Teoria Sterowania

Sterowanie Inteligentne: Adaptacyjny kontroler ewolucyjny

1. Wprowadzenie

Idea niniejszej instrukcji jest bardzo podobna, do instrukcji wcześniejszej, tj. Ewolucyjnej identyfikacji obiektu sterowania. Tym razem stajemy przed podobnym problemem, ale zamiast identyfikować obiekt sterowania, odnajdywać będziemy transmitancję najlepszego kontrolera PID do sterowania obiektem. Istotne różnice sprowadzają się do następujących punktów:

A: Istotne jest dla nas przeprowadzenie procesu w sposób ograniczający prawdopodobieństwo destabilizacji obiektu. Zakładamy jednak, że obiekt wykazuje naturalne tłumienie a maksymalna amplituda sygnału sterującego jest ograniczona – więc w przpadku wykrycia niewłaściwej strategii sterowania możemy sterowanie zatrzymać i poczekać na naturalne ustabilizowanie obiektu.
B: Cały proces przeprowadzany jest przy założeniu, że obiekt może zmieniać się w czasie.
C: Każda próba sterowania obiektem reprezentuje rzeczywisty test wykonany na rzeczywistym obiekcie – niezbędne więc staje się ograniczenie prób (tzn. "sprawdzeń funkcji celu" do minimum. Zadowalamy się sterowaniem "wystarczająco dobrym" osiąganym w krótkim czasie niż sterowaniem "idealnym" które wymaga setek lub tysięcy prób do osiągnięcia swojej skuteczności.

Ze względu na te założenia najlepszym podejściem do rozwiązania problemu w sposób ewolucyjny okazuje się metoda 1+1 – ma szansę zadziałać dobrze z racji relatywnej prostoty zadania optymalizacyjnego a wymaga wyłącznie jednego testu w każdej iteracji.

Zadanie indywidualne

Przed rozpoczęciem zajęć należy podzielić przez 5 sumę liter swojego imienia i nazwiska. Reszta z tego dzielenia stanowi numer przepisu na "indywidualne zadanie". Aby uruchomić swój przepis na indywidualne zadanie, należy wczytać plik randomWalk z numerem swojego indywidualnego zadania a następnie użyć go jako argumentu funkcji. Osoba, o nr. Indywidualnego zadania "2" uruchomi więc swoje indywidualne zadanie poprzez wywołanie funkcji w następujący sposób:

```
load randomWalk_2
[A,B,C,D] = MyObject(randomWalk_2(1,:))
```

2. Optymalizacja kontrolera dla obiektu statycznego:

Wczytamy obiekt zdefiniowany w przestrzeni stanu. Zwróćmy uwagę na (0) w wywołaniu funkcji. Na razie pozostawiamy tą wartość bez zmian. W przyszłości poprzez zmiany argumentu będziemy mieli wpływ na zmiany obiektu w czasie.

```
MyObject = str2func('StateSpaceNonStationary_rank2');
[A,B,C,D] = MyObject(randomWalk_test(0))
```

Warto obejrzeć otrzymane macierze aby przekonać się, że obiekt ma dwa wejścia i jedno wyjście. Jedno z tych wejść potraktujemy jako wejście zakłócane – i dostarczać tam będziemy sygnał

szumu, na który kontroler nie będzie miał wpływu. Drugie z tych wejść będzie wejściem sterowania, a celem całego procesu będzie zminimalizowanie odpowiedzi obiektu w funkcji czasu. Zdefiniujmy zatem wektor czasowy będący podstawą do symulacji, oraz zestaw wejść i wyjść symulacji:

```
TimeStep = 0.1;
T = [0:TimeStep:1000];
Inputs = randn(length(T),2);
x = zeros(rank(A),length(Inputs)); % State vector
y = zeros(1,length(Inputs));       % System output
```

Będziemy również potrzebowali parametrów początkowych dla naszego regulatora PID. Przygotujemy od początku wszystkie trzy parametry, choć początkowo skorzystamy tylko z części proporcjonalnej:

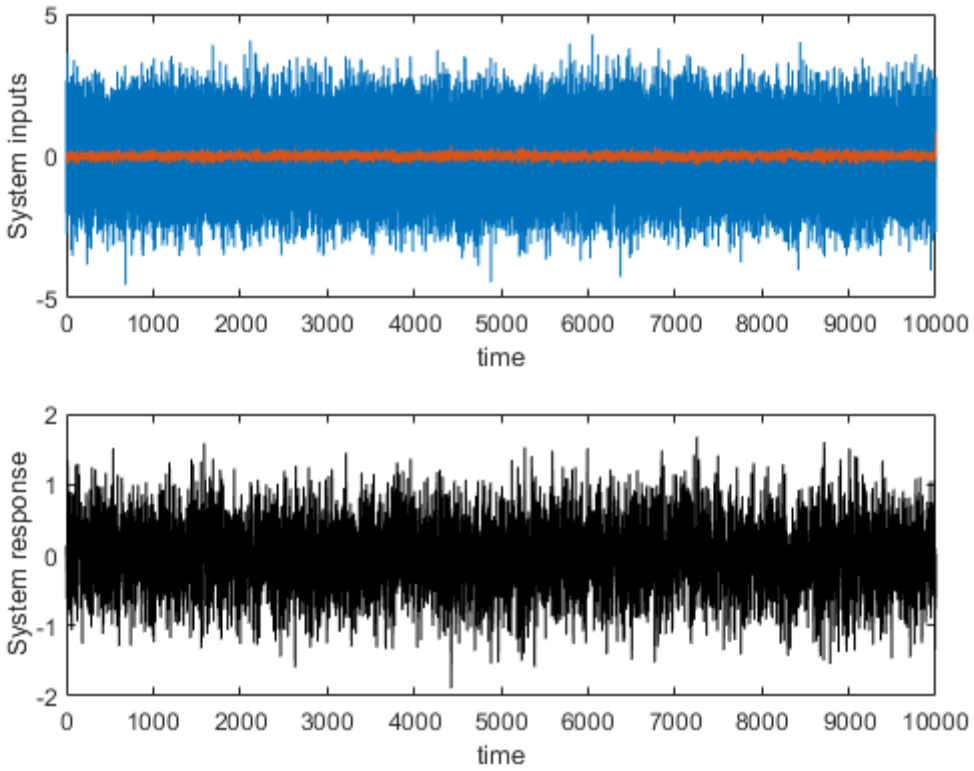
```
P =  1 * rand();
I =  0 * rand();
D =  0 * rand();
```

Ze względu na fakt, że musimy być w stanie "dostać się" do sterowania w trakcie jego trwania a obiekt ma być wymuszany za pomocą losowego szumu przez cały czas trwania symulacji i adaptacji – konieczne staje się napisanie pętli symulacyjnej ręcznie. Będziemy wykonywać następujące operacje aż do końca wektora T:

```
for k = 1:length(T)-1
    y(:,k) = C*x(:,k);
    if (k > 1)
        Sm = Sm + y(:,k);
        Inputs(k,2) = -y(:,k)'*P' - (y(:,k)-y(:,k-1))'*D'- Sm'*I';
        Inputs(k,2) = min(max(Inputs(k,2),...
     -SafetyThreshold),SafetyThreshold);
    end
    dot_X(:,k) = A*x(:,k) + B*Inputs(k,:)';
    x(:,k+1) = x(:,k) + TimeStep*(dot_X(:,k));
end

figure(1)
subplot(2,1,1);
plot(T,Inputs);
xlabel('time'); ylabel('System inputs')
subplot(2,1,2);
plot(T,y','k'); hold on
xlabel('time'); ylabel('System response')

rms(y)     % Measure of vibration at the output
```

Zwróćmy uwagę, że oprócz wykonania cyklu symulacji oraz wyznaczenia wartości podawanej na wejście regulacyjne wykonujemy również sprawdzenie, czy wejście mieści się w zdefiniowanym zakresie – odpowiada za to stała *SafetyThreshold*. Jeśli doprowadzilibyśmy do destabilizacji obiektu, *SafetyThreshold* uniemożliwi wpompowanie w obiekt niekontrolowanej energii. Ustawmy początkowo jego wartość na 1. Efekt sterowania może wyglądać jak na Rys. 1. Widzimy, że

sterownik "coś robi" (drugie wejście nie jest zerowe), trudno jednak powiedzieć na ile sprawnie radzi sobie z postawionym zadaniem – można jednak porównać finalne wartości RMS drgań dla różnych losowań parametrów regulatora.



Rys 1 – Przykładowe rezultaty zadania

Zadanie 2.1: Uruchom pętlę sterowania swoim ***indywidualnym*** obiektem, przetestuj zachowanie regulatora przy dodaniu stałych odpowiadających za część różniczkującą i całkującą – w którym przypadku otrzymuje się statystycznie niższe wartości RMS drgań na wyjściu? Zapisz wyniki przynajmniej 5 testów regulatora P, PI, PD oraz PID.

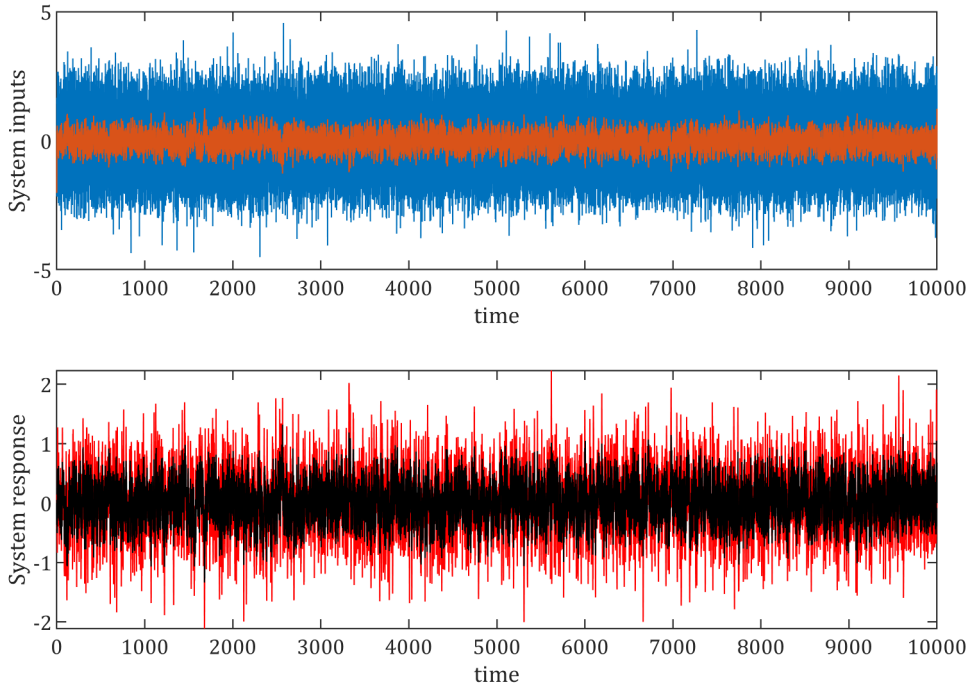
Lepszym sposobem oceny efektów pracy naszego regulatora byłoby wykonanie symulacji dwukrotnie dla tego samego wektora szumu zakłócającego – raz z zastosowaniem regulatora i raz bez niego. Wykonajmy więc to małe oszustwo (w praktyce nie mamy zazwyczaj możliwości odpowiedzieć na pytanie "co by było, gdyby sterowanie wyglądało inaczej a zakłócenia wyglądałyby tak samo"). W tym celu będziemy potrzebowali zdublować pętlę symulacji zerując drugą kolumnę macierzy Inputs oraz dostarczając nowych wektorów x, x_dot oraz y do wypełnienia treścią. Nazwijmy je *x_ref*, *y_ref* oraz *dot_x_ref*.

```
x_ref = zeros(rank(A),length(Inputs));
y_ref = zeros(1,length(Inputs));
Inputs_ref = [Inputs(:,1),zeros(size(Inputs(:,2)))];
```

Po wykonaniu drugiej pętli symulacyjnej, tym razem już bez sterowania, możemy dodać nasze nowe wyniki do poprzedniego wykresu:

```
plot(T,y_ref,'r'); hold on
```

I zaobserwować np. coś takiego (Rys 2):



Rys 2 – Wynik działania regulatora PD na przypadku testowym

Wygląda na to, że sterowanie działa: drgania są mniejsze, niż gdyby regulatora nie było. Spróbujmy teraz zoptymalizować parametry regulatora, aby uzyskać dalszą poprawę. W tym celu wykorzystamy następującą koncepcję: Co określoną liczbę iteracji sterowania zapiszemy miarę jakości sterowania oraz parametry sterownika który został wykorzystany w tym casie – a następnie zmodyfikujemy te parametry korzystając z podejścia 1+1 – wybierając jako punkt odniesienia sterowanie historycznie najlepsze. Zdefiniujmy zmienną *LengthOfControlCycle* i przypiszmy jej wartość 1000 (tzn. Co 1000 cykli sterowania będziemy wykonywali jego ewaluację), zainicjalizujmy również wartością 0 iterator *ControlCycleNumber*, który przechowywać będzie numer aktualnego cyklu sterowania, zdefiniujmy też wartość kroku mutacji odpowiadającego za zmiany.

Następnie, wewnątrz pętli sterowania skorzystajmy z następującego kodu:

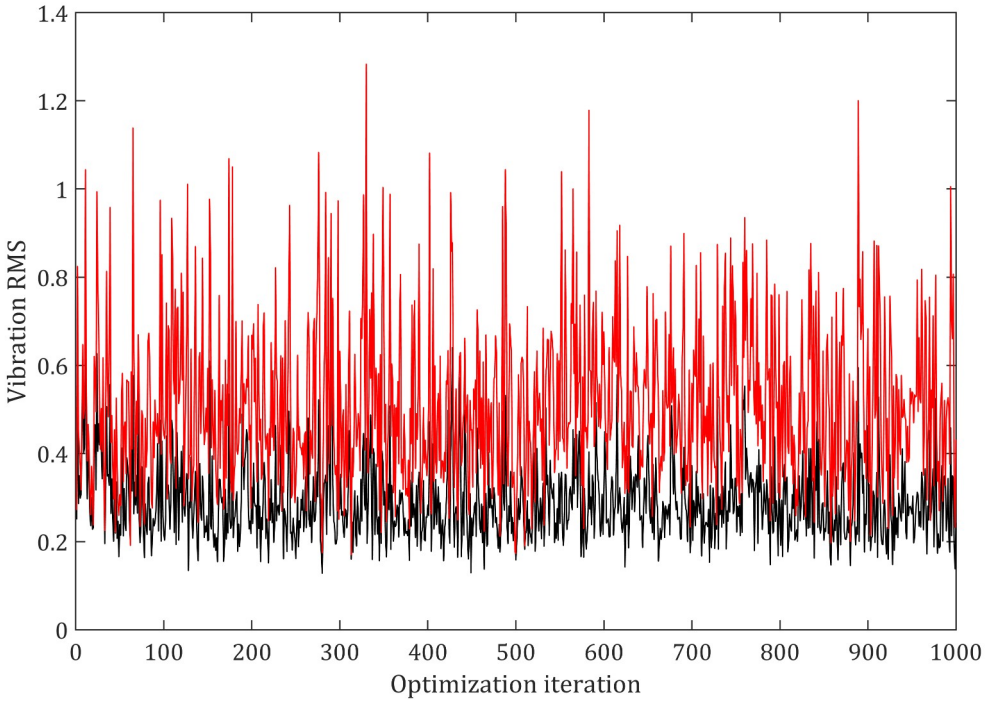
```
if(mod(k,LengthOfControlCycle) == 0)
    clc
    fprintf('Percent of calculations done: %f\', ...
   100*ControlCycleNumber/(length(T)/LengthOfControlCycle))
    ControlCycleNumber = ControlCycleNumber + 1;
    ControlResults(ControlCycleNumber).Quality = rms(y(k-LengthOfControlCycle+1:k));
    ControlResults(ControlCycleNumber).Parameters.P = P;
    ControlResults(ControlCycleNumber).Parameters.D = D;
    ControlResults(ControlCycleNumber).Parameters.I = I;

    [m, in] = min([ControlResults(:).Quality]);
 %   P = ControlResults(in).Parameters.P + Step*randn(size(ControlResults(in).Parameters.P));
 %   D = ControlResults(in).Parameters.D + Step*randn(size(ControlResults(in).Parameters.D));
 %   I = ControlResults(in).Parameters.I + Step*randn(size(ControlResults(in).Parameters.I));
end
```

Odkomentowanie odpowiednich linijek "odblokuje" optymalizację. Spróbujmy zacząć od regulatora proporcjonalnego i zobaczmy do jakiego wyniku będziemy w stanie dojść po naszych 100 000 iteracjach sterowania (tj. 100 iteracjach algorytmu 1+1). Aby zobaczyć jakość sterowania,

skorzystać można z zapisanej w strukturze ControlResults wartości Quality. Jeśli w analogiczny sposób zapiszemy ją dla pętli referencyjnej, można będzie porównać obie te wartości ze sobą w funkcji iteracji algorytmu optymalizacyjnego.

```
figure(3);
plot([ControlResults(:).Quality],'k'); hold on
plot([ReferenceResults(:).Quality],'r'); hold on
xlabel('Optimization iteration')
ylabel('Vibration RMS')
```

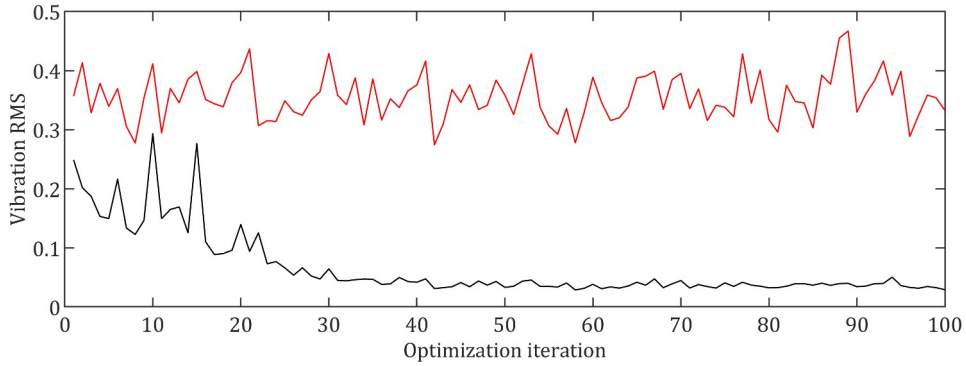
W moim przypadku, o ile sterowanie cały czas działa (tj. Regulator cały czas powoduje drgania mniejsze, niż gdyby go nie było), poza kilkoma pierwszymi iteracjami optymalizacji nie widać już właściwie żadnej poprawy:



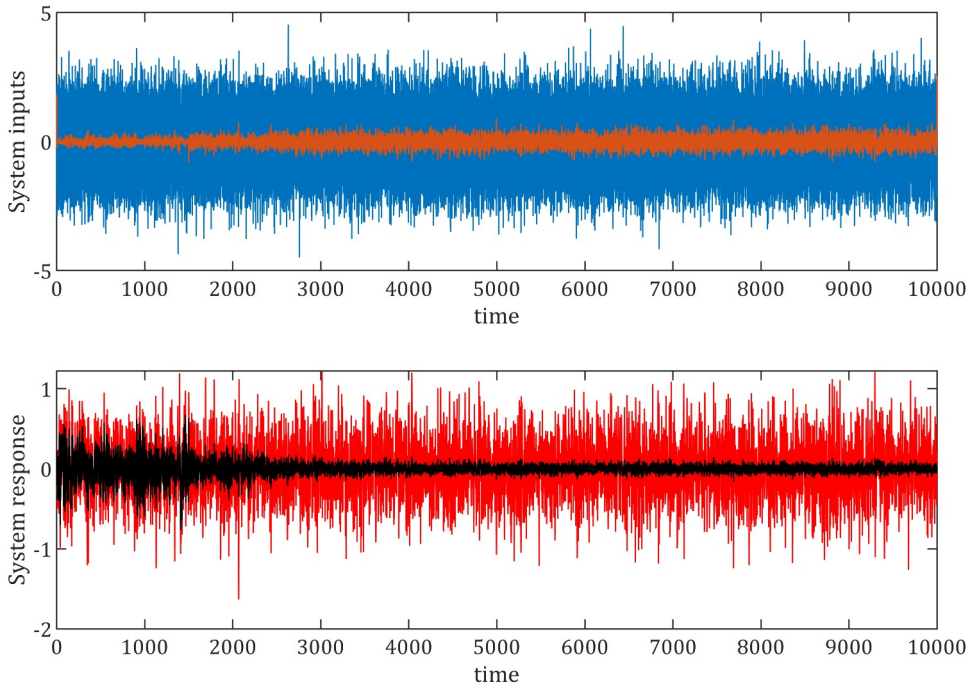
Rys 3 – Krzywa konwergencji algorytmu optymalizacji sterownika proporcjonalnego

Zadanie 2.2: Uruchom pętlę sterowania swoim ***indywidualnym*** obiektem z zastosowaniem sterownika optymalizowanego algorytmem 1+1. Przetestuj działanie optymalizacji przy regulatorze P oraz PD. Zapisz otrzymane krzywe konwergencji

Problemów jest tutaj kilka jednocześnie. Po pierwsze losowe wejścia wymagają prawdopodobnie uśrednienia w większym czasie, po drugie, krok optymalizacji i wartości początkowe nie są dobrane właściwie. Po trzecie, być może warto zastosować tutaj regulator badziej zaawansowany (z członem różniczkującym, całkującym lub obydwoma?), Po czwarte może warto pozwolić regulatorowi na generowanie sygnałów kontrolnych o większej amplitudzie – porównywalnej do amplitudy wejścia zakłócanego? Po piąte wreszcie horyzont czasowy symulacji musi być prawdopodobnie znacznie większy. Po kilku zmianach dla obiektu testowego możliwe jest uzyskanie wyników jak na Rys. 4 i 5.



Rys 4 – Krzywa konwergencji algorytmu optymalizacji sterownika po lepszej konfiguracji metaparametrów sterownika i optymalizatora



Rys 5 – Wyniki sterowania z zastosowaniem sterownika optymalizowanego algorytmem 1+1

Zadanie 2.3 – Dobierz metaparametry sterownika i regulatora, sprawdź jak skuteczne sterowanie uda Ci się uzyskać dla Twojego ***indywidualnego*** obiektu sterowania w długim horyzoncie czasowym (100 000 iteracji, 10 000 sekund). Zapisz wyniki cząstkowe w formie krzywych konwergencji oraz przypisanych do nich nastawów regulatora.

Zadanie 2.4 – Czy optymalizacja byłaby dużo łatwiejsza, gdyby sygnał zakłócający był również pod naszą kontrolą? Niech sygnał zakłócający będzie równy 0 z wyjątkiem pierwszej iteracji sterowania w każdym nowym cyklu, niech wtedy będzie równy 1. Sprawdź jakie wyniki optymalizacji można uzyskać w takich warunkach, a następnie przetestuj tak zoptymalizowany regulator PID w zadaniu sterowania obiektem wymuszanym za pomocą szumu.

Zadania dodatkowe i rozszerzające

Zadanie 2.5 - Nasz regulator "widzi" jedynie wyjście. Czy zadanie byłoby prostsze czy trudniejsze, gdyby regulator miał dostęp do wszystkich elementów wektora x? Przetestuj działanie takiego regulatora, posiadającego odpowiednio po dwie stałe P, I oraz D, porównaj z odpowiadającymi regulatorami PD oraz P. Czy uzyskiwane wyniki są lepsze czy gorsze niż regulator znany z zadań 2.2 – 2.3? Dlaczego? Zadanie wykonaj na podstawie obiektu ze swojego ***indywidualnego*** zadania

Zadanie 2.6 – Jaka jest górna granica jakości regulacji w tym przypadku? Wykonaj zadanie 2.3 z zastosowaniem algorytmu genetycznego, zastosuj dużą nadmiarowość metody (bardzo dużo iteracji optymalizatora, długi czas uśredniania). Porównaj otrzymane wyniki z wynikami zadania 2.3.

Zadanie 2.7 – Rozwiąż przedstawione zadanie na sposób klasyczny. Zidentyfikuj transmitancję korzystając z rozwiązania z instrukcji 1 a następnie zaprojektuj dla zidentyfikowanej transmitancji odpowiedni regulator. Przetestuj działanie tego regulatora poprzez porównanie go z regulatorem wykorzystywanym w zadaniu 2.3 (wykorzystaj go do sterowania obiektem w ramach naszej symulacji).