Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Sztuczna Inteligencja w Automatyce

Sprawozdanie z projektu 1
Polecenie 5
Zadanie 3, 4
Dokumentacja

Michał Kwarciński, Bartosz Gałecki

Warszawa, 2022

Spis treści

1. Zadanie 3

1.1. Numeryczny algorytm regulacji typu SL z ograniczeniami

Napisaliśmy algorytm SL z ograniczeniami. Różni się on do rozmytego DMC: sposobem liczenia macierzy M i M_p (w SL w odróżnieniu od podejścia PDC nie urzywamy bezpośrednio modeli lokalnych, tylko liczymy rozmytą odpowiedź skokową i na jej podstawie budujemy macierze M i M_p). Kolejną różnicą jest też liczenie sterowania, w rozmytym DMC liczyliśmy je analitycznie, poprzez operacje na macierzach, w SL liczymy numerycznie, używamy *quadprog*, która minimalizuje funkcję celu względem parametru D_u.

$$cel = (Y_{zad} - Y_0 - M \cdot D_u)^T \cdot (Y_{zad} - Y_0 - M \cdot D_u) + \lambda \cdot D_u^T \cdot D_u \tag{1.1}$$

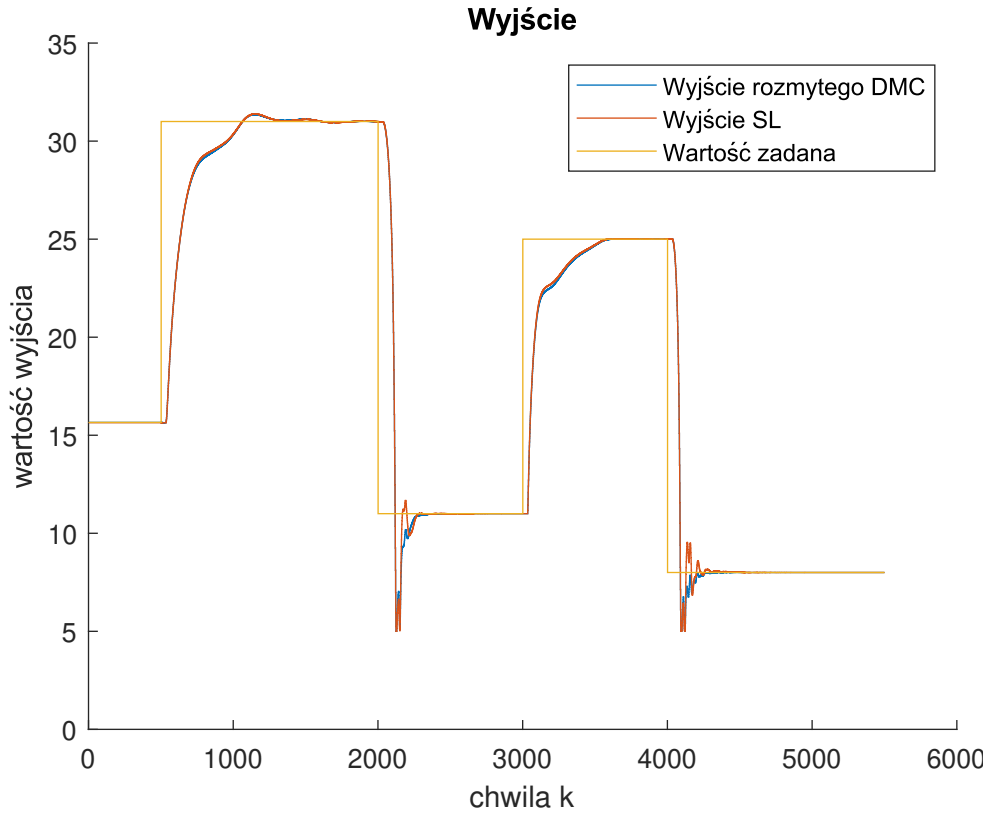
By skorzystać z funkcji *quadprog* trzeba obliczyć macierz H i wektor f, gdzie

$$H = 2 \cdot M^T \cdot M + 2 \cdot \lambda \cdot I \tag{1.2}$$

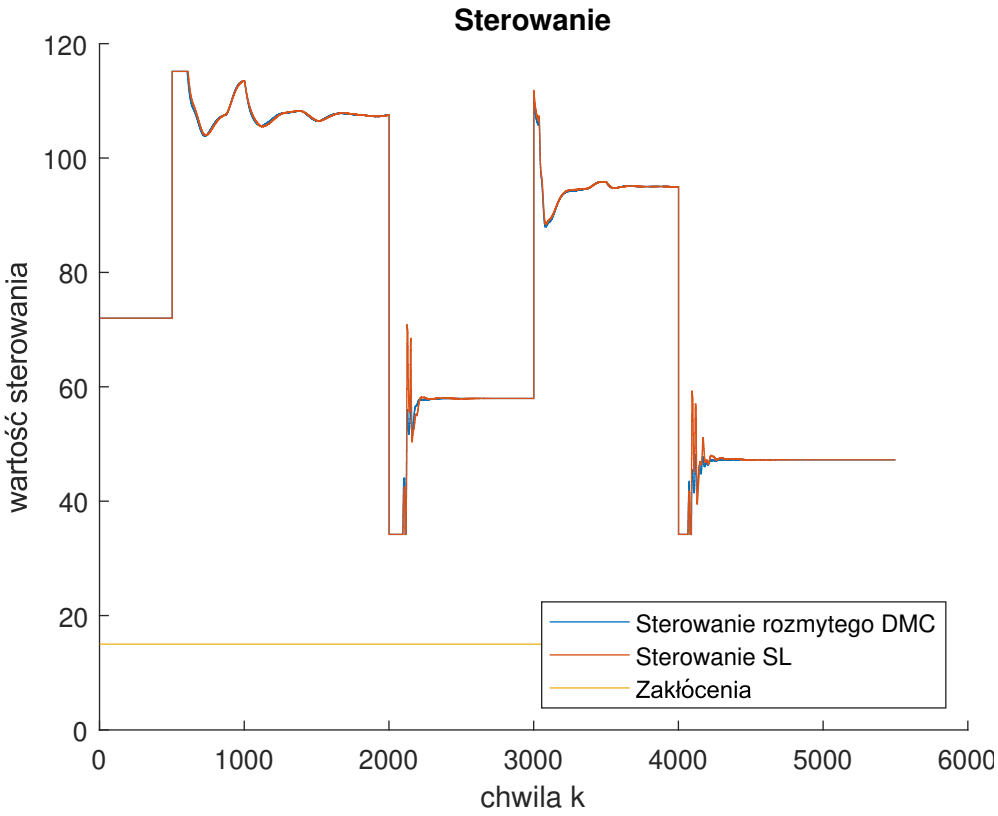
I - macierz jednostkowa wymiaru $N_u \times N_u$

$$f = -2 \cdot (Y_{zad} - Y_0)^T \cdot M \tag{1.3}$$

Testowaliśmy go na parametrach: $N = 132$, $N_u = 1$, $\lambda = 0.32$.
Otrzymaliśmy rezultaty w porównaniu z rozmytym DMC (oba algorytmy działają na 5 modelach lokalnych i gaussowskich funkcjach przynależności):
Błąd dla SL: $9.0415 \cdot 10^4$.
Błąd dla rozmytego DMC: $9.0081 \cdot 10^4$.
Dla tego testu konieczna była zmiana zmian wartości zadanej dlatego błąd dla rozmytego DMC jest delikatnie inny niż w poprzednich testach.
Porównując błędy widać, że SL działa lepiej, chodź wykresy są bardzo podobne.



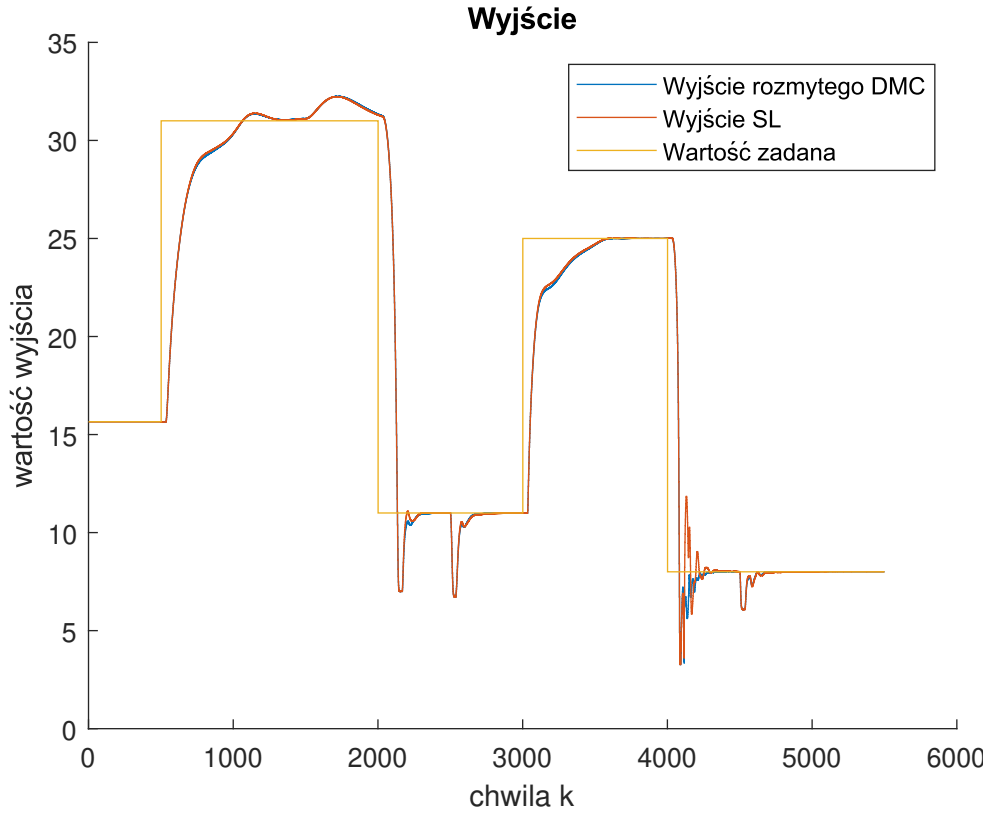
Rys. 1.1: Algorytm SL

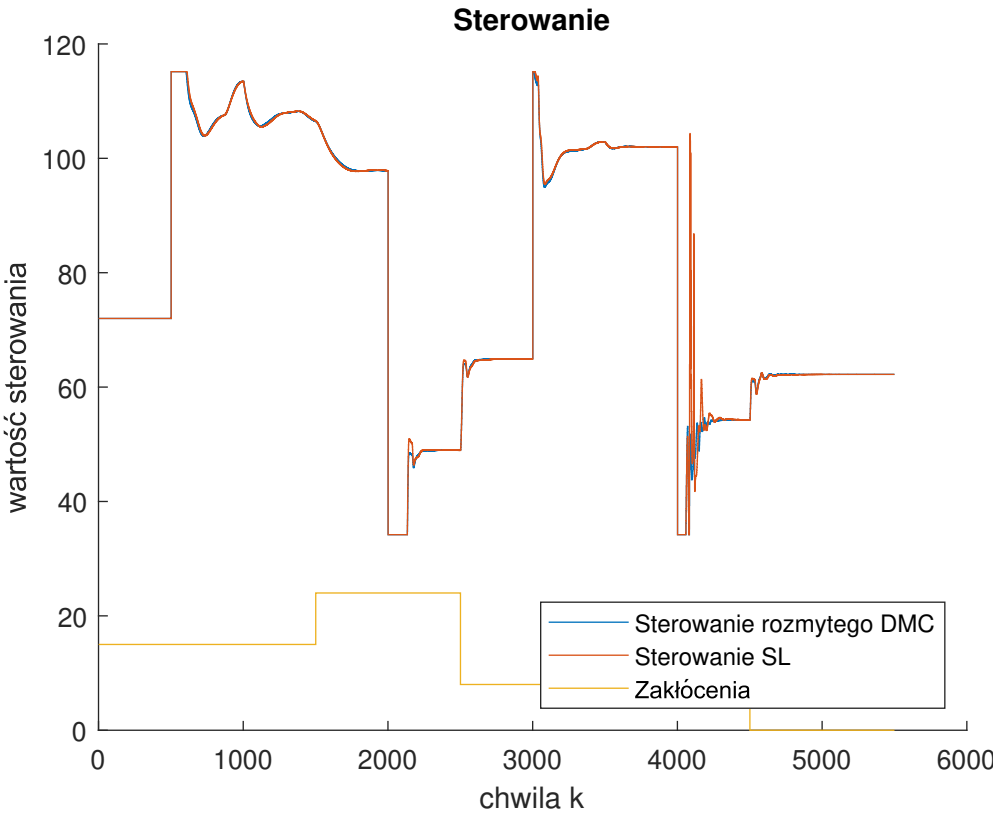


Rys. 1.2: Algorytm SL

1.2. Zakłócenia

Przetestowaliśmy odporność algorytmów na zakłócenia.





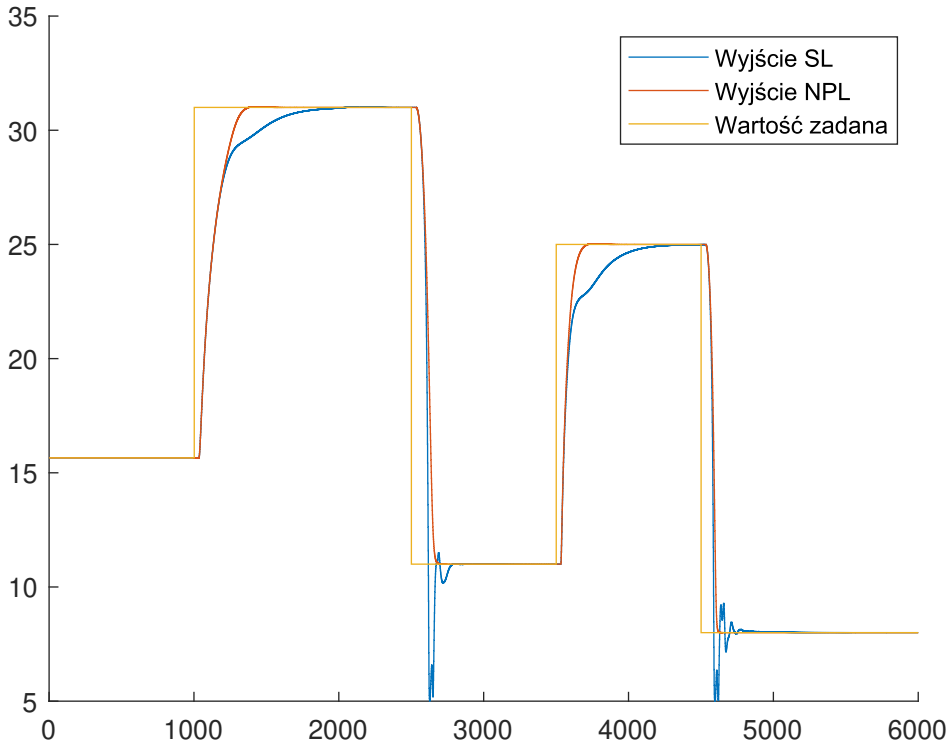
Porównując błędy z zakłóceniami między regulatorem rozmytym DMC, a SL możemy zauważyć, że błędy są mniejsze w SL.

2. Zadanie 4

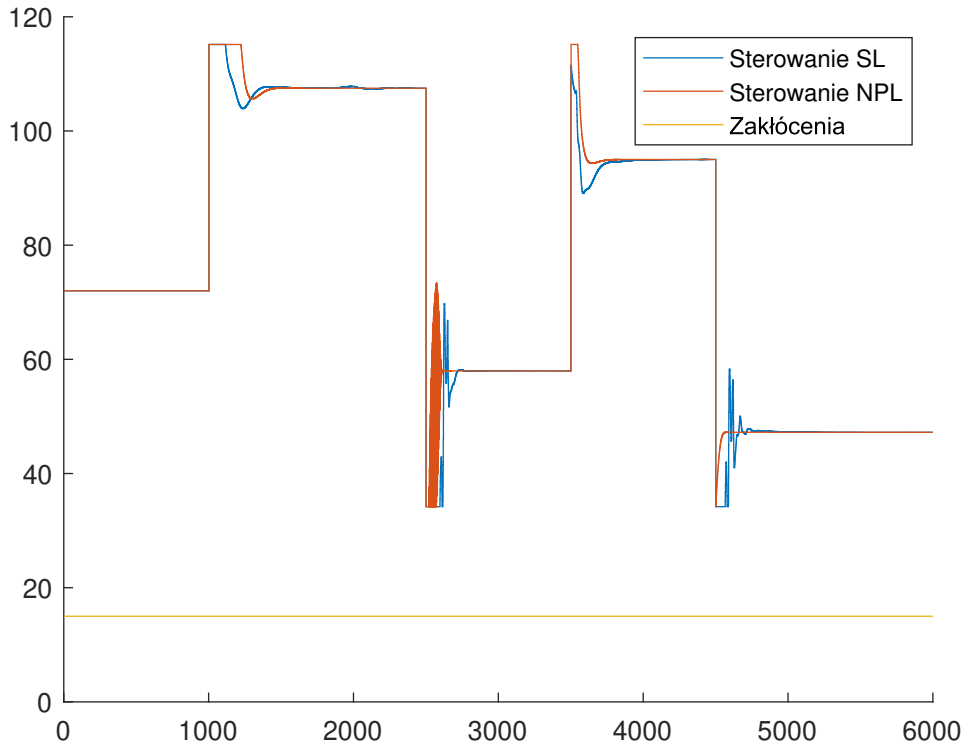
2.1. Numeryczny, rozmyty NPL i porównanie z SL

Zmiana między algorytm SL a NPL polega jedynie na innym sposobie obliczania odpowiedzi swobodnej obiektu regulacji. Dokładnie jest to opisane w skrypcie.

NPL bez zakłóceń:



Rys. 2.1: Porówna

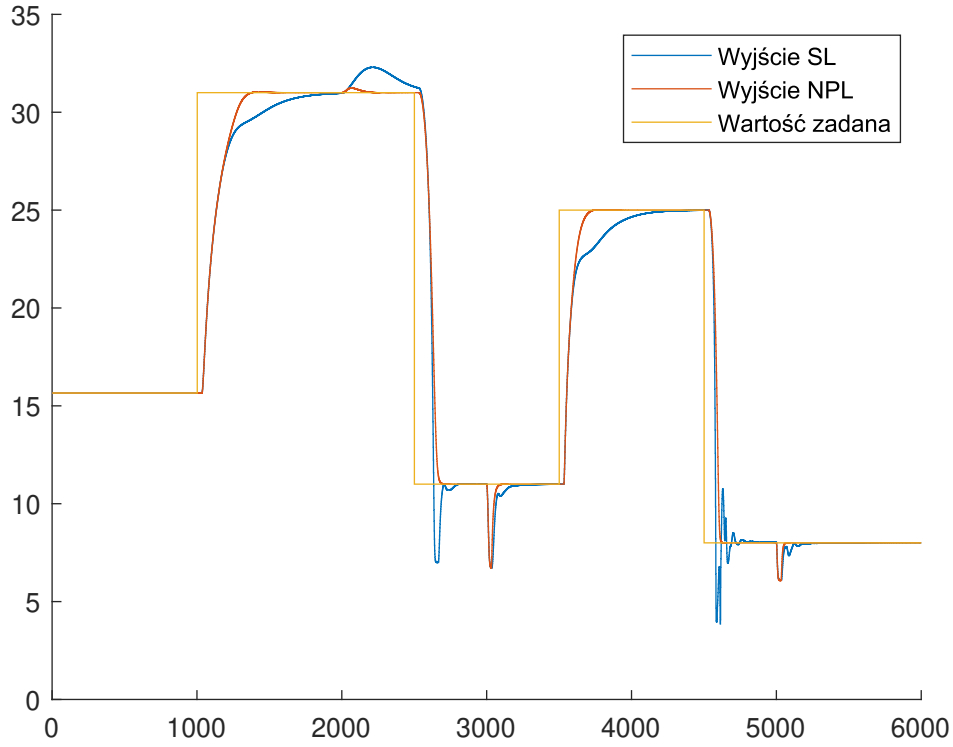


Rys. 2.2: Sterowanie NPL

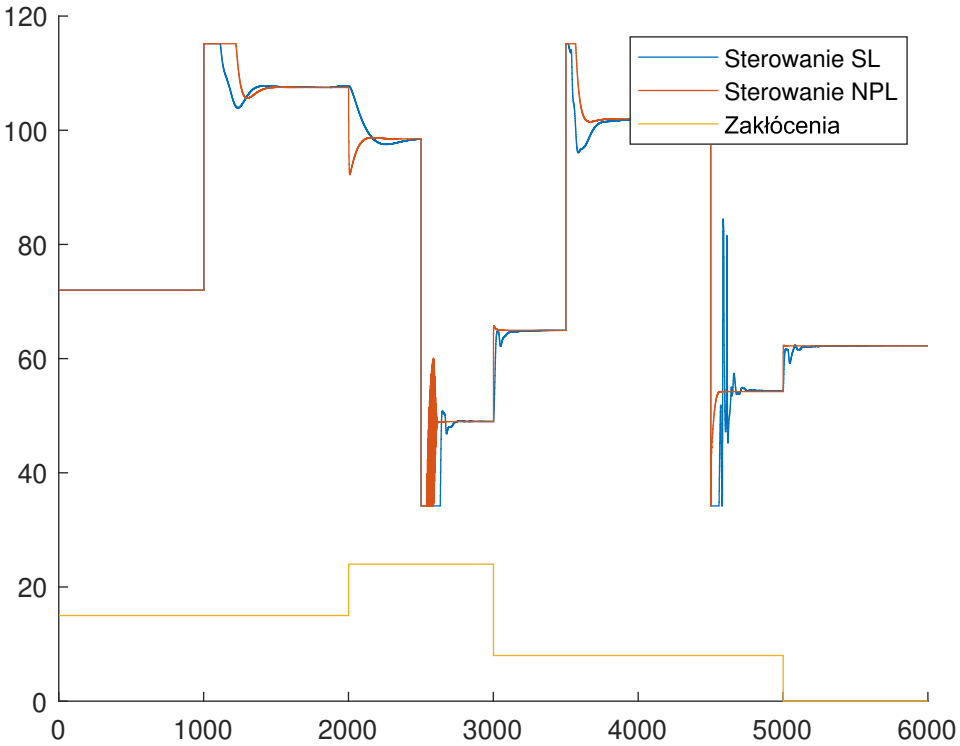
Błąd SL: $8,9821 \cdot 10^4$, błąd NPL: $9,1095 \cdot 10^4$.
Po wykresach widać, że algorytm NPL radzi sobie zdecydowanie lepiej niż SL. Szybko i jednolicie dochodzi do wartości zadanej.

2.2. NPL z zakłóceniami, porównanie z SL

:



Rys. 2.3: Wyjście NPL



Rys. 2.4: Sterowanie NPL

Błąd SL: $9,3629 \cdot 10^4$, NPL: $9,3390 \cdot 10^4$. Podobnie jak w przypadku bez zakłóceń, przy tym teście algorytm NPL wypada lepiej. Widać to i w wykresach, jak również w błędzie.

3. Podsumowanie

3.1. Podsumowanie

Z przeprowadzonych testów wynika, że dla nieliniowych obiektów regulatory rozmyte działają lepiej, niż konwencjonalne. Niestety przy bardziej skomplikowanych algorytmach i większej ilości modeli lokalnych stają się bardziej złożone obliczeniowo. Regulatory rozmyte lepiej radzą sobie z zakłóceniami i generują lepsze sterowanie (mniej oscylacji). Łatwiej też można je poprawić (zwiększając parametr λ). Na wykresach widoczne są również charakterystyczne momenty "przełączania się" między poszczególnymi modelami lokalnymi.

4. Dokumentacja

Skrypty:

"*PunktPracy.m*" - skrypt do sprawdzenia poprawności punktu pracy. Ma wpisane równania nieliniowe obiektu i solver *ode*45 rozwiązuje owe równania dla parametrów punktu pracy. Rysuje wykres jak obiekt dąży do punktu pracy.

Linaeryzacja.m - skrypt służący do linearyzacji nieliniowych równań obiektu w danych punktach linearyzacji (punkty linearyzacji: poziom cieczy w zbiorniku 1 - $h1$, poziom cieczy w zbiorniku 2 - $h2$, sterowanie - $F1$ i zakłócenia - Fd).

DMCStrojenie.m - skrypt służący do znalezienia algorytmem *ga* optymalnych parametrów i przetestowania na nich działalności konwencjonalnego regulatora DMC. Można testować wersja bez jak i z zakłóceniami. W zależności od od komentowanych linijek, rysuje wykresy: wyjścia obiektu ($h2$) i wartości zadanej ($h2_{zad}$) i sterowania ($F1$) i zakłócenia (Fd). Zwraca jeszcze błąd między wyjściem obiektu a wartością zadaną.

FuzzyModel.m - Skrypt symulujący działanie modelu rozmytego. Można ustawić ilość modeli lokalnych, i typ funkcji przynależności. Rysuje odpowiedź modlelu rozmytego, razem z odpowiedziami modelu nieliniowego, oraz zlineartyzowanego, my móc je porównać.

FuzzyDmcStrojenie.m - skrypt służący do testowania parametrów rozmytego regulatora DMC. Można testować wersja bez jak i z zakłóceniami. W zależności od od komentowanych linijek, rysuje wykresy: wyjścia obiektu ($h2$) + wartości zadanej ($h2_{zad}$) i sterowania ($F1$) + zakłócenia (Fd). Zwraca jeszcze błąd między wyjściem obiektu a wartością zadaną.

porownianie_FDMC_SL.m - skrypt porównujący działanie algorytmu FDMC i SL. Rysuje też wykresy do porównania wizualnego przebiegów wartości wyjścia i sterowania dla obu regulatorów.

porownianie_SL_NPL.m - skrypt porównujący działanie algorytmu SL i NPL. Rysuje też wykresy do porównania wizualnego przebiegów wartości wyjścia i sterowania dla obu regulatorów.

Funkcje:

StepRespons.m - Uruchomienie przykładową komendą: "StepRespons(92, 19, 16, 0)", Pobiera parametry: $F1$ - wartość sterowania, do którego z punktu pracy skoczy sterowanie, $h1l$ - punkt linearyzacji poziomu cieczy w zbiorniku 1 i $h2l$ - punkt linearyzacji poziomu cieczy w zbiorniku 2, *rysowanie* - gdy jej wartość wynosi 1, to wykonują się wykresy przebiegów. Zwraca: *respons* - gotowa do użycia w DMC przeskalowana odpowiedź obiektu po linearyzacji.

SuperDMC.m - Uruchomienie przykładową komendą: "SuperDMC([150, 2, 15], 0, 1)", Pobiera parametry: N - horyzont predykcji, Nu - horyzont sterowania, λ - parametr kary, *zakucenia* - wartość 1 włącza zakłócenie w symulacji, *rysowanie* - wartość 1 włącza ry-

sowanie rysunków. Zwraca: E - łączny błąd, $h1$ - zmiany poziomu cieczy w zbiorniku 1, $h2$ - zmiany w poziomie cieczy w zbiorniku 2, $h2_{zad}$ - wartość zadaną poziomu cieczy w z zbiorniku 2, $F1$ - sterowanie, Fd - zakłócenia (wszystkie elementy prócz błędu są zwracane jako wartości dla wszystkich k chwil czasu symulacji obiektu).

StepResponsesFuzzy.m - Uruchomienie przykładową komendą: "StepResponsesFuzzy(4, [5, 35], 0)", Pobiera parametry: *liczbaregulatorw* - liczba przedziałów, na który ma zostać podzielona "przestrzeń" między parametrami ograniczającymi ją, *functiontype* - ustala używany rodzaj funkcji przynależności (do wyboru *tr* - funkcja przynależności trójkątna i *gaus* - funkcja przynależności gaussowska), y_{min} - dolne ograniczenie przestrzeni i y_{max} - górne ograniczenie przestrzeni, *rysowanie* - wartość 1 włącza rysowanie rysunków. Zwraca: *functions* - obiekt MATLAB cell, który zawiera, odpowiedzi skokowe w ilości takiej , która została przekazana w parametrach.

MembershipFunction.m - Uruchomienie przykładową komendą: "MembershipFunction(4, "gaus", 5, 35, 0)", Pobiera parametry: *liczbaregulatorw* - liczba przedziałów, na który ma zostać podzielona "przestrzeń" między parametrami ograniczającymi ją i [y_{min} - dolne ograniczenie przestrzeni i y_{max} - górne ograniczenie przestrzeni], *rysowanie* - wartość 1 włącza rysowanie rysunków. Zwraca: *responses* - obiekt MATLAB cell, który zawiera, ustaloną w parametrach, liczbę, gotowych do użycia w rozmytym DMC ,przeskalowanych odpowiedzi obiektu po linearyzacji.

```
width = y_max-y_min;
functions = cell(1, liczba_regulatorow);
for i = 1:1:liczba_regulatorow
    if typ_funkcji == "tr"
        functions{i} = ...
            @(x) trimf(x, [y_min+(i-1)*width/(liczba_regulatorow+1),...
            y_min+(i)*width/(liczba_regulatorow+1)...
            y_min+(i+1)*width/(liczba_regulatorow+1)]);
    elseif typ_funkcji == "gaus"
        functions{i} =...
            @(x) gaussmf(x, [sqrt((width/(liczba_regulatorow+1))), ...
            y_min+i*width/(liczba_regulatorow+1)]);
    end
end
```

FuzzyDmc.m - Uruchomienie przykładową komendą: "FuzzyDmc([150, 2, 15], 4, "gaus", 1, 1)", Pobiera parametry: [N - horyzont predykcji, Nu - horyzont sterowania, λ - parametr kary], *iloregulatorwlokalnych* i *typfunkcji* - rodzaj funkcji przynależności (trójkątna lub gaussowska), której używa regulator, *zakucenia* - wartość 1 włącza zakłócenie w symulacji, *rysowanie* - wartość 1 włącza rysowanie rysunków. Zwraca: to samo co *SuperDmc.m* 4.

Liczenie sterowania podejściem PDC:

```
for i=1:liczba_regulatorow
    sum_mi = sum_mi + functions{i}(h2(k));
end

DU = cell(1,liczba_regulatorow);
for i=1:liczba_regulatorow
     %Pomiar wyjścia
```

```
        Y = ones(N, 1) * h2(k);
        Y_zad = ones(N, 1) * h2_zad(k);
        DU{i} = (K{i} * (Y_zad -  M_p{i} * DU_p - Y));
    end
    u=0;
    for i =1:liczba_regulatorow
        u = u + (functions{i}(h2(k))/sum_mi)*DU{i}(1);
    end
```

SL.m - Uruchomienie przykładową komendą: "SL([150, 2, 15], 4, "gaus", 1, 1)", Pobiera i zwraca to samo co *FuzzyDmc.m* 4.

Oblicznie rozmytej odpowiedzi skokowej:

```
    %symulacja obiektu
    [h1_prov, h2_prov] = func(h1(k-1), h2(k-1), F1(k-1-tau), Fd(k));
    [h1_mid, h2_mid] = func(h1(k-1)+h1_prov*T/2, h2(k-1)+h2_prov*T/2, F1(k
    h1(k) = h1(k-1) + T*h1_mid ;
    h2(k) = h2(k-1) + T*h2_mid;

    %Obliczenie części macierzy DMC

    sum_mi = 0;
    for i=1:liczba_regulatorow
        sum_mi = sum_mi + max(functions{i}(h2(k)), 0.01);
    end

    s_average = zeros(1, D);
    for j=1:D
        for i=1:liczba_regulatorow
            s_average(j) = s_average(j) + max(functions{i}(h2(k)), 0.01)*o
        end
    end
```

Obliczanie macieży M i M_p w algorytmie SL

```
    M = zeros(N, N_u);
    for column=1:N_u
        for row=1:N
           if (row>=column)
             if(row-column+1<=D)
                M(row,column)=s_average(row-column+1);
             else
               M(row,column)=s_average(D);
             end
          end
        end
    end

    M_p = zeros(N, D-1);
    for column=1:(D-1)
        for row=1:N
```

```
            if row + column > D
                if column>D
                    M_p(row, column) = 0;
                else
                    M_p(row, column) = s_average(D) - s_average(column);
                end
            else
                M_p(row, column) = s_average(row + column) - s_average(col
            end
        end
    end
```

NPL.m - Uruchomienie przykładową komendą: "NPL([150, 2, 15], 2, "gaus", 1, 1)", Pobiera i zwraca to samo co *FuzzyDmc.m* 4.

Obliczenie odpowiedzi swobodnej dla algorytmu NPL:

```
    for i=1:N
        if i>1
            [h01_prov, Y0_prov] = func(h01(i-1), Y0(i-1),...
            F1(min(k+i-1-tau,k-1)), Fd(k));
            [h01_mid, Y0_mid] = func(h01(i-1)+h01_prov*T/2, Y0(i-1)+ ...
            Y0_prov*T/2, F1(min(k+i-1-tau,k-1)),Fd(k));
            h01(i) = h01(i-1) + T*h01_mid ;
            Y0(i) = Y0(i-1) + T*Y0_mid;
        else
            [h01_prov, Y0_prov] = func(h1(k), h2(k), F1(k-tau), Fd(k));
            [h01_mid, Y0_mid] = func(h1(k)+h01_prov*T/2, h2(k)+Y0_prov*T/2,
            h01(i) = h1(k) + T*h01_mid ;
            Y0(i) = h2(k) + T*Y0_mid;
        end
        Y0(i) = max(y_min + 0.001, Y0(i));
        Y0(i) = min(y_max - 0.001, Y0(i));
    end
```