Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Sterowanie Procesami

Sprawozdanie z projektu II, zadania 58

Adam Sokołowski

Warszawa, 2024

Spis treści

1. Transmitancje

Obiekt regualcji opisany jest poniższą transmitancją:

$$G(s) = \frac{K_o e^{-T_o s}}{(T_1 s + 1)(T_2 s + 1)} \tag{1.1}$$

gdzie $K_o = 4,7, T_o = 5, T_1 = 1,83, T_2 = 5,45$.

1.1. Wyznaczanie transmitancji dyskretnej

Do wyznaczenia transmitancji dyskretnej skorzystano z polecenia c2d w MATLAB w następujący sposób:

```
Gz = c2d(Gs, Tp, 'zoh');
```

Otrzymana transmuitancja wygląda następująco:

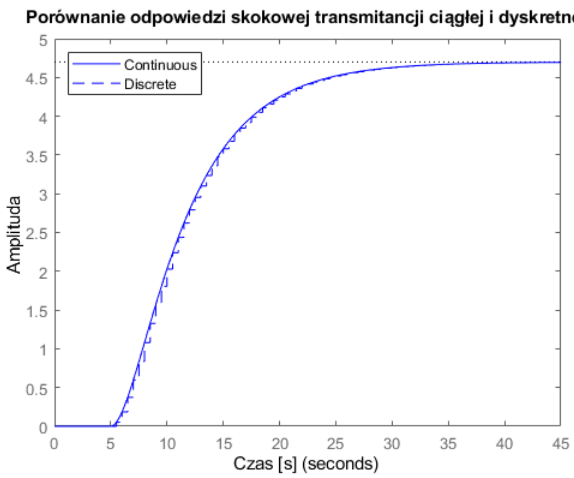
$$G(z) = z^{-10} \frac{0.05224z + 0.04626}{z^2 - 1.673z + 0.6942} \tag{1.2}$$

Jako argumenty funkcji podano kolejno transmitancję G(s), Tp, czyli okres próbkowania równy 0,5s oraz 'zoh', który określa, że używany ektrapolatora zerowego rzędu.

Transmitancję G(s) uzyskano w MATLAB w następujący sposób:

```
Gs = tf(Ko,conv([T1 1], [T2 1]),'inputdelay',To);
```

Następnie porównano odpowiedzi skokowe obu transmitancji i współczynniki wzmocienia statycznego.



Rys. 1.1. Odpowiedzi skokowe

Odpowiedzi skokowe się pokrywają, można zaobserwować, że odpowiedź skokowa modelu dyskretnego jest "schodkowa", i wynika to z tego, że liczymy jej kolejne wartości tylko w kolejnych okresach prógkowania czyli co kolejne 0,5s.
Wzmocnienia statyczne dla obu transmitancji były takie same i wyniosły 4,7. Tego wyniku można się było spodziewać gdyż jest to ten sam obiekt tylko przedstawiony w innej postaci. Ponadto na powyższym rysunku odpowiedzi skokowe stabilizowały się dla tej samej wartości.
Do otrzymania odpowiedzi skokowych skorzystano z funkcji step.

```
step(Gs, '-', Gz, '--');
legend('Continuous', 'Discrete');
title('Porównanie odpowiedzi skokowych');
xlabel('Czas [s]');
ylabel('Amplituda');
```

Do otrzymania wzmocnienia statycznego dcgain.

```
K_stat_continuous = dcgain(Gs);
K_stat_discrete = dcgain(Gz);

disp(['Kstat transmitancji ciągłej: ', num2str(K_stat_continuous)]);
disp(['Kstat transmitancji dyskretnej: ', num2str(K_stat_discrete)]);
```

1.2. Równanie różnicowe

Na podstawie transmitancji dyskretnej wyznaczono również równanie różnicowe służące do obliczenia wielkości y(k) na podstawie sygnałów wejściowych i wyjściowych z chwil poprzednich. Sposób wyznaczenia:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.05224z^{-11} + 0.04626z^{-12}}{1 - 1.673z^{-1} + 0.6942z^{-2}} \tag{1.3}$$

$$Y(z)(1 - 1.673z^{-1} + 0.6942z^{-2}) = U(z)(0.05224z^{-11} + 0.04626z^{-12}) \tag{1.4}$$

$$y(k) - 1.673y(k-1) + 0.6942y(k-2) = 0.05224u(k-11) + 0.04626u(k-12) \tag{1.5}$$

$$y(k) = 1.673y(k-1) - 0.6942y(k-2) + 0.05224u(k-11) + 0.04626u(k-12) \tag{1.6}$$

```
syms k u(k) y(k)
y(k) = -Gz.Denominator{1}(2)*y(k-1) - Gz.Denominator{1}(3)*y(k-2) ...
    + Gz.Numerator{1}(2)*u(k-11) + Gz.Numerator{1}(3)*u(k-12);
```

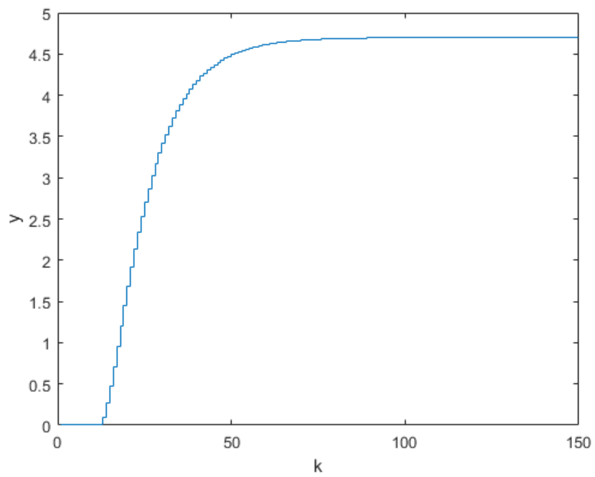
Do jego wyznaczenia w MATLAB użyto zmiennych symbolicznych oraz skorzystano z wcześniej wyznaczonej transmitancji G(z).

Zapis MATLAB'owy odpowiada następującemu równaniu

$$y(k) = -a_1y(k-1) - a_0y(k-2) + b_1u(k-11) + b_2u(k-12) \tag{1.7}$$

gdzie $a_1 = 1.673$, $a_0 = 0.6942$, $b_1 = 0.05224$, $b_2 = 0.04626$.

Poprawność równania różnicowego sprawdzono porównując kolejne wyjścia obliczonego równania z odpowiedzią skokową transmitancji dyskretnej



Rys. 1.2. Kolejne wyjścia obliczone z równania różnicowego

Równanie wyjścia obiektu zostało poprawnie wyznaczone, ponieważ odpowiedzi skokowe wyznaczone z transmitancji są bardzo zbliżone do kolejnych wyjść wyznaczonych za pomocą równania.

2. Regulator PID

Regulator PID składa się z trzech części: proporcjonalnej (P), całkującej (I) i różniczkującej (D). PID reguluje wyjście systemu, aby minimalizować błąd w jak najkrótszym czasie i z minimalnymi oscylacjami.
Wzory na regulator w czasie ciągłym i dyskretnym

$$u(t) = K\left[1 + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right]e(t) \tag{2.1}$$

i dyskretnym

$$u(k) = K\left[1 + \frac{T}{2T_i}\left(\frac{1+z^{-1}}{1-z^{-1}}\right) + \frac{T_d}{T}(1-z^{-1})\right]e(k) \tag{2.2}$$

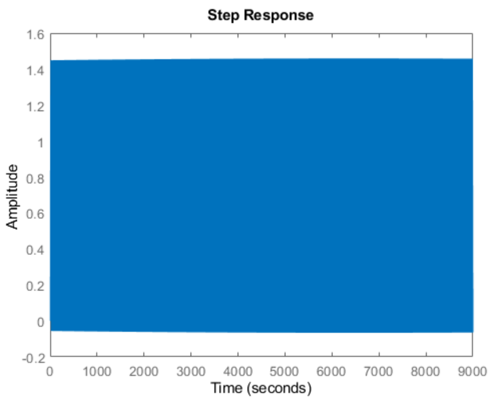
2.1. Dobór ciągłego regulatora PID metodą Zieglera-Nicholsa

Metoda Zieglera-Nicholsa polega na znalezieniu takiego K_{kryt} dla którego w odpowiedzi skokowej obiektu z regulatorem uzyskamy oscylacje niegasnące i nierosnące. Stosujemy regulator P, gdzie jako K podajemy K_{kryt}, a pozostałe człony (I i D) zerujemy $T_i = inf$, $T_d = 0$, . Następnie odczytujemy T_{kryt}, które jest okresem oscylacji. Po otrzymamaniu K_{kryt} i T_{kryt} wyznaczamy parametry ciagłego regulatora PID ($K_r = 0,6K_{kryt}$, $T_i = 0,5T_{kryt}$, $T_d = 0,12T_{kryt}$).

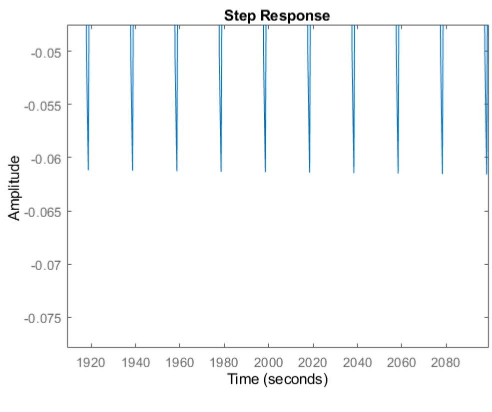
Implementacja w MATLAB'ie wygląda następująco

```
Kkryt = 0.48812; Ti = inf; Td = 0;
s = tf('s');
sys = (Ko*exp(-To*s))/((T1*s+1)*(T2*s+1));
C0 = pidstd(Kkryt, Ti, Td);
C1 = feedback(C0*sys, 1);
step(C1)
```

Odpowiedź skokowa dla $K_{kryt} = 0.48812$ (Rys. 2.1).



Rys. 2.1. Oscylacje niegasnące i nierosnące



Rys. 2.2. Przybliżenie, aby otrzymać T_{kryt}

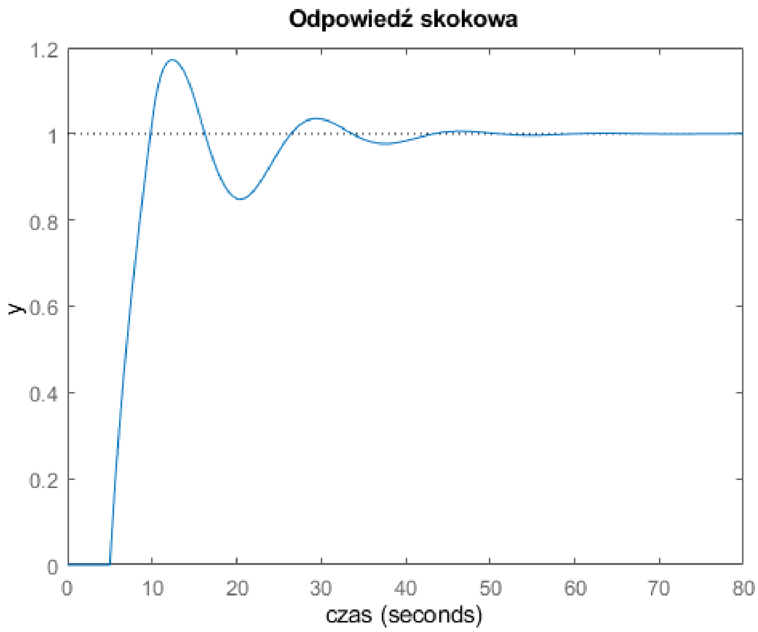
Widać, że dobrane K_{kryt} jest odpowiednie i dla niego $T_{kryt} = 20s$ (z Rys. 2.2).

2.1.1. Implementacja i działanie regulatora PID

```
Kr = 0.6 * Kkryt; Ti = 0.5 * Tkryt; Td = 0.12 * Tkryt;
s = tf('s');
sys = (Ko*exp(-To*s))/((T1*s+1)*(T2*s+1));
C0 = pidstd(Kr, Ti, Td);
C1 = feedback(C0*sys, 1);
step(C1, 80)
```

Do obliczenia parametrów skorzystano z wcześniej wyznaczonych K_{kryt} i T_{kryt} oraz wzorów dostarczonych przez prowadzącego.

Odpowiedź skokowa dla regulatora PID z parametrami wyznaczonymi metodą Zieglera-Nicholsa.



Rys. 2.3. Odpowiedź skokowa dla ciągłego regulatora PID

Regulator z parametrami wyznaczonymi metodą Zieglera-Nicholsa działa dobrze, jednak nie idealnie, gdyż sygnał wyjściowy nie stabilizuje się od razu w wartości zadanej tylko oscyluje przez jakiś czas.

2.2. Symulacja cyfrowego algorytmu PID

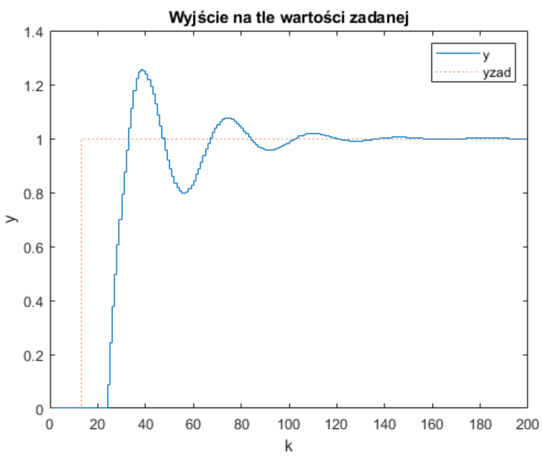
Korzystając z obliczonych nastaw ciągłego regulatora PID wyznaczono parametry r_0, r_1, r_2 dyskretnego regulatora PID.

```
r2 = Kr * Td / Tp;
r1 = Kr * (Tp/(2*Ti)-2*Td/Tp-1);
r0 = Kr * (1+Tp/(2*Ti)+Td/Tp);
```

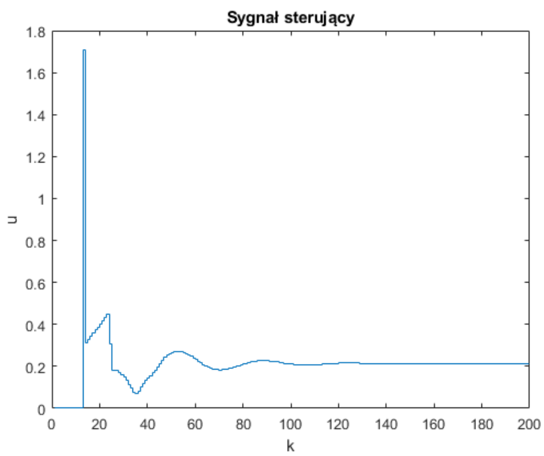
W MATLAB zaimplementowano program do symulacji cyfrowego algorytmu PID.

```
kk = 200;
a1 = Gz.Denominator{1}(2);
a0 = Gz.Denominator{1}(3);
b1 = Gz.Numerator{1}(2);
b0 = Gz.Numerator{1}(3);
u(1:12) = 0;
y(1:12) = 0;
e(1:12) = 0;
yzad(1:14) = 0;
yzad(13:kk) = 1;
for k = 13:kk
    y(k) = b1*u(k-11)+b0*u(k-12)-a1*y(k-1)-a0*y(k-2);

    e(k) = yzad(k)-y(k);

    u(k) = r2*e(k-2)+r1*e(k-1)+r0*e(k)+u(k-1);
end
```

Przyjęto stałą trajektorię referencyjną dla całego horyzontu predykcji, do wyznaczenia odpowiedzi skokowej i symulacji obiektu wykorzystano wcześniej wyznaczone równanie różnicowe (Wzór 1.2).



Rys. 2.4. Sygnał wyjściowy



Rys. 2.5. Sygnał sterujący

Znów widać, że regulator działa poprawnie ale nie idealnie. Sygnał sterujący ma bardzo duży skok na początku przebiegu, co nie jest pożądane. Ponadto odpowiedzi skokowe dla obu wersji regulatora są bardzo podobne. Podsumowując metoda Zieglera-Nicholsa ma swoje zalety: jest szybka i intuicyjna, oraz wady: słaba dokładność.

3. Algorytm DMC

3.1. Realizacja w MATLAB

Zainicjowano parametry

```
% parametry symulacji
tmax = 100;
y_zad = 1;
% wartosci wyjsciowe
wyu = 0;
wyy = 0;
% parametry obiektu
a1 = Gz.Denominator{1}(2);
a0 = Gz.Denominator{1}(3);
b1 = Gz.Numerator{1}(2);
b0 = Gz.Numerator{1}(3);
% horyzonty
D = 80; N = 70; Nu = 70;
% wspolczynniki s
sv = step(Gz);
sv(1) = []; % uwzględniając opóźnienie
% współczynnik kary
lambda = 1;
% warunki początkowe
y(1:12) = 0;
u(1:12) = 0;
```

Stworzono macierze M i M_p korzystając z następujących wzorów

$$M = \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ s_2 & s_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ s_N & s_{N-1} & \cdots & s_{N-N_u+1} \end{bmatrix} \tag{3.1}$$

$$M^P = \begin{bmatrix} s_2 - s_1 & s_3 - s_2 & \cdots & s_D - s_{D-1} \\ s_3 - s_1 & s_4 - s_2 & \cdots & s_{D+1} - s_{D-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N+1} - s_1 & s_{N+2} - s_2 & \cdots & s_{N+D-1} - s_{D-1} \end{bmatrix} \tag{3.2}$$

```
M = zeros(N, Nu);
for i = 1:N
    for j = 1:Nu
        if i-j+1 > 0
            M(i, j) = sv(i-j+1);
        end
    end
end
Mp = zeros(N, D-1);
for i = 1:N
    for j = 1:D-1
        if j + i <= D
            Mp(i, j) = sv(i+j) - sv(j);
        else
            Mp(i, j) = sv(D) - sv(j);
        end
    end
end
```

Obliczono parametry regulatora

```
% obliczenie parametrów regulatora
I = eye(Nu);
K = inv((M'*M + lambda*I))*M';
Ku = K(1,:)*Mp;
Ke = sum(K(1,:));
```

Stworzono główną pętlę programu

```
for k=13:tmax
    % obiekt
    y(k) = b1*u(k-11)+b0*u(k-12)-a1*y(k-1)-a0*y(k-2);

    % regulator
    ek = y_zad - y(k);

    deltauk = Ke*ek-Ku*deltaupk';
    for n=D-1:-1:2
        deltaupk(n)=deltaupk(n-1);
    end
    deltaupk(1)=deltauk;
    u(k) = u(k-1)+deltaupk(1);

    wyu(k) = u(k);
    wyy(k) = y(k);
end
```

Wzór wykorzystany do liczenia $\Delta u(k)$

$$\Delta u(k) = \Delta u(k \mid k) = k^e \left(y^{zad}(k) - y(k)\right) - \sum_{j=1}^{D-1} k_j^u \Delta u(k-j) \tag{3.3}$$

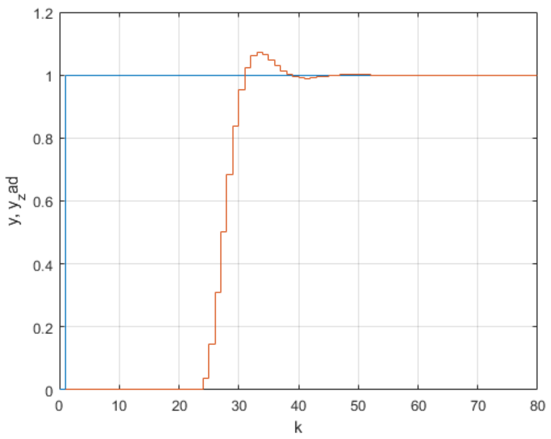
gdzie

$$k^e = \sum_{p=1}^{N} k_{1,p} \tag{3.4}$$

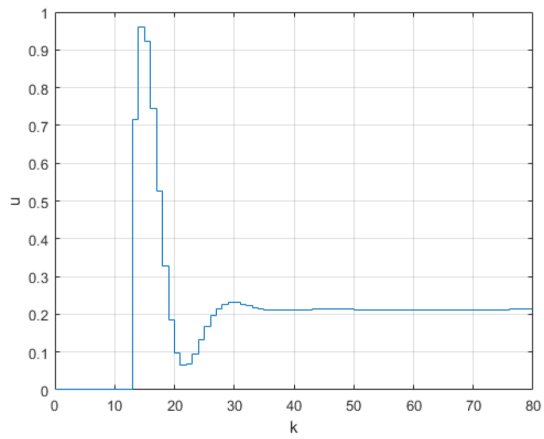
$$k^u = \overline{K}_1 M_j^P \quad dla \quad j = 1, 2, ..., D-1 \tag{3.5}$$

3.2. Dobieranie parametrów algorytmu DMC

Zaczęto od określenia horyzontu dynamiki na podstawie odpowiedzi skokowej. Wybrano $D = 79$, gdyż taka była długość wektora wartości odpowiedzi skokowych modelu. Przyjęto $\lambda = 1$ oraz $N = N_u = D$ i sprawdzono poprawność działania regulatora.

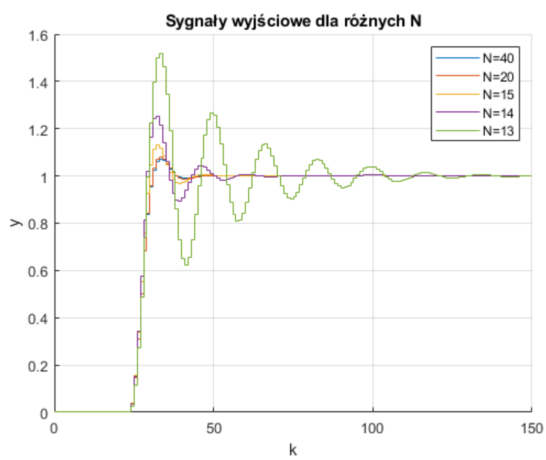


Rys. 3.1. Sygnał wyjściowy

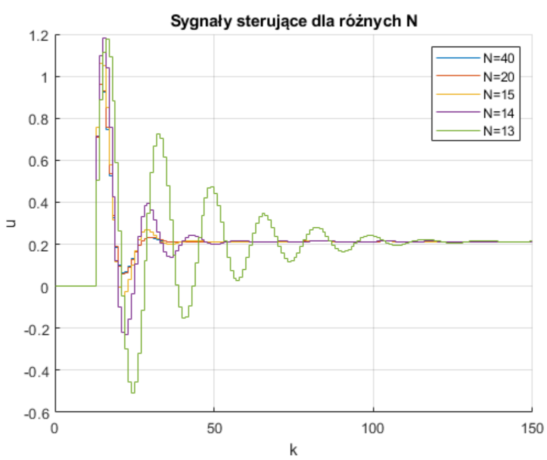


Rys. 3.2. Sygnał sterujący

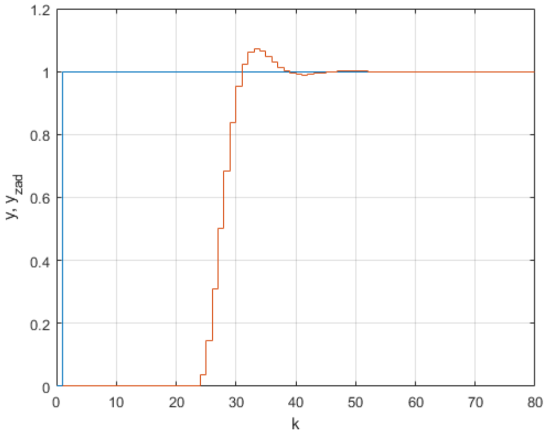
Regulator działa poprawnie.
Następnie zaczęto stopniowo skracać stopień predykcji przy zachowaniu warunku $N_u = N$ i sprawdzano czy regulator dalej działa. Zatrzymano się na wartości $N = 13$, ponieważ dla mniejszych wartości zaczęły się pojawiać bardzo duże oscylacje, co wynika z opóźnienia obiektu.



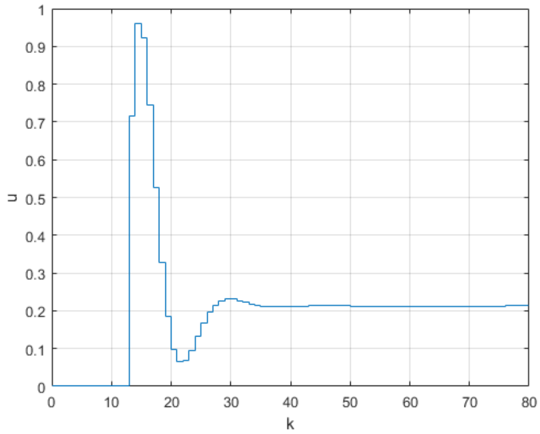
Rys. 3.3. Sygnały wyjściowe



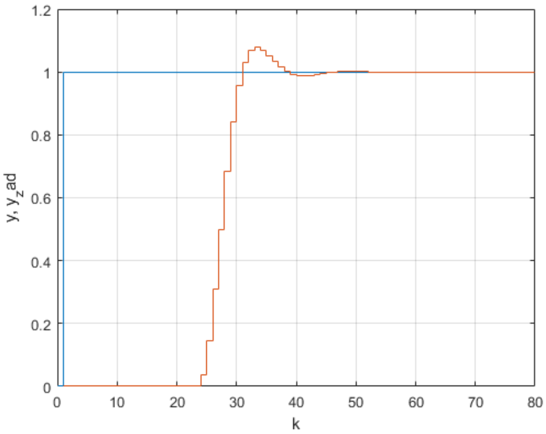
Rys. 3.4. Sygnały sterujące



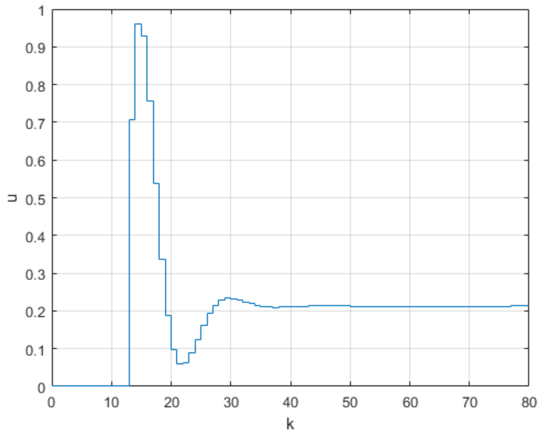
Rys. 3.5. Sygnał wyjściowy dla $N = N_u = 40$



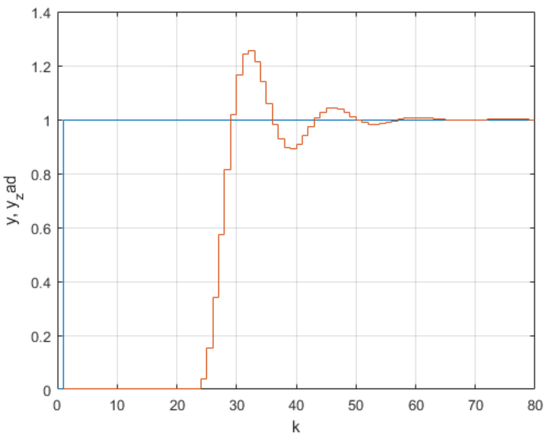
Rys. 3.6. Sygnał sterujący dla $N = N_u = 40$



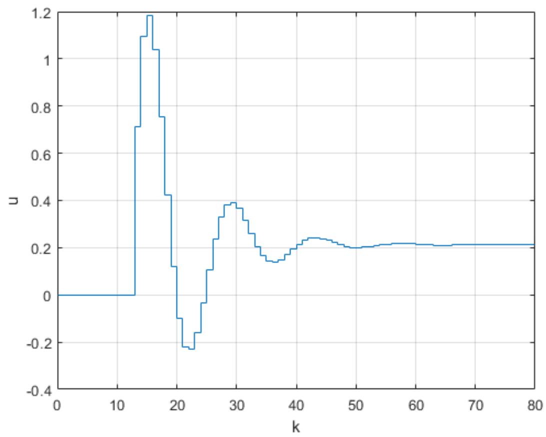
Rys. 3.7. Sygnał wyjściowy dla $N = N_u = 20$



Rys. 3.8. Sygnał sterujący dla $N = N_u = 20$



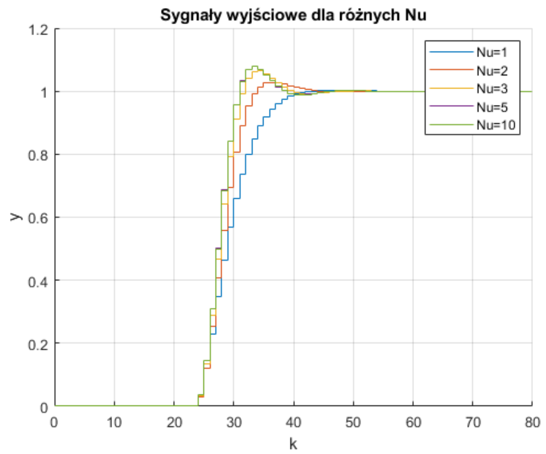
Rys. 3.9. Sygnał wyjściowy dla $N = N_u = 14$



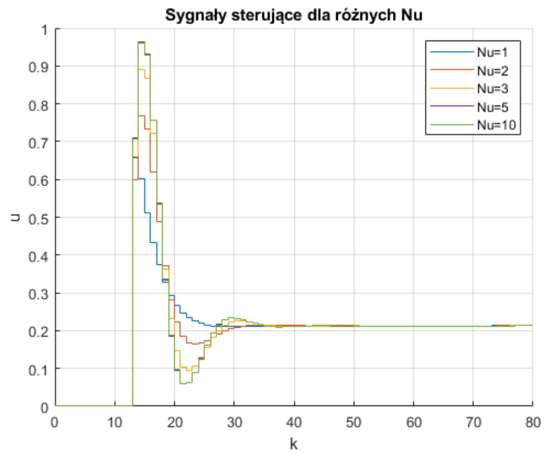
Rys. 3.10. Sygnał sterujący dla $N = N_u = 14$

Zdecydowano się na $N = 20$.

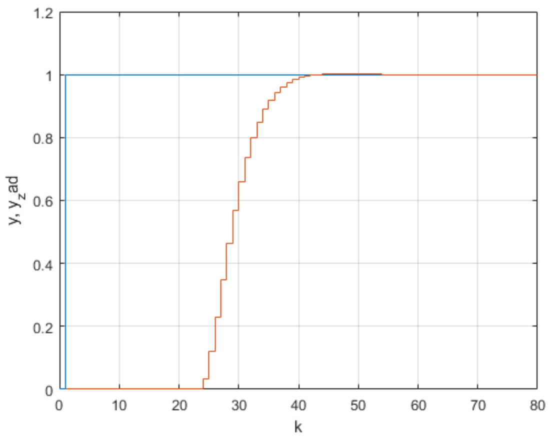
Zbadano wpływ horyzontu sterowania na jakość regulacji.



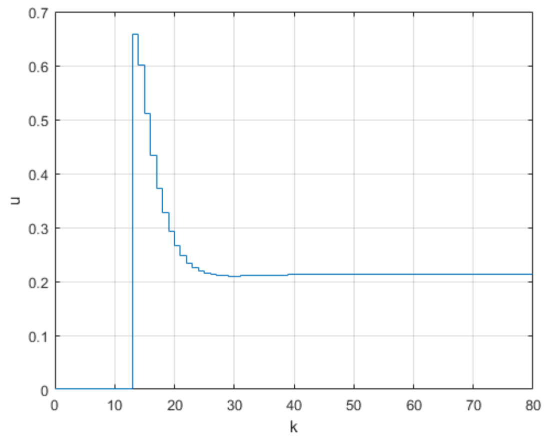
Rys. 3.11. Sygnały wyjściowe



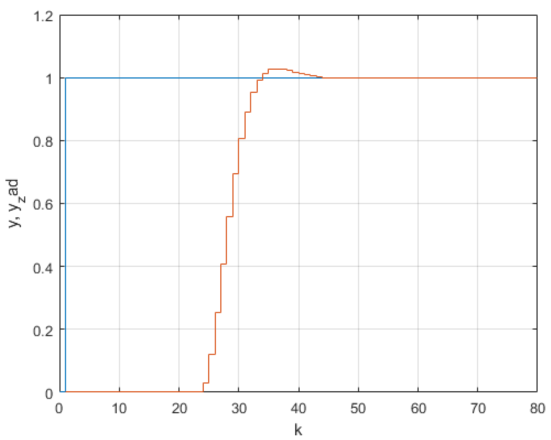
Rys. 3.12. Sygnały sterujące



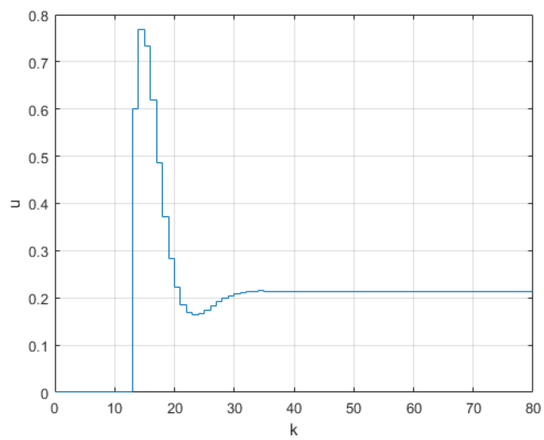
Rys. 3.13. Sygnał wyjściowy dla $N = 20,\ N_u = 1$



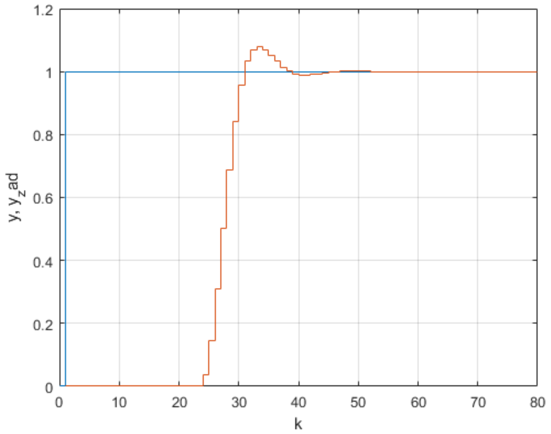
Rys. 3.14. Sygnał sterujący dla $N = 20,\ N_u = 1$



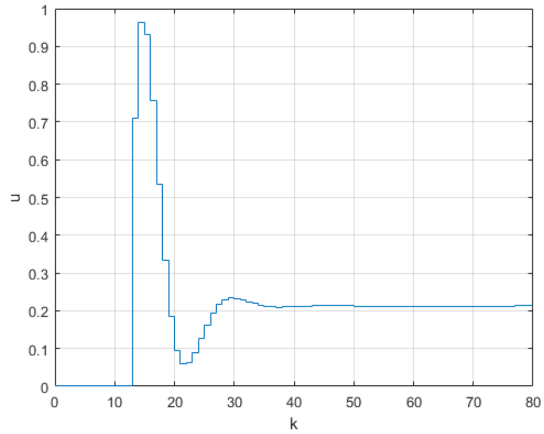
Rys. 3.15. Sygnał wyjściowy dla $N = 20,\ N_u = 2$



Rys. 3.16. Sygnał sterujący dla $N = 20,\ N_u = 2$



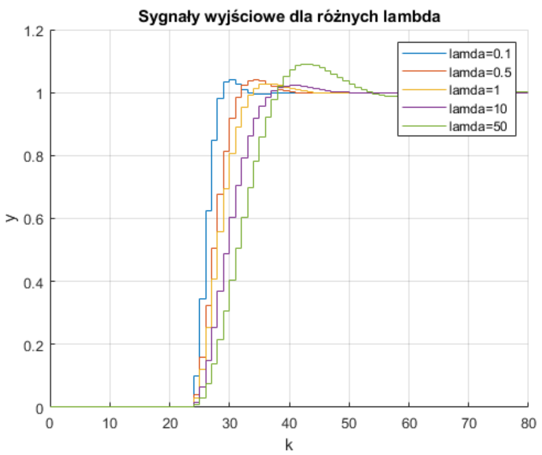
Rys. 3.17. Sygnał wyjściowy dla $N = 20,\ N_u = 5$



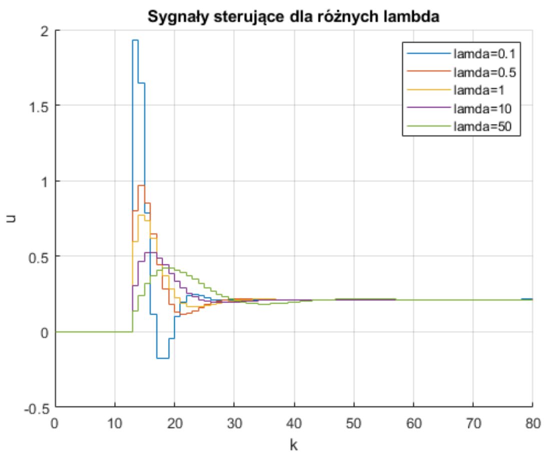
Rys. 3.18. Sygnał sterujący dla $N = 20,\ N_u = 5$

Na podstawie powyższych odpowiedzi skokowych i charakterystyk sygnału sterującego można stwierdzić, że najlepszą wartością horyzontu sterowania jest $N_u = 2$, gdyż najszybciej osiąga wartość zadaną i praktycznie nie występuje przeregulowanie. Widać też, że wydłużenie horyzontu sterowania pogarsza działanie programu, ponieważ im większa jego wartość tym bardziej skacze sygnał sterowania, co ma negatywny wpływ na działanie regulatora.

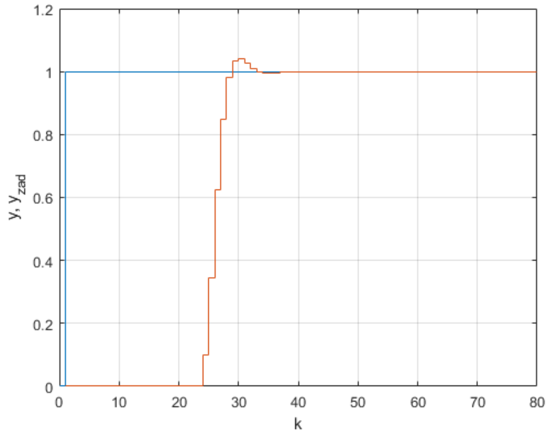
Aby poprawnie dobrać parametr λ sprawdzono przebiegi wyjścia oraz sygnału sterującego dla różnych jego wartości (zaczęto od 0.1 i stopniowo zwiększano)



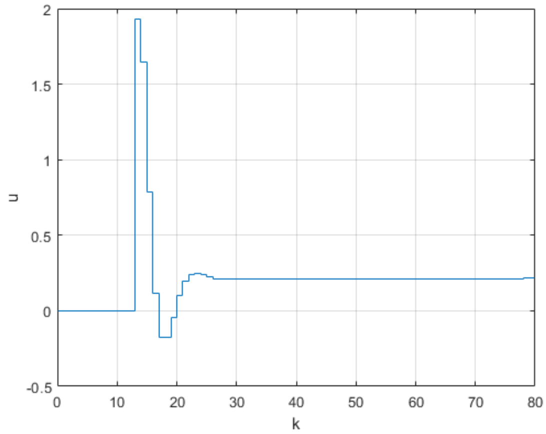
Rys. 3.19. Sygnały wyjściowe



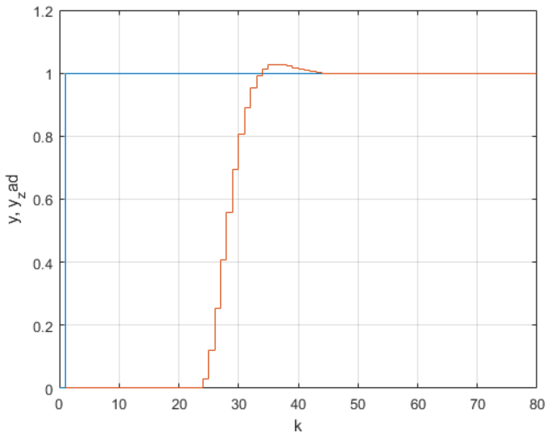
Rys. 3.20. Sygnały sterujące



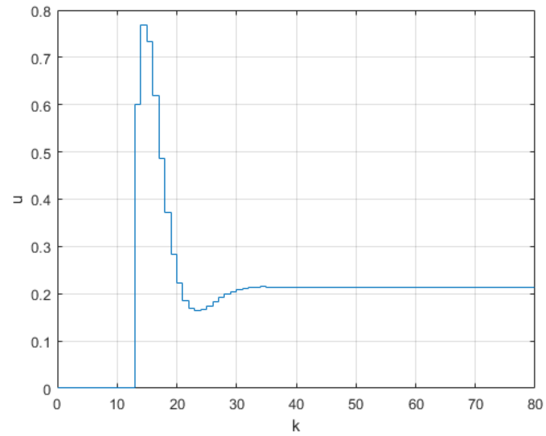
Rys. 3.21. Sygnał wyjściowy dla $\lambda = 0.1$



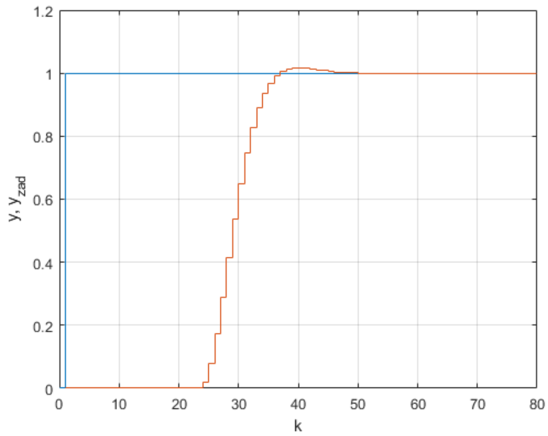
Rys. 3.22. Sygnał sterujący dla $\lambda = 0.1$



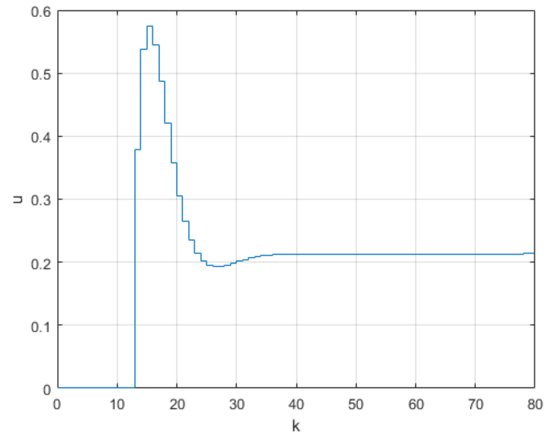
Rys. 3.23. Sygnał wyjściowy dla $\lambda = 1$



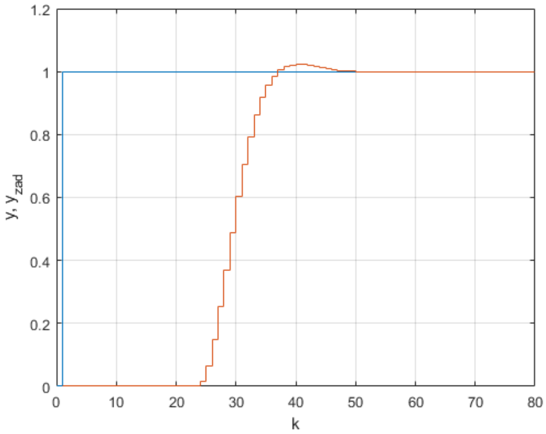
Rys. 3.24. Sygnał sterujący dla $\lambda = 1$



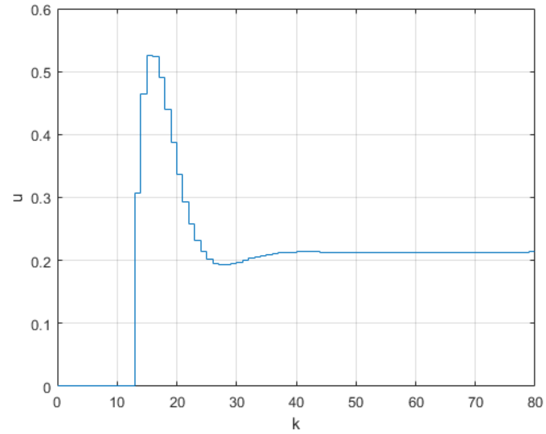
Rys. 3.25. Sygnał wyjściowy dla $\lambda = 5$



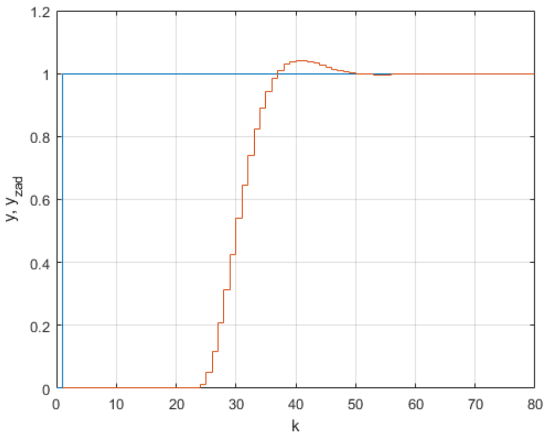
Rys. 3.26. Sygnał sterujący dla $\lambda = 5$



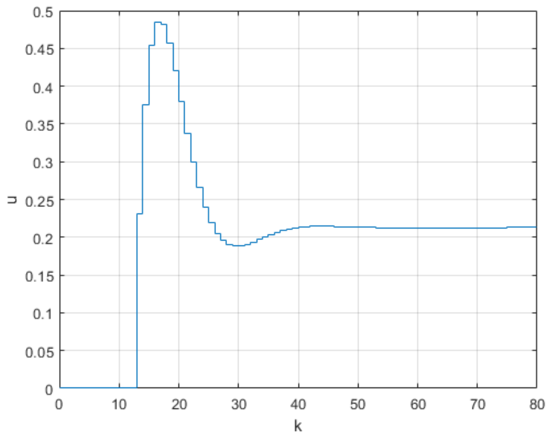
Rys. 3.27. Sygnał wyjściowy dla $\lambda = 10$



Rys. 3.28. Sygnał sterujący dla $\lambda = 10$



Rys. 3.29. Sygnał wyjściowy dla $\lambda = 20$

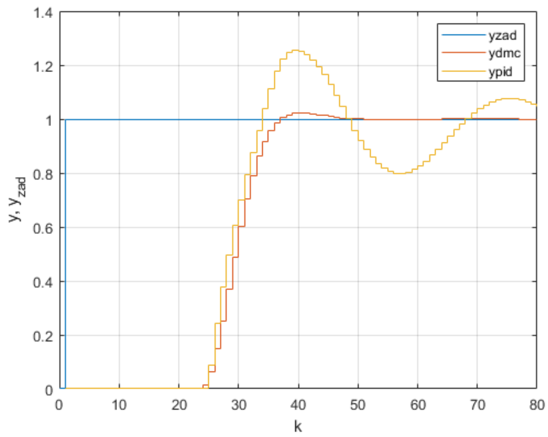


Rys. 3.30. Sygnał sterujący dla $\lambda = 20$

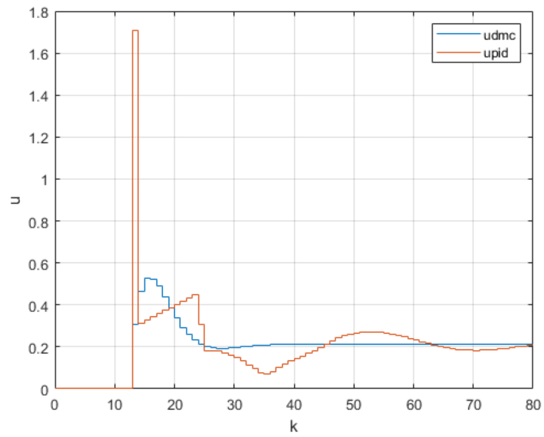
Przy doborze parametru λ, należy zwrócić szczególną uwagę na wygląd sygnału sterującego. Może się wydawać, że $\lambda = 0.1$ jest najlepsza, gdyż najszybciej osiąga wartość zadaną, jednak sygnał sterujący dla tej wartości jest tragiczny. Jest to praktycznie pionowy słupek idący do bardzo dużej wartości.

Zdecydowano się na kompromis między szybkością regulacji i jakością sygnału sterującego i wybrano $\lambda = 10$, gdyż sygnał sterujący jest już dobrej jakości, a wartość zadana jest całkiem szybko osiągana.

3.3. Porównanie DMC i PID



Rys. 3.31. Sygnały wyjściowe dla DMC i PID



Rys. 3.32. Sygnały sterujące dla DMC i PID

Na powyższych rysunkach widać, że algorytm DMC znacznie lepiej radzi sobie z regulacją. Dużo szybciej osiąga wartość zadaną (pełen przebieg tego jak regulator PID osiąga wartość zadaną na Rys 2.4). Ponadto jego sygnał sterujący jest dużo lepszej jakości. Regulator DMC pod każdym względem jest lepszy od regulatora PID. Wynika to z tego, że metoda Zieglera-Nicholsa jest niedokładna i z tego, że algorytm DMC jest algorytmem regulacji predykcyjnej.

3.4. Wyznaczenie obszarów stabilności algorytmów DMC i PID

Aby poprawnie wyznaczyć obszary stabilności sprawdzono dla jakich odchyleń parametrów od ich wartości nominalnych w odpowiedzi skokowej pojawiały się stałe oscylacje (Podobnie jak podczas szukania K_{kryt} w metodzie Zieglera-Nicholsa). W celu zrealizowania tego w MATLAB zmodyfikowano programy. Wyznaczając transmitancję dyskretna dodano zmienną pomocniczą nazwaną w kodzie "pom", którą zwiększamy o 1 za każdym razem jak zwiększamy T_o o 0.5 (zwiększamy tylko o taką wartość - wynika z treści zadania), gdyż wtedy zmienia się transmitancja i równanie różnicowe z którego korzystamy w regulatorach.

Poniżej przykład zastosowania zmiennej p w regulatorze PID.

```
u(1:12+pom) = 0; y(1:12+pom) = 0; e(1:12+pom) = 0;
yzad(1:14+pom) = 0; yzad(13+pom:kk) = 1;
for k = (13+pom):kk
    y(k) = b1*u(k-11-pom)+b0*u(k-12-pom)-a1*y(k-1)-a0*y(k-2);
    e(k) = yzad(k)-y(k);
    u(k) = r2*e(k-2)+r1*e(k-1)+r0*e(k)+u(k-1);
end
```

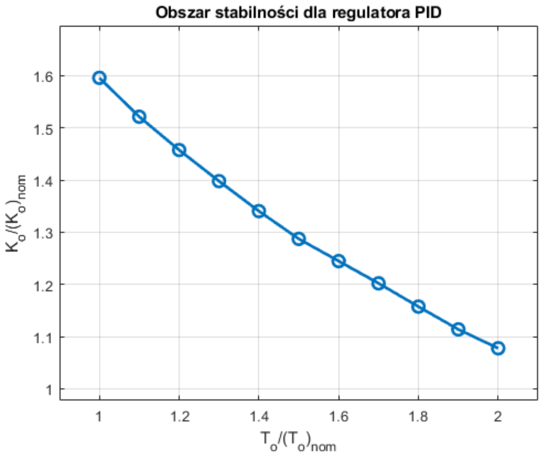
Rówania różnicowe kolejno dla $T_o = 5$ i $T_o = 5.5$

$$y(k) = -a_1 y(k-1) - a_0 y(k-2) + b_1 u(k-11) + b_2 u(k-12) \tag{3.6}$$

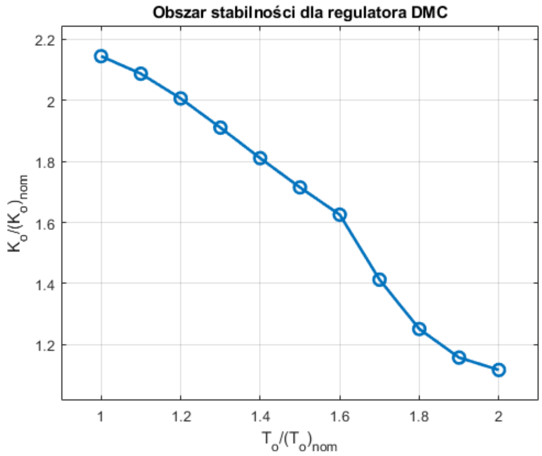
$$y(k) = -a_1 y(k-1) - a_0 y(k-2) + b_1 u(k-12) + b_2 u(k-13) \tag{3.7}$$

Dla kolejnych wartości T_o zmieniają się indeksy przy u. Ponadto należy pamiętać o tym, że zmieniamy tylko obiekt, a regulator zostaje ten sam, Jest to ważne w algorytmach DMC i GPC, które korzystają z macierzy M i M^P, które wyznaczane są z opowiedzi skokowej obiektu (z parametrami o wartościach nominalnych).

Poniżej rysunki wyznaczonych obszarów stabilności



Rys. 3.33. Obszar stabilności algorytmu PID



Rys. 3.34. Obszar stabilności algorytmu DMC

Zgodnie z oczekiwaniem, zmniejszenie opóźnienia To zwiększa dopuszczalny zakres zmian współczynnika wzmocnienia K. Algorytm DMC jest trochę bardziej odporny na zmiany K. Wkres dla PID jest prawie liniowy. Dla DMC w okolicach $T_o/(T_o)_{nom} = 1.6$ następuje lekkie załamanie i na końcu charakterystyki się odkształca. Otrzymane rysunki należy interpretować tak, że pod otrzymaną charakterystyką jest obszar w którym algorytm jest stabilny, a ponad nią jest niestabilny. Dzieki temu wiemy jak bardzo możliwe jest zwiększenie współczynnika wzmocnienia regulatora PID, co prowadzi do przyspieszenia przebiegów przejściowych. Charakterystyki pokazują też, że dla obiektów, które trochę się różnią od tego dla którego projektowaliśmy regulator algorytm dalej by działał.

4. Algortym GPC

4.1. Realizacja w MATLAB

Najpierw zainicjowano parametry oraz przyjęto takie parametry regulatora jak te finalne, dobrane dla algorytmu DMC.

```
kk = 100;
wyu = zeros(1, kk);
wyy = zeros(1, kk);
yzad = 1;
s = step(Gz);
s(1) = [];
D = 79;
N = 20;
Nu = 2;
lambda = 10;
a1 = Gz.Denominator{1}(2);
a0 = Gz.Denominator{1}(3);
b1 = Gz.Numerator{1}(2);
b0 = Gz.Numerator{1}(3);
y(1:12) = 0;
u(1:12) = 0;
```

Stworzono macierz M tak samo jak w algorytmi DMC.
Algorytm GPC różni się od DMC sposobem obliczania $\Delta u(k)$.

$$\Delta u(k) = \Delta u(k \mid k) = \sum_{p=1}^{N} k_{1,p} \left(y^{zad}(k+p \mid k) - y^0(k+p \mid k) \right) \tag{4.1}$$

Składową swobodną $y^0(k+p \mid k)$ oblicza się na podstawie predykcji wyjścia. Wyraża się ona zależnością

$$\begin{aligned} y^0(k+p \mid k) = & \sum_{i=1}^{N_{un}(p)} b_i u(k-1) + \sum_{i=N_{un}(p)+1}^{n_B} b_i u(k-i+p) \\ & - \sum_{i=1}^{N_{\hat{y}}(p)} a_i y^0(k-i+p \mid k) - \sum_{i=N_{\hat{y}}(p)+1}^{n_A} a_i y(k-i+p) + d(k) \end{aligned} \tag{4.2}$$

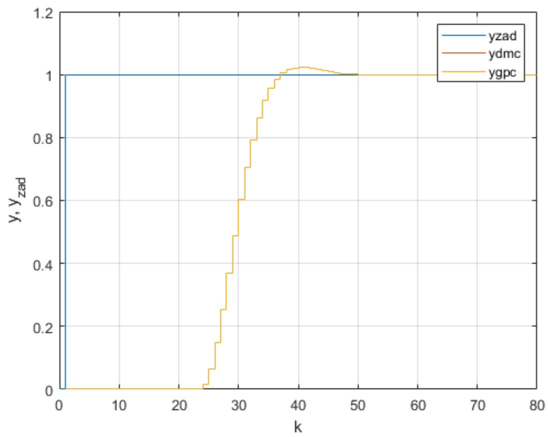
Powyższy wzór pozwala obliczać odpowiedź swobodną w każdej iteracji w sposób rekurencyjny i właśnie z tej metody skorzystano w implementacji programu.

Poniżej implementacja głównej pętli programu w MATLAB

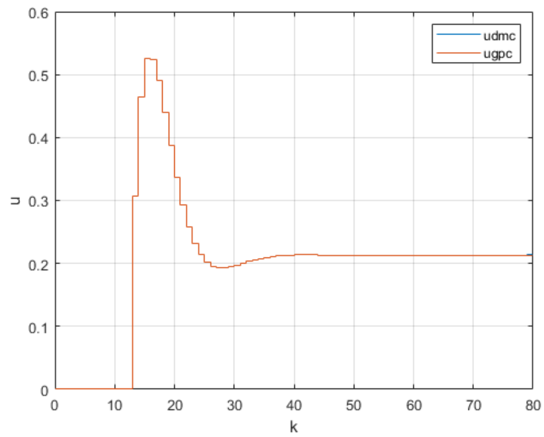
```
for k = 13:kk
    y(k) = b1*u(k-11)+b0*u(k-12)-a1*y(k-1)-a0*y(k-2);
    d(k) = y(k) - (b1*u(k-11)+b0*u(k-12)-a1*y(k-1)-a0*y(k-2));
    y_0 = zeros(1, N);
    y_0(1) = -a1*y(k) - a0*y(k-1) + b1*u(k-10) + b0*u(k-11) + d(k);
    y_0(2) = -a1*y_0(1) - a0*y(k) + b1*u(k-9) + b0*u(k-10) + d(k);
    for p=3:N
        if p <= 10
            y_0(p) = -a1*y_0(p-1) - a0*y_0(p-2) + b1*u(k-11+p)  ...
                  + b0*u(k-12+p) + d(k);
        elseif p == 11
            y_0(p) = -a1*y_0(p-1) - a0*y_0(p-2) + b1*u(k-1) ...
                  + b0*u(k-2) + d(k);
        else
            y_0(p) = -a1*y_0(p-1) - a0*y_0(p-2) + b1*u(k-1) ...
                  + b0*u(k-1) + d(k);
        end
    end
    deltauk = K(1, :)*(yzad*ones(1, N) - y_0)';
    u(k) = u(k-1) + deltauk;
    wyu(k) = u(k);
    wyy(k) = y(k);
end
```

4.2. Porównanie algorytmu GPC i DMC

Algorytmy porównano dla takich samych parametrów.
Najpierw przetestowano ich zachowanie dla skokowej zmiany wartości zadanej z 0 na 1.



Rys. 4.1. Porównanie sygnałów wyjściowych GPC i DMC



Rys. 4.2. Porównanie sygnałów sterujących GPC i DMC

Na rysunkach widać, że zarówno sygnały wyjściowe jak i sterujące nakładają się na siebie. Regulatory pracują tak samo dobrze dla skokowej zmiany wartości zadanej.

Następnie przetestowano regulatory przy skokowej zmianie niemierzalnego zakłócenia dodanego do wyjścia obiektu (o wartości 0.05 od 40 kroku) i stałej wartości zadanej.

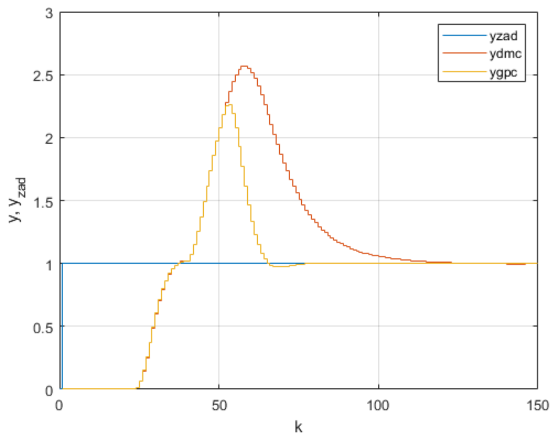
Inicjalizacja zakłóceń w MATLAB

```
zaklocenia(1:40) = 0;
zaklocenia(41: tmax) = 0.05;
```

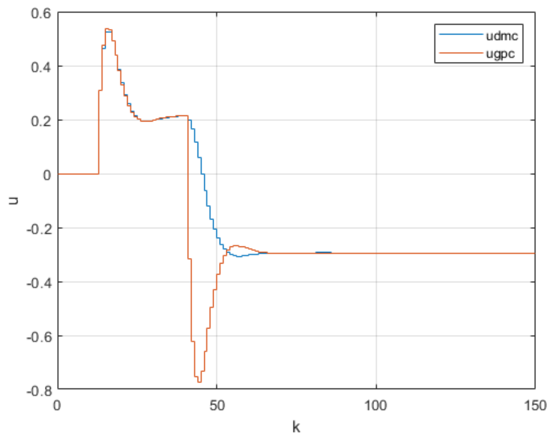
Dodanie zakłóceń do wyjścia w MATLAB

```
y(k) = b1*u(k-11)+b0*u(k-12)-a1*y(k-1)-a0*y(k-2)+zaklocenia(k);
```

Poniżej wykresy otrzymane z symulacji wpływu zakłóceń na pracę regulatorów



Rys. 4.3. Porównanie sygnałów sterujących GPC i DMC z zakłóceniami



Rys. 4.4. Porównanie sygnałów sterujących GPC i DMC z zakłóceniami

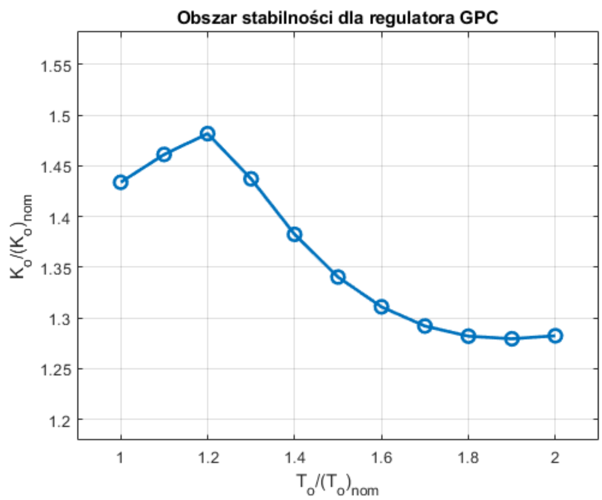
Na powyższych rysunkach widać, że algorytm GPC dużo lepiej poradził sobie z regulacją przy dodatkowych zakłóceniach. Patrząc na sygnał sterujący widać, że dużo szybciej i lepiej zareagował w momencie pojawienia się zakłócenia. Podsumowując, kiedy nie ma zakłóceń algorytmy prezentują się tak samo, jednak kiedy pojawiają się zakłócenia algorytm GPC wypada dużo lepiej, ponieważ progonozuje się w nim niemierzalne zakłócenie wyjścia d(k).

4.3. Wyznaczenie obszaru stabilności algorytmu GPC

Obszar stabilności dla algorytmu GPC wyznaczono tą samą metodą co dla algorytmu PID i DMC. Należało jednak dodatkowo uwzględnić wcześniej omówioną zmienną pomocniczą "pom" w obliczaniu składowej swobodnej. Oraz nowych współczynników "b1_new" i "b2_new" w celu zmiany obiektu.

```
for k = (13+pom):tmax
    y(k) = b1_new*u(k-11-pom)+b0_new*u(k-12-pom)-a1*y(k-1)-a0*y(k-2);
    d(k) = y(k) - (b1*u(k-11-pom)+b0*u(k-12-pom)-a1*y(k-1)-a0*y(k-2));
    y_0 = zeros(1, N);
    y_0(1) = -a1*y(k) - a0*y(k-1) + b1*u(k-10-pom) ...
            + b0*u(k-11-pom) + d(k);
    y_0(2) = -a1*y_0(1) - a0*y(k) + b1*u(k-9-pom) ...
            + b0*u(k-10-pom) + d(k);
    for p=3:N
        if p <= 10
            y_0(p) = -a1*y_0(p-1) - a0*y_0(p-2) + b1*u(k-11+p) ...
                    + b0*u(k-12+p) + d(k);
        else
            y_0(p) = -a1*y_0(p-1) - a0*y_0(p-2) + b1*u(k-1) ...
                    + b0*u(k-1) + d(k);
        end
    end
    deltauk = K(1, :)*(yzad*ones(1, N) - y_0)';
    u(k) = u(k-1) + deltauk;
    wyu_gpc(k) = u(k);
    wyy_gpc(k) = y(k);
end
```

Poniżej wykres przedstawiający obszar stabliności algorytmu GPC



Rys. 4.5. Obszar stabilności dla algorytmu GPC

Krzywa stabliności lekko przypomina tą otrzymaną dla algorytmu DMC Rys. 3.34. Różnice są w tym, że dla T_o mniejszych od $T_o = 1.2(T_o)_{nom}$ zwiększa się dopuszczalny zakres zmian współczynnika wzmocnienia K. Dopiero dla T_o większych od $T_o = 1.2(T_o)_{nom}$, zwiększenie opóźnienia To zmniejsza dopuszczalny zakres zmian współczynnika wzmocnienia K. Algorytm GPC jest mniej odporny na zmiany K od algorytmu DMC, ponieważ stosunki $K_o/(K_o)_{nom}$ dla GPC są mniejsze dla danego To.

Podsumowując algorytm GPC jest lepszym wyborem od algorytmu DMC, kiedy mamy do czynienia z niemierzalnymi zakłóceniami, a DMC będzie lepszym wyborem kiedy skupiamy się na odporności na odchylenia współczynnika wzmocnienia K.