Teoria Sterowania

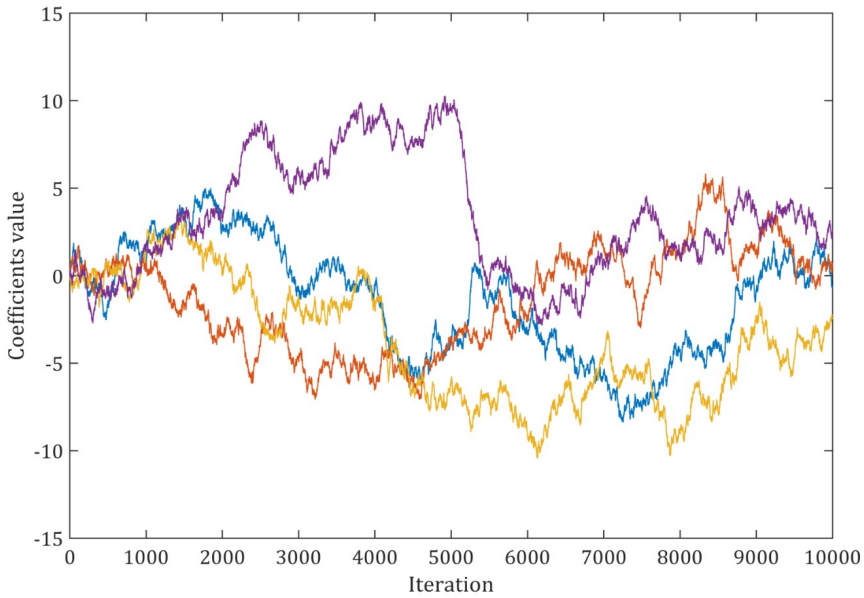
Sterowanie Inteligentne: Sterowanie adaptacyjne

1. Wprowadzenie

Niniejsza instrukcja opiera się na wynikach instrukcji sterowanie inteligentne 2 i 3 – punktem wyjścia są przygotowane w ramach w.w. ćwiczeń kontroler PID ewoluowany z zastosowaniem metody 1+1, oraz kontroler predykcyjny, neuronowy, zbudowany w ramach instrukcji 3. Oba te kontrolery będą w tym ćwiczeniu przetestowane w zadaniu adaptacyjnym, w którym obiekt sterowania zmienia się w czasie.

1. Losowe błądzenie jako definicja systemu zmiennego w czasie

System wczytywać i wykorzystywać będziemy dokładnie tak samo, jak w instrukcjach 2 i 3. Jedyna różnica polega na sposobie wywołania metody. W każdej iteracji sterowania do funkcji przekazywać będziemy inny zestaw wartości modyfikujących parametry wewnętrzne sterowanego obiektu. Wartości te dane są w pliku randomWalk o numerze odpowiadającym numerowi indywidualnego zadania. Przykładowe wartości mogą wyglądać np. tak (Rys1):



Rys 1 – zmiany wartości współczynników w funkcji numeru iteracji

Wczytanie obiektu do indywidualnego zadania

Wczytywać obiekt będziemy tak samo jak w poprzednich dwóch instrukcjach. Na zielono zaznaczony jest komponent definiujący zadanie indywidualne, na żółto informacja o tym, jakiego typu przepis na zmiany wczytujemy. Rozpoczniemy od przepisu interpolowanego, tj. posiadającego stukrotnie wolniejsze zmiany:

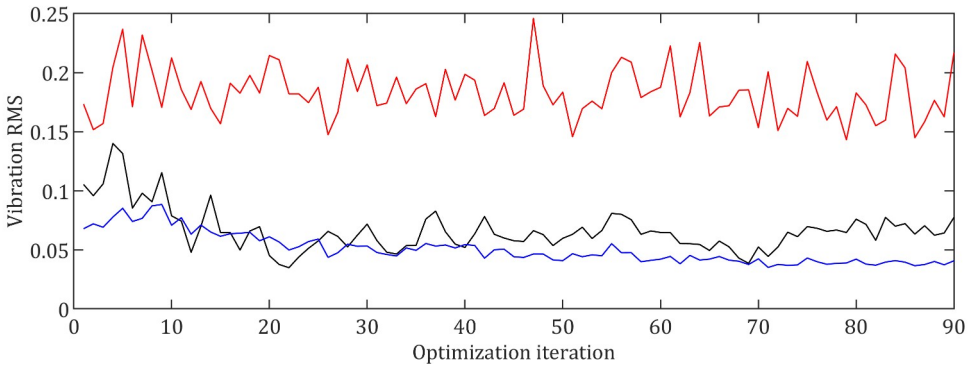
```
MyObject = str2func('StateSpaceNonStationary_rank2');
load randomWalk_5_interpolated
[A,B,C,D] = MyObject(randomWalk(1,:))
```

Sterowanie adaptacyjne z zastosowaniem sterownika PID oraz optymalizatora 1+1

Zmodyfikujmy kod z instrukcji 2 (adaptacyjny kontroler ewolucyjny), w celu spowodowania, by model zaczął się zmieniać w czasie, w trakcie adaptacji. Jedyna zmiana, jaką w tym celu będziemy musieli wprowadzić, to powiązanie parametrów obiektu z aktualnym numerem iteracji, poprzez wstawienie następującej linijki na początek głównej pętli:

```
[A,B,C,D] = MyObject(randomWalk(k,:))
```

Sprawdźmy teraz efekt sterowania obiektem stacjonarnym oraz zmiennym w czasie przez tak samo ewoluowany kontroler (Rys 2):



Rys 2 - Efekt sterowania obiektem stacjonarnym (niebieski) i niestacjonarnym (czarny) przez przez kontroler optymalizowany ewolucyjnie, opracowany w ramach instrukcji 2.

Aby umożliwić kontrolerowi podążanie za zmianami obiektu, musimy zmodyfikować sposób, w jaki optymalizator znajduje punkt, od którego należy wykonać kolejny krok optymalizacji. Do tej pory używaliśmy następującej linii:

```
[m, in] = min([ControlResults(:).Quality]);
```

Linia ta pozwalała na znalezienie indeksu (in) odpowiadającego historycznie najlepszemu osiągniętemu wynikowi. Teraz będziemy chcieli znaleźć wynik najmniejszy w pewnym określonym oknie. Zdefiniujmy na początku kodu to okno za pomocą zmiennej *HistoricalQualityWindow*, przypiszmy jej początkowo wartość 10 a następnie zamiast powyższej linii kodu zastosujmy następujący kod:

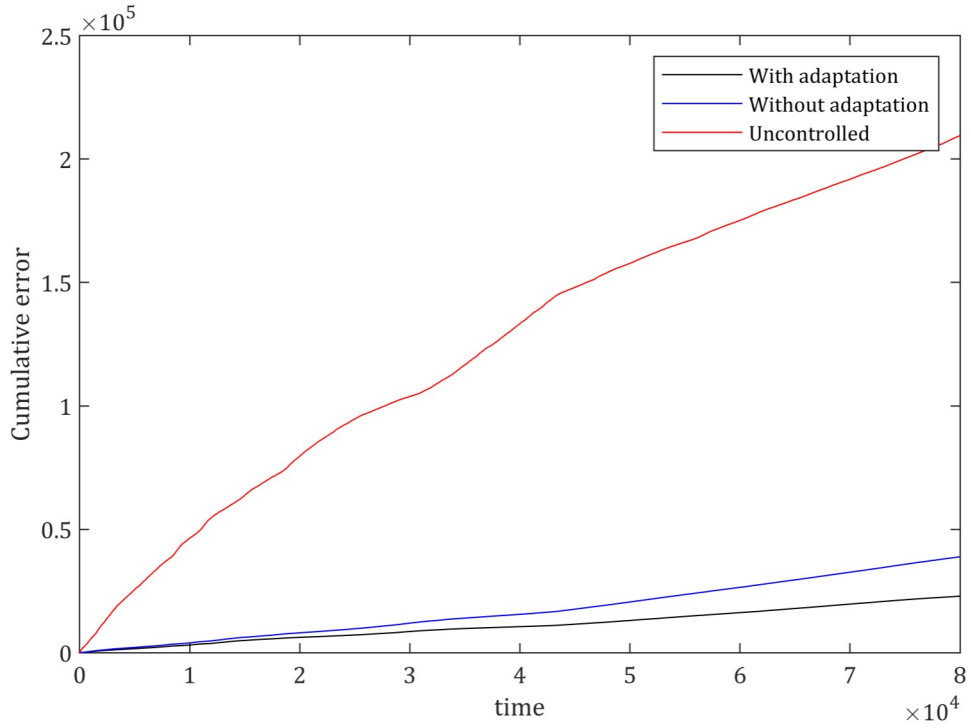
```
% [m, in] = min([ControlResults3(:).Quality]);
QualityVector = [ControlResults(:).Quality];
[~,order] = sort(QualityVector);
in = order(find(order > ControlCycleNumber-HistoricalQualityWindow,1))
```

Kod ten pozwala na posortowanie wektora zawierającego historyczne jakości a następnie znalezienie pierwszego elementu tego wektora, który jest większy od numeru iteracji pomniejszonego o długość naszego okna, czyli znajduje się wewnątrz okna.

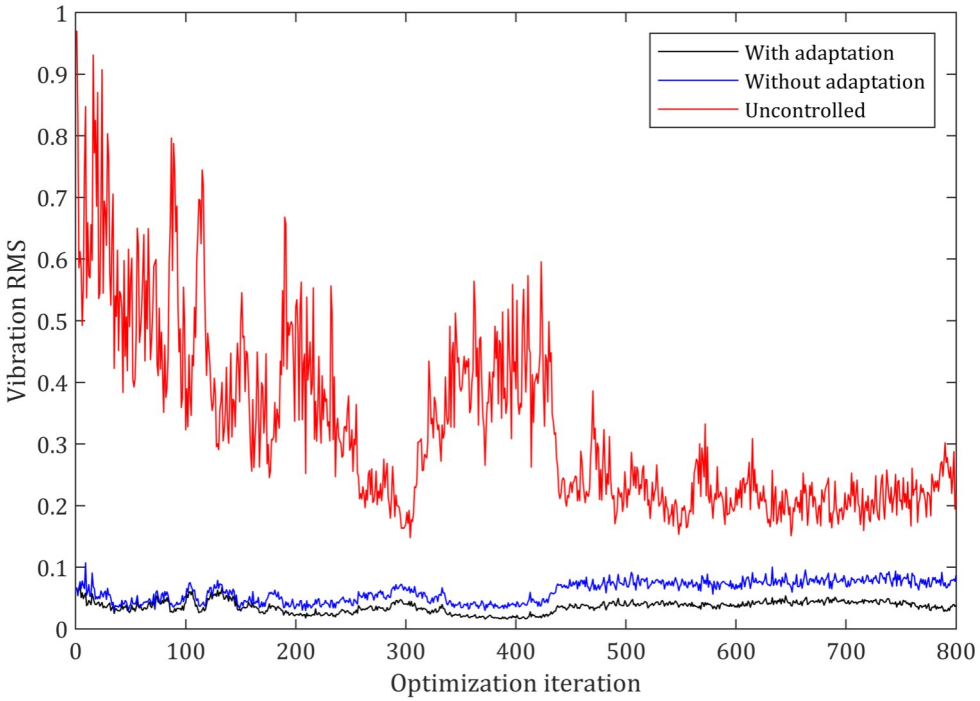
Aby ocenić skuteczność działania takiej modyfikacji kodu można wykonać pętlę sterowania trzykrotnie: z optymalizacją adaptacyjną, z optymalizacją globalną (jak w instrukcji 2) oraz bez sterowania. Jeśli efekty działania tych trzech symulacji zapisane zostaną odpowiednio do wektorów *y_adacon, y_con* oraz *y_ref*, można wyświetlić skumulowany poziom błędu dla wszystkich trzech podejść:

```
figure(71);
% Cumulative errors:
plot(T,cumsum(abs(y_adacon)),'k'); hold on
plot(T,cumsum(abs(y_con)),'b'); hold on
plot(T,cumsum(abs(y_ref)),'r'); hold on
legend('With adaptation','Without adaptation','Uncontrolled')
xlabel('time'); ylabel('Cumulative error')
```

Wykres taki może wyglądac np. tak jak na Rys 3, a odpowiadające mu krzywe konwergencji, tworzone analogicznie jak w instrukcji 2 – jak na Rys. 4.



Rys 3 – Zakumulowana suma błędów dla trzech algorytmów sterowania dla 80 000 sekund symulacji



Rys 4 – Krzywe konwergencji dla trzech przypadków testowych w długim horyzoncie czasowym. Tutaj wyniki przedstawiono dla czasu T(end) = 80 000 s

Zadanie 4.1: Przetestuj działanie adaptacyjnego kontrolera 1+1 w sterowaniu niestacjonarnym systemem modyfikowanym z zastosowaniem swojego ***indywidualnego*** przepisu. Zastosuj interpolowany przepis na zmiany obiektu sterowania w czasie, czas symulacji równy 50 000s i krok symulacji równy 0.2s, *LengthOfControlCycle* = 1000 oraz *HistoricalQualityWindow* = 50.

Zadanie 4.2: Porównaj kontroler z zadania 4.1z kontrolerem z instrukcji 2 działającym na tym samym problemie (tj. dla zmieniającego się w czasie systemu, przez tyle samo iteracji oraz z takimi samymi metaparametrami algorytmu) oraz z przypadkiem braku kontroli. Zaprezentuj krzywe konwergencji oraz zakumulowane sumy błędów

Zadanie 4.3: Przetestuj działanie adaptacyjnego kontrolera 1+1 w sterowaniu niestacjonarnym systemem. Jakie musi być minimalne okno sterowania (*LengthOfControlCycle*) i okno z którego wybieramy najlepszy kontroler (*HistoricalQualityWindow*), aby kontroler miał szansę zadziałać zgodnie z oczekiwaniami? Jakie muszą być powyższe parametry, aby kontroler działał powtarzalnie i zapewniał skuteczną adaptację do zmian w systemie w większości przypadków testowych? Oceń możliwość zastosowania tego kontrolera do problemu zdefiniowanego za pomocą przepisu standardowego (nie interpolowanego).

Sterowanie adaptacyjne z wykorzystaniem sterownika neuronowego, predykcyjnego

Tym razem korzystać będziemy początkowo z nieinterpolowanego przepisu na zmiany, wykorzystać więc należy następującą linię (z numerem odpowiadającym numerowi naszego indywidualnego zadania):

```
MyObject = str2func('StateSpaceNonStationary_rank2');
load randomWalk_5
```

Zasada działania sterowania adaptacyjnego z zastosowaniem predykcyjnej sieci neuronowej będzie dokładnie taka sama, jak sterownika budowanego na poprzednich zajęciach. Jedyna modyfikacja, jaką trzeba będzie dodać do kodu polegać będzie na cyklicznym douczaniu sieci w celu dostosowania jej do nowego stanu systemu. Potrzebować zatem będziemy dodatkowej zmiennej *TrainingWindow* która określi nam długość okna po jakim następuje douczenie sieci neuronowej. Początkiem sterowania, jak poprzednio, będzie wymuszenie systemu za pomocą szumu losowego podawanego na wejście sterujące. Zauważmy, że w każdej iteracji do funkcji generującej obiekt przekazywany jest inny zestaw modyfikatrów (zaznaczony w kodzie na żółto):

```
TrainingWindow = 100;

for k = 1:TrainingWindow
    [A,B,C,D] = MyObject(randomWalk(k,:));
    dot_X(:,k) = A*x(:,k) + B*Inputs(k,:)';
    y(:,k) = C*x(:,k);
    x(:,k+1) = x(:,k) + TimeStep*(dot_X(:,k));
end
```

Po uruchomieniu tego kodu dysponujemy pierwszymi danymi do uczenia sieci neuronowej. Dysponujemy również iteratorem k, którego wartość po zakończeniu pętli jest równa *TrainingWindow* – od tej zatem iteracji rozpoczynać będziemy kolejny cykl sterowania. Nasza pętla sterowania będzie pętlą *while* która sprawdzać będzie, czy kolejny cykl sterowania zmieści się jeszcze w zadanym wektorze T:

```
while k+TrainingWindow<length(T)
      ...
end
```

Wewnątrz tej pętli będziemy cyklicznie wykonywali następujące czynności:

1. Przypisanie danych z poprzedniego okna uczącego do macierzy uczących dla sieci neuronowej:

```
    TrainingInputs = [Inputs(k-TrainingWindow+1:k-1,:)';x(:,k-TrainingWindow+1:k-1)];
    TrainingTargets = y(k-TrainingWindow+2:k);
```

2. Wytrenowanie lub dotrenowanie sieci neuronowej
3. Wykonanie sterowania przez ilość iteracji równą długości okna treningowego. Do tego kroku zastosować można następujący kod:

```
    for z = k:k+TrainingWindow-1
        k = k+1;
        [A,B,C,D] = MyObject(randomWalk(k,:));
        NetData = [ones(size(PossibleControlsVector))*Inputs(k,1); ...
                   PossibleControlsVector;x(:,k)*ones(size(PossibleControlsVector))];
        PossibleResults = sim(OurNet,NetData);
        [mi in] = min(abs(PossibleResults));
        Inputs(k,2) = PossibleControlsVector(in);
        dot_X(:,k) = A*x(:,k) + B*Inputs(k,:)';
        y(:,k) = C*x(:,k);
        x(:,k+1) = x(:,k) + TimeStep*(dot_X(:,k));
    end
```

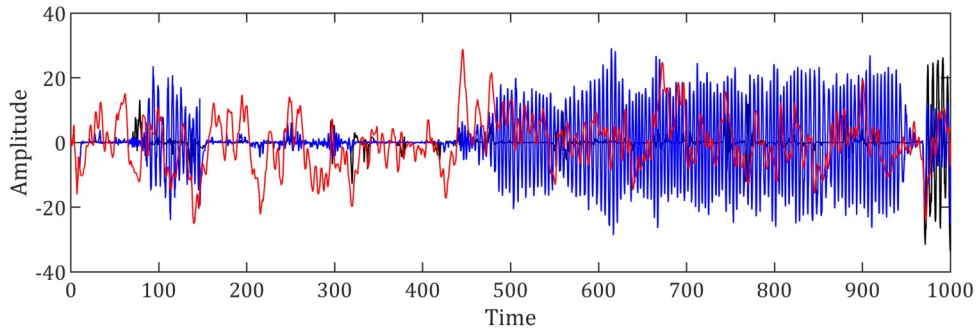
4. Możemy jeszcze pokusić się o umieszczenie jakiegoś wskaźnika postępu naszego kodu:

```
clc
fprintf('Done %d out of %d iterations',k,length(T))
```

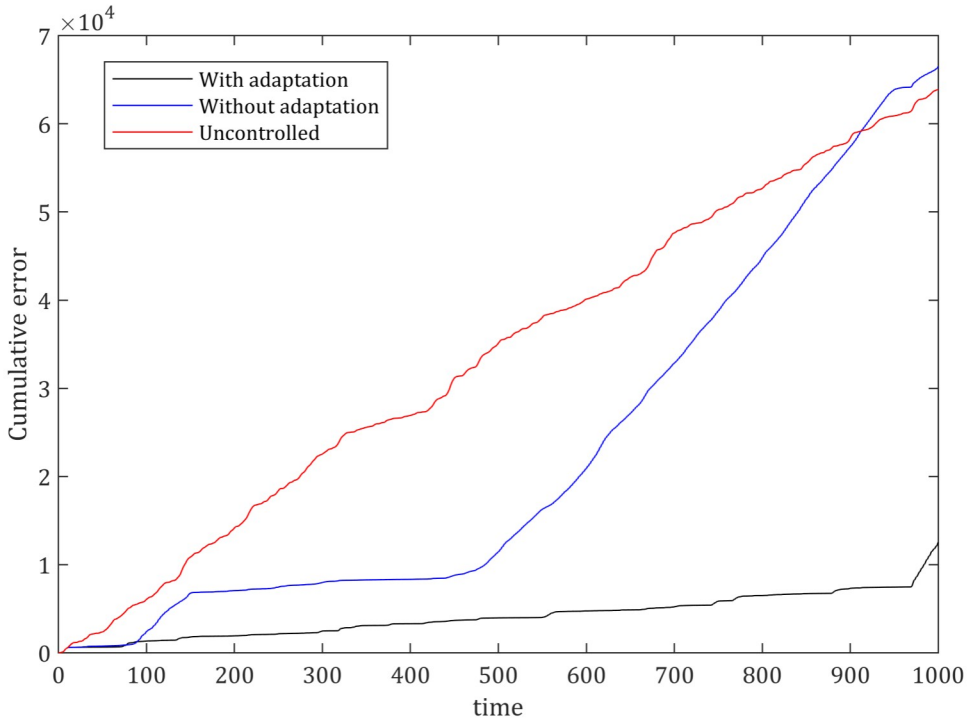
Po poprawnym zaimplementowaniu powyższego planu powinniśmy otrzymać wynik sterowania w postaci wektora *y* równego długością wektorowi *T*. Zapiszmy w takim razie wektor *y* jako wektor *y_adacon* i sprawdźmy wynik tego sterowania:

```
figure(3)
subplot(2,1,1);
plot(T,Inputs);
xlabel('time'); ylabel('System inputs with control')
subplot(2,1,2);
plot(T,y_adacon','k'); hold on
```

Brakuje nam jeszcze punktu odniesienia, do którego moglibyśmy porównać skuteczność naszego sterownika. Niech pierwszym takim punktem odniesienia będzie tak samo zmieniający się układ, zakłócany takim samym szumem, ale po początkowym przypadkowym wymuszeniu nie kontrolowany w ogóle, a drugim punktem odniesienia tak samo zmieniający się układ, zakłócany takim samym szumem, w którym sieć neuronowa po początkowym wyuczeniu na podstawie jednego okna treningowego nie będzie już więcej douczana. Przedstawmy te trzy wyniki na jednym wykresie (Rys 5) wraz ze zakumulowaną sumą błędów (Rys 6):



Rys 5 – Efekt sterowania z zastosowaniem neuronowego predykcyjnego kontrolera z adaptacją (czarny), bez adaptacji (niebieski) oraz braku kontroli (czerwony)



Rys 6 – zakumulowane sumy błędów dla sterowania z zastosowaniem neuronowego predykcyjnego kontrolera z adaptacją (czarny), bez adaptacji (niebieski) oraz braku kontroli (czerwony)

Zadanie 4.4: Zaimplementuj cykliczne douczanie do swojego predykcyjnego kontrolera neuronowego. Przetestuj działanie systemu na przykładzie problemu drugiego rzędu, modyfikowanego z zastosowaniem swojego ***indywidualnego*** przepisu w wersji interpolowanej. Wykonaj 10 000 iteracji sterowania (tzn. 1000 sekund czasu symulacji z krokiem 0.1s).

Zadanie 4.5: Porównaj cyklicznie douczany predykcyjny kontroler neuronowy z zadania 4.4 z kontrolerem uczonym jednokrotnie (na początku symulacji) oraz brakiem kontroli. Zaprezentuj wyjścia z systemu we wszystkich trzech przypadkach oraz skumulowane sumy błędów. Wykonaj to zadanie dla przepisu interpolowanego (*randomWalk_numer_interpolated*) oraz standardowego (*randomWalk_numer*). Wyciągnij wnioski.

Zadania dodatkowe i rozszerzające:

Zadanie 4.6: Zoptymalizuj parametry neuronowego predykcyjnego kontrolera pod kątem zapewnienia maksymalnej skuteczności sterowania (uwzględniając parametry sieci, wielkość okna uczenia i zakres wartości z których losowane są możliwe sterowania). Do optymalizacji wykorzystaj pierwsze 500 sekund symulacji z krokiem 0.1s dla indywidualnego problemu nieinterpolowanego. Wykonaj następnie test na ostatnich 500 sekundach symulacji i zweryfikuj poprawność optymalizacji.

Zadanie 4.7: Przetestuj adaptacyjny algorytm 1+1 w zadaniu sterowania sytemem 4 rzędu. Jakiego rodzaju problemy pojawiają się w tym przypadku? Dobierz metaparametry metody tak, aby zmaksymalizować skuteczność sterowania mierzoną statystycznie (na podstawie szeregu uruchomień zadania).

Zadanie 4.8: Adaptacyjny algorytm 1+1 w zadaniu sterowania systemem 4 rzędu będzie wykazywał tendencje do destabilizacji obiektu – zwłaszcza dla wybranego dużego kroku optymalizacji. Zmodyfikuj kod metody w ten sposób, aby wykrywać przypadek destabilizacji gdy tylko się pojawi (np. na podstawie amplitudy drgań zwiększającej się zbyt szybko (powyżej pewnego progu w stosunku do poprzedniego cyklu) przy jednoczesnym zwiększaniu amplitudy sygnału sterującego) i reagować poprzez:

a) Wyłączenie kontrolera na ten cykl symulacji oraz przypisanie mu wartości *Quality* równej *Inf*
b) Zmniejszenie wzmocnienia kontrolera o 50%

Który z tych dwóch przypadków pozwala na większą skuteczność sterowania (mierzoną statystycznie)?

Zadanie 4.9: Wykonaj zadania 4.4 i 4.5 korzystając z rekurencyjnej sieci neuronowej: Naucz sieć neuronową do przewidywania wartości y(t+1) oraz wektora x(t+1) po przekazaniu x(t), y(t) oraz Inputs(t). Następnie w sterowaniu przekazuj do sieci wartości wyjść sieci z wcześniejszej iteracji, bez przekazywania faktycznych wartości wektora x.

Zadanie 4.10: Wykonaj zadania 4.4 i 4.5 modyfikując zadanie sterowania tak, aby sterowany system stał się systemem nieliniowym. Zmodyfikuj wyjście z obiektu w ten sposób, by zamiast:

```
y(:,k) = C*x(:,k);
```

wstawić linię

```
y(:,k) = C*(sqrt(abs(x(:,k))).*sign(x(:,k)));
```

Porównaj w tym zadaniu działanie sieci o jednej warstwie ukrytej [10] z siecią o dwóch warstwach ukrytych [5,5]. W którym przypadku sterowanie jest skuteczniejsze?

Zadanie 4.11: Wykonaj zadania 4.4 i 4.5 modyfikując zadanie sterowania tak, aby sieć neuronowa nie była cyklicznie uczona od zera, ale jedynie douczana (tzn. Linię "newff" wykonaj tylko raz przy pierwszym uczeniu sieci. Porównaj tak przygotowany kontroler z tym z zadań 4.4 i 4.5.