**Wydział Elektroniki i Technik Informacyjnych**
**Politechnika Warszawska**

# Projektowanie układów sterowania (projekt grupowy)

## Sprawozdanie z projektu i ćwiczenia laboratoryjnego nr 4, zadanie nr 12

**Paulina Dąbrowska, Miłosz Kowalewski,**
**Adam Rybojad, Mikołaj Wewiór**

**Warszawa, 2024**

# Spis treści

# 1. CZĘŚĆ PROJEKTOWA

## 1.1. Sprawdzenie poprawności punktu pracy

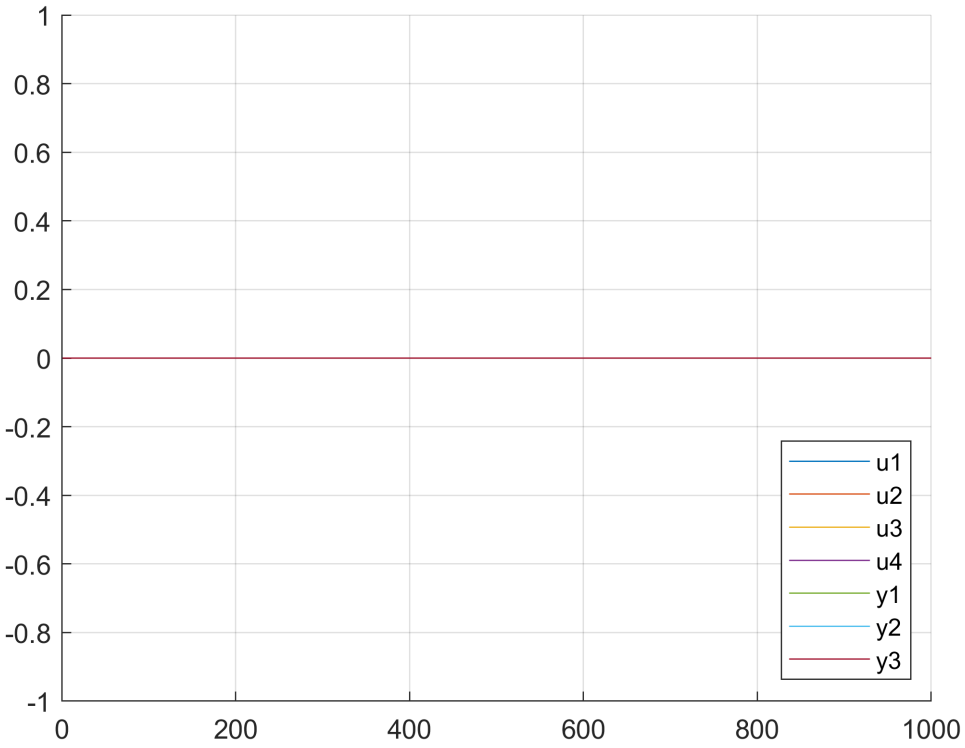
Punkt pracy sprawdzany jest dla sterowań

$$u_1 = u_2 = u_3 = u_4 = 0$$

i oczekiwany stanem wyjść jest

$$y_1 = y_2 = y_3 = 0$$

Zgodnie z oczekiwaniami wartości te ustalają się w zerze:




Rys. 1. Sprawdzenie punktu pracy

## 1.2. Odpowiedzi skokowe

Odpowiedzi skokowe wyznaczamy dla 12 torów procesu

$$y_m(u_n)$$

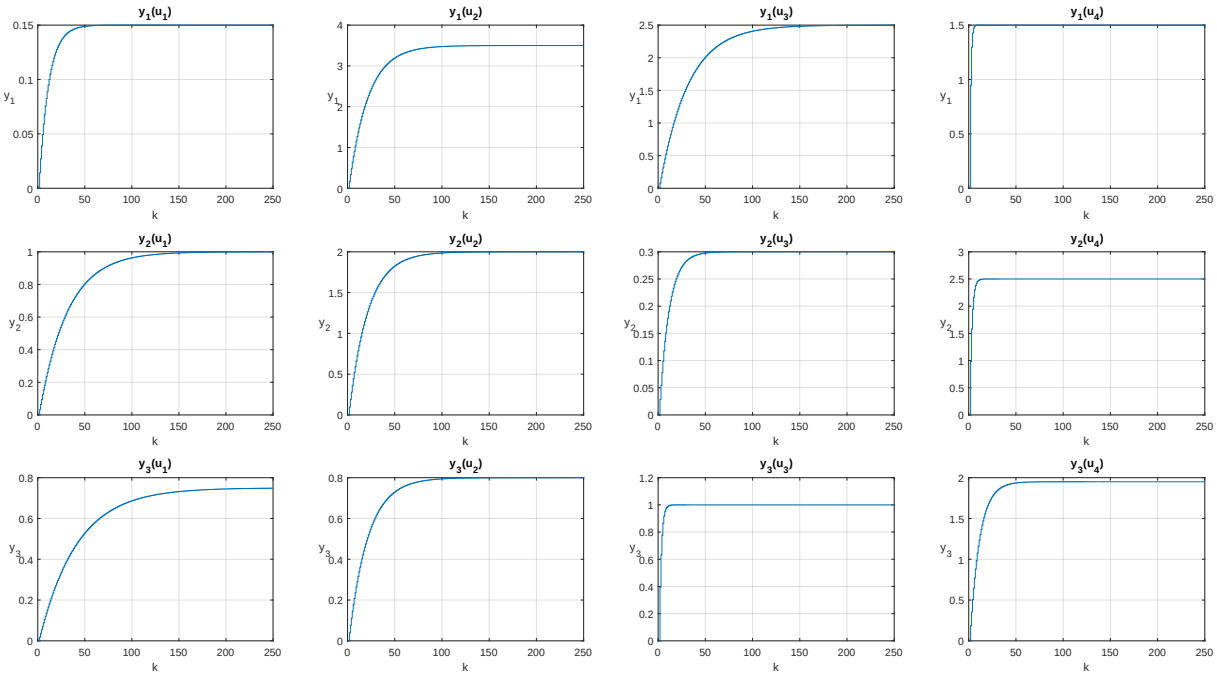
gdzie

$$m = 1, 2, 3$$

zaś

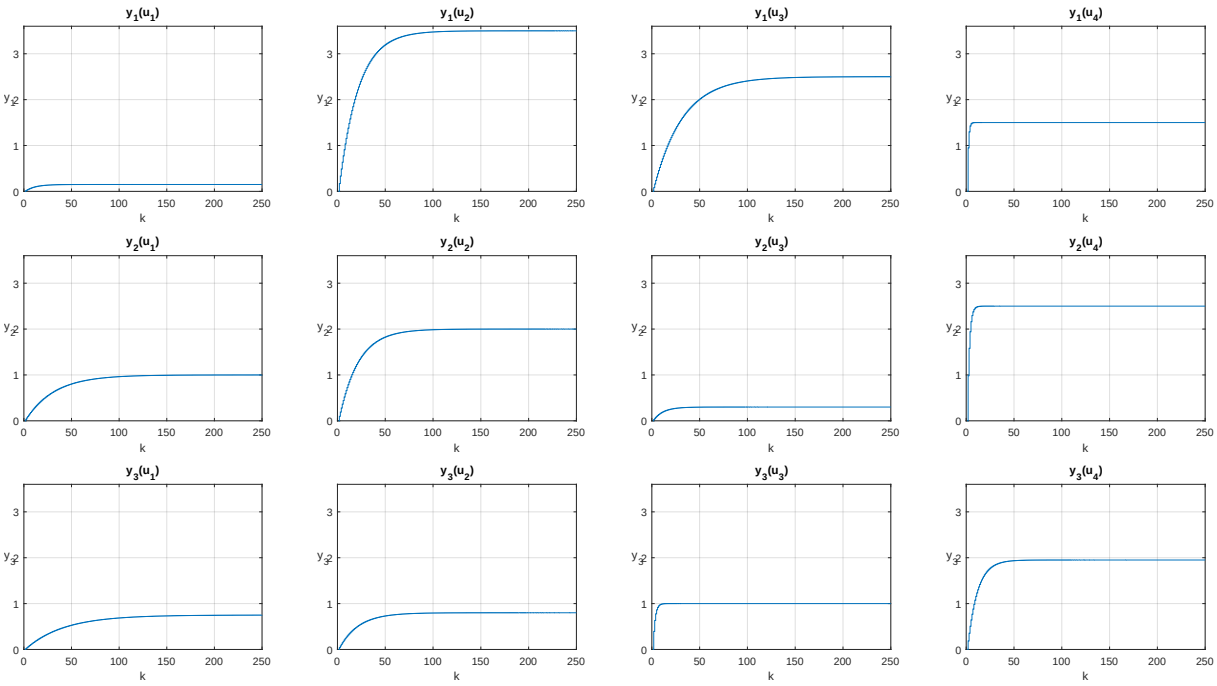
$$n = 1, 2, 3, 4$$

Wykresy odpowiedzi skokowych:




Rys. 2. Odpowiedzi każdego z wyjść na skok danego wejścia

Oraz te same odpowiedzi skokowe w jednej skali




Rys. 3. Odpowiedzi każdego z wyjść na skok danego wejścia

Przypisywanie kolejnych współczynników odpowiedzi skokowej realizujemy w następujący sposób:

```
%% ODPOWIEDŹ SKOKOWA

S = cell(len, 1);
nullS = zeros(3, 4);
%             ny, nu

for i = 1:len
    S{i} = nullS;
end


for wejscie = 1:nu

    if      wejscie == 1
        u1(start:n) = 1;
    elseif  wejscie == 2
        u2(start:n) = 1;
    elseif  wejscie == 3
        u3(start:n) = 1;
    elseif  wejscie == 4
        u4(start:n) = 1;
    end

    for k = start:len+1

        [y1(k),y2(k),y3(k)] = symulacja_obiektu12y_p4( ...
            u1(k-1), u1(k-2), u1(k-3), u1(k-4), ...
            u2(k-1), u2(k-2), u2(k-3), u2(k-4), ...
            u3(k-1), u3(k-2), u3(k-3), u3(k-4), ...
            u4(k-1), u4(k-2), u4(k-3), u4(k-4), ...
            ...
            y1(k-1), y1(k-2), y1(k-3), y1(k-4), ...
            y2(k-1), y2(k-2), y2(k-3), y2(k-4), ...
            y3(k-1), y3(k-2), y3(k-3), y3(k-4) ...
        );

        p = k-start+1;

        S{p}(1, wejscie) = y1(k);
        S{p}(2, wejscie) = y2(k);
        S{p}(3, wejscie) = y3(k);

    end


    [u1, u2, u3, u4, y1, y2, y3] = zerowanie(n);

end

```

```
50
51  for k=1:len-start+1
52
53      p = k;
54
55      % kolumna 1 : Sp^(ny 1)
56      S11(p) = S{k}(1, 1);
57      S21(p) = S{k}(2, 1);
58      S31(p) = S{k}(3, 1);
59
60      % kolumna 2 : Sp^(ny 2)
61      S12(p) = S{k}(1, 2);
62      S22(p) = S{k}(2, 2);
63      S32(p) = S{k}(3, 2);
64
65      % kolumna 3 : Sp^(ny 3)
66      S13(p) = S{k}(1, 3);
67      S23(p) = S{k}(2, 3);
68      S33(p) = S{k}(3, 3);
69
70      % kolumna 4 : Sp^(ny 4)
71      S14(p) = S{k}(1, 4);
72      S24(p) = S{k}(2, 4);
73      S34(p) = S{k}(3, 4);
74
75  end
```

### 1.3. Implementacja algorytmu PID

Dla regulatora wielowymiarowego nie możemy skorzystać z typowego równania:

$$u(k) = r_2 \cdot e(k-2) + r_1 \cdot e(k-1) + r_0 \cdot e(k) + u(k-1)$$

Musimy rozdzielić część powiązaną z uchybem w bieżącej chwili od poprzedniej wartości sterowania dlatego pojawiają się zmienne $out_{pid_i}$, które są obecnym przyrostem sterowania wyliczonym przez każdy z pojedynczych regulatorów. Zatem równanie opisujące ten regulator będzie wyglądał w następujący sposób:

$$u_i(k) = u_1(k-1) + w_1 \cdot out_{pid_1} + ... + w_n \cdot out_{pid_n}$$

gdzie $n$ to ilość regulatorów a $w$ to wagi, przez które wymnażane są przyrosty od każdego regulatora oraz

$$out_{pid_i} = r_2 \cdot e(k-2) + r_1 \cdot e(k-1) + r_0 \cdot e(k)$$

Wagi zostały dobrane odpowiednio według tego jak silnie dane wejście oddziałuje na dane wyjście. Ostatecznie, przyrost w bieżącej chwili jest średnią ważoną uchybów z każdego regulatora.

Zgodnie z powyższym implementacja regulatora PID w wersji MIMO wygląda następująco:

```
function [u1k, u2k, u3k, u4k] =
        pid_mimo(T, k, U, Ypom, Yzad, Pid1, Pid2, Pid3)

u1_prev = U(k-1, 1);
u2_prev = U(k-1, 2);
u3_prev = U(k-1, 3);
u4_prev = U(k-1, 4);

y1 = Ypom(:,1);
y2 = Ypom(:,2);
y3 = Ypom(:,3);

yzad1 = Yzad(:,1);
yzad2 = Yzad(:,2);
yzad3 = Yzad(:,3);

K_pid1 = Pid1(1); Ti1 = Pid1(2); Td1 = Pid1(3);
K_pid2 = Pid2(1); Ti2 = Pid2(2); Td2 = Pid2(3);
K_pid3 = Pid3(1); Ti3 = Pid3(2); Td3 = Pid3(3);

% parametry r regulatorów PID
r0_1 = K_pid1*(1+T/(2*Ti1)+Td1/T);
r1_1 = K_pid1*(T/(2*Ti1)-2*Td1/T-1);
r2_1 = K_pid1*Td1/T;

r0_2 = K_pid2*(1+T/(2*Ti2)+Td2/T);
r1_2 = K_pid2*(T/(2*Ti2)-2*Td2/T-1);
r2_2 = K_pid2*Td2/T;

r0_3 = K_pid3*(1+T/(2*Ti3)+Td3/T);
r1_3 = K_pid3*(T/(2*Ti3)-2*Td3/T-1);
r2_3 = K_pid3*Td3/T;

% uchyby regulacji
e1 = yzad1-y1;
e2 = yzad2-y2;
e3 = yzad3-y3;

%sygnał sterujący regulatorów PID
out_pid1 = r2_1*e1(k-2) + r1_1*e1(k-1) + r0_1*e1(k);
out_pid2 = r2_2*e2(k-2) + r1_2*e2(k-1) + r0_2*e2(k);
out_pid3 = r2_3*e3(k-2) + r1_3*e3(k-1) + r0_3*e3(k);

u1k = u1_prev + (out_pid1*0.15 + out_pid2*1 + out_pid3*0.75)/1.9;
u2k = u2_prev + (out_pid1*3.5 + out_pid2*2 + out_pid3*0.8)/6.3;
u3k = u3_prev + (out_pid1*2.5 + out_pid2*0.3 + out_pid3*1)/3.8;
u4k = u4_prev + (out_pid1*1.5 + out_pid2*2.5 + out_pid3*1.9)/5.9;
end
```

Jak widać zamieszczony kod zawiera w sobie dobrane wagi dla konkretnych $out_{pidi}$, co opisane było w równaniach nad listingiem kodu.

### 1.4. Implementacja algorytmu DMC

W wielowymiarowym algorytmie DMC do predykcji wykorzystuje się model odpowiedzi skokowych, zdefiniowany zestawem D (horyzont dynamiki) macierzy o wymiarowości $n_y \times n_u$:

$$S = \begin{bmatrix} s_p^{1,1} & \dots & s_p^{1,n_u} \\ \vdots & \ddots & \vdots \\ s_p^{n_y,1} & \dots & s_p^{n_y,n_u} \end{bmatrix}$$

gdzie $n_y$ - liczba wyjść, $n_u$ - liczba wejść.

Oznaczenie $s_p^{m,n}$ należy rozumieć jako: p-ty współczynnik odpowiedzi skokowej przy skoku (jednostkowym) sygnału n-tego wejścia, który jest obserwowany dla m-tego wyjścia.

$$K = \begin{bmatrix} \overline{K}_1 \\ \vdots \\ \overline{K}_{N_u} \end{bmatrix} = \begin{bmatrix} K_{1,1} & \dots & K_{1,N} \\ \vdots & \ddots & \vdots \\ K_{N_u,1} & \dots & K_{N_u,N} \end{bmatrix}$$

$$M^p = \begin{bmatrix} M_1^p & \dots & M_{D-1}^p \end{bmatrix}$$

$$\Delta u(k|k) = \begin{bmatrix} \Delta u_1(k|k) \\ \vdots \\ \Delta u_{n_u}(k|k) \end{bmatrix} = K^e(y^{zad}(k) - y(k)) - \sum_{i-1}^{D-1} K_i^u \Delta u(k-i)$$

$$K^e = \sum_{p=1}^{N} K_{1,p}$$

$$K_i^u = \overline{K}_1 M_i^p$$

Implementację algorytmu podzieliliśmy na dwa pliki. W pierwszym wyliczamy jednorazowo macierze $K$ oraz $M_P$. Drugim plikiem jest tzw. część online, która jest wykonywana w każdej iteracji działania algorytmu.

```
function [K, Mp] =
    dmc_mimo_offline(S, D, N, ny, Nu, nu, Mi, Lambda)

    % Mi = {mi1, mi2, mi3};
    % Lambda = {lambda1, lambda2, lambda3, lambda4};

    nullS = zeros(ny, nu);

    m_rows = ny * N;              % size(S{1}, 1) * N
    m_cols = nu * Nu;             % size(S{1}, 2) * (N-Nu+1)
    M = zeros(m_rows, m_cols);  % N x Nu

    for c = 1:nu:m_cols % kolumny
        C = (c-1)/nu + 1;

        for r = 1:ny:m_rows % wiersze
            R = (r-1)/ny + 1;

```

```
            if (R - C + 1) <= 0
                M(r:r+ny-1, c:c+nu-1) = nullS;

            elseif (R - C + 1) > D
                M(r:r+ny-1, c:c+nu-1) = S{D};

            else
                M(r:r+ny-1, c:c+nu-1) = S{R-C+1};

            end
        end
    end

    mp_rows = ny * N;                % size(S{1}, 1) * N
    mp_cols = nu * (D-1);            % size(S{1}, 2) * (D-1)
    Mp = zeros(mp_rows, mp_cols);    % N x (D-1)

    for c = 1:nu:mp_cols % kolumny
        C = (c-1)/nu + 1;

        for r = 1:ny:mp_rows % wiersze
            R = (r-1)/ny + 1;

            if R + C > D
                Scr = S{D};
            else
                Scr = S{C+R};
            end

            if C > D
                Sc = S{D};
            else
                Sc = S{C};
            end

            Scur = Scr - Sc;
            Mp(r:r+ny-1, c:c+nu-1) = Scur;
        end
    end


    PSI = eye(N*ny);
    Mi_enum = Mi;

    for J = 1:N
        for j = 1:ny
            diag = (J-1)*ny+j;
            PSI(diag, diag) = Mi_enum{j};
        end
    end

```

```
70
71      LAMBDA = eye(Nu*nu);
72      Lambda_enum = Lambda;
73
74      for J = 1:Nu
75          for j = 1:nu
76              diag = (J-1)*nu+j;
77              LAMBDA(diag, diag) = Lambda_enum{j};
78          end
79      end
80
81
82      K = ((M' * PSI * M + LAMBDA)^(-1)) * M' * PSI;
83
84  end
```

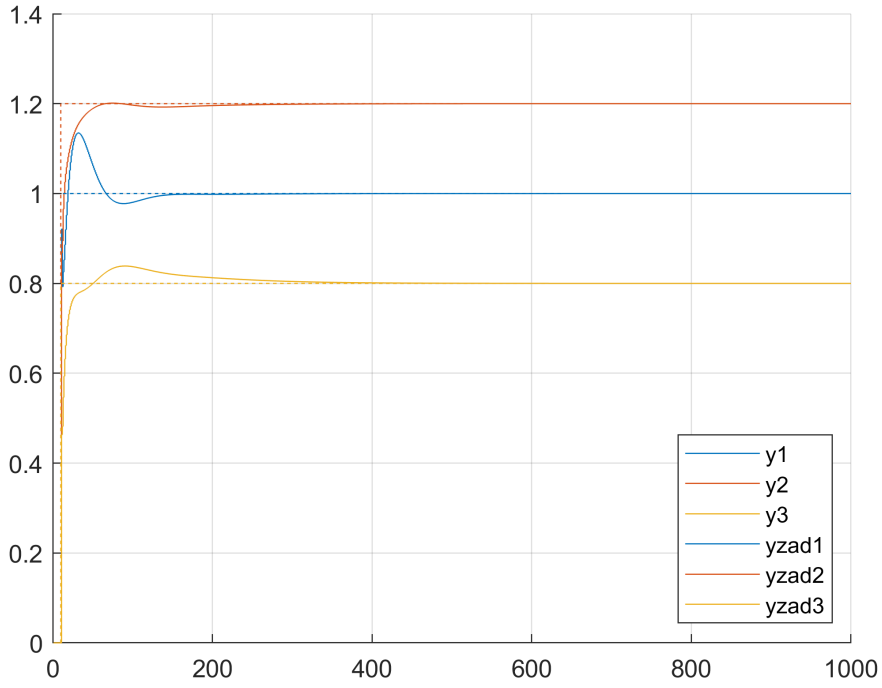
```
1   function [u1k, u2k, u3k, u4k, Y0k] =
2       dmc_mimo(k, U, Y, Yzad, D, N, ny, nu, K, Mp)
3
4       u1 = U(:, 1);
5       u2 = U(:, 2);
6       u3 = U(:, 3);
7       u4 = U(:, 4);
8       U_enum = {u1, u2, u3, u4};
9
10      yzad1 = Yzad(k,1);
11      yzad2 = Yzad(k,2);
12      yzad3 = Yzad(k,3);
13      Yzad_enum = {yzad1, yzad2, yzad3};
14
15      y1 = Y(k, 1);
16      y2 = Y(k, 2);
17      y3 = Y(k, 3);
18      Y_enum = {y1, y2, y3};
19
20      Y0k = [yzad1 - y1; yzad2 - y2; yzad3 - y3];
21
22      Ke = zeros(nu, ny);
23      for J=1:N
24          cur = (J-1)*ny+1;
25          Ke = Ke + K(1:nu, cur:cur+ny-1);
26      end
27
28      K1 = K(1:nu,:);
29      Ku = {D-1};
30      for i = 1:D-1
31          cur = (i-1)*nu+1;
32          Mpj = Mp(:, cur:cur+nu-1);
33          Ku{i} = K1 * Mpj;
34      end
35
```

```
36
37      elem = zeros(nu, 1);
38      for i = 1:D-1
39
40          if k-i <= 1
41              du = zeros(nu, 1);
42          else
43              du = [
44                   u1(k-i) - u1(k-i-1);
45                   u2(k-i) - u2(k-i-1);
46                   u3(k-i) - u3(k-i-1);
47                   u4(k-i) - u4(k-i-1)
48              ];
49          end
50
51          elem = elem + Ku{i}*du;
52      end
53
54
55      dUk = Ke * Y0k - elem;
56
57      u1k = u1(k-1) + dUk(1);
58      u2k = u2(k-1) + dUk(2);
59      u3k = u3(k-1) + dUk(3);
60      u4k = u4(k-1) + dUk(4);
61
62  end
```

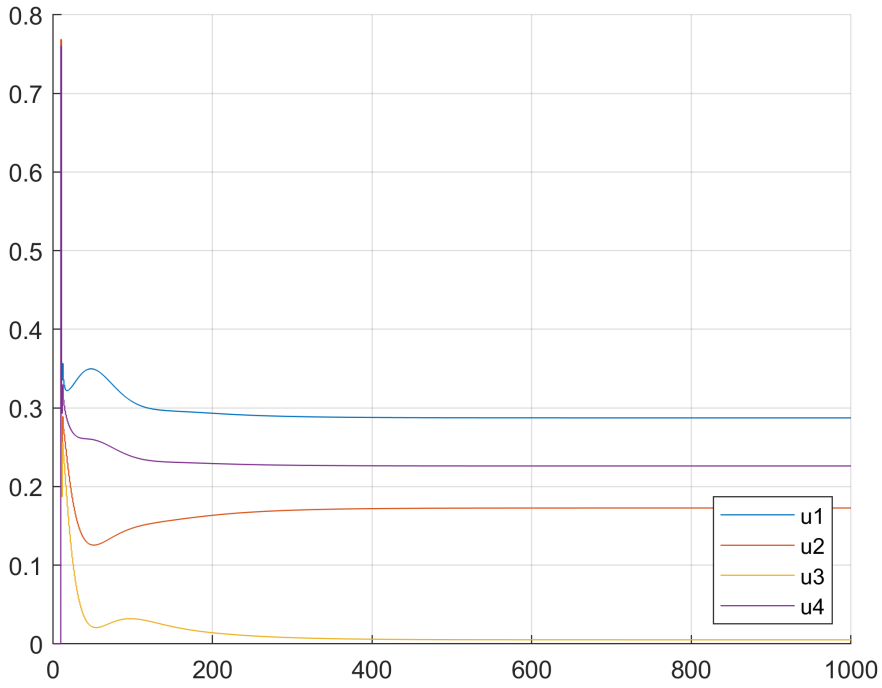
## 1.5. Dobór nastaw metodą eksperymentalną oraz w wyniku optymalizacji wskaźnika jakości

### 1.5.1. Regulator PID

Podczas doboru nastaw regulatora PID rozpoczęliśmy od ustawienia parametrów wzmocnień, a także wag, o których mowa była w punkcie 1.3. Jakość regulacji dla dobranych wartości wzmocnień oraz poziomu całki (redukcja uchybów ustalonych, które występowały dla regulatorów P). Najpierw przedstawione będą wykresy dla regulatorów działających relatywnie dobrze, lecz z wyzerowanym wpływem różniczki.




Rys. 4. Przebieg wyjść

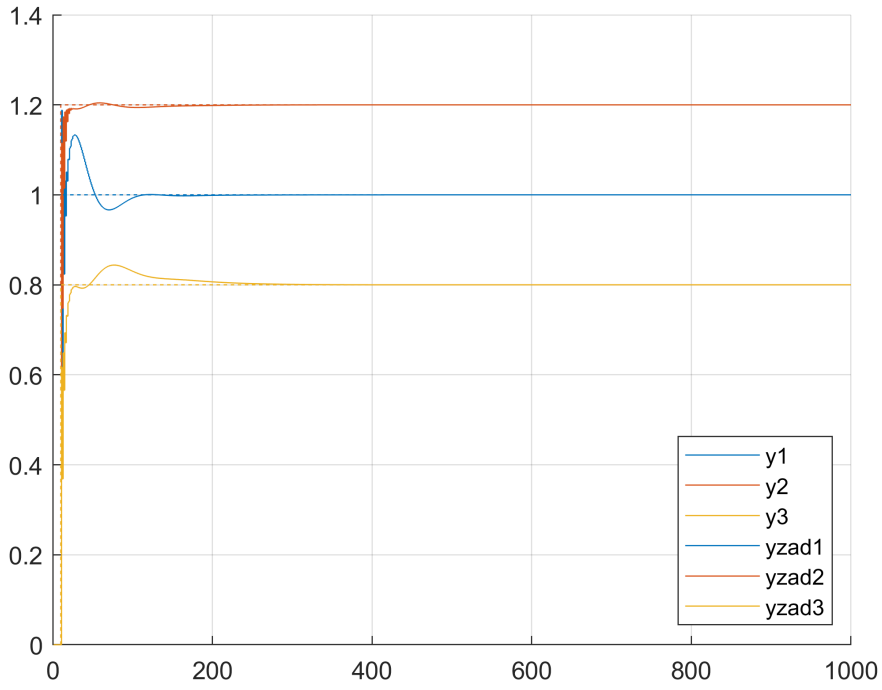



Rys. 5. Przebieg sterowań

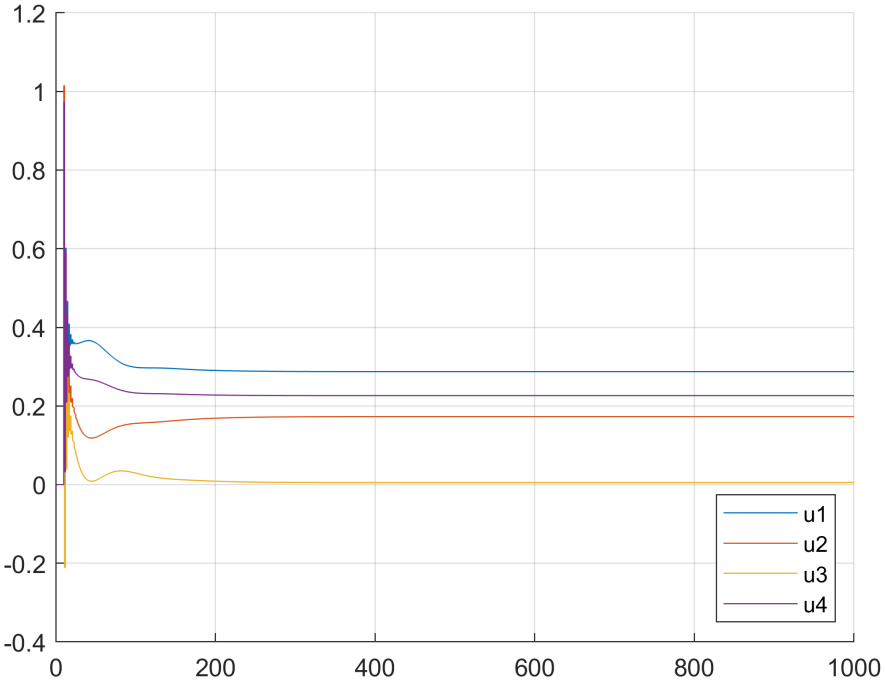
Dla powyższych przebiegów wskaźnik jakości wyniósł

$$E = 4,60893944331713$$

Następnie zdecydowaliśmy się dodać człon różniczkujący i udało uzyskać się lepszy wskaźnik jakości. Przebiegi dla optymalnych nastaw:




Rys. 6. Przebieg wyjść




Rys. 7. Przebieg sterowań

Dla powyższych przebiegów wskaźnik jakości wyniósł

$$E = 4,33412274443403$$

Mimo lepszej wartości wskaźnika jakości warto zaznaczyć, że dla pierwszych nastaw wykresy są bardziej gładkie przez co jakościowa ocena była sprzeczna z ilościową. Stąd zdecydowaliśmy się zamieścić obydwie wersje.

Optymalnymi nastawami dla kolejnych regulatorów okazały się:

$$K = 0{,}7, T_i = 4, T_d = 0{,}2$$

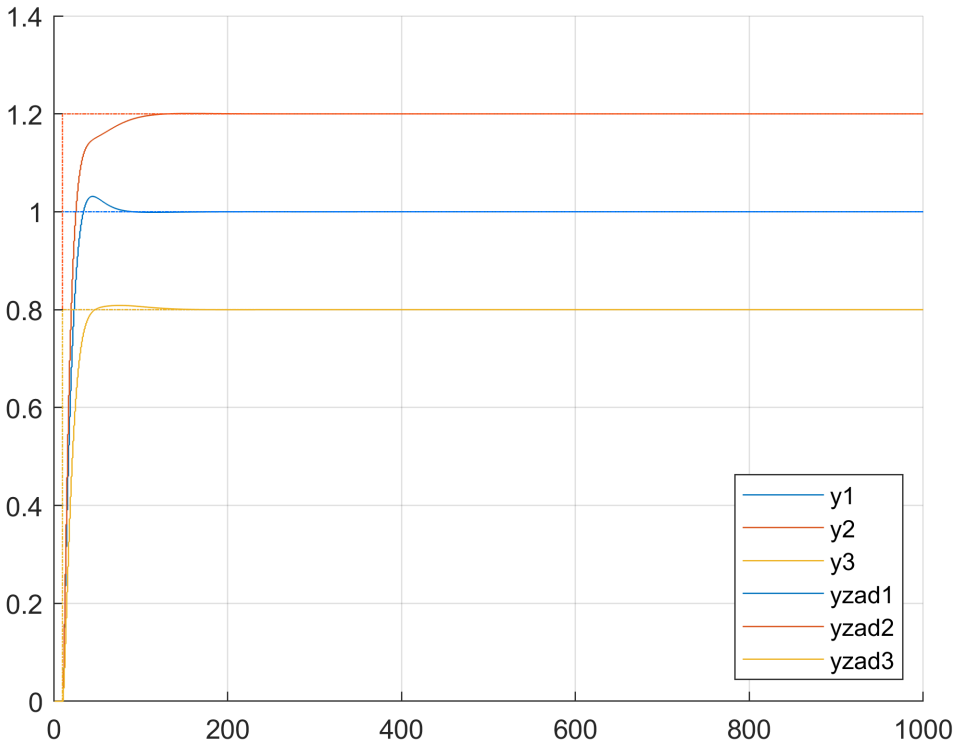
$$K = 0{,}7, T_i = 1{,}5, T_d = 0{,}07$$

$$K = 0{,}7, T_i = 3{,}5, T_d = 0{,}15$$

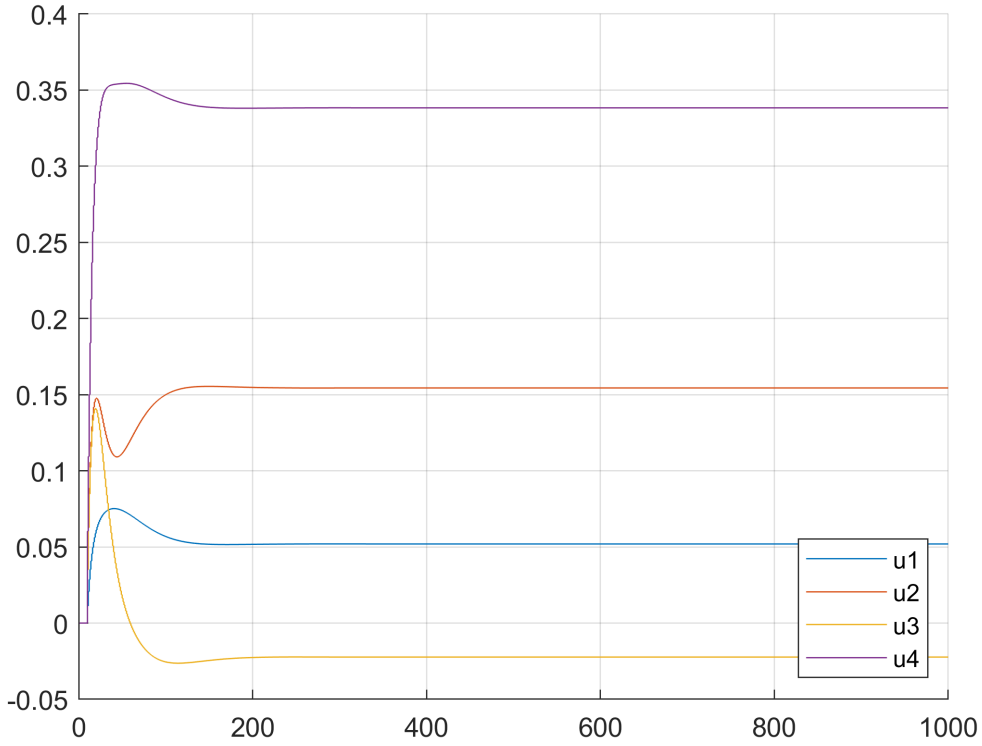
### 1.5.2. Regulator DMC

W wyniku doboru parametrów regulatora DMC poprzez optymalizację wskaźnika jakości uzyskaliśmy również regulację najlepszą jakościowo tj. przebiegi wyjść były gładkie oraz zmiany sterowań w kolejnych próbkach są relatywnie niskie. Dla wszystkich testów przyjęliśmy jednakowe wartości horyzontów, równe horyzontowi dynamiki:

$$D = N = N_u = 220$$




Rys. 8. Przebieg wyjść




Rys. 9. Przebieg sterowań

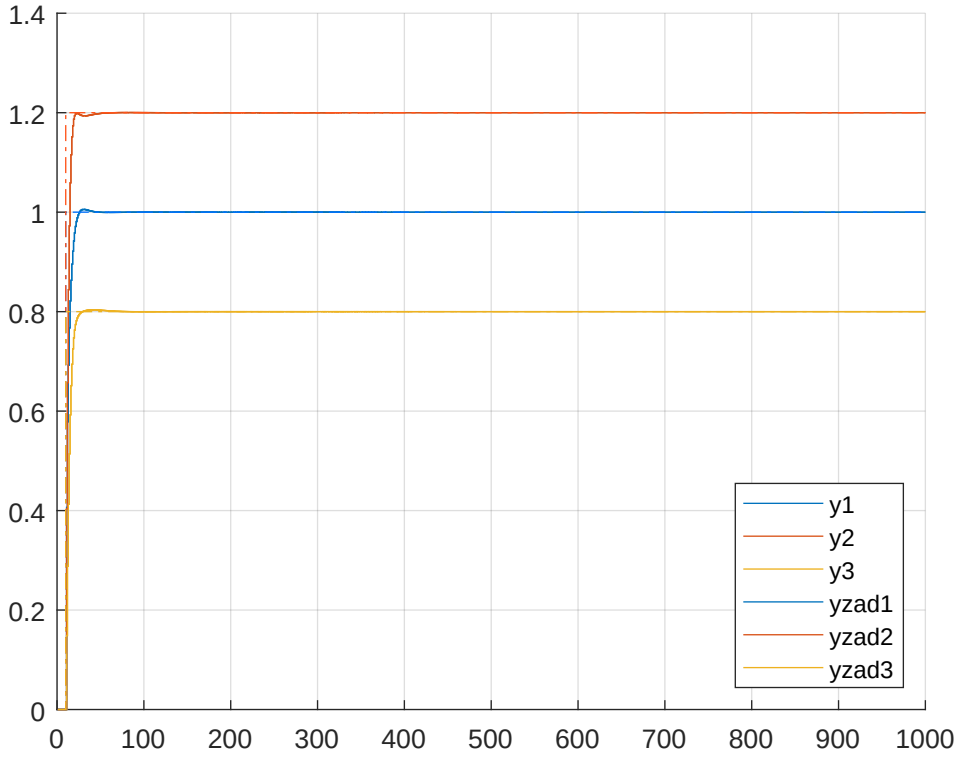
Dla powyższych przebiegów wskaźnik jakości wyniósł

$$E = 17,4270321349678$$

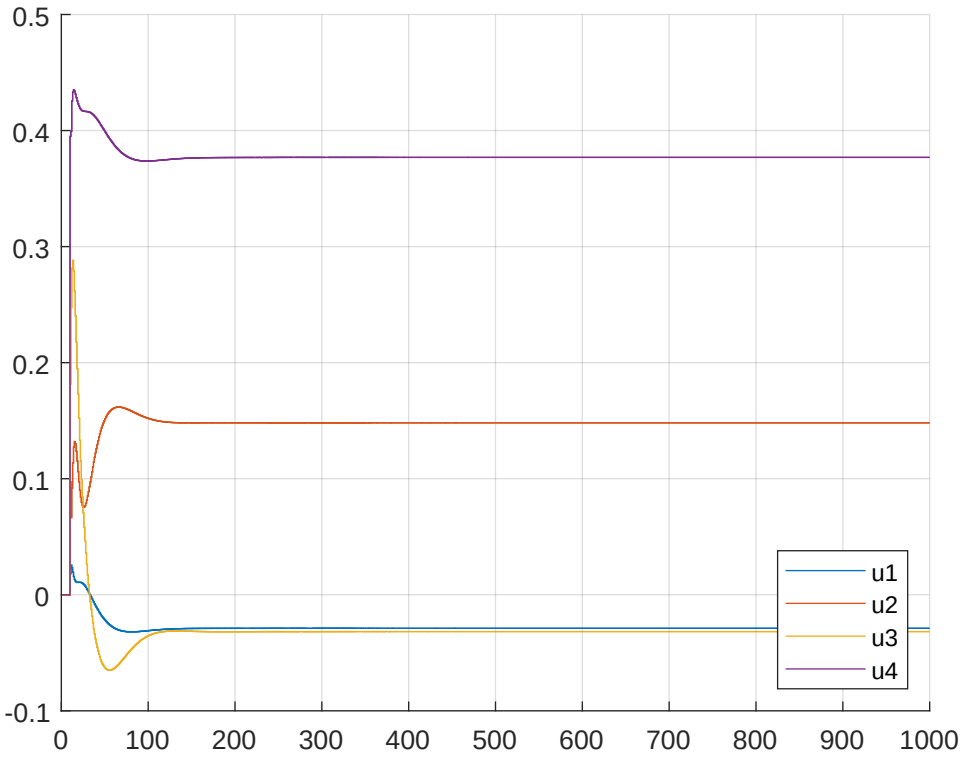
Taki wynik uzyskaliśmy dla poniższych wartości nastaw:

$$\lambda_1 = 2, \lambda_2 = 2, \lambda_3 = 2, \lambda_4 = 4$$
$$\mu_1 = 0{,}01, \mu_2 = 0{,}005, \mu_3 = 0{,}01$$

Udało nam się również uzyskać dużo szybsze przebiegi. Kosztem tego były jednak bardziej dynamiczne i poszarpane przebiegi sterowania. Zaowocowało to prawie trzykrotnym zmniejszeniem wskaźnika jakości błędu.




Rys. 10. Przebieg wyjść




Rys. 11. Przebieg sterowań

Dla powyższych przebiegów wskaźnik jakości wyniósł

$$E = 5{,}976\,427\,061\,029\,243$$

Taki wynik uzyskaliśmy dla poniższych wartości nastaw:

$$\lambda_1 = 4, \lambda_2 = 4, \lambda_3 = 4, \lambda_4 = 5$$

$$\mu_1 = 0{,}7, \mu_2 = 0{,}6, \mu_3 = 0{,}6$$

Powyższe przebiegi pokazują, że wyższe wartości współczynników $\mu$, skutkują szybszym ustaleniem się wartości wyjść. Równocześnie należało zwiększyć współczynniki $\lambda$, ponieważ poprzednie nastawy generowały oscylacje sygnałów sterujących.

## 1.6. Implementacja algorytmu DMC w wersji klasycznej

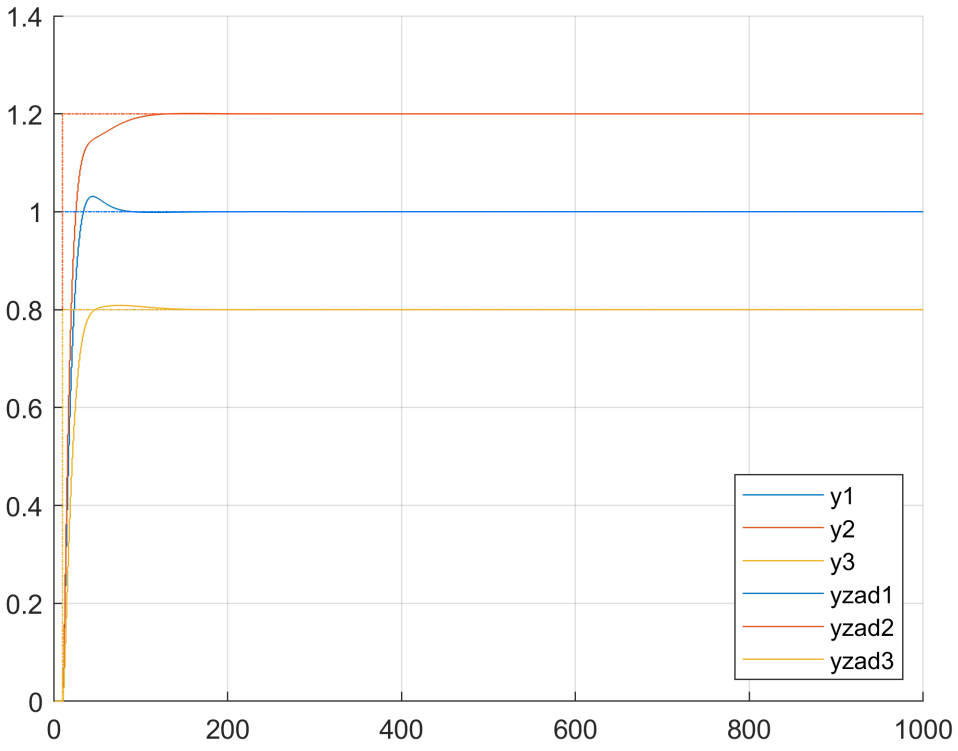
Implementacja algorytmu różni się jedynie w przypadku wyliczania następnych przyrostów sterowania w drugim pliku związanym z algorytmem. Pierwszy algorytm, tzw. offline, pozostaje bez zmian.

Listing kodu regulatora DMC w wersji klasycznej:

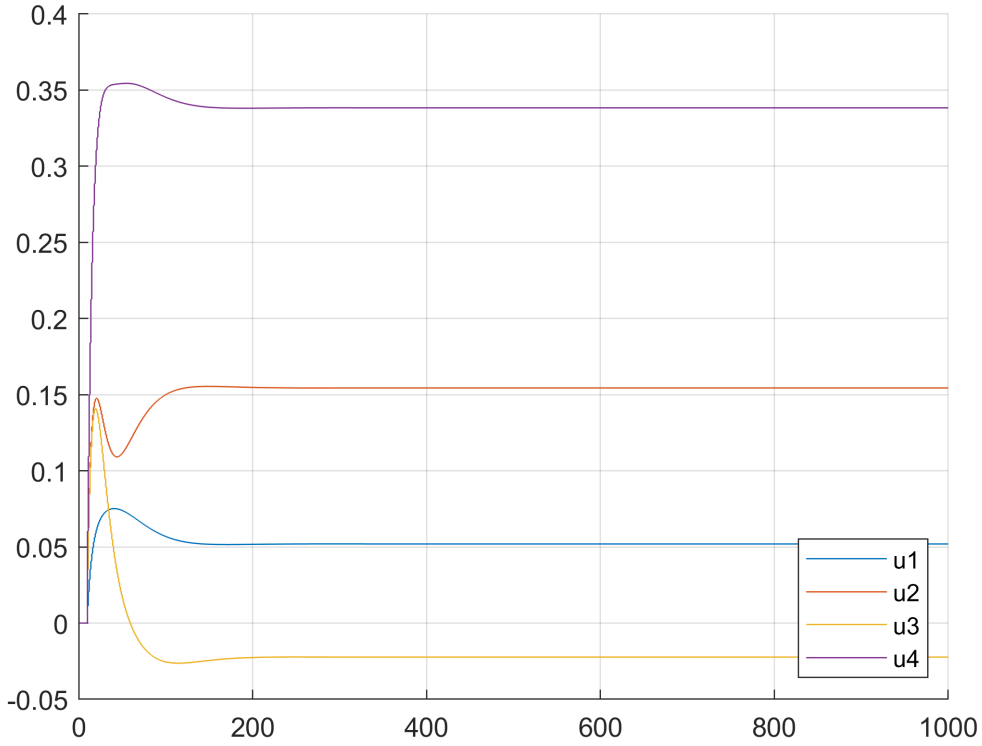
```
function [u1k, u2k, u3k, u4k, Y0k] =
    dmc_mimo(k, U, Y, Yzad, D, N, ny, nu, K, Mp)

    u1 = U(:, 1);
    u2 = U(:, 2);
    u3 = U(:, 3);
    u4 = U(:, 4);
    U_enum = {u1, u2, u3, u4};

    yzad1 = Yzad(k,1);
    yzad2 = Yzad(k,2);
    yzad3 = Yzad(k,3);
    Yzad_enum = {yzad1, yzad2, yzad3};

    y1 = Y(k, 1);
    y2 = Y(k, 2);
    y3 = Y(k, 3);
    Y_enum = {y1, y2, y3};


    Y0k = zeros(N*ny, 1);

    for J = 1:N
        for j = 1:ny
            cur = (J-1)*ny+j;
            Y0k(cur) = Yzad_enum{j} - Y_enum{j};
        end
    end


    DeltaUpk = zeros(nu*(D-1), 1);

    for J = 1:D-1
        for j = 1:nu

            if k-J-1 <= 1
```

```
                chwila = 2;
            else
                chwila = k-J;
            end

            cur = (J-1)*nu+j;
            DeltaUpk(cur) = U_enum{j}(chwila) - U_enum{j}(chwila-1);

        end
    end


    dUk = K * (Y0k - Mp*DeltaUpk);

    u1k = u1(k-1) + dUk(1);
    u2k = u2(k-1) + dUk(2);
    u3k = u3(k-1) + dUk(3);
    u4k = u4(k-1) + dUk(4);

end
```

Przebiegi wyjść i sterowań dla takich samych parametrów jak w pierwszym (łagodniejszym) przypadku punktu 1.5.2:




Rys. 12. Przebieg wyjść




Rys. 13. Przebieg sterowań

Dla powyższych przebiegów wskaźnik jakości wyniósł

$$E = 17,4270321349679$$

Zgodnie z przewidywaniami jakość regulacji jest taka sama. Wskaźnik jakości rozbieżny jest dopiero na trzynastym miejscu po przecinku. Wynika to bezpośrednio z błędów reprezentacji. Tryb klasyczny lub oszczędny wpływa jedynie na złożoność obliczeniową problemu, dlatego nie widzimy różnic w regulacji.

# 2. CZĘŚĆ LABORATORYJNA

## 2.1. Sprawdzenie możliwości sterowania i pomiaru w komunikacji ze stanowiskiem

W celu sprawdzenia możliwości sterowania i pomiaru w komunikacji ze stanowiskiem posłużyliśmy się gotowymi programami utworzonymi na potrzeby pracy ze stanowiskiem; są to matlabowy `socket_plc` oraz utworzonych na sterowniku PLC `Modbus` i `SocketComm`. Wartości zadane zadawaliśmy bezpośrednio w sterowniku programowalnym poprzez funkcjonalność *watch*.

Punkt pracy:

$$G1 = 27$$

$$G2 = 32$$

$$W1 = W3 = 50$$

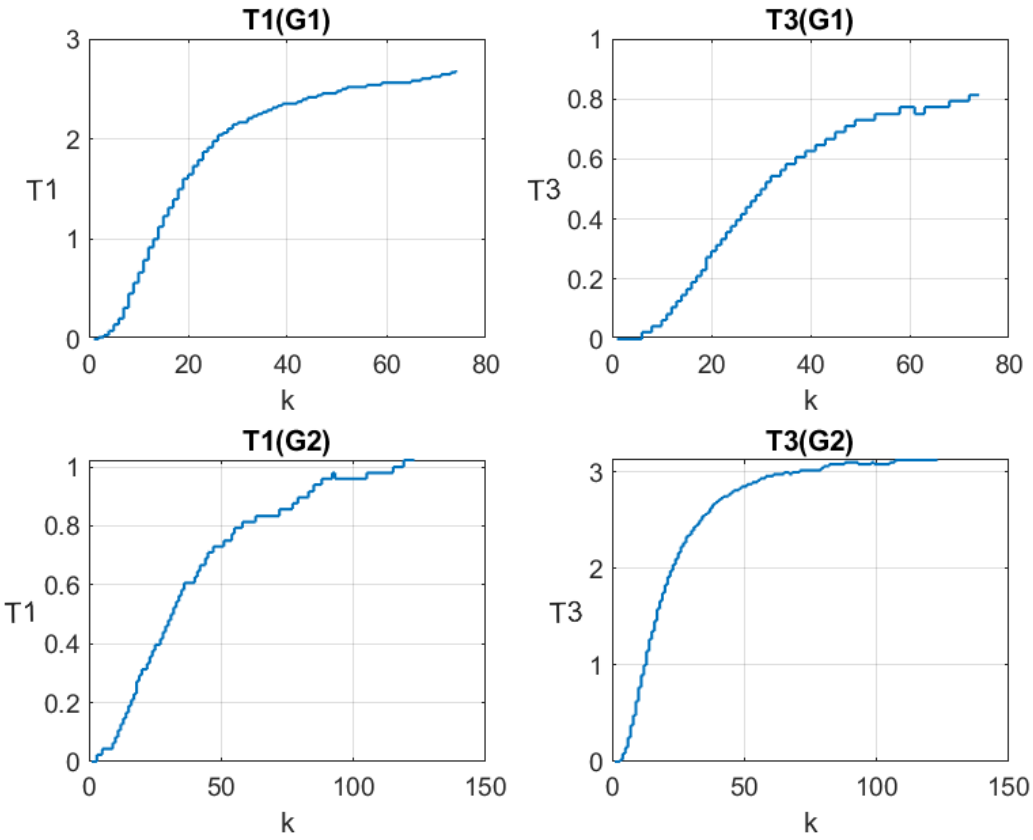
Wartości temperatur w punkcie pracy:

$$T1 = 34{,}2$$

$$T3 = 35{,}7$$

### 2.1.1. Wpływ sygnałów sterujących na wyjścia dla obiektu MIMO

W obiektach z wieloma wejściami i wieloma wyjściami można spodziewać się, że sygnał sterujący będzie wpływał na wiele wyjść. W celu ocenienia wpływu grzałki G1 i G2 na odczyt temperatury T1 i T3 zmierzono odpowiedzi skokowe dla zmiany pojedynczego sygnału sterującego z 100 na 300.
Poniżej przedstawiono odpowiedzi skokowe przeliczone na wartości $s_1$ oraz $s_2$ wykorzystywane w algorytmie DMC - a więc odpowiedź skokową dla zmiany sterowania z 0 na 1.




Rys. 14. Odpowiedzi skokowe obiektu dla obu sterowań

Jak widać na rysunku 14 grzałka G1 ma znacznie większy wpłw na pomiar temperatury T1 oraz G2 silniej wpływa na T3. W związku z tym, nie ma potrzeby, aby algorytm PID był rozmywany, wystarczy jedynie dopasować odpowiednie wejście sterujące do odpowiedniego wyjścia. Stąd, zaimplementowano regulatory PID1 i PID2 o wejściu i wyjściu odpowiednio G1-T1, G2-T3.

## 2.2. Implementacja mechanizmu zabezpieczającego przed przegrzaniem

```
IF (T1 > 25000) THEN
    GRZALA1 := 0;
END_IF;
IF (T3 > 25000) THEN
    GRZALA2 := 0;
END_IF;
```

Kiedy jeden z czujników temperatury wykryje temperaturę powyżej 250°C wyłącza grzałkę, przy której jest ten czujnik.

## 2.3. Implementacja regulatora PID na sterowniku

Implementacja regulatora PID dwupętlowego sprowadza się, do zaimplementowania dwóch niezależnych regulatorów.

W algorytmie PID równanie różnicowe regulatora ma następującą postać:

$$u(k) = r_2 e(k-2) + r_1 e(k-1) + r_0 e(k) + u(k-1)$$

gdzie:

$$r_0 = K\left(1 + \frac{T}{2T_i} + \frac{T_d}{T}\right)$$

$$r_1 = K\left(\frac{T}{2T_i} - 2\frac{T_d}{T} - 1\right)$$

$$r_2 = \frac{KT_d}{T}$$

$$e(k) = y_{zad}(k) - y(k)$$

Każdy regulator w systemie dwupętlowym skupia się na regulacji jednej zmiennej procesowej. W naszym przypadku regulowanymi niezależnymi zmiennymi są T1 i T3.

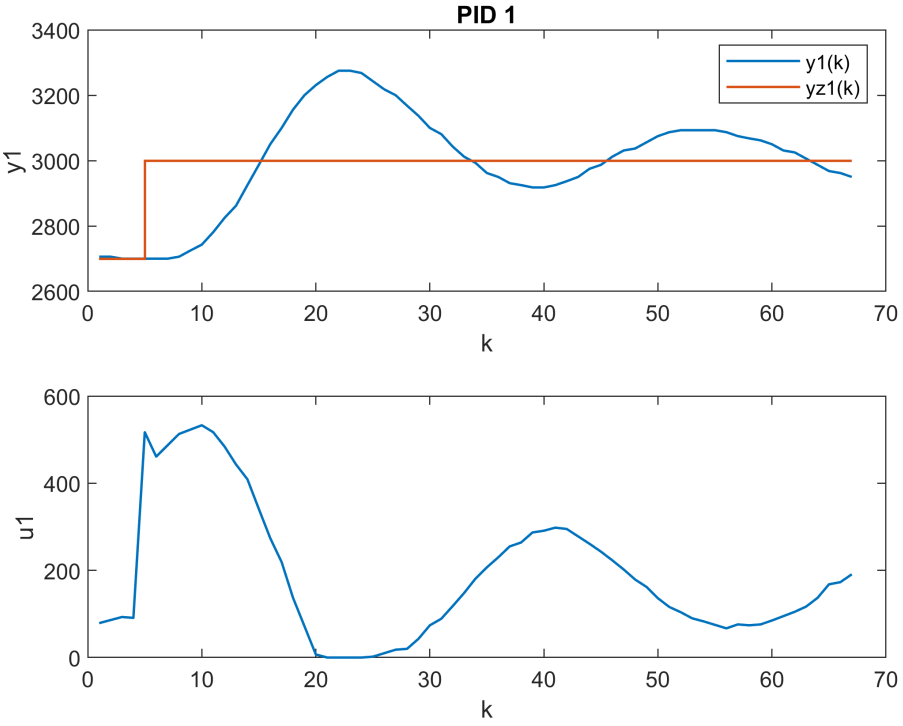
Implementacja:

```
...
IF trybPID THEN
    K_pid1:= 0.42;
    Ti1 := 50;
    Td1 := 0.8;

    K_pid2 := 0.42;
    Ti2 := 50;
    Td2 := 0.8;

    r0_1 := K_pid1*(1+Tprob/(2*Ti1)+Td1/Tprob);
    r1_1 := K_pid1*(Tprob/(2*Ti1)-2*Td1/Tprob-1);
    r2_1 := K_pid1*Td1/Tprob;

    r0_2 := K_pid2*(1+Tprob/(2*Ti2)+Td2/Tprob);
    r1_2 := K_pid2*(Tprob/(2*Ti2)-2*Td2/Tprob-1);
    r2_2 := K_pid2*Td2/Tprob;

    y1_k := TEMP1;
    y2_k := TEMP3;

    Err1_k := T1_ZAD - y1_k;
    Err2_k := T3_ZAD - y2_k;

    u1_k := r2_1*Err1_k_2 + r1_1*Err1_k_1 + r0_1*Err1_k + u1_k_1;
    u2_k := r2_2*Err2_k_2 + r1_2*Err2_k_1 + r0_2*Err2_k + u2_k_1;

    // uwzglednienie ograniczen
    IF u1_k > 1000 THEN
```

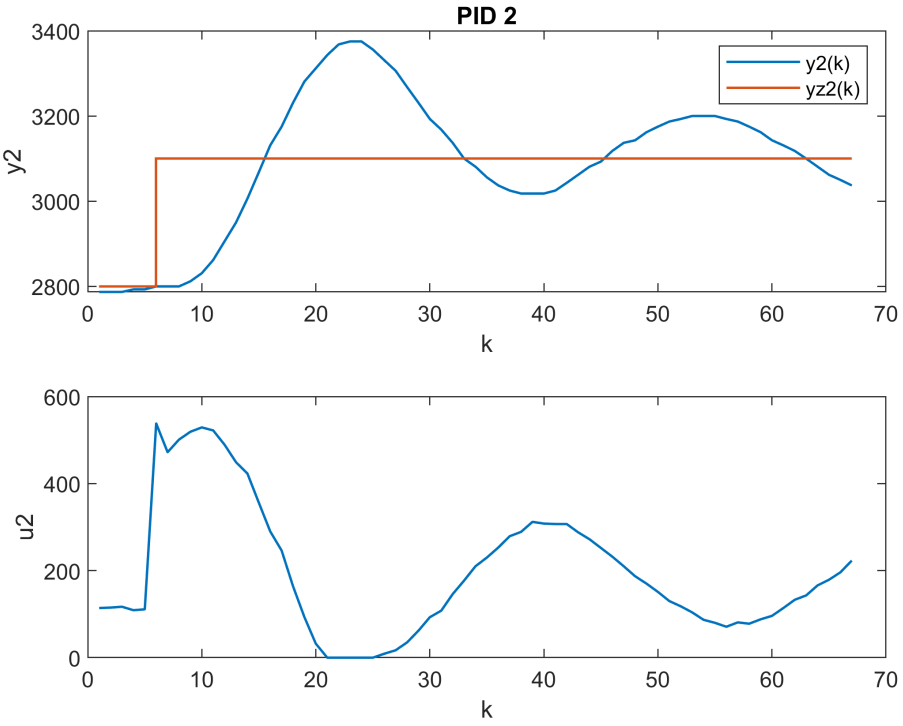
```
30          u1_k := 1000;
31      ELSIF u1_k < 0 THEN
32          u1_k := 0;
33      END_IF;
34
35      IF u2_k > 1000 THEN
36          u2_k := 1000;
37      ELSIF u2_k < 0 THEN
38          u2_k := 0;
39      END_IF;
40
41      GRZALA1 := REAL_TO_INT(u1_k);
42      GRZALA2 := REAL_TO_INT(u2_k);
43
44      y1_k_1 := y1_k;
45      u1_k_1 := u1_k;
46      y2_k_1 := y2_k;
47      u2_k_1 := u2_k;
48
49      Err1_k_2 := Err1_k_1;
50      Err2_k_2 := Err2_k_1;
51
52      Err1_k_1 := Err1_k;
53      Err2_k_1 := Err2_k;
54  ...
55
56  END_IF;
```

Strojenie regulatora PID dwupętlowego zostało przeprowadzone metodą prób i błędów. Na początku, postanowiliśmy ustalić wartość wzmocnienia. Tym samym, wyłączyliśmy człon całkujący i różniczkujący, a modyfikowaliśmy jedynie wartość wzmocnienia.

$K = 1{,}1$, $T_i = 1\,000\,000$, $T_d = 0$




Rys. 15.



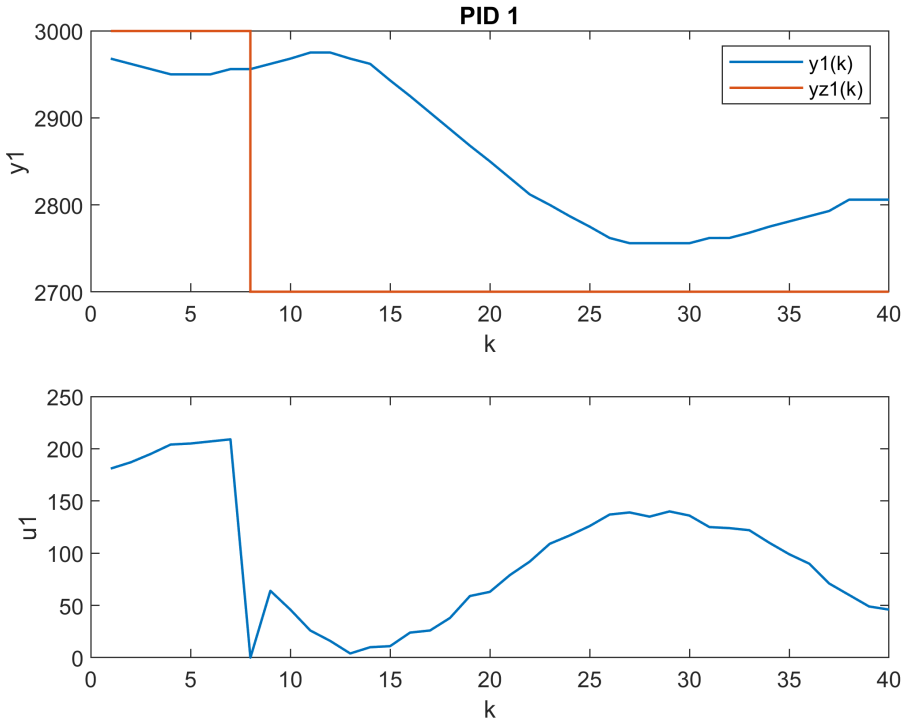

Rys. 16.

Jak można zauważyć, przy tej wartości wzmocnienia występują gasnące oscylacje. Z tego względu, postanowiliśmy obniżyć wartość wzmocnienia do takiej, aby te oscylacje nie występowały. Finalnie wartość wzmocnienia osiągnęła wartość $K = 0{,}42$.

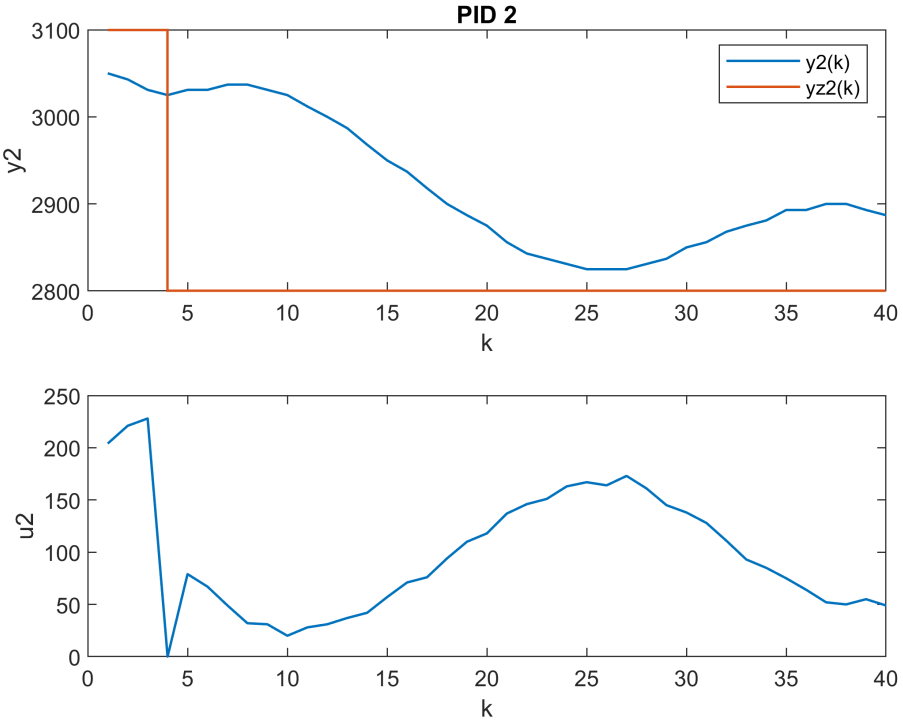
Następnie w regulatorze dodaliśmy człon całkujący, w celu eliminacji błędów ustalonych, a tym samym zwiększeniu dokładności regulacji. Na koniec, dobraliśmy parametr $T_d$ członu różniczkującego, w celu zapobiegnięciu nadmiernym oscylacjom i nadmiernemu przeregulowaniu systemu.

Niestety, nie wszystkie kroki doboru nastaw, zostały udokumentowane. Jednakże, ocalał jeden plik formatu **mat** jako przykład błędnych nastaw.

$K = 1{,}1$, $T_i = 100$, $T_d = 0$



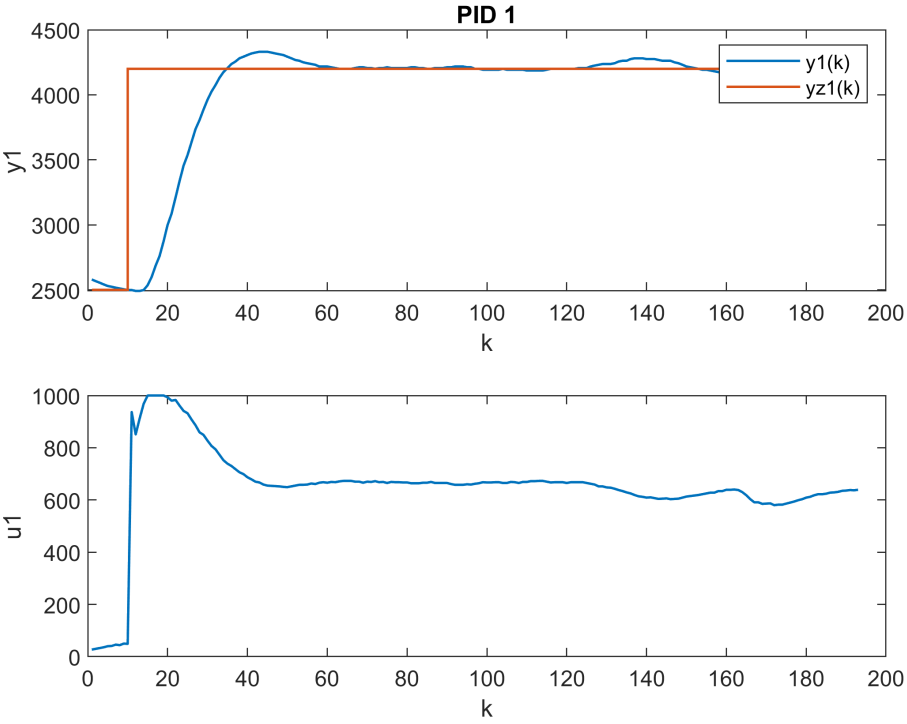

Rys. 17.



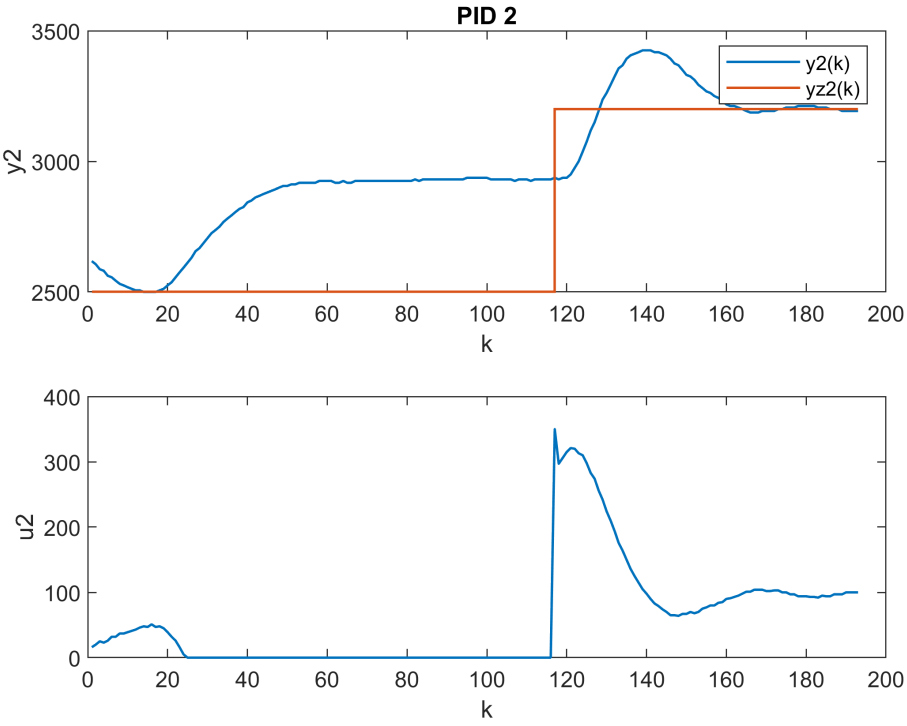

Rys. 18.

Ostatecznie zdecydowaliśmy się na następujące nastawy: $K = 0{,}42$, $T_i = 50$, $T_d = 0{,}8$. Jak można zauważyć na rysunkach 19-20 proces dobrze sobie radzi z dochodzeniem do wartości zadanej w dość sprawnym tempie.

Na wykresie 20 można zaobserwować rozbieżność między wartością wyjściową, a wartością zadaną dla $k \approx 20, ..., 120$. Wynika to z charakterystyki statycznej obiektu. W chwili $k \approx 20$, grzałce 1 została zadana wartość 4000. Nagrzanie się grzałki 1 spowodowało również wzrost temperatury w grzałce 2. Ograniczenia nałożone na wartość sygnału sterującego, uniemożliwiły obniżenie wartości sterowania do wartości $< 0$. Charakterystyka obiektu nie pozwala na tak duże rozbieżności między grzałkami - dopóki zadane wartości sterowania są wystarczająco bliskie, to oboma grzałkami można sterować stosunkowo niezależnie, wpływ jednej na drugą kompensuje się. W przypaku tak dużych rozbieżności w zadanych temepraturach (jak na wykresie 20), które są bliskie, tym które uzyskuje się dla zerowego sterowania, wpływ grzałek na siebie nie pozwala osiągnąć takich wartości wyjściowych. Po zwiększeniu wartości zadanej grzałki 2 w chwili $k \approx 120$, rozbieżności przestały występować, a sygnał wyjściowy odpowiednio zaczął osiągać wartością zadaną.




Rys. 19.




Rys. 20.

## 2.4. Implementacja regulatora DMC 2x2 w wersji analitycznej na sterowniku

Implementacja regulatora DMC 2x2 sprowadza się do implementacji dwóch niezależnych regulatorów, z których każdy jest odpowiedzialny za regulację jednej z dwóch zmiennych procesowych.

W najprostszej (oszczędnej obliczeniowo) wersji algorytmu DMC przyrost sygnału sterującego w chwili $k$ można zapisać jako:

$$\Delta u(k) = \Delta u(k|k) = K_e \cdot e(k) - \sum_{j=1}^{D-1} k_j^u \cdot \Delta u(k-j)$$

gdzie

$$K_e = \sum_{i=1}^{N} K_{1,i}$$

$$k_j^u = \bar{K_1} \cdot M_j^P$$

dla $j = 1, ..., D$, przy czym: $\bar{K_1}$ - wiersz 1 macierzy $K$, $M_j^P$ - kolumna j macierzy $M^P$.

Implementacja:

W środowisku MATLAB zostały przeprowadzone obliczenia off-line algorytmu, a następnie wartości poszczególnych parametrów zostały przekazane do implementacji DMC na sterowniku w języku ST.

MATLAB:

```
% *********** ODPOWIEDŹ SKOKOWA ***********
D = 85;
len = D;
start = 5;
nullS = zeros(2, 2);
S = cell(len, 1);

ws1 = load('odp_skokowa_g1_300.mat');
ws2 = load('odp_skokowa_g2_300.mat');

for k=start:D+start
    p = k-start+1;
    y1o1 = (ws1.y1(k) - ws1.y1(start))/300;
    y2o1 = (ws1.y2(k) - ws1.y2(start))/300;

    y1o2 = (ws2.y1(k) - ws2.y1(start))/300;
    y2o2 = (ws2.y2(k) - ws2.y2(start))/300;

    S{p} = [
        y1o1,   y1o2;
        y2o1,   y2o2;
    ];
end

for k=1:len-start+1
    p = k;
    S11(p) = S{k}(1, 1);
```

```
    S21(p) = S{k}(2, 1);

    S12(p) = S{k}(1, 2);
    S22(p) = S{k}(2, 2);
end

N = 50;
Nu = 10;
nu = 2; ny =2;
Mi = 1;
Lambda = 1;
```

```
% ****************** OBLICZENIA OFF-LINE ******************
[K, Mp] = dmc_mimo_offline(S, D, N, ny, Nu, nu, Mi, Lambda);

% oszczędna wersja DMC - parametry
Ke = 0;
for i = 1 : N
    Ke = Ke + K(1, i);
end
ku = K(1, :)*Mp


Ke = zeros(nu, ny);
for J=1:N
    cur = (J-1)*ny+1;
    Ke = Ke + K(1:nu, cur:cur+ny-1);
end


K1 = K(1:nu,:);
Ku = {D-1};
for i = 1:D-1
    cur = (i-1)*nu+1;
    Mpj = Mp(:, cur:cur+nu-1);
    Ku{i} = K1 * Mpj;
end


for i =1:length(ku)
    fprintf('Ku[%d] := %f;\n',i-1,ku(i));
end
for i=1:75
    fprintf('dU1[%d] := 0.0;\n',i-1);
    fprintf('dU2[%d] := 0.0;\n',i-1);
end
...
```

```
function [K, Mp] = dmc_mimo_offline(S, D, N, ny, Nu, nu, Mi, ...
    Lambda)

```

```
    nullS = zeros(ny, nu);

    m_rows = ny * N;            % size(S{1}, 1) * N
    m_cols = nu * Nu;           % size(S{1}, 2) * (N-Nu+1)
    M = zeros(m_rows, m_cols);  % N x Nu

    for c = 1:nu:m_cols % kolumny
        C = (c-1)/nu + 1;

        for r = 1:ny:m_rows % wiersze
            R = (r-1)/ny + 1;

            if (R - C + 1) <= 0
                M(r:r+ny-1, c:c+nu-1) = nullS;

            elseif (R - C + 1) > D
                M(r:r+ny-1, c:c+nu-1) = S{D};

            else
                M(r:r+ny-1, c:c+nu-1) = S{R-C+1};

            end
        end
    end

    mp_rows = ny * N;               % size(S{1}, 1) * N
    mp_cols = nu * (D-1);           % size(S{1}, 2) * (D-1)
    Mp = zeros(mp_rows, mp_cols);   % N x (D-1)

    for c = 1:nu:mp_cols % kolumny
        C = (c-1)/nu + 1;

        for r = 1:ny:mp_rows % wiersze
            R = (r-1)/ny + 1;

            if R + C > D
                Scr = S{D};
            else
                Scr = S{C+R};
            end

            if C > D
                Sc = S{D};
            else
                Sc = S{C};
            end

            Scur = Scr - Sc;
            Mp(r:r+ny-1, c:c+nu-1) = Scur;
        end
    end
```

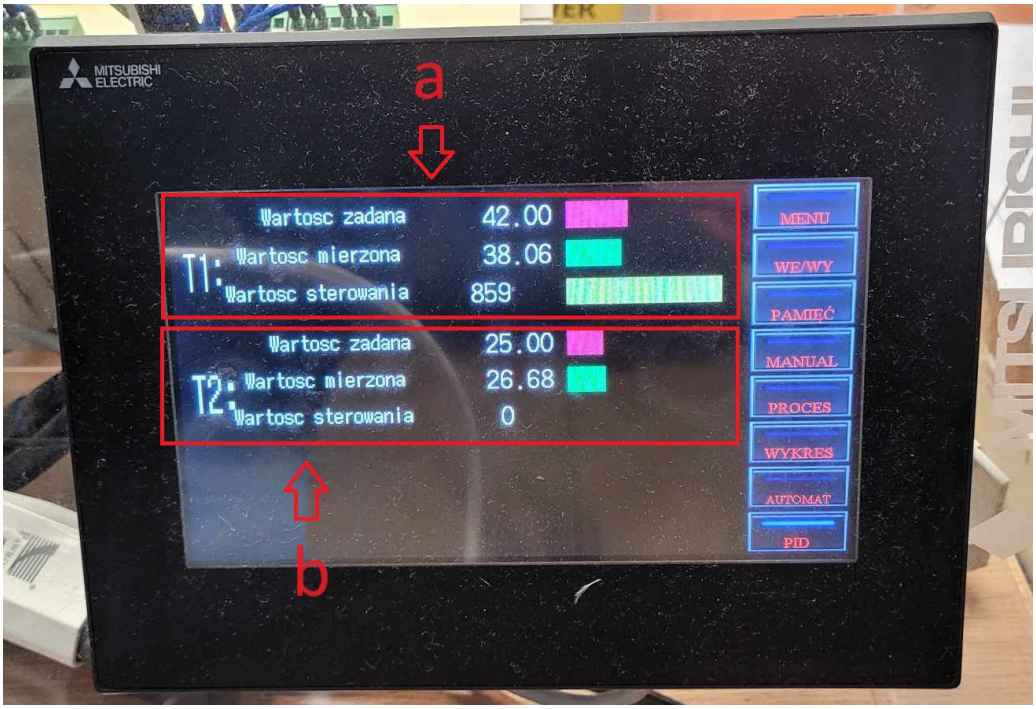
```
55
56
57      PSI = eye(N*ny);
58      Mi_enum = Mi;
59
60      for J = 1:N
61          for j = 1:ny
62              diag = (J-1)*ny+j;
63              PSI(diag, diag) = Mi_enum;%{j};
64          end
65      end
66
67      LAMBDA = eye(Nu*nu);
68      Lambda_enum = Lambda;
69
70      for J = 1:Nu
71          for j = 1:nu
72              diag = (J-1)*nu+j;
73              LAMBDA(diag, diag) = Lambda_enum;%{j};
74          end
75      end
76
77      K = ((M' * PSI * M + LAMBDA)^(-1)) * M' * PSI;
78  end
```

ST:

```
1   ELSIF trybDMC THEN
2       Ku[0] := 0.470232;
3       Ku[1] := 0.018026;
4       ...
5       dU1[0] := 0.0;
6       dU2[0] := 0.0;
7       dU1[1] := 0.0;
8       dU2[1] := 0.0;
9       ...
10      Ke[0] := 0.782273;
11      Ke[1] := -0.063985;
12      Ke[2] := 0.044315;
13      Ke[3] := 0.774380;
14
15      T1o := TEMP1/100.0;
16      T3o := TEMP3/100.0;
17
18      U_1 := GRZALA1;
19      U_2 := GRZALA2;
20
21      Ke1_tp := Ke[0]*(T1_ZAD/100.0 - T1o) + Ke[1]*(T3_ZAD - T3o);
22      Ke2_tp := Ke[2]*(T1_ZAD/100.0 - T1o) + Ke[3]*(T3_ZAD - T3o);
23
24      Ku_tmp[0] := 0;
25      Ku_tmp[1] := 0;
```

```
26
27      FOR licznik := 0 TO 74 BY 1 DO
28          Ku_tmp[0] := Ku_tmp[0] + Ku[licznik]*dU1[licznik];
29          Ku_tmp[1] := Ku_tmp[1] + Ku[licznik+75]*dU2[licznik];
30      END_FOR;
31
32
33      FOR licznik := 74 TO 1 BY -1 DO
34          dU1[licznik] := dU1[licznik-1];
35          dU2[licznik] := dU2[licznik-1];
36      END_FOR;
37
38      dU1[0] := Ke1_tp - Ku_tmp[0];
39      dU2[0] := Ke2_tp - Ku_tmp[0];
40
41      U_1 := (U_1 + dU1[0]);
42      U_2 := (U_2 + dU2[0]);
43
44
45      //ograniczenia
46      IF(T1o > 15000) THEN
47          GRZALA1 := 0;
48          U_1 := 0;
49      END_IF;
50
51      IF (U_1 > 1000) THEN
52          GRZALA2 := 1000;
53          U_2 := 1000;
54      END_IF;
55
56      IF (U_1 < 0) THEN
57          GRZALA1 := 0;
58          U_1 := 0;
59      END_IF;
60
61      IF (U_2 < 0) THEN
62          GRZALA2 := 0;
63          U_2 := 0;
64      END_IF;
65
66      GRZALA1 := REAL_TO_INT(U_1*100.0);
67      GRZALA2 := REAL_TO_INT(U_2*100.0);
68
69  END_IF;
```

## 2.5. Panel operatora




Rys. 21. Panel operatorski: a - wartość mierzona, zadana oraz sterowania T1; b - wartość mierzona, zadana oraz sterowania T2.

W projektowaniu panelu operatora staraliśmy się, aby był on prosty, czytelny i intuicyjny w odbiorze.

Na rysunku 21 możemy zauważyć zaznaczone na czerwono dwa obszary (**a** i **b**). W obszarze **a** znajdują się kolejno wartość zadana, mierzona oraz sterowania T1. Analogicznie do obszaru **a**, w obszarze **b** znajdują się informacje o tych samych wartościach tylko, że dla T3 (oznaczenie T2 zostało zmienione na T3, ale niestety nie zostało to udokumentowane :(). Obok każdej zmiennej wyświetlana jest ich liczbowa reprezentacja oraz reprezentacja w postaci słupka. Są one w zakresie od 0 do 100 dla wartości temperaturowych, czyli dla *wartości zadanej* oraz *wartości mierzonej*, a sterowanie jest w zakresie 0-1000. Jak można również zauważyć, słupki opisujące te same wartości są w tym samym kolorze - zwiększa to estetyczność i czytelność panelu.

## 2.6. Implementacja automatu stanów

Na poniższym listingu przedstawiony jest automat stanów, w którym w zależności od aktualnego stanu zmieniana jest wartość zadana T1 i T3.

```
1 TIM_MAIN(PT := T#100s);
2 TIM_MAIN.IN := 1;
3 stan = Stan_MAIN;
4
5 CASE Stan_MAIN OF
6     0:
7     IF TIM_MAIN.Q THEN
8         T1_ZAD := 4400;
```

```
9           T3_ZAD := 4000;
10          TIM_MAIN.IN := 0;
11          Stan_MAIN := 1;
12      END_IF;
13
14      1:
15      TIM_MAIN.IN := 1;
16      Stan_MAIN := 2;
17
18      2:
19      IF TIM_MAIN.Q THEN
20          T1_ZAD := 2900;
21          T3_ZAD := 3200;
22          TIM_MAIN.IN := 0;
23          Stan_MAIN := 3;
24      END_IF;
25
26      3:
27      TIM_MAIN.IN := 1;
28      Stan_MAIN := 4;
29
30      4:
31      IF TIM_MAIN.Q THEN
32          T1_ZAD := 3400;
33          T3_ZAD := 3650;
34          TIM_MAIN.IN := 0;
35          Stan_MAIN := 5;
36      END_IF;
37
38      5:
39      TIM_MAIN.IN := 1;
40      Stan_MAIN := 0;
41
42  END_CASE;
```

Na samym początku programu zostaje zdefiniowany czas, który ma zliczać timer (100 s), a następnie zostaje on w 2 linii kodu aktywowany. W bloku CASE znajduje się definicja poszczególnych stanów automatu.

Stan 0:
Jeżeli timer odliczy czas zdefiniowany przez PT (`TIM_MAIN.Q` == 1), zadawana jest nowa wartość zadana `T1_ZAD` oraz `T3_ZAD` odpowiednio na wartości 4400 oraz 4000. Oprócz tego, zostaje wyłączone zliczanie czasu przez timer, a aktualnym stanem staje się 1.

Stan 1:
Aktywacja timera poprzez ustawienie wartości `TIM_MAIN.IN` na 1. Timer rozpoczyna zliczanie czasu. Przejście do stanu 2.

Stan 2:
Jeżeli timer odliczy czas zdefiniowany przez PT, zadawana jest nowa wartość zadana `T1_ZAD` oraz `T3_ZAD` odpowiednio na wartości 2900 oraz 3200. Oprócz tego, zostaje wyłączone zliczanie

czasu przez timer, a aktualnym stanem staje się 3.

Stan 3:
Aktywacja timera poprzez ustawienie wartości `TIM_MAIN.IN` na 1. Timer rozpoczyna zliczanie czasu. Przejście do stanu 4.

Stan 4:
Jeżeli timer odliczy czas zdefiniowany przez PT, zadawana jest nowa wartość zadana `T1_ZAD` oraz `T3_ZAD` odpowiednio na wartości 3400 oraz 3650. Oprócz tego, zostaje wyłączone zliczanie czasu przez timer, a aktualnym stanem staje się 5.

Stan 5:
Aktywacja timera poprzez ustawienie wartości `TIM_MAIN.IN` na 1. Timer rozpoczyna zliczanie czasu. Przejście do stanu 0.

# 3. Stanowisko Inteco

Tej części zadań laboratoryjnych nie wykonaliśmy.