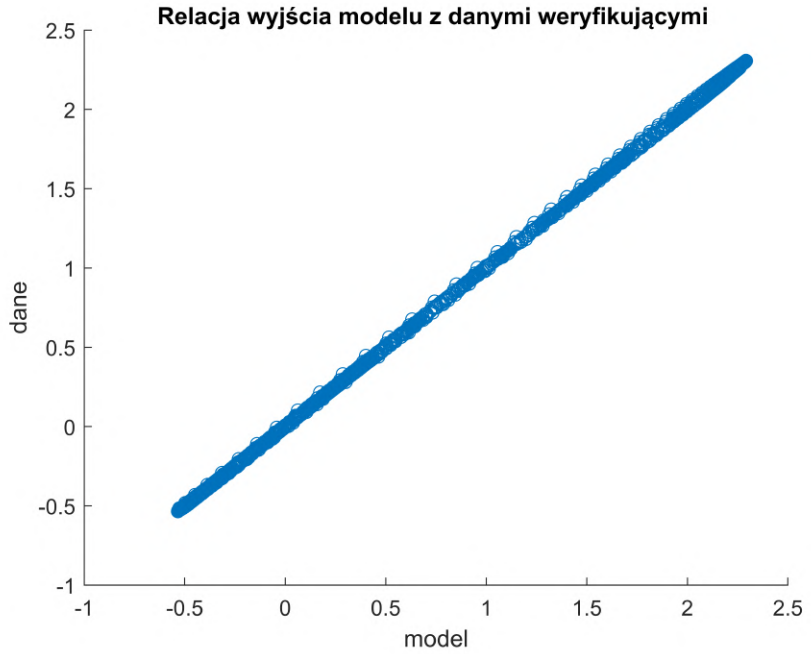
Rys. 2.8: Porównanie modelu z danymi uczącymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



Rys. 2.9: Porównanie modelu z danymi weryfikującymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



Rys. 2.10: Relacja wyjścia modelu z danymi uczącymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.



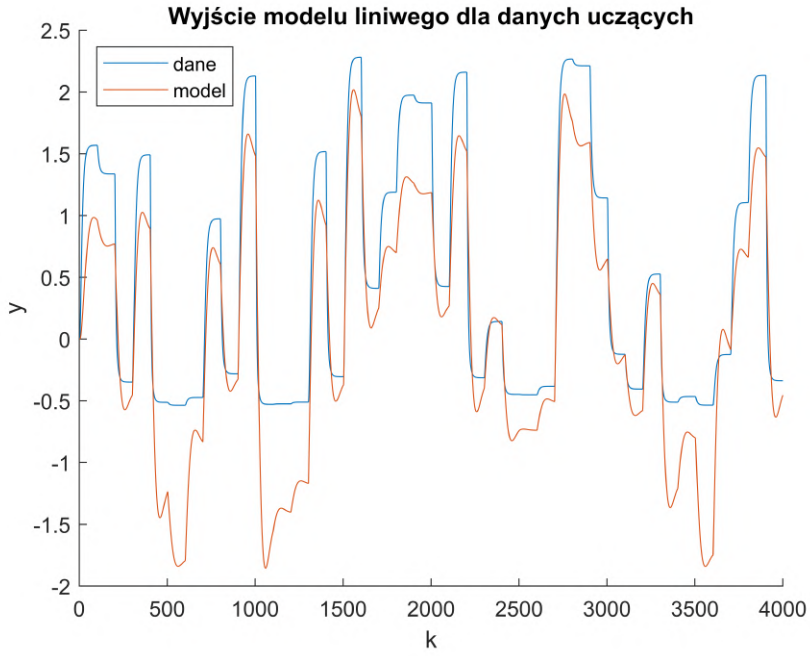
Rys. 2.11: Relacja wyjścia modelu z danymi weryfikującymi dla najlepszego modelu neuronowego - 8 neuronów ukrytych.

Błędy modeli:

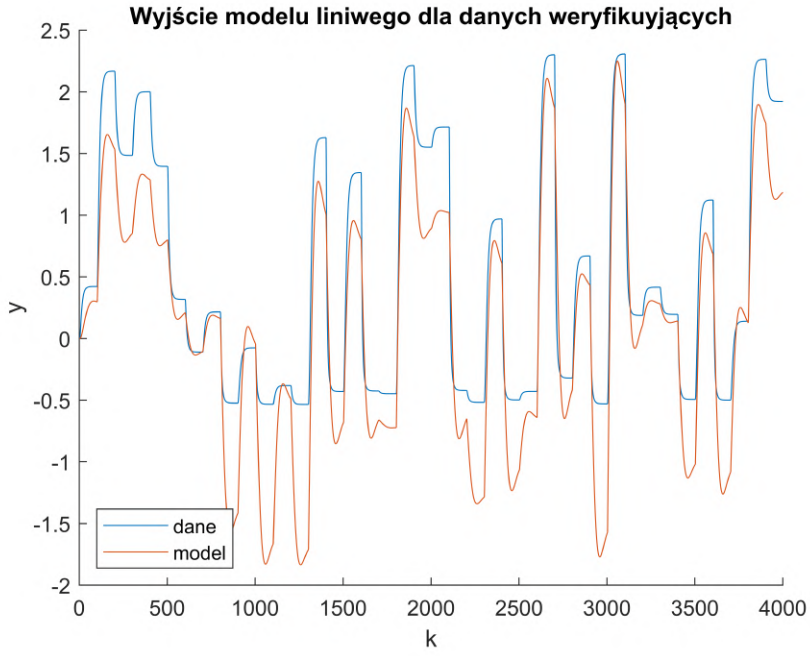
— błąd modelu na zbiorze uczącym = 0,0514.
— błąd modelu na zbiorze weryfikującym = 0,1677.

2.8. Podpunkt 8 - Wyznaczenie modelu liniowego o dynamice drugiego rzędu

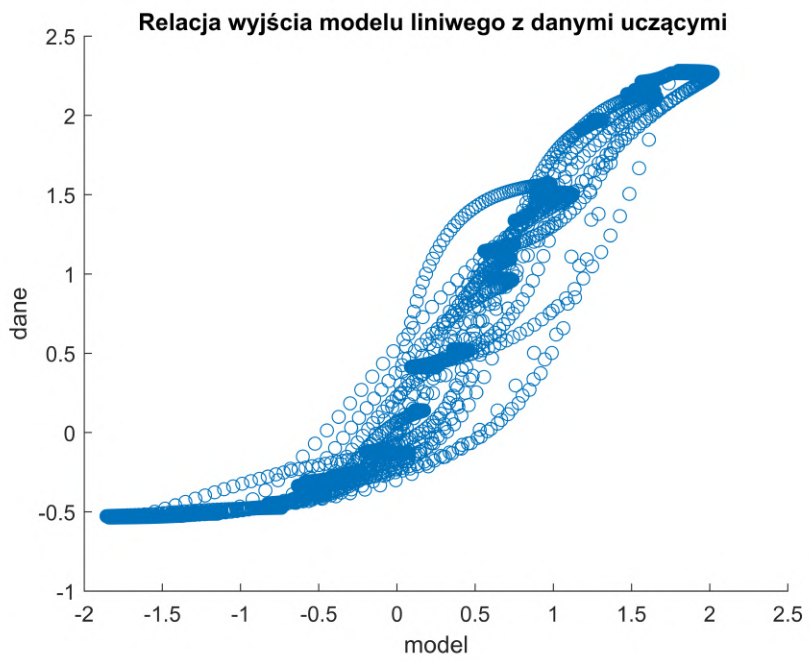
Wyznaczyliśmy model liniowy metodą najmniejszych kwadratów korzystając z operacji $W = M\backslash Y$. Wyniki symulacji modelu liniowego przedstawiamy poniżej:



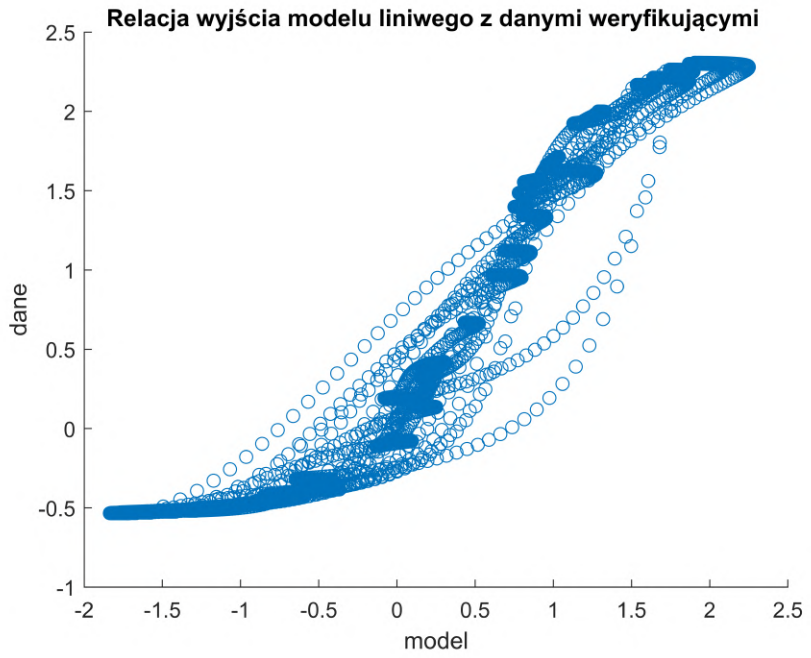
Rys. 2.12: Porównanie modelu z danymi uczącymi dla modelu liniowego.



Rys. 2.13: Porównanie modelu z danymi weryfikującymi dla modelu liniowego



Rys. 2.14: Relacja wyjścia modelu z danymi uczącymi dla modelu liniowego



Rys. 2.15: Relacja wyjścia modelu z danymi weryfikującymi dla modelu liniowego

Błędy modeli:

— błąd modelu na zbiorze uczącym = 1166,2769.
— błąd modelu na zbiorze weryfikującym = 6181,4619.

Model liniowy kiepsko dostosowuje się do danych uczących jak i weryfikujących. Obserwujemy duże rozbieżności wyjścia modelu w porównaniu do danych. Ponadto wykresy relacji wyjść

modelu z danymi weryfikującymi i uczącymi nie układają się w prostą o nachyleniu 45 stopni. Błędy są bardzo duże w porównaniu do modeli neuronowych.

3. Zadanie 3 - Modelowanie procesu za pomocą przyborników programu MATLAB

3.1. Podpunkt 1 - Wybranie przybornika programu MATLAB

Do modelowania procesu za pomocą sieci neuronowych wybraliśmy przybornik MATLAB Deep Learning Toolbox. Deep Learning Toolbox to zestaw narzędzi do projektowania, uczenia i implementacji sieci neuronowych i głębokich modeli uczenia maszynowego. Toolbox ten oferuje funkcje i narzędzia do tworzenia, trenowania różnych architektur sieci i wizualizacji wyników. Dzięki Deep Learning Toolbox można efektywnie eksplorować, rozwijać i wdrażać zaawansowane modele głębokiego uczenia się.

3.2. Podpunkt 2 - Uczenie kilku modeli neuronowych za pomocą wybranego przybornika

Przeprowadziliśmy uczenie kilku sieci neuronowych przy użyciu *DeepLearningToolBox*, struktura modeli była taka sama jak najlepszym modelu z zadania 2 - 8 neuronów w warstwie ukrytej. Modele były uczone w trybie ARX z wykorzystaniem algorytmu Levenberga-Marquardta oraz algorytmu gradientów sprzężonych Poljaka-Polaka-Ribiery.

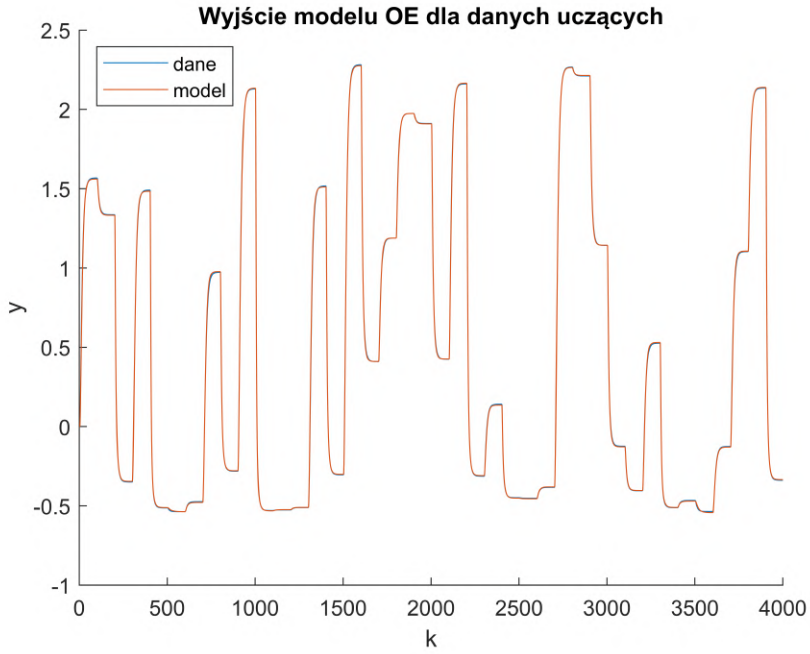
Tab. 3.1: Błędy modeli na zbiorze uczącym

alg. Uczenia	Alg Levenberga-Marquardta					Alg Poljaka-Polaka-Ribiery				
nr modelu	1	2	3	4	5	1	2	3	4	5
ARX	0,0255	0,0082	0,0062	0,0029	0,0053	0,1512	1,5783	0,2763	1,3491	1,2971
OE	1,5814	1,3592	0,8025	0,2057	0,5420	22,5900	54,9008	7,3064	833,9546	215,8179

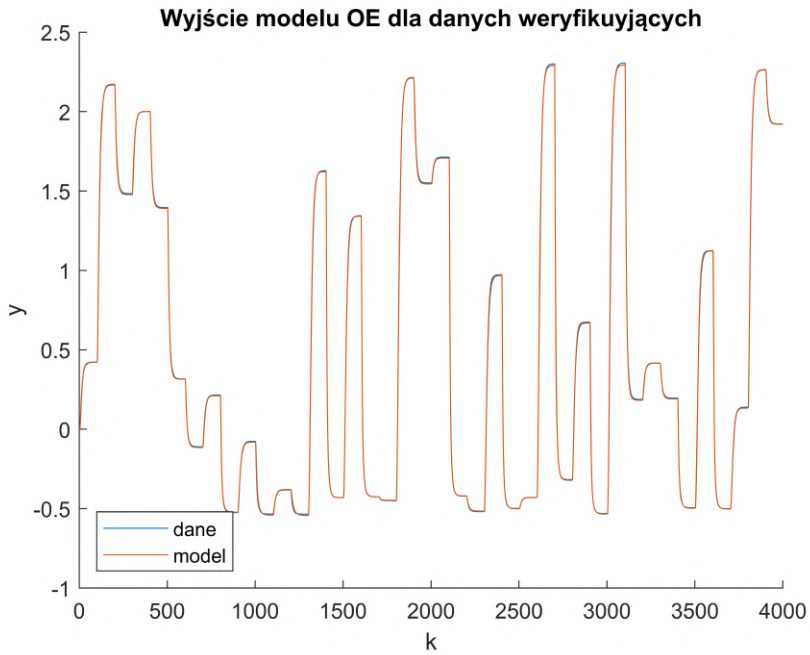
Tab. 3.2: Błędy modeli na zbiorze weryfikującym

alg. Uczenia	Alg Levenberga-Marquardta					Alg Poljaka-Polaka-Ribiery				
nr modelu	1	2	3	4	5	1	2	3	4	5
ARX	0,0224	0,0127	0,0090	0,0036	0,0095	0,1539	1,1715	0,4185	1,2249	2,7774
OE	2,3528	2,2852	0,7564	0,4102	1,7323	28,6707	51,6568	30,5286	147,9874	257,8369

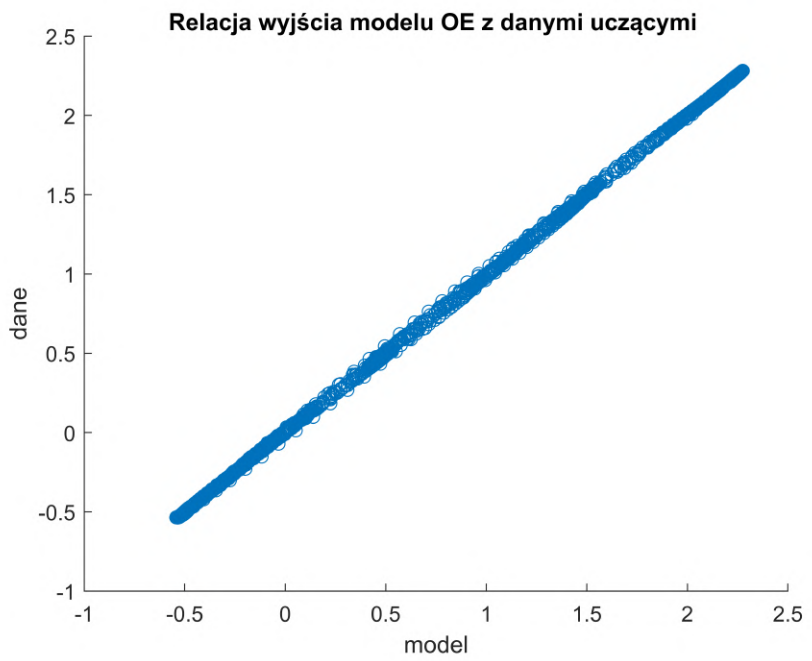
3.3. Podpunkt 3 - Symulacja wybranego modelu



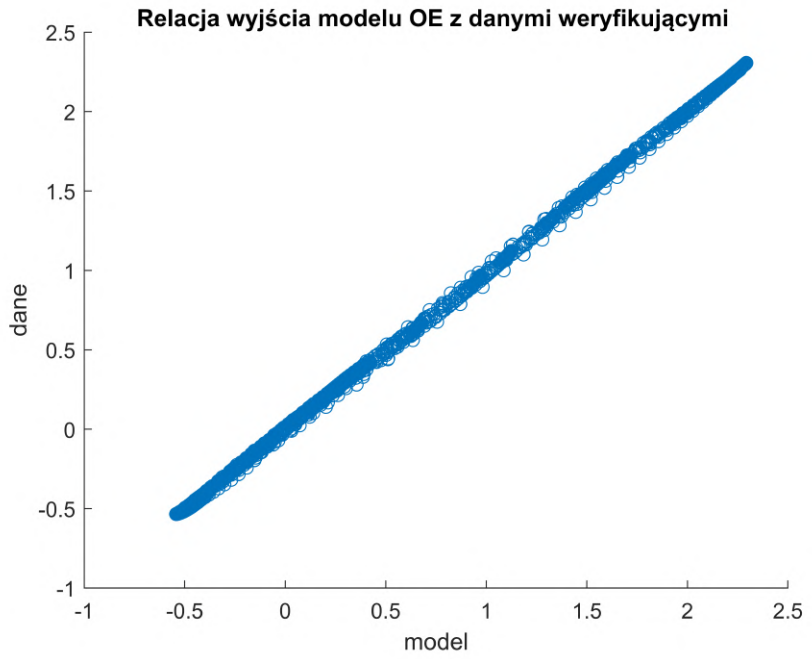
Rys. 3.1: Porównanie modelu symulowanego w trybie OE z danymi uczącymi dla najlepszego modelu.



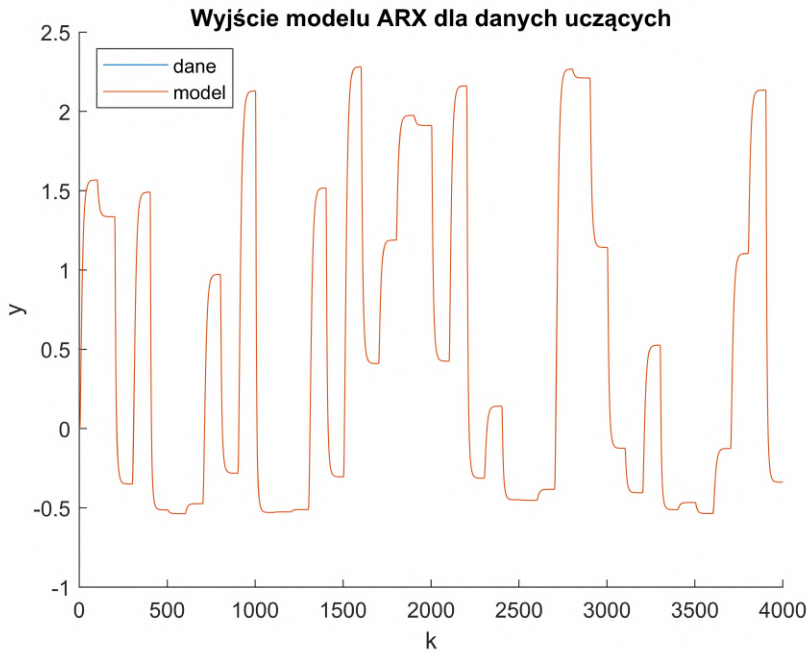
Rys. 3.2: Porównanie modelu symulowanego w trybie OE z danymi weryfikującymi dla najlepszego modelu.



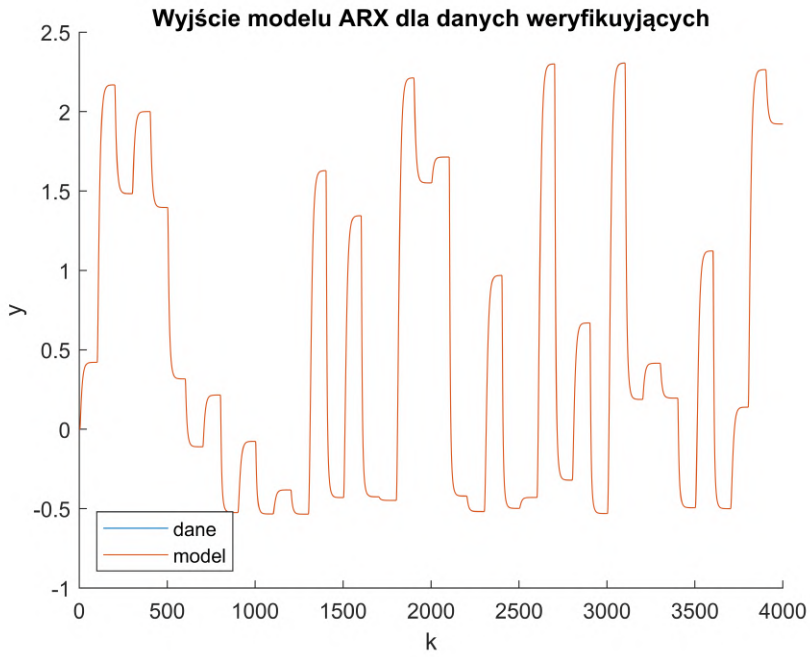
Rys. 3.3: Relacja wyjścia modelu z danymi uczącymi dla najlepszego modelu neuronowego symulowanego w trybie OE.



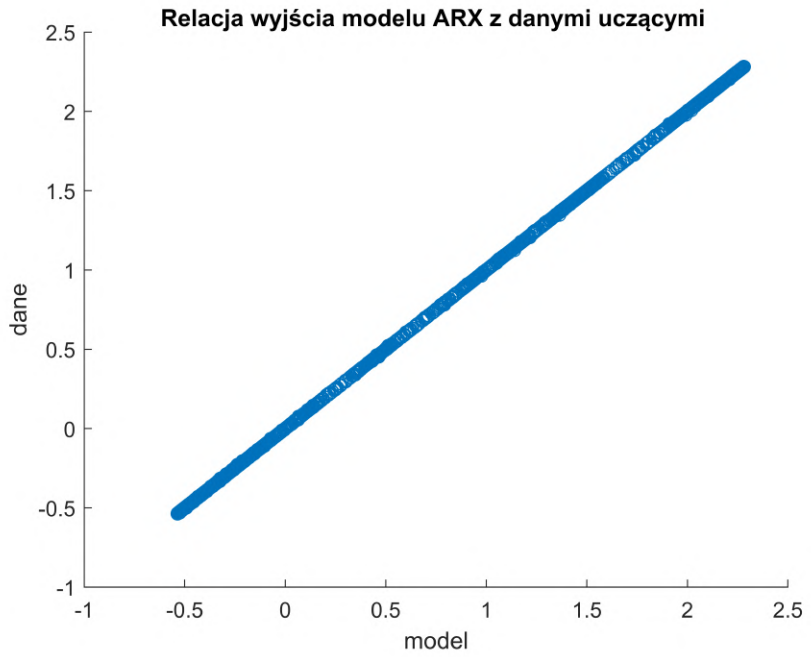
Rys. 3.4: Relacja wyjścia modelu z danymi weryfikującymi dla najlepszego modelu neuronowego symulowanego w trybie OE.



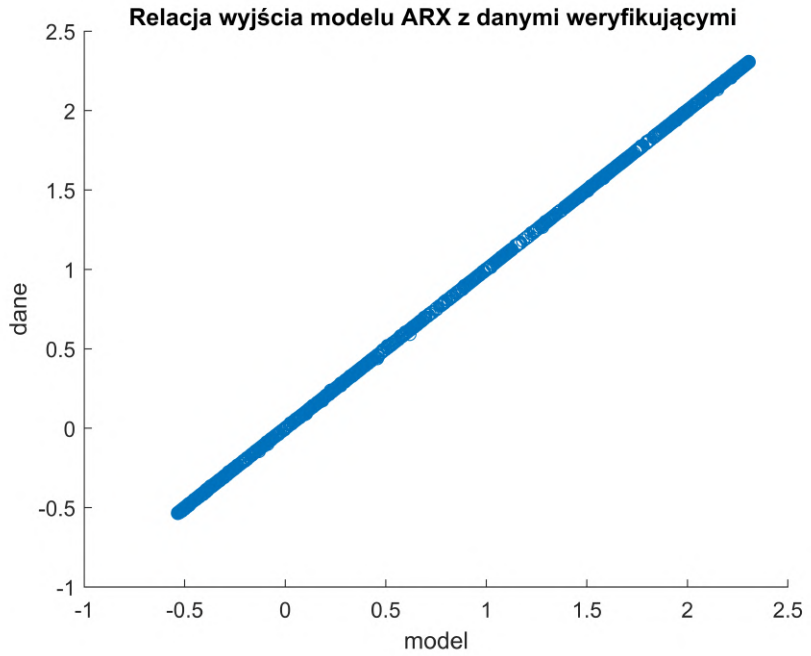
Rys. 3.5: Porównanie modelu symulowanego w trybie ARX z danymi uczącymi dla najlepszego modelu.



Rys. 3.6: Porównanie modelu symulowanego w trybie ARX z danymi weryfikującymi dla najlepszego modelu.



Rys. 3.7: Relacja wyjścia modelu z danymi uczącymi dla najlepszego modelu neuronowego symulowanego w trybie ARX.



Rys. 3.8: Relacja wyjścia modelu z danymi weryfikującymi dla najlepszego modelu neuronowego symulowanego w trybie ARX.

3.4. Podpunkt 4 - Porównanie jakości wybranego modelu

Modele uzyskane przy pomocy przybornika, uczone w trybie ARX porównujemy do modeli uzyskanych w zadaniu 2.6 także uczonych w trybie ARX.

Tab. 3.3: Porównanie modeli otrzymanych za pomocą programu sieci.exe i Deep Learning ToolBox

dane/model	sieci.exe	DL ToolBox
dane weryfikujące	0,1677	0,0036

Modele uzyskane przybornikiem Deep Learning Toolbox są pod względem błędów są lepsze od modeli uzyskanych programem sieci. Przebiegi na tle danych uczących i weryfikujących są niemalże nie do odróżnienia między modelami. Jedynie na wykresach relacji danych, w przypadku modelu z programu *sieci.exe*, obserwujemy nieznaczne "odchylenia" od prostej 45°,co nie występuje na wykresie relacji danych modelu z przybornika. Wartym odnotowania jest także nieco krótszy czas obliczeń przybornika.

4. Zadanie 4 - Regulacja procesu

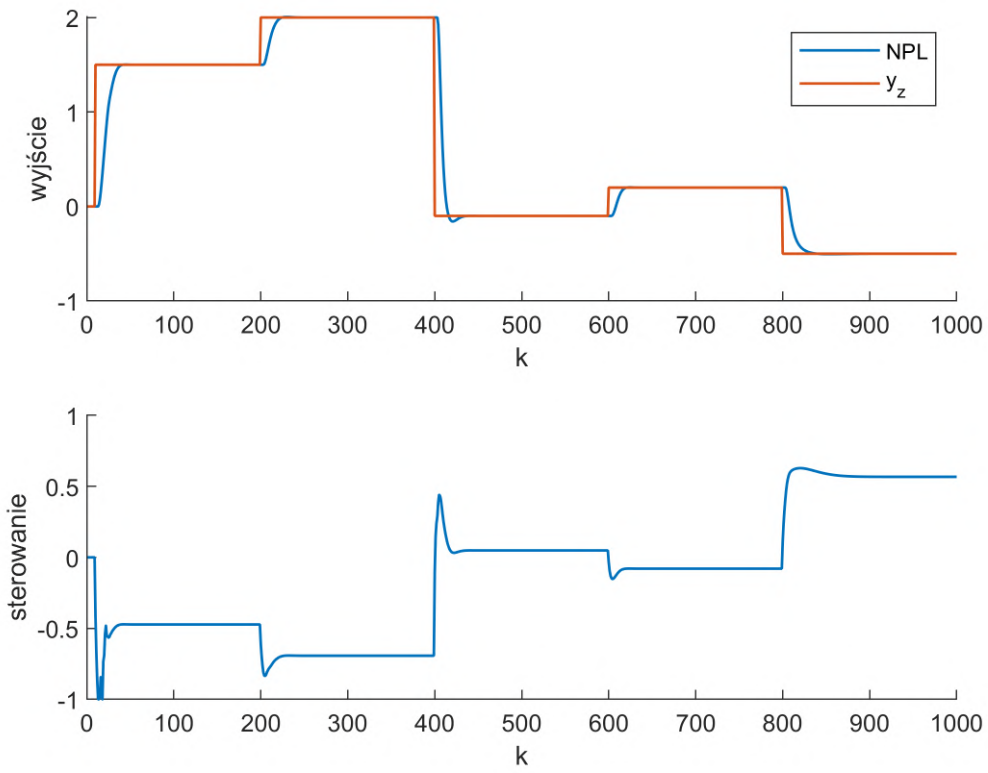
4.1. Podpunkt 1 - Implementacja NPL

Zaimplementowaliśmy algorytm regulacji predykcyjnej NPL w wersji analitycznej, bazujący na najlepszym modelu wyznaczonym w zadaniu 2 (w punkcie 2.3, jest to model o 8 neuronach uczony w trybie rekurencyjnym). Implementacja algorytmu przedstawiona jest w pliku $Zad_4.m$

4.2. Podpunkt 2 - Strojenie NPL

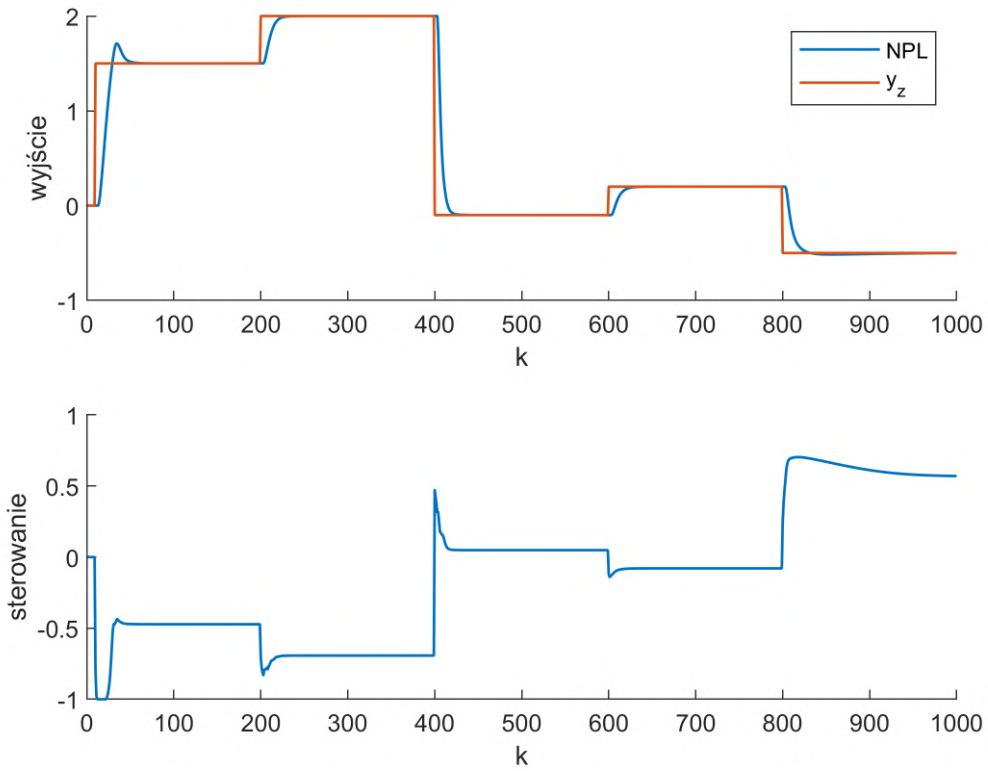
Przeprowadziliśmy strojenie regulatora NPL dobierając wartości parametrów N, N_u, λ. Trajektoria zadana została wygenerowana na podstawie charakterystyki statycznej 3.8 - skoki wartości zadanej są z przedziału [-0.5; 2.0].

Działanie regulatora NPL dla początkowych parametrów $N = 100$, $N_u = 50$, $\lambda = 10$ uzyskano błąd regulacji = 60,5235. Działanie regulatora:



Rys. 4.1: Przebiegi regulatora NPL dla początkowych parametrów $N = 100$, $N_u = 50$, $\lambda = 10$, błąd regulacji = 60,5235.

W wyniku strojenia poprawiliśmy jakość regulacji - dla parametrów $N = 20$, $N_u = 10$, $\lambda = 10$ otrzymaliśmy błąd regulacji równy 55,8270.

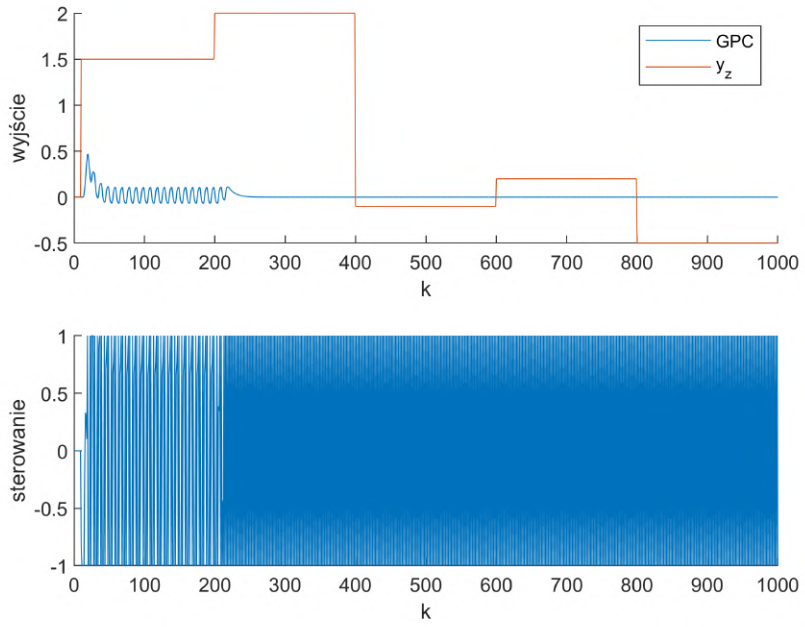


Rys. 4.2: Przebiegi nastrojonego regulatora NPL o parametrach$N = 20$, $N_u = 10$, $\lambda = 10$, błąd regulacji = 55,8270.

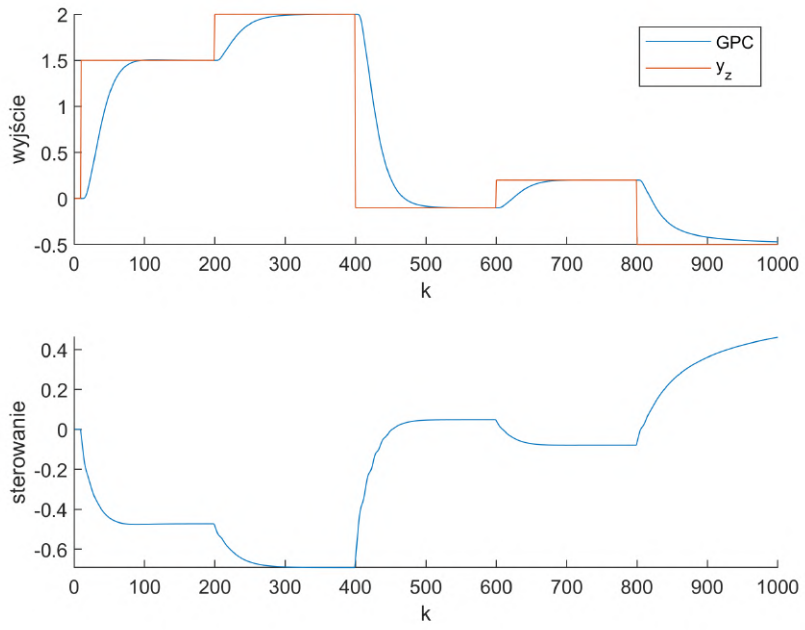
Z przedstawionych wykresów widzimy że regulator NPL dobrze radzi sobie z regulacją nieliniowego procesu.

4.3. Podpunkt 3 - Implementacja GPC

W tym samym pliku co algorytm regulacji NPL - *zad3.m* zaimplementowaliśmy algorytm regulacji predykcyjnej GPC w wersji analitycznej. Oba algorytmy regulacji korzystają z jednakowych parametrów - $N = 20$, $N_u = 1$, $\lambda = 10$. W przeciwieństwie do NPL, algorytm GPC korzysta z modelu liniowego, co może negatywnie wpływać na jakość regulacji w przypadku sterowania procesem nieliniowym.



Rys. 4.3: Przebiegi regulatora GPC, błąd regulacji = 1259,9916



Rys. 4.4: Przebiegi regulatora GPC z zwiększoną lambdą, błąd regulacji = 233,1571

Jak widzimy na przedstawionych wykresach regulator GPC działa niepoprawnie - nie jest w stanie osiągnąć wartości zadanej i niemalże od razu sterowanie oscyluje w maksymalnym przedziale wartości. Złe działanie wynika z używania przez GPC modelu liniowego podczas pracy na nieliniowym obiekcie. Znaczne zwiększenie parametru lambda (do wartości 1000) pozwoliły na osiąganie wartości zadanej.

4.4. Regulator PID (zadanie dodatkowe)

Regulator PID w wersji dyskretnej został zaimplementowany w tym samym pliku co poprzednie regulatory - *zadanie4.mat*.

Regulator został dostrojony metodą Zieglera - Nicholsa:

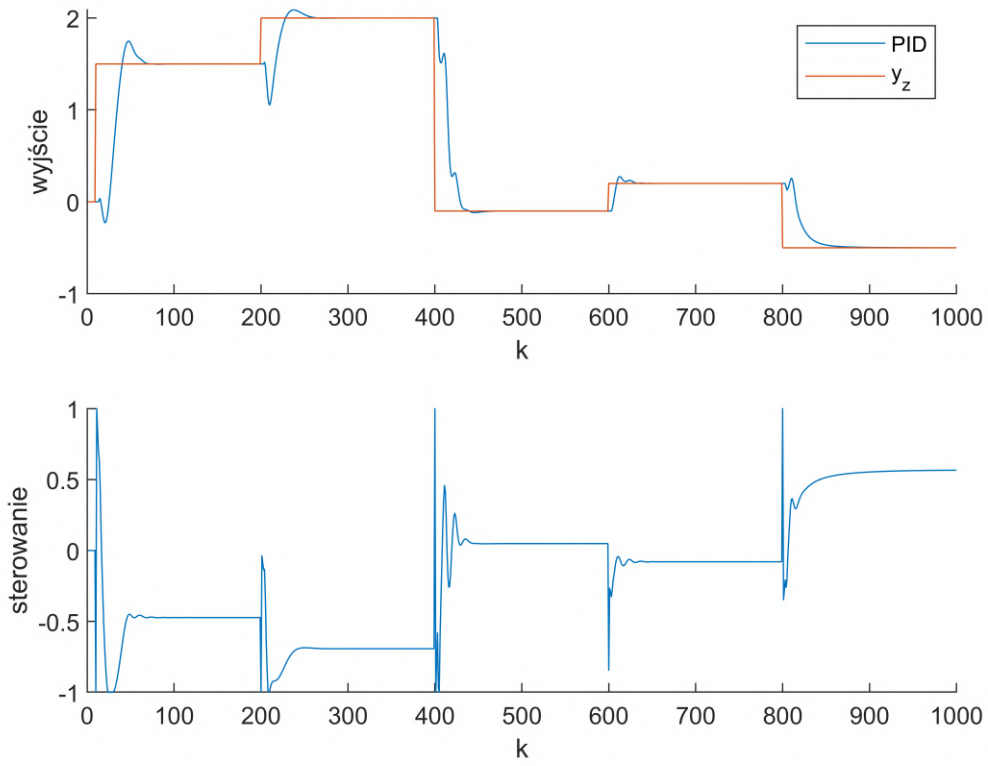
— Wzmocnienie krytyczne K_u = -2,5
— Okres oscylacji $T_u = 18$

Zatem nastawy PID wynoszą:

— $K = 0,6 * K_u = -1,5$
— $T_i = 0,5 * T_u = 9$
— $T_d = 0,125 * T_u = 2,25$

Ponieważ przebiegi wyjścia i sterowania regulatora PID nastrojonego metodą Z-N wpadały w oscylacje, regulator dostroiliśmy ręcznie. Finalne nastawy PID:

— $K = 0,6 * K_u = -0,9$
— $T_i = 0,5 * T_u = 9$
— $T_d = 0,125 * T_u = 2,25$

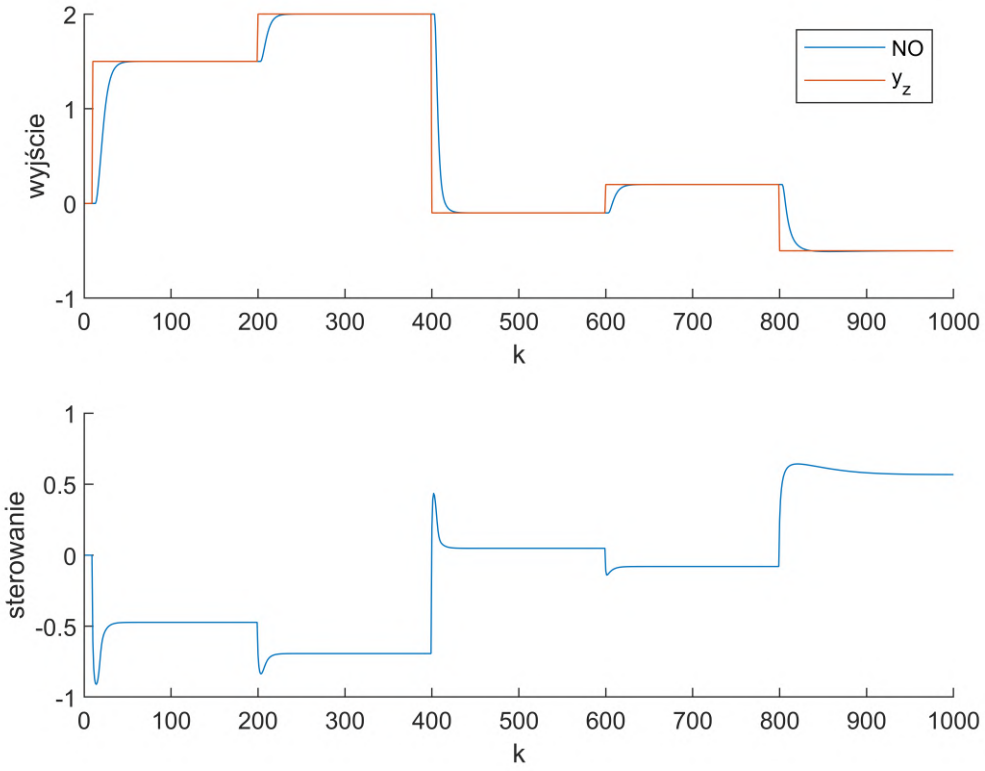


Rys. 4.5: Przebiegi regulatora PID, błąd regulacji = 119,14260.

Regulator PID, mimo że jest prostym, liniowym regulatorem, działa stosunkowo dobrze z analizowanym obiektem nieliniowym. Jest on z pewnością gorszy w porównaniu do regulatora NPL (błąd regulacji ponad 2 krotnie większy), ponadto przebiegi regulacji pozostawiają sporo do życzenia - obserwujemy spore przeregulowanie, w niektórych momentach występują oscylacje sygnału wyjściowego, a sygnał sterujący po każdym skoku wartości zadanej oscyluje.

4.5. Regulator NO (zadanie dodatkowe)

Na koniec zaimplementowaliśmy i przetestowaliśmy regulator z nieliniową optymalizacją (NO). Przyjęliśmy te same parametry jak w przypadku regulatora NPL i GPC - $N = 20, N_u = 1, \lambda = 10$. Trajektoria zadana jest również identyczna jak dla wszystkich wyżej testowanych regulatorach.



Rys. 4.6: Przebiegi regulatora NO, błąd regulacji = 57,0628

Regulator NO działa bardzo podobnie do regulatora NPL, cechuje się niskim błędem regulacji, szybkim osiąganiem wartości zadanej oraz brakiem przeregulowania. Jednak na niekorzyść regulatora NO przemawia o wiele dłuższy czas wykonywania obliczeń w porównaniu do NPL, a wyniki są bardzo zbliżone.