Politechnika Śląska **Gliwice**

Wydział Automatyki Elektroniki i Informatyki **Rok akademicki 2021/2022**

Kierunek: Automatyka i Robotyka **Semestr letni**

Metody Optymalizacji
Projekt

Strojenie układu regulacji

Wykonali:

Mateusz DERA

Grupa SPiI

1. Wstęp

Celem projektu było znalezienie nastaw regulatora PID dla obiektu tak, aby zminimalizować zadane wskaźniki przy zmianie wartości zadanej z 0 na 0.5.

Regulacja PID ma na celu wypracowanie odpowiedniego sterowania na podstawie uchybu czyli różnicy pomiędzy wartością otrzymaną, a wartością zadaną. Regulator opisany jest następującym wzorem:

$$u(t) = k_p\left[\varepsilon(t) + \frac{1}{T_i}\int_0^t \varepsilon(\tau)d\tau + T_d\frac{d\varepsilon(t)}{dt}\right]$$

gdzie:

u(t) – wartość sterowania w chwili *t*

e(t) – wartość uchybu w chwili *t*

k_p – wzmocnienie regulatora

T_i – stała całkowania

T_d – stała różniczkowania

Algorytm genetyczny w różnych wariantach zadania minimalizował następujące wskaźniki jakości:

$$ISE = \int_0^\infty e^2(t)dt$$

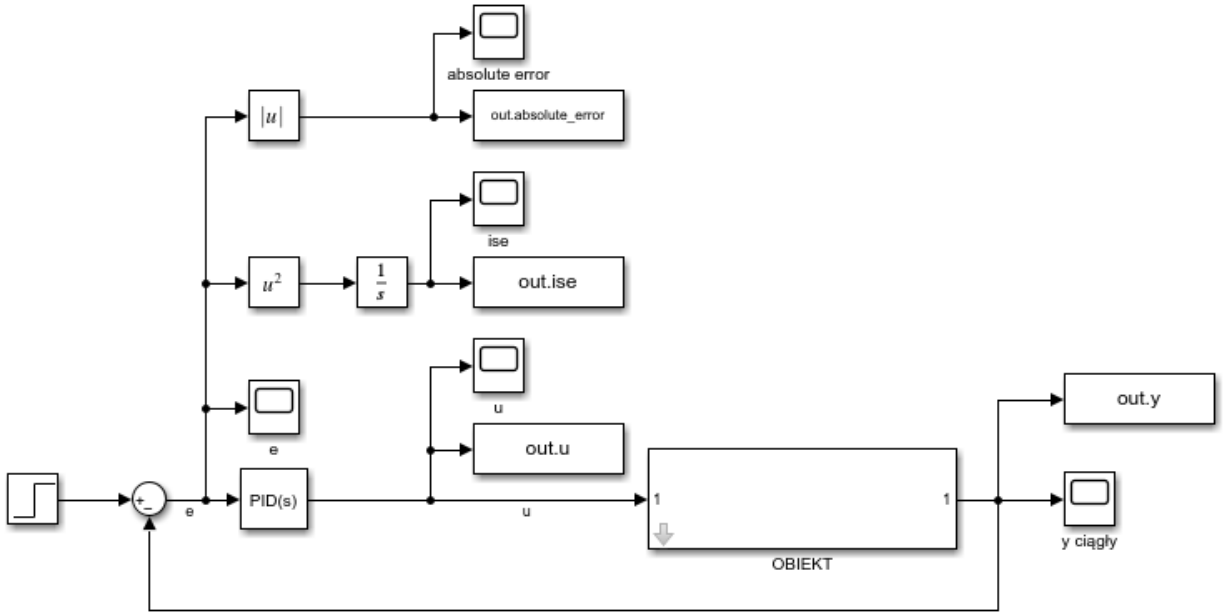
$$\bar{e} = \frac{\sum_{t=20}^{100} e(t)}{N}$$

$$ISE + ISC = \int_0^\infty e^2(t)dt + \int_0^\infty ((u(\infty) - u(t))^2)dt$$

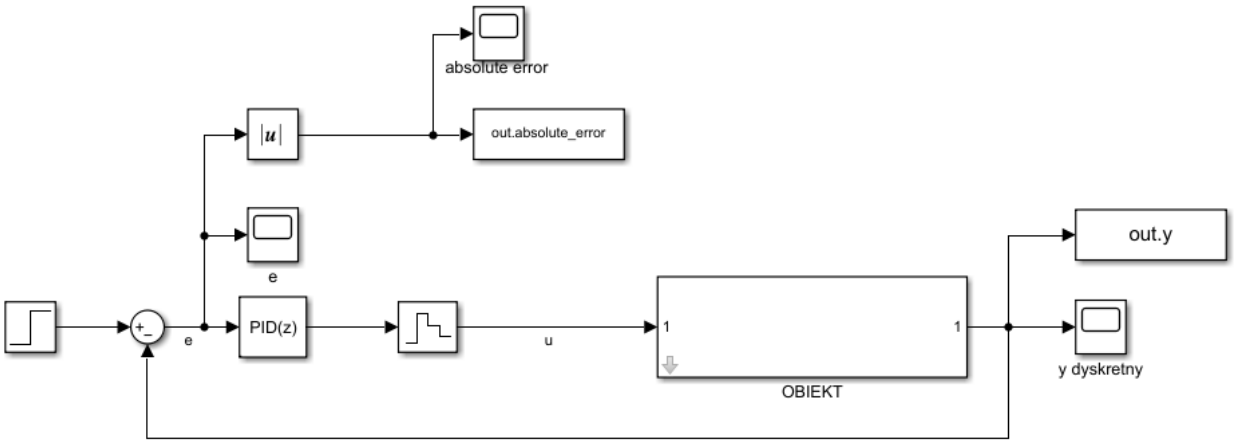
Zadanie	Wskaźnik	Typ regulatora
1	ISE w czasie 100 s	Ciągły
2	Średni błąd w okresie 20-100 s	Ciągły
3	ISE + ISC w 100 s	Ciągły
4	Średni błąd w okresie 20-100 s	Dyskretny

W celu rozwiązania zadań projektowych zaimplementowano algorytm genetyczny – ze względu na brak informacji o obiekcie regulacji uznano, że implementacja algorytmu genetycznego jest najlepszym wyborem do rozwiązania zadań.

Podczas projektu wykorzystano modele w Simulinku przedstawione poniżej.



Rysunek 1 Model układu ciągłego



Rysunek 2 Model układu dyskretnego

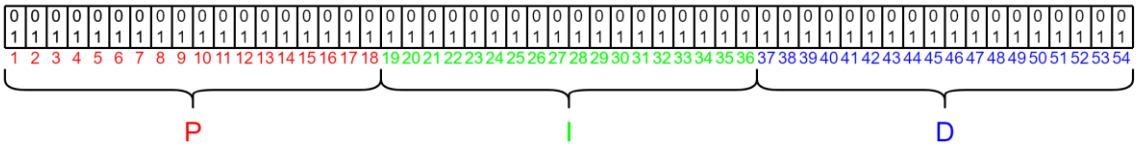
2. Podstawowy algorytm

Podczas implementacji algorytmu kierowano się poniższymi krokami:

1) Inicjalizacja – wybór losowej, początkowej populacji
2) Wybór osobników do reprodukcji
3) Tworzenie osobników potomnych
4) Tworzenie nowej generacji
5) Jeśli nie został spełniony warunek stopu – powrót do kroku nr 2
6) Koniec

3. Implementacja

Pierwszym krokiem w implementacji było zdefiniowanie postaci „chromosomu" – zdecydowano, że jeden gen będzie odpowiadał jednemu bitowi (kodowanie binarne), a chromosom będzie miał długość 54 genów, co daje nam 18 genów („N") na każdą z nastaw regulatora PID („variables_number"). Ponadto zdecydowano, że każda z nastaw (18 bitów) po zdekodowaniu reprezentuje liczbę od 0 („a") do 20 („b").



Rysunek 3 Graficzna prezentacja genotypu

W celu implementacji algorytmu należało również określić takie parametry jak: rozmiar populacji („population_size"), rozmiar turnieju („tournament_size"),, współczynnik krzyżowania („crossover_rate", określa część populacji, która powinna zostać zastąpiona w wyniki krzyżowania) oraz współczynnik mutacji („mutation_rate", określa część populacji, która ulec mutacji). Zdecydowano, że populacja składa się z 40 osobników, do turnieju dobiera się losowo 8 osobników, a współczynnik krzyżowania i mutacji wynosi kolejno 0.8 (32 nowe osobniki w populacji) i 0.1 (4 osobniki ulegające mutacji).

Ważną częścią w implementacji algorytmu są również warunki stopu: maksymalna liczba generacji („max_generation") oraz maksymalna liczba „stałych" generacji („max_stall_iterations") – określa maksymalną liczbę generacji, w których najlepsze wartości wskaźnika nie ulegają znacznej poprawie. Zdecydowano, że wynoszą one kolejno 30 oraz 8.

Inicjalizacje wyżej opisanych parametrów dla algorytmu genetycznego oraz populacji początkowej realizuje przedstawiona poniżej funkcja StructInitializer:

```
function out = StructInitializer(variables_number, ...
    population_size, crossover_rate, mutation_rate, tournament_size, ...
```

```
    max_generation, max_stall_iterations, N, a, b, task_number)

out.variables_number = variables_number;
out.population_size = population_size;

% crossover and mutation parameters
out.crossover_rate = crossover_rate;
out.mutation_rate = mutation_rate;
out.tournament_size = tournament_size;

% stop conditions parameters
out.max_generation = max_generation;
out.max_stall_iterations = max_stall_iterations;

% decoding information
out.N = N;
out.a = a;
out.b = b;

% generating initial population
rng('shuffle');

out.task_number = task_number;

out.current_population = randi([0 1], out.population_size, out.variables_number *
out.N);

end
```

Pojawiająca się w kodzie zmienna „task_number”, jak sugeruje nazwa, określa numer zadania projektowego (od 1 do 4). Powyższa funkcja zwraca strukturę zawierającą wszystkie potrzebne dane do inicjalizacji algorytmu genetycznego.

Kolejnym krokiem była implementacja algorytmu genetycznego. Kolejne kroki zaimplementowanego algorytmu można przedstawić za pomocą poniższej listy:

1) Zdekodowanie każdego osobnika w populacji – realizowane przez funkcje DecodePopulation oraz Decode przedstawione poniżej:

```
function decoded_population = DecodePopulation(population, k1, k2, N,
variables_number, population_size)

decoded_population = zeros(population_size, variables_number);
for m = 1:population_size
    for i = 1:variables_number
        decoded_population(m, i) = Decode(population(m, (1 + (i - 1) * N:i * N)), k1,
k2);
    end
end
```

```
end


function out = Decode(variable, k1, k2)

v = 1;
s = 0;

for i = 1:length(variable)
    s = s + v * variable(length(variable) - i + 1);
    v = v * 2;
end

out = k1 * s + k2;

end
```

zmienne „k1” oraz „k2” są zmiennymi potrzebnymi do zdekodowania obliczonymi w poniższy sposób:

```
mx = 2^input.N - 1;
k1 = (input.b - input.a) / mx;
k2 = input.a;
```

2) Wyznaczenie wartości wskaźnika dla każdego osobnika w populacji wykorzystując funkcję Fitness:

```
function f = Fitness(x, task_number)

assignin('base',"x",x);

if task_number == 1
    out = sim('Obiekt123.slx',[0 100]);
    f = out.ise.Data(end);
elseif task_number == 2
    out = sim('Obiekt123.slx',[0 100]);
    f = mean(out.absolute_error.Data(find(out.absolute_error.Time==20):end));
elseif task_number == 3
    out = sim('Obiekt123.slx',[0 100]);
    u_end = out.u.Data(end);
    isc = 0;
    for i=1:length(out.u.Data)-1
        isc = isc + (u_end - out.u.Data(i))^2 * (out.u.Time(i+1) - out.u.Time(i));
    end
    f = out.ise.Data(end) + isc;
else
    out = sim('Obiekt123_dyskretny.slx',[0 100]);
    f = mean(out.absolute_error.Data(find(out.absolute_error.Time==20):end));
```

```
end

end
```

3) Wyznaczenie rodziców za pomocą odpowiedniej ilości turniejów wykorzystując funkcję Tournament

```
function [winner, ind] = Tournament(population, fit_vector, population_size,
tournament_size)

% we cannot have more tournament participants than population size
if tournament_size > population_size
    tournament_size = population_size;
end

% getting random tournament participants
tournament_members = zeros(tournament_size, 2);
tournament_members(:, 1) = randperm(population_size, tournament_size);

for i = 1:tournament_size
    tournament_members(i, 2) = fit_vector(tournament_members(i, 1));
end

% choosing winner of the tournament
[~, index] = min(tournament_members(:, 2));
ind = tournament_members(index, 1);

winner = population(ind, :);

end
```

4) Stworzenie dzieci z wybranych rodziców wykorzystując funkcję OnePointCrossover

```
function [child1, child2] = OnePointCrossover(parent1, parent2, l)

point = randi([1 l]);

child1 = zeros(l, 1);
child2 = zeros(l, 1);

child1(1:point) = parent1(1:point);
child1(point + 1:end) = parent2(point + 1:end);

child2(1:point) = parent2(1:point);
child2(point + 1:end) = parent1(point + 1:end);

end
```

5) Wybranie najbardziej przystosowanych osobników i zachowanie ich w nowej generacji

6) Zmutowanie nowej generacji realizowane za pomocą funkcji Mutation

```
function population = Mutation(population, population_size, l, mutation_number)

% getting random gen to be mutated
x = randperm(population_size, mutation_number);
y = randperm(l, mutation_number);

for i = 1:mutation_number
    population(x(i), y(i)) = 1 - population(x(i), y(i));
end

end
```

7) Sprawdzenie warunków stopu – jeśli nie są spełnione to powrót do pkt 1
8) Wyznaczenie najbardziej przystosowanego osobnika i zakończenie algorytmu

Zaimplementowany algorytm realizujący powyższe punkty jest realizowany prze funkcję GeneticAlgorithm i został przedstawiony poniżej:

```
function output = GeneticAlgorithm(input)

if ~ any([1, 2, 3, 4] == input.task_number)
   input.task_number = 1;
end

contraints = [0.00005, 0.00001, 0.00005, 0.00001];
contraint = contraints(input.task_number);

children_number = ceil(input.population_size * input.crossover_rate);
mutation_number = ceil(input.population_size * input.mutation_rate);

if mod(children_number, 2) ~= 0
    if children_number < input.population_size
        children_number = children_number + 1;
    elseif children_number == input.population_size
        children_number = children_number - 1;
    end
end

remaining_parents_number = input.population_size - children_number;

output.iterations = 0;
stall_iterations = 0;
flag_stall_iter = true;

%  vectors for storing informations about each generation
output.best_fit_vector = zeros(input.max_generation + 1, 1);
output.mean_fit_vector = zeros(input.max_generation + 1, 1);

% variables for decoding
if input.a > input.b
    [input.b, input.a] = deal(input.a, input.b);
```

```
end
mx = 2^input.N - 1;
k1 = (input.b - input.a) / mx;
k2 = input.a;

% length of the genotype
l = input.N * input.variables_number;

% generataing next generations
while output.iterations <= input.max_generation && flag_stall_iter

    % decoding each individual
    v = DecodePopulation(input.current_population, k1, k2, input.N, ...
        input.variables_number, input.population_size);

    fit_vector = zeros(input.population_size, 1);

    % calculating fitness value for each individual
    for i = 1:input.population_size
        fit_vector(i) = Fitness(v(i, :), input.task_number);
    end

    % there's no need to generate 'max_generation + 1' generation
    if output.iterations < input.max_generation

        % vector for storing parents
        parents = zeros(children_number, l);
        temp_population = input.current_population;
        temp_population_size = input.population_size;

        % choosing parents in tournament
        for i = 1:children_number
            [parents(i, :), index] = Tournament(temp_population, ...
             fit_vector, temp_population_size, input.tournament_size);

            % we don't want situation when we get 2 childern from 1 parent
            if mod(i,2) ~= 0
                temp_population(index, :) = [];
                temp_population_size = temp_population_size - 1;
            else
                temp_population = input.current_population;
                temp_population_size = input.population_size;
            end

        end

        % vector for storing children
        children = zeros(children_number, l);

        % generating children
        for i = 1:2:children_number - 1
            [children(i, :), children(i + 1, :)] = ...
                OnePointCrossover(parents(i, :), parents(i + 1, :), l);
        end
```

```
        % choosing best parents that will remain in next generation
        [~, best_parents_index] =...
            mink(fit_vector, remaining_parents_number);

        % creating next generation
        input.current_population(1:remaining_parents_number, :) = ...
            input.current_population(best_parents_index, :);

        input.current_population(remaining_parents_number + 1:end, :) =...
            children;

        % mutating generation
        input.current_population = Mutation(input.current_population,...
            input.population_size, l, mutation_number);
    end

    % calculationg best and mean fit's value
    output.best_fit_vector(output.iterations + 1) = min(fit_vector);

    output.mean_fit_vector(output.iterations + 1) = mean(fit_vector);


    % checking if we get any progress in generating new generations
    if output.iterations > 0 &&...
            abs(output.best_fit_vector(output.iterations + 1) -...
            output.best_fit_vector(output.iterations)) < contraint
        stall_iterations = stall_iterations + 1;
    else
        stall_iterations = 0;
    end

    % text with informations about generation
    text = ['Iteration: ' num2str(output.iterations) ', Best: '...
     num2str(output.best_fit_vector(output.iterations + 1), 5)...
     ', Mean: ' num2str(output.mean_fit_vector(output.iterations + 1), 5)...
     ', Stall iteration: ' num2str(stall_iterations)];

    disp(text);

    if stall_iterations == input.max_stall_iterations
        flag_stall_iter = false;
        output.best_fit_vector(output.iterations + 2:end) = [];
        output.mean_fit_vector(output.iterations + 2:end) = [];
    end

    output.iterations = output.iterations + 1;

end

output.iterations = output.iterations - 1;

[output.solution, output.solution_index] = min(fit_vector);
output.solution_parameters = v(output.solution_index, :);

end
```

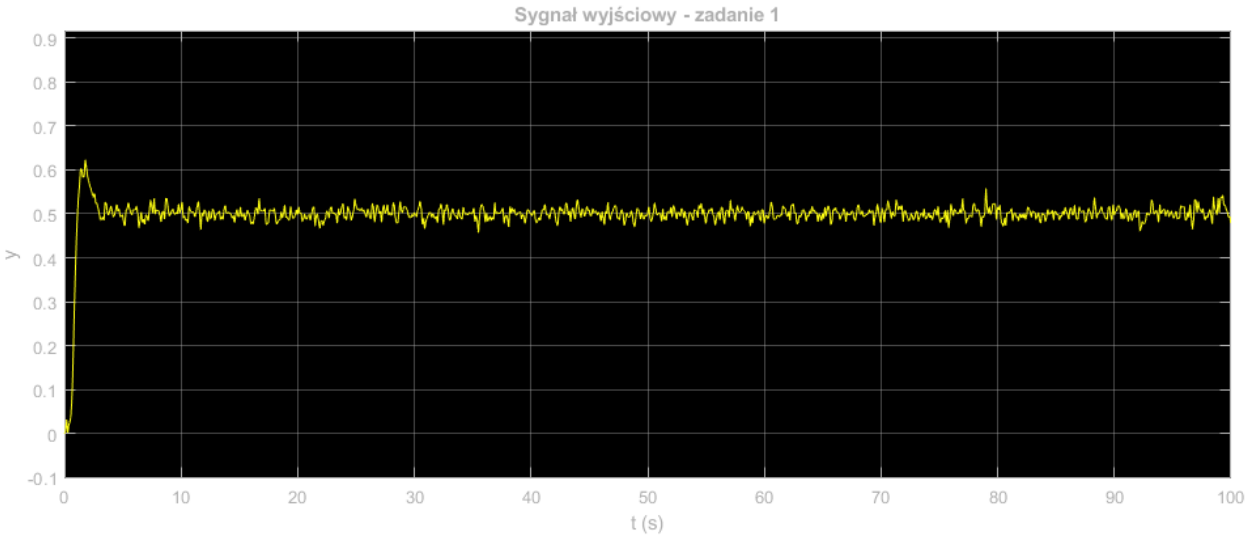
4. Wyniki

Otrzymane wyniki dla każdego zadania przestawiono w poniższej tabeli.

Numer zadania	Wartość parametru P	Wartość parametru I	Wartość parametru D	Wartość wskaźnika
1	4.9301	1.2503	2.0527	0.1958
2	2.8613	0.8317	1.4172	0.0092
3	0.4625	0.3735	0.0134	0.4785
4	3.4911	0.7324	1.686	0.0089

Dodatkowo sprawdzono otrzymane wyniki (nastawy) z każdego zadania pod kątem jakości regulacji – sprawdzono wartości maksymalnego przeregulowania. Otrzymanego wyniki przedstawiono poniżej.

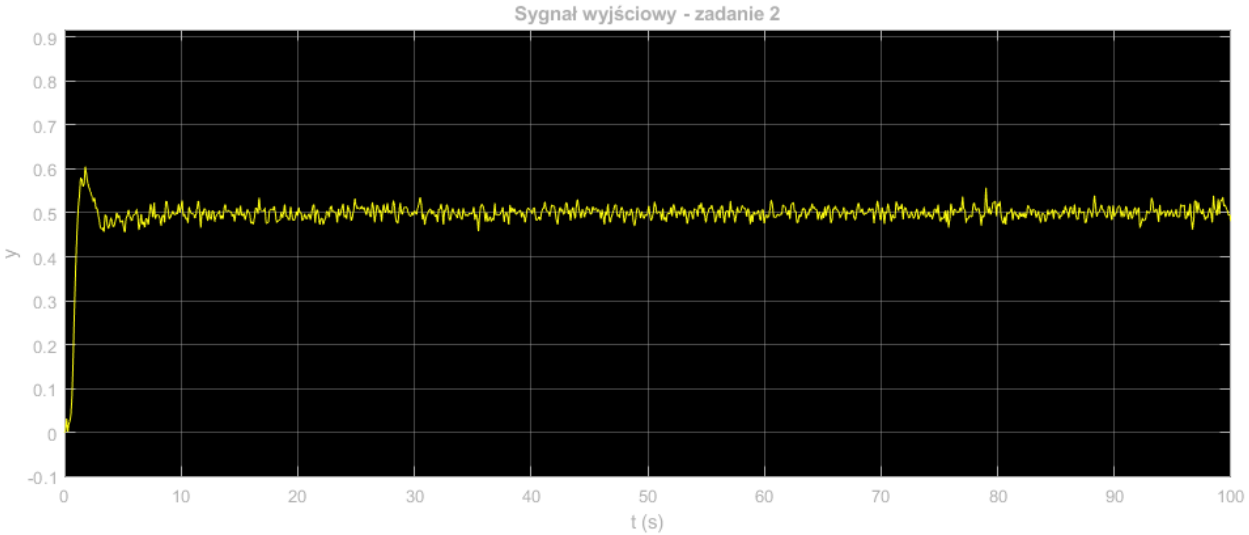
1) Zadanie 1



Rysunek 4 Sygnał wyjściowy dla zadania 1

Wartość maksymalnego przeregulowania: 0.123

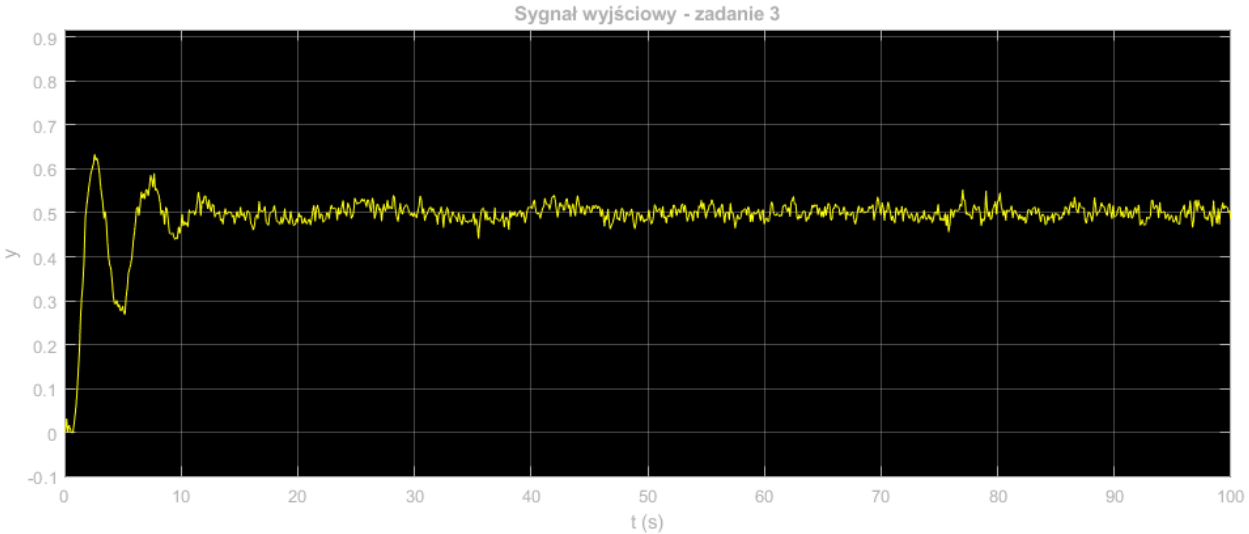
2) Zadanie 2



Rysunek 5 Sygnał wyjściowy dla zadania 2

Wartość maksymalnego przeregulowania: 0.105

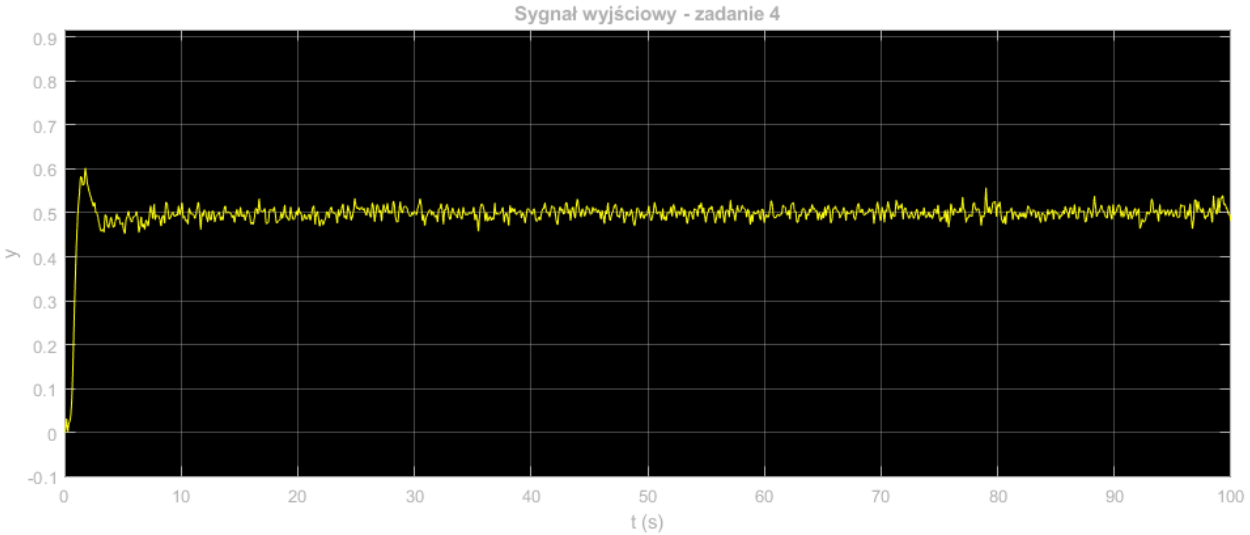
3) Zadanie 3



Rysunek 6 Sygnał wyjściowy dla zadania 3

Wartość maksymalnego przeregulowania: 0.133

Zadanie 4



Rysunek 7 Sygnał wyjściowy dla zadania 4

Wartość maksymalnego przeregulowania: 0.102

5. Wnioski

Realizowane ćwiczenie pozwoliło nam zmierzyć się z problemem optymalizacji nastaw regulatora PID. Bardzo rozwijająca była samodzielna implementacja algorytmu genetycznego. We wszystkich z zadań udało się wyznaczyć nastawy regulatora, które pozwoliły osiągnąć stabilny przebieg wyjścia bez dużych przeregulować oraz oscylacji. Z uwagi na fakt, że każde z zadań minimalizowało inny wskaźnik jakości, nie dało się ich porównać. Jako kryterium porównawcze przyjęliśmy więc maksymalne przeregulowanie – najmniejsze maksymalne przeregulowanie uzyskano dla nastaw otrzymanych w ramach zadania 4, a największe – dla zadania 3. Dodatkowo dla zadania 3 otrzymano oscylacje wartości wyjściowej. Stało się tak dlatego, że zminimalizowano składnik, którego składową było ISC, które jest niskie, jeżeli wartość sterująca nie zmienia się zbytnio względem sterowania końcowego. Z tego powodu zostały wybrane nastawy o niskich wartościach, które powodowały małe zmiany wartości sterującej. Konsekwencją było nietłumienie oscylacji. Z drugiej strony pozwoli to na przedłużenie żywotności elementu wykonawczego.