

SPRAWOZDANIE Z DRUGIEGO PROJEKTU
Z PRZEDMIOTU
„SZTUCZNA INTELIGENCJA W
AUTOMATYCE”

Numer zadania: 10

Wykonawcy:

Daniel Giełdowski

Piort Chachuła

Spis treści

| | |
|--|----|
| 1. Symulacja procesu | 2 |
| 1.1. Charakterystyka statyczna | 2 |
| 1.2. Zbiory danych | 3 |
| 2. Modelowanie procesu | 5 |
| 2.1. Opóźnienie | 5 |
| 2.2. Dobór liczby neuronów | 6 |
| 2.3. Model z algorytmu BFGS | 7 |
| 2.4. Symulacja modelu z algorytmu BFGS | 8 |
| 2.5. Model z algorytmu najszybszego spadku | 9 |
| 2.6. Model z algorytmu BFGS z uczeniem bez rekurencji | 10 |
| 2.7. Symulacja modelu z algorytmu BFGS z uczeniem bez rekurencji | 11 |
| 2.8. Model metodą najmniejszych kwadratów | 12 |
| 3. Regulacja procesu | 13 |
| 3.1. Implementacja NPL | 13 |
| 3.2. Strojenie NPL | 13 |
| 3.3. GPC | 19 |
| 4. Zadania dodatkowe | 21 |
| 4.1. PID | 21 |
| 4.2. NO | 22 |

1. Symulacja procesu

1.1. Charakterystyka statyczna

Zadany układ opisany jest równaniami:

$$\begin{cases} x_1(k) = -\alpha_1 x_1(k-1) + x_2(k-1) + \beta_1 g_1(u(k-3)) \\ x_2(k) = -\alpha_2 x_1(k-1) + \beta_2 g_1(u(k-3)) \\ y(k) = g_2(x_1(k)) \end{cases} \quad (1.1)$$

gdzie u -sygnał wejściowy, y -sygnał wyjściowy, x_1, x_2 - zmienne stanu, $\alpha_1 = -1,422574$, $\alpha_2 = 0,466776$, $\beta_1 = 0,017421$, $\beta_2 = 0,013521$ oraz

$$g_1(u(k-3)) = \frac{\exp(5u(k-3)) - 1}{\exp(5u(k-3)) + 1}, \quad g_2(x_1(k)) = 1 - \exp(-1.5x_1(k)) \quad (1.2)$$

Podany punkt pracy układu to $u = y = x_1 = x_2 = 0$, więc w wersji statycznej:

$$\begin{cases} x_1 = -\alpha_1 x_1 + x_2 + \beta_1 g_1(u) \\ x_2 = -\alpha_2 x_1 + \beta_2 g_1(u) \\ y = g_2(x_1) \end{cases} \quad (1.3)$$

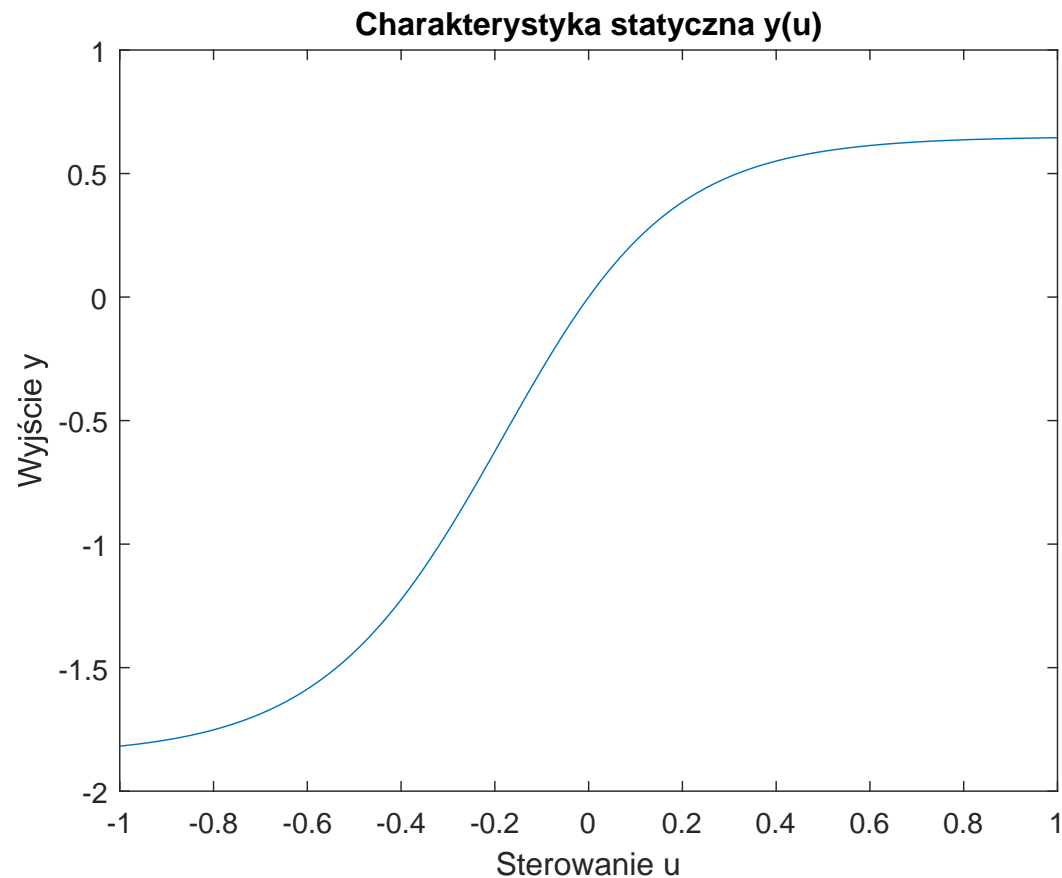
Po przekształceniach:

$$x_1 = \frac{(\beta_1 + \beta_2)g_1(u)}{1 + \alpha_1 + \alpha_2} \quad (1.4)$$

Po podstawieniu równania (1.4) do y otrzymujemy

$$y(u) = g_2\left(\frac{(\beta_1 + \beta_2)g_1(u)}{1 + \alpha_1 + \alpha_2}\right) \quad (1.5)$$

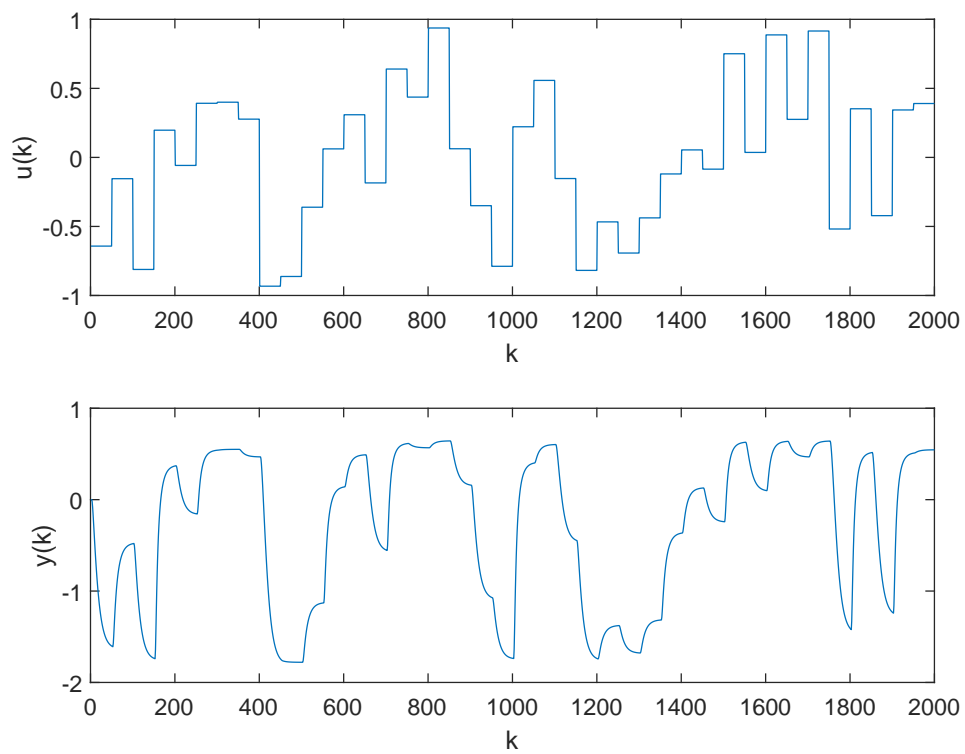
Wykres wyznaczonej charakterystyki statycznej dla zadanego zakresu wartości sterowania ($u^{\min} = -1, u^{\max} = 1$) przedstawiony został na wykresie 1.1. Wykres został wygenerowany za pomocą skryptu *charakterystyka_statyczna.m*.



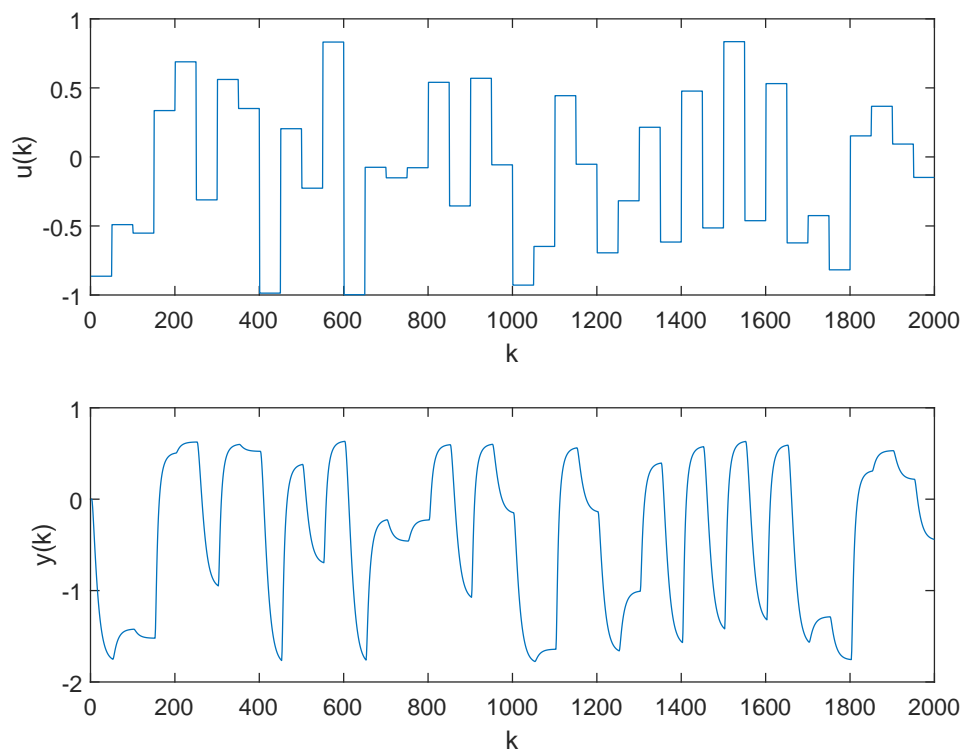
Rys. 1.1. Charakterystyka statyczna procesu

1.2. Zbiory danych

W celu przygotowania do uczenia sieci neuronowych wygenerowaliśmy dwa zbiory danych. Dane zostały wygenerowane poprzez zasymulowanie zadanego procesu dla sygnału sterowania złożonego o wartości zmieniającej się skokowo co 50 próbek. Obydwa zbiory danych mają po 2000 próbek. Zostały one przedstawione na wykresach 1.2 i 1.3. Użyte zostały skrypty: *generowanie_danych.m* (do wygenerowania danych) oraz *wykres_danych.m* (do narysowania wykresów).



Rys. 1.2. Dane uczące

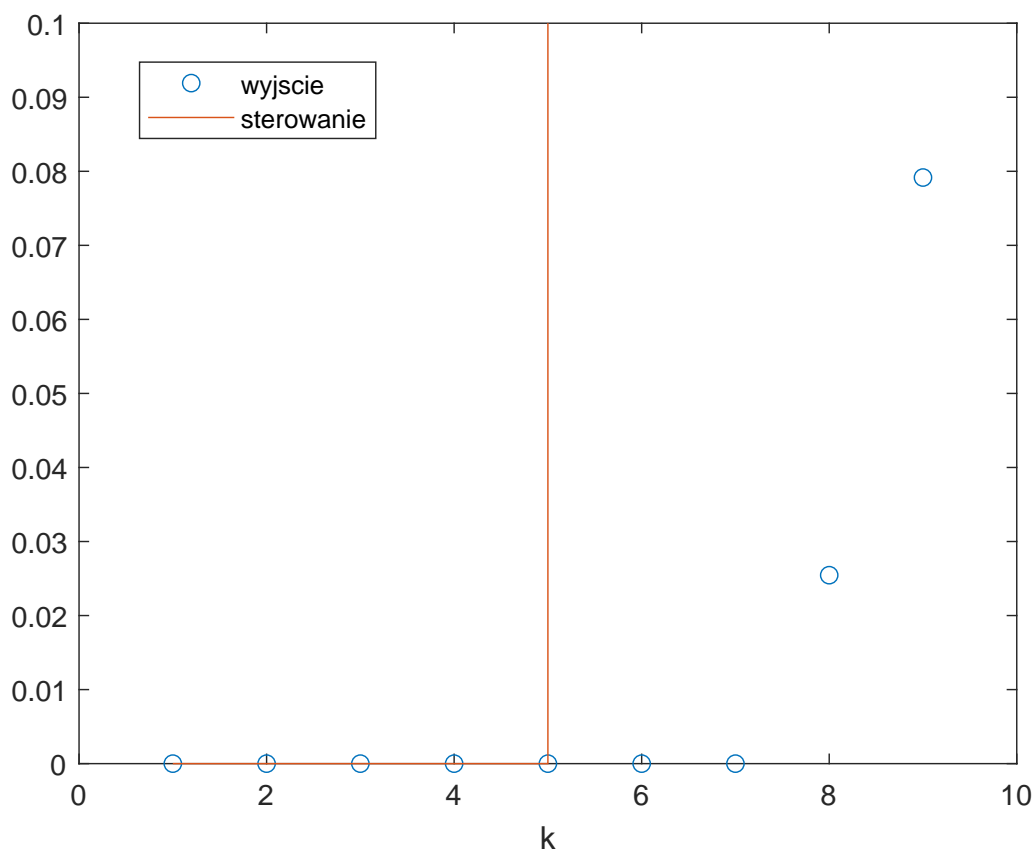


Rys. 1.3. Dane weryfikujące

2. Modelowanie procesu

2.1. Opóźnienie

W celu zdefiniowania opóźnienia τ procesu zasymulowaliśmy go dla pojedynczego skoku sterowania. Wyniki symulacji przedstawione są na wykresie 2.1. Skok sterowania nastąpił w 5 kroku działania programu, natomiast wyjście procesu zmieniło się dopiero w kroku 8. Oznacza to, że poszukiwane przez nas opóźnienie wynosi $\tau = 3$. Użyty przez nas skrypt to *tauwiz.m*.



Rys. 2.1. Wizualizacja opóźnienia procesu

2.2. Dobór liczby neuronów

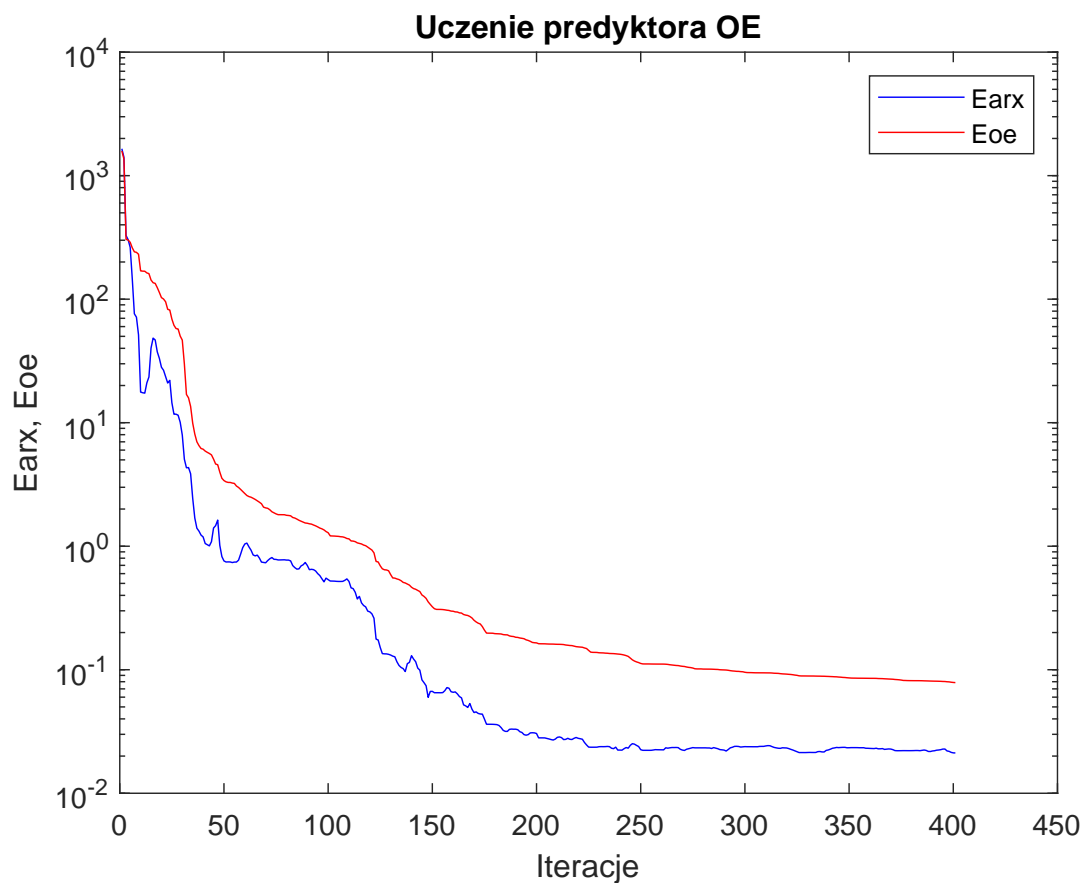
W celu dobrania odpowiedniej liczby neuronów dla sieci zastosowaliśmy wielokrotne uczenie z użyciem programu *sieci.exe*. Dla każdej ilości neuronów ukrytych od 1 do 10 dokonaliśmy 5 procesów uczenia za pomocą algorytmu BFGS z wykorzystaniem rekurencji. W tym celu wyko-rzystaliśmy skrypt *modelowanie.m*. Najmniejszy uzyskany błąd uczenia wraz ze skojarzonym z nim błędem weryfikacji przedstawiony został w tabeli poniżej. Najmniejszy błąd dla obydwu zbiorów występuje dla 9 neuronów. Ostatecznie jednak zdecydowaliśmy się na używanie sieci z pięcioma neuronami ukrytymi. Powodem tego jest mała poprawa w stosunku do większych ilości neuronów oraz chęć zmniejszenia nakładu obliczeń. Dodatkowo sieci o zbyt dużej ilości neuronów ukrytych mają tendencję do przetrenowywania się, w wyniku którego sieć przystosowuje się nie tyle do procesu co do samych danych uczących.

| Liczba neuronów | Błąd uczenia | Błąd weryfikacji |
|-----------------|--------------|------------------|
| 1 | 3.070626e+01 | 5.548815e+01 |
| 2 | 4.977413e-01 | 1.060318e+00 |
| 3 | 3.206039e-01 | 5.111444e-01 |
| 4 | 1.479096e-01 | 2.625729e-01 |
| 5 | 8.734595e-02 | 1.534512e-01 |
| 6 | 7.765994e-02 | 2.087909e-01 |
| 7 | 2.614618e-02 | 1.727668e-01 |
| 8 | 1.509561e-02 | 1.095385e-01 |
| 9 | 1.355132e-02 | 6.725641e-02 |
| 10 | 2.105601e-02 | 1.136051e-01 |

Tab. 2.1. Błędy modelu dla różnej ilości neuronów

2.3. Model z algorytmu BFGS

Na wykresie 2.2 przedstawione zostały błędy predyktorów ARX i OE dla kolejnych iteracji uczenia modelu. Zgodnie z ustaleniami z poprzednich punktów zastosowane zostały następujące parametry: $\tau = 3$, $\text{neurony ukryte} = 5$. Końcowe błędy dla obydwu predyktorów wynosiły odpowiednio: $E_{oe} = 0.0787$ i $E_{arx} = 0.0212$. Jak widać błędy te są dosyć małe jak na 2000 próbek co oznacza, że sieć dobrze nauczyła się modelu.



Rys. 2.2. Zmiany błędów predyktora ARX i OE dla kolejnych iteracji uczenia modelu algorytmem BFGS z użyciem rekurencji

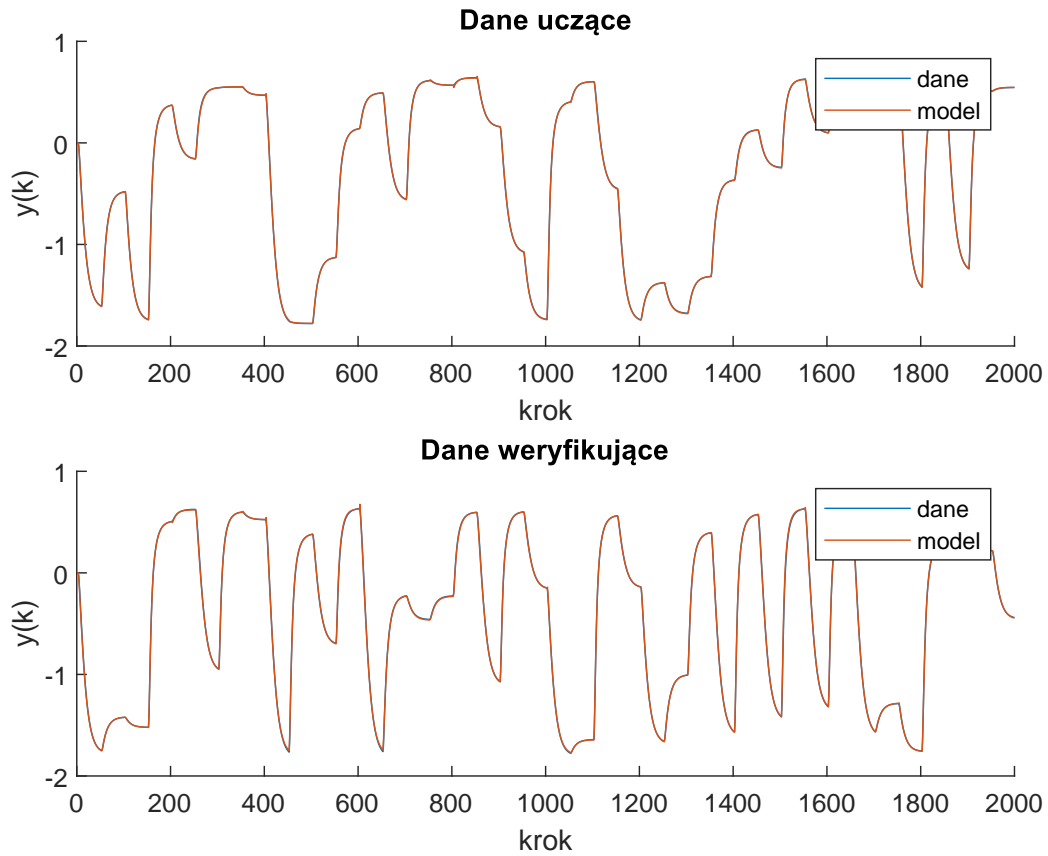
2.4. Symulacja modelu z algorytmu BFGS

Model z poprzedniego punktu został zasymulowany w trybie rekurencyjnym dla uczącego oraz weryfikującego zbioru danych. $Eucz = 0.0787$ $Ewer = 0.2172$ Błędy otrzymywane były ze wzoru

$$E = (y(S : end) - y^M(S : end))' * (y(S : end) - y^M(S : end)) \quad (2.1)$$

gdzie $S = \max(n_A, n_B) + 1$, oraz n_A i n_B są współczynniki modelu opisanego wzorem

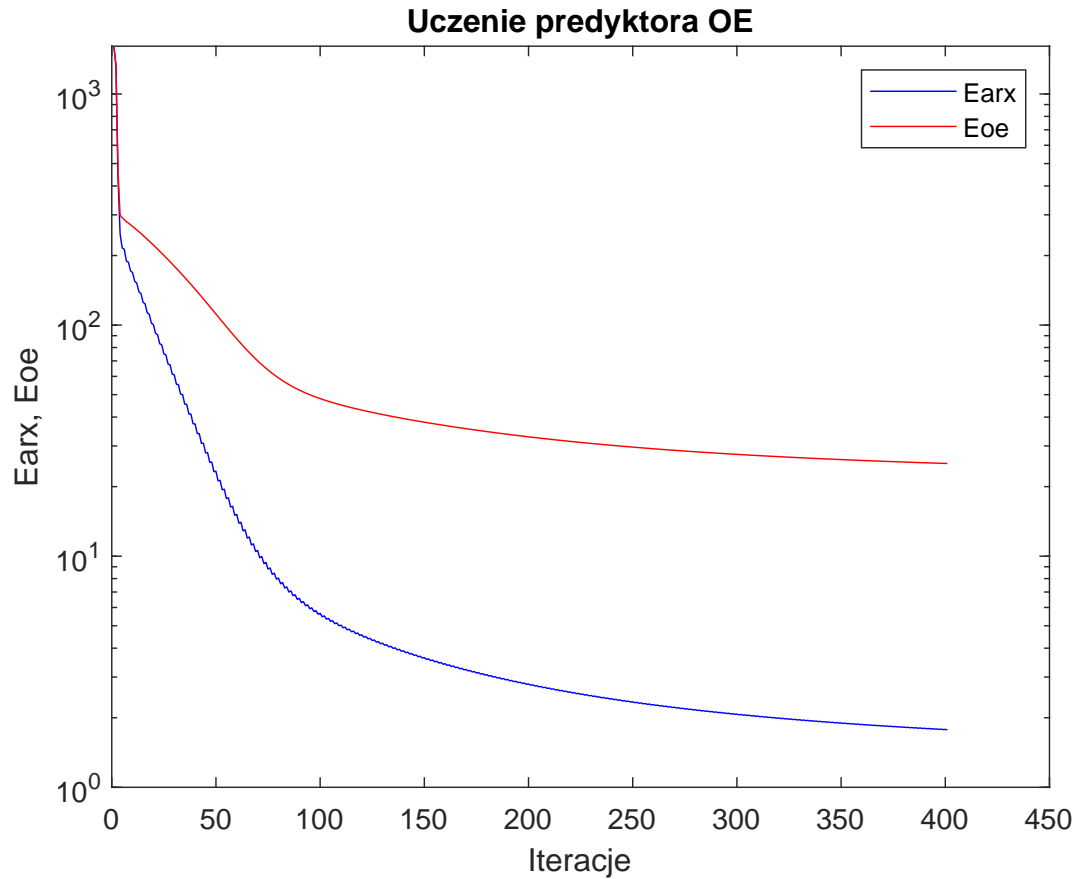
$$\hat{y}(k) = f(u(k - \tau), \dots, u(k - n_B), y(k - 1), \dots, y(k - n_A)) \quad (2.2)$$



Rys. 2.3. Symulacja modelu uczonego algorytmem BFGS z rekurencją na danych uczących i weryfikujących

2.5. Model z algorytmu najszybszego spadku

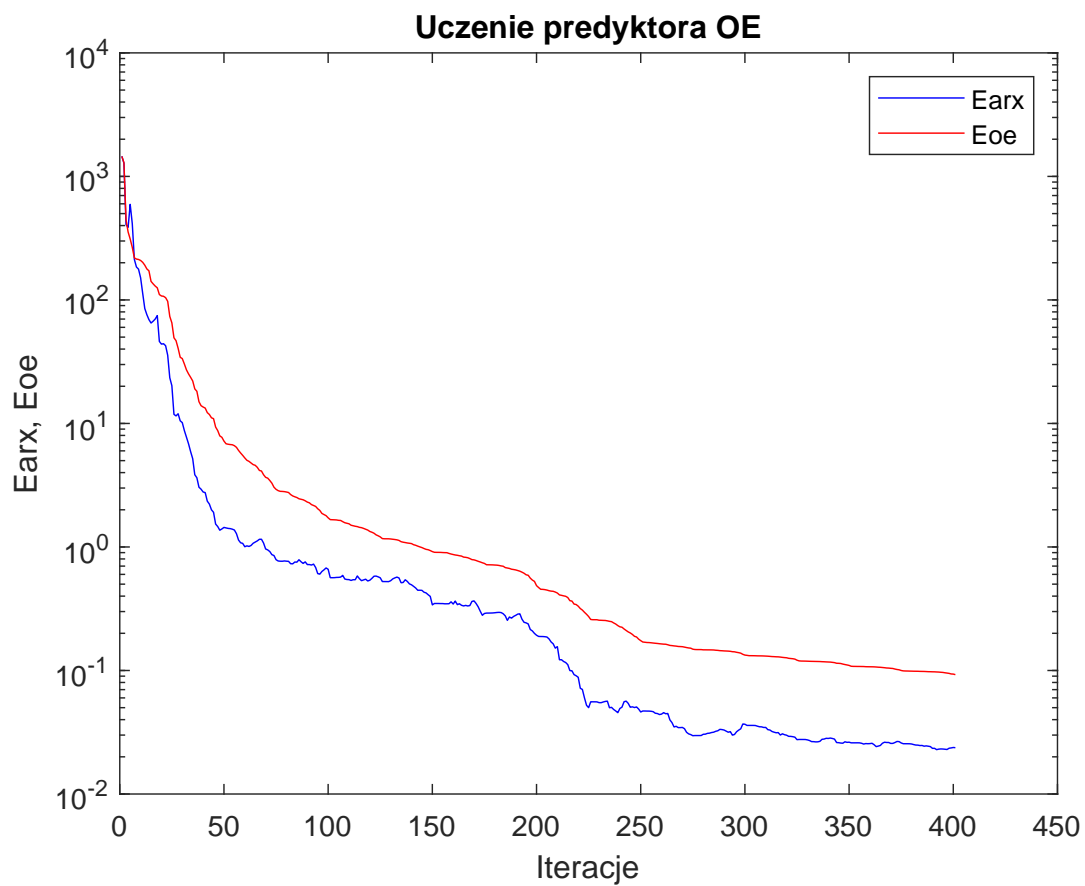
Jak widać po poniższych wartościach błędów uczenia oraz z wykresu 2.4 algorytm najszybszego spadku nie radzi sobie najlepiej co skutkuje dużą niedokładnością modelu. $E_{oe} = 25.1864$
 $E_{arx} = 1.7730$



Rys. 2.4. Zmiany błędów predyktora ARX i OE dla kolejnych iteracji uczenia modelu algorytmem najszybszego spadku z użyciem rekurencji

2.6. Model z algorytmu BFGS z uczeniem bez rekurencji

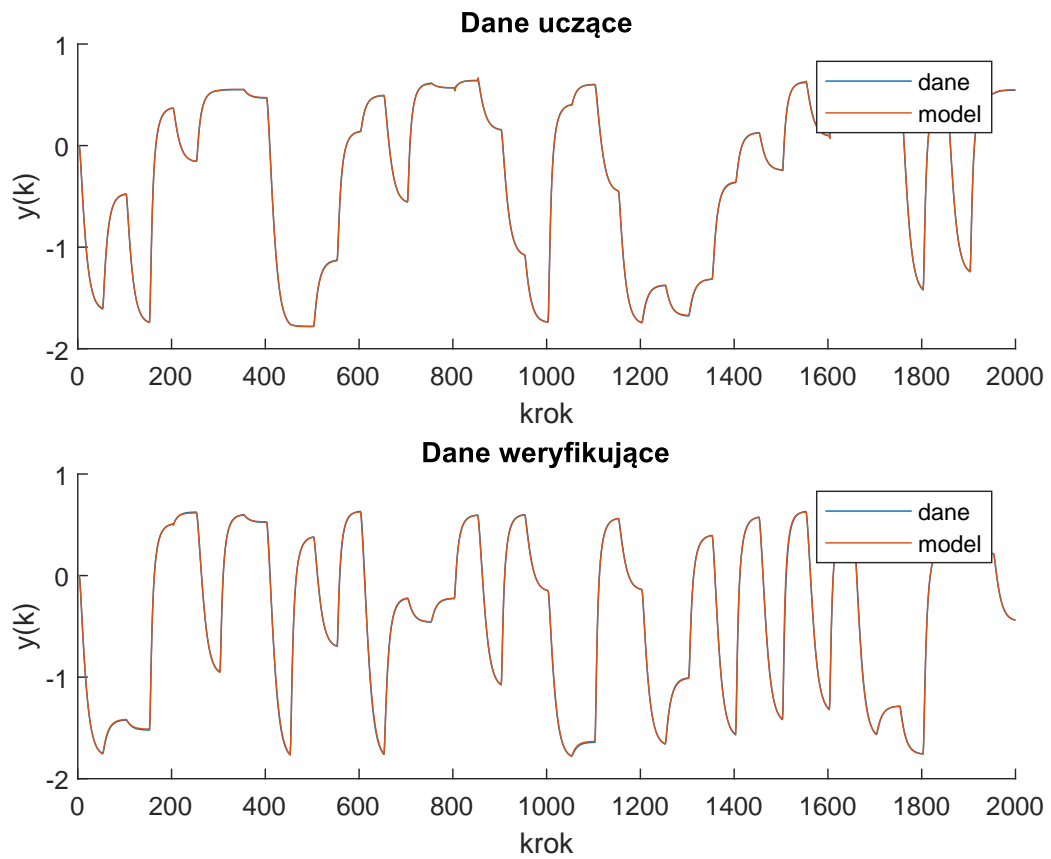
$$E_{oe} = 0.0925 \quad E_{arx} = 0.0237$$



Rys. 2.5. Zmiany błędów predyktora ARX i OE dla kolejnych iteracji uczenia modelu algorytmem BFGS bez użycia rekurencji

2.7. Symulacja modelu z algorytmu BFGS z uczeniem bez rekurencji

$E_{ucz} = 0.0925$ $E_{wer} = 0.3542$



Rys. 2.6. Symulacja modelu uczonego algorytmem BFGS bez rekurencji na danych uczących i weryfikujących

2.8. Model metodą najmniejszych kwadratów

Uczenie modelu metodą najmniejszych kwadratów realizowane jest następującym wzorem

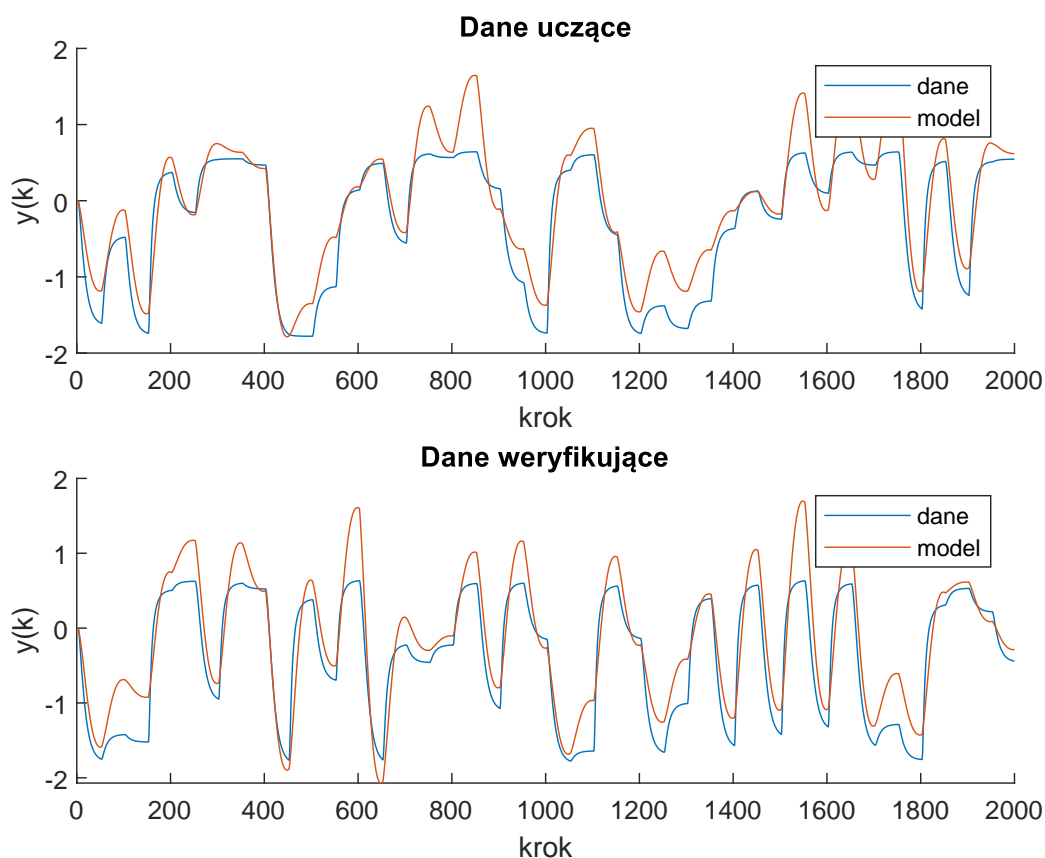
$$w = \begin{bmatrix} b_3 \\ b_4 \\ a_1 \\ a_2 \end{bmatrix} = M \backslash y_{ucz}(5 : end) \quad (2.3)$$

gdzie

$$M = \begin{bmatrix} x_{ucz}(2) & x_{ucz}(1) & y_{ucz}(4) & y_{ucz}(3) \\ \vdots & \vdots & \vdots & \vdots \\ x_{ucz}(n - \tau) & x_{ucz}(n - \tau - 1) & y_{ucz}(n - 1) & y_{ucz}(n - 2) \end{bmatrix} \quad (2.4)$$

Oraz n = liczba próbek. Otrzymane błędy były równe:

Ewer = 294.6949 Eucz = 299.7408 Jak widać jakość tego modelu jest bardzo zła i błędy są większe, niż w przypadku modelu uczonego algorytmem najszybszego spadku. Przebiegi przedstawiające działanie modelu nad danych uczących pokazane są poniżej.



Rys. 2.7. Symulacja modelu wykonanego za pomocą metody najmniejszych kwadratów

3. Regulacja procesu

3.1. Implementacja NPL

NPL jest algorytmem regulacji predykcyjnym z Nieliniową predykcją i z linearyzacją oznacza to że do wyznaczania trajektorii swobodnej (która zależy tylko od przeszłych sterowań) używamy nieliniowego modelu neuronowego:

$$y^0(k+1) = w_2 + w_2 * \tanh(w_1 + w_1 * x(k)) + dk \quad (3.1)$$

gdzie

$$dk = y(k) - y^M(k) \quad (3.2)$$

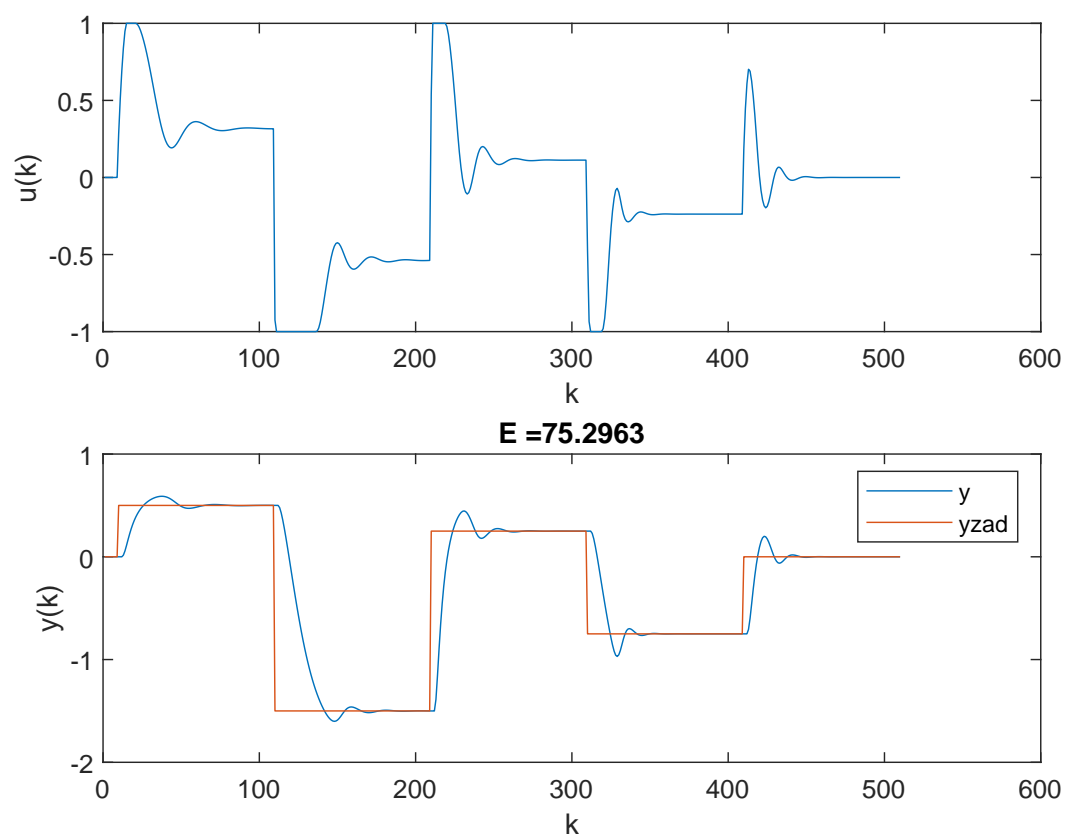
$$x(k) = \begin{bmatrix} u(\min(k - \tau + n, k - 1)) \\ u(\min(k - \tau - 1 + n, k - 1)) \\ y(k - 1 + n) \\ y(k - 2 + n) \end{bmatrix} \quad (3.3)$$

n = ilość chwil w przyszłość. Warto dodać, że we wzorze 3.3 dla chwil czasu dalszych od k zakłada się że $y(p > k) = y^0(p)$.

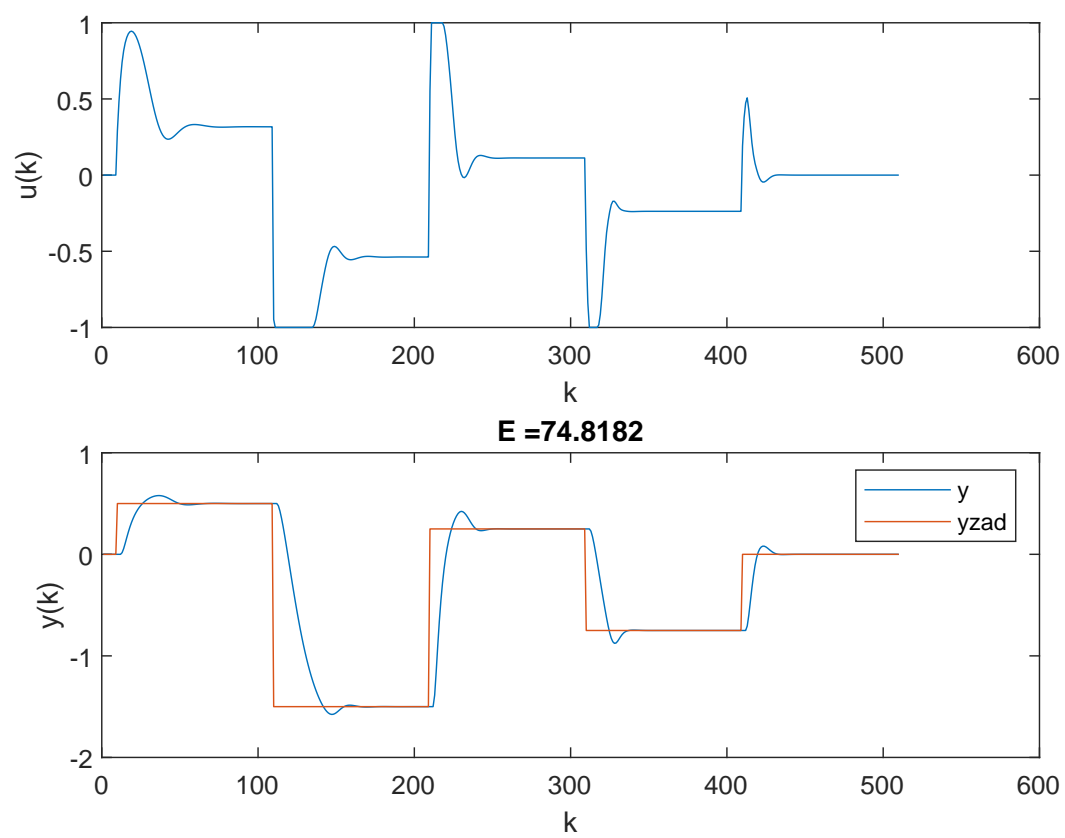
Aby móc rozwiązać algorytm analitycznie dokonuje się linearyzacji predykcji wyjścia modelu w przyszłości. Współczynniki b_3, b_4, a_1, a_2 we wzorze

$$y(k) = b_3 u(k - \tau = k - 3) + b_4 (k - \tau - 1 = k - 4) - a_1 y(k - 1) - a_2 y(k - 2) \quad (3.4)$$

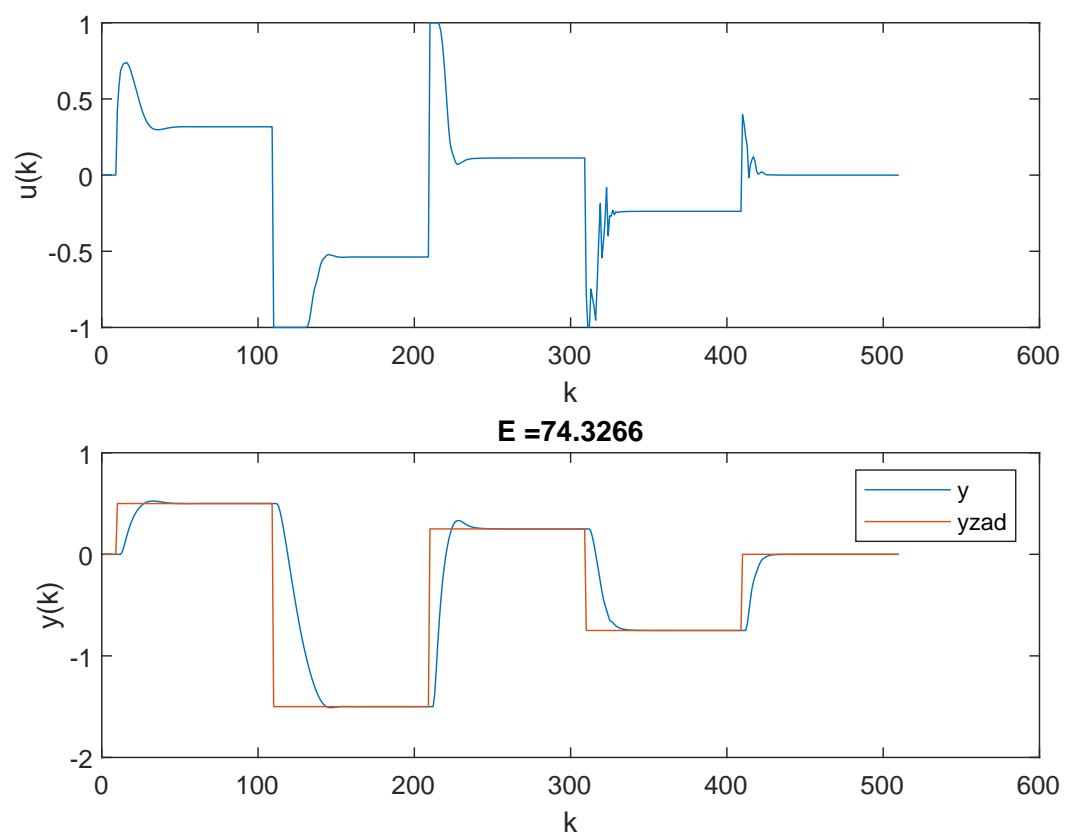
3.2. Strojenie NPL



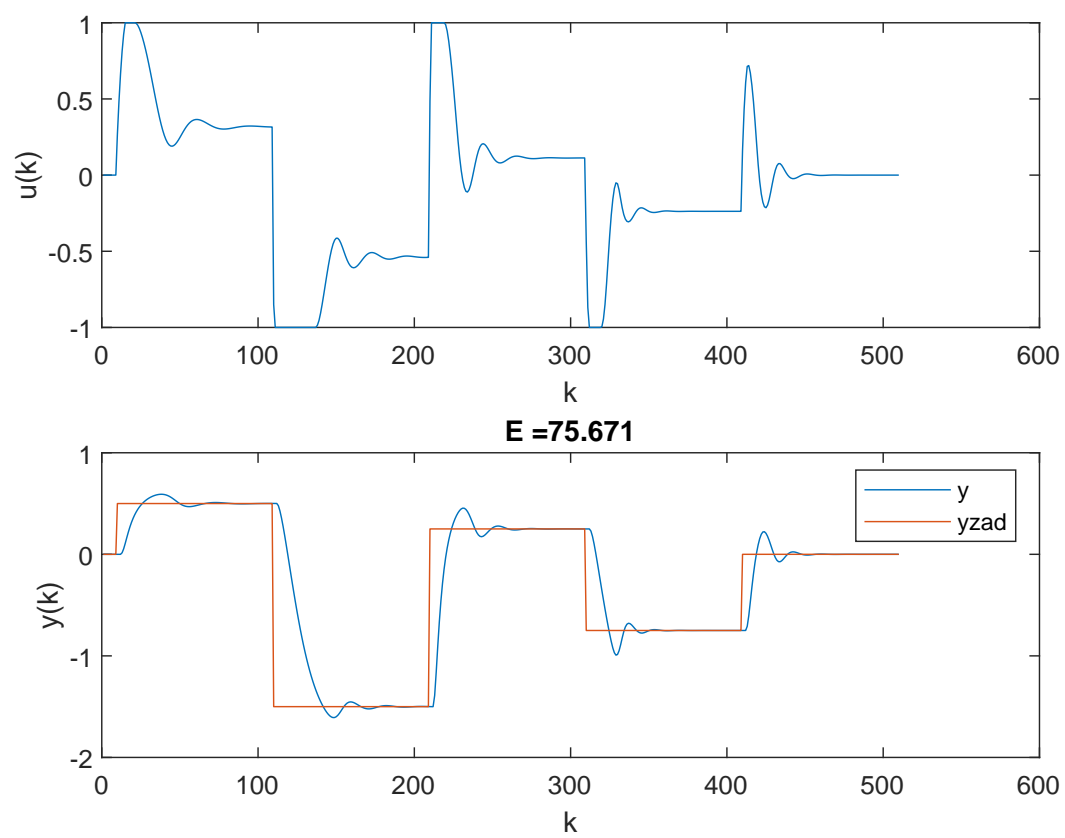
Rys. 3.1. Działanie regulatora NPL z nastawami $N=10$, $N_u=2$, $\lambda=1$



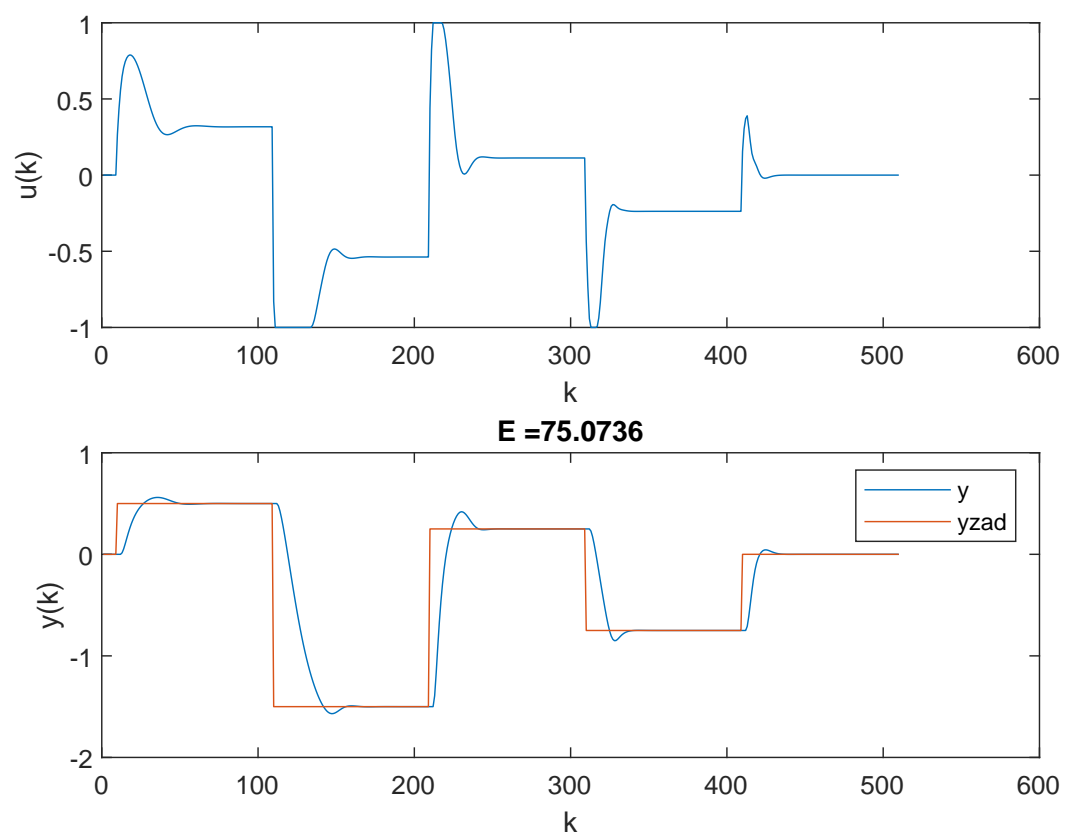
Rys. 3.2. Działanie regulatora NPL z nastawami $N=20$, $N_u=2$, $\lambda=1$



Rys. 3.3. Działanie regulatora NPL z nastawami $N=20$, $N_u=1$, $\lambda=1$

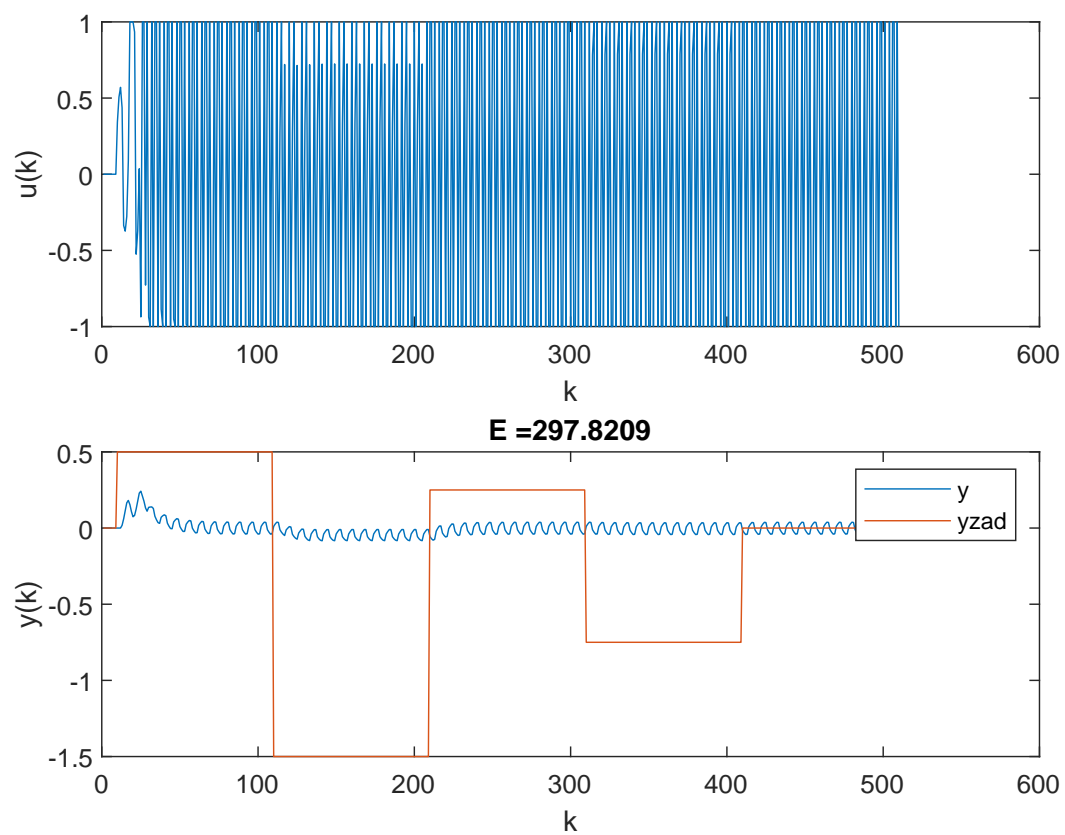


Rys. 3.4. Działanie regulatora NPL z nastawami $N=20$, $N_u=3$, $\lambda=1$

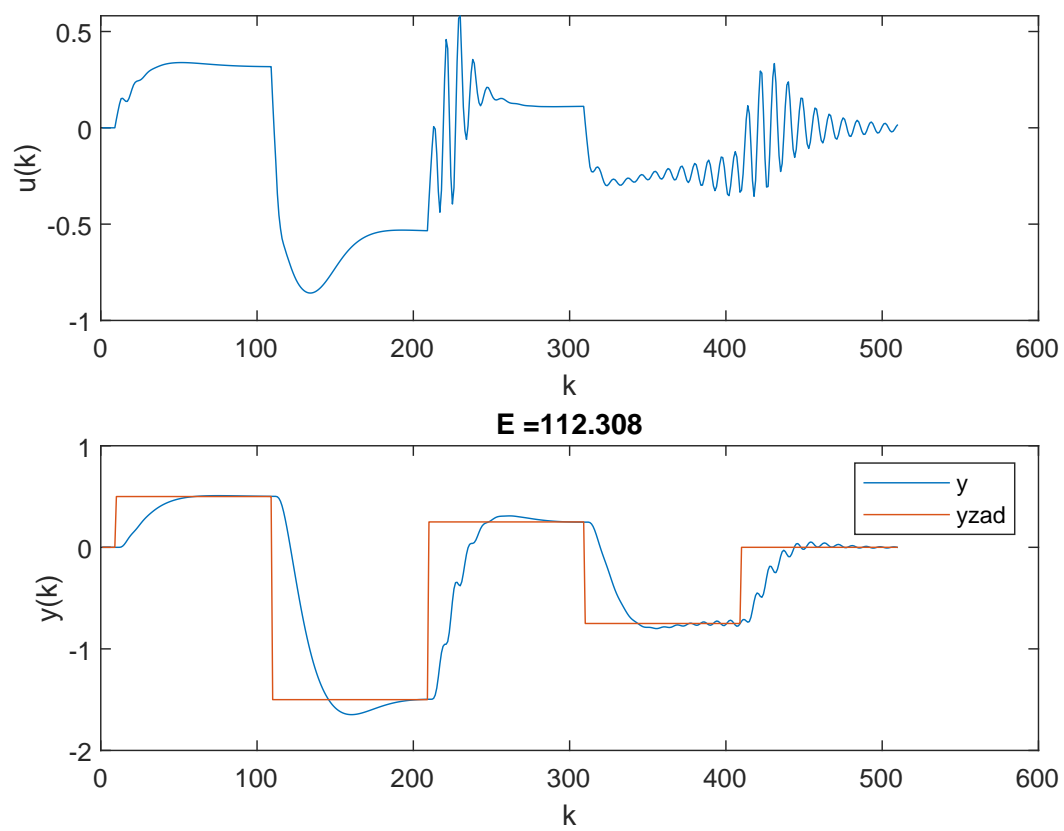


Rys. 3.5. Działanie regulatora NPL z nastawami $N=20$, $N_u=2$, $\lambda=2$

3.3. GPC



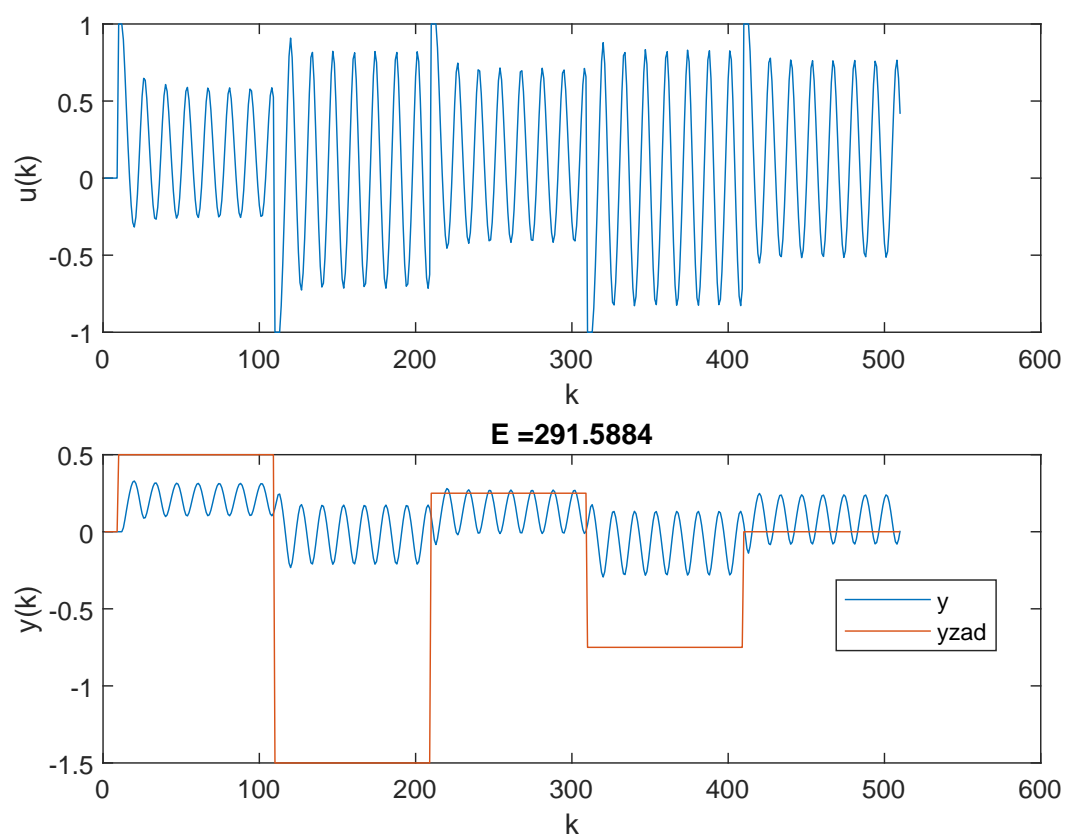
Rys. 3.6. Działanie regulatora GPC z nastawami $N=20$, $N_u=2$, $\lambda=2$



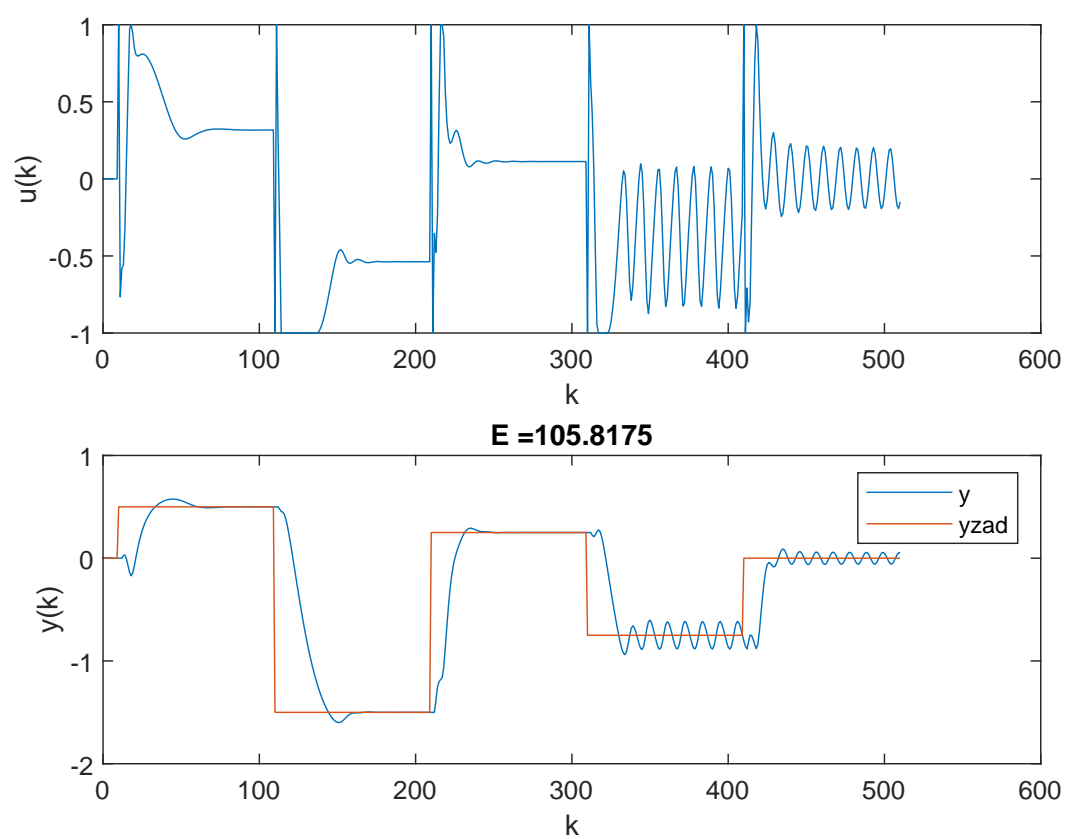
Rys. 3.7. Działanie regulatora GPC z nastawami $N=20$, $N_u=2$, $\lambda=100$

4. Zadania dodatkowe

4.1. PID

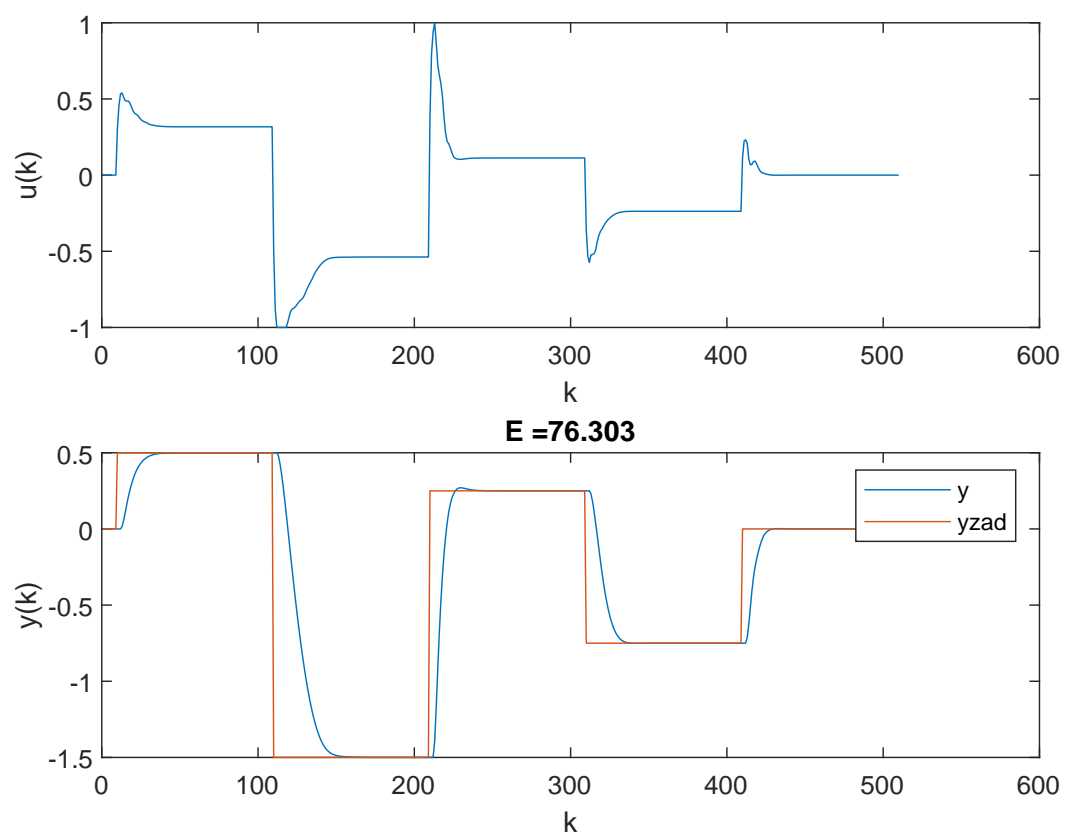


Rys. 4.1. Działanie regulatora PID z nastawami $K_p = 4$, $T_i = \text{Inf}$, $T_d = 0$



Rys. 4.2. Działanie regulatora PID z nastawami $K_p = 2.4$, $T_i = 6.5$, $T_d = 1.625$

4.2. NO



Rys. 4.3. Działanie regulatora NO z nastawami $N=20$, $N_u=2$, $\lambda=2$