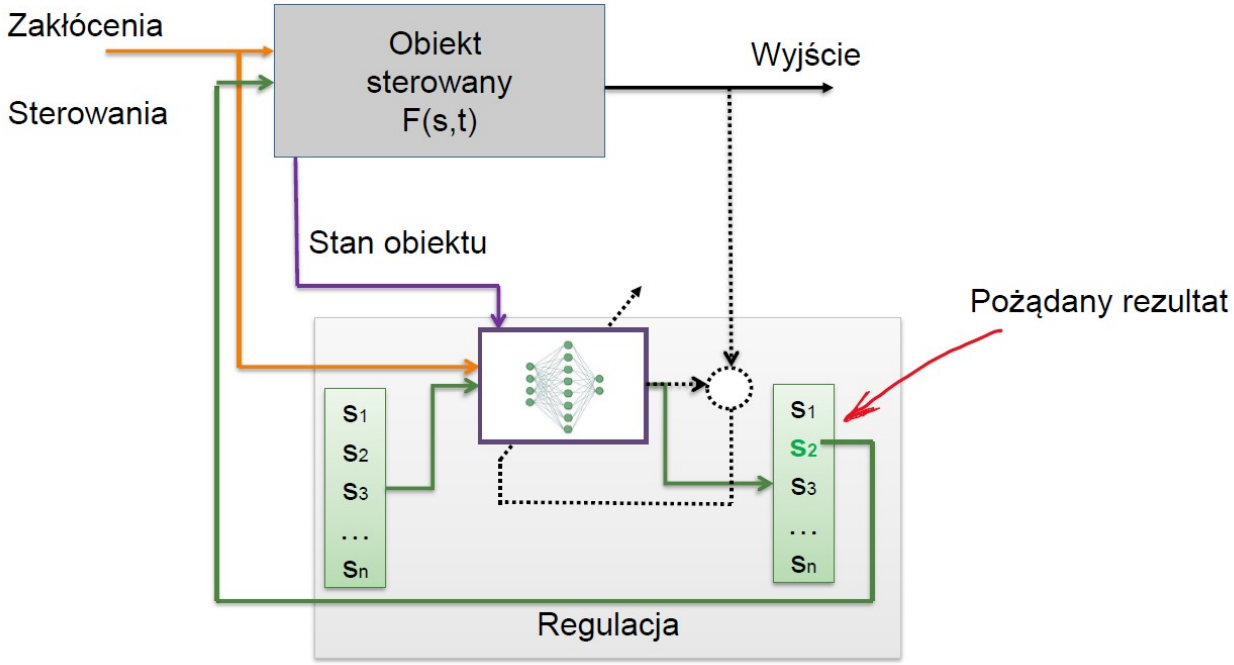
Teoria Sterowania

Sterowanie Inteligentne: *Artificial Neural Network Model Predictive Control*

1. Wprowadzenie

Zaprezentowana w niniejszej instrukcji metoda ANN - Model Predictive Control (ANN-MPC) działa według schematu przedstawionego na Rys. 1:



ys 1 – Schemat ANN-MPC

Rejestrowane są wymuszenia i stany obiektu sterowania a następnie wykonywane jest zagadnienie predykcyjne będące z punktu widzenia uczenia maszynowego zagadnieniem regresji. Sieć neuronowa uczona jest odpowiadania na pytanie

Jak zachowa się obiekt w kolejnej chwili czasowej, biorąc pod uwagę jego stan w danej chwili oraz wzbudzenie?

Jeśli taka sieć neuronowa odpowiadać będzie z dużą dokładnością, można ją następnie wykorzystać do sterowania, poprzez iteracyjne zadawanie jej pytania:

Jak zachowałby się obiekt w tym stanie, gdybym zastosował wymuszenie X?

Poprzez podstawienie wielu możliwych sterowań za X uzyskujemy szereg odpowiedzi, które następnie możemy porównać ze względu na wybrane kryterium jakościowe i wybrać taki X, który maksymalizuje skuteczność sterowania.

Definicja indywidualnego zadania

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(1,:))
```

Uczenie sieci neuronowej do MPC

Niech dany będzie obiekt zdefiniowany następująco:

```
clear all
close all
clc

% Plant definition:
load randomWalk_2
MyObject = str2func('StateSpaceNonStationary_rank2');
[A,B,C,D] = MyObject(randomWalk(1,:))
```

Warto zauważyć, że obiekt ma dwa wejścia i jedno wyjście. Jedno z wejść obiektu jest wejściem sterującym a drugie z wejść przekazuje na obiekt zakłócenia na które nie mamy wpływu, ale które możemy mierzyć. Celem sterowania będzie uzyskanie na wyjściu sygnału 0 niezależnie od obecności zakłóceń.

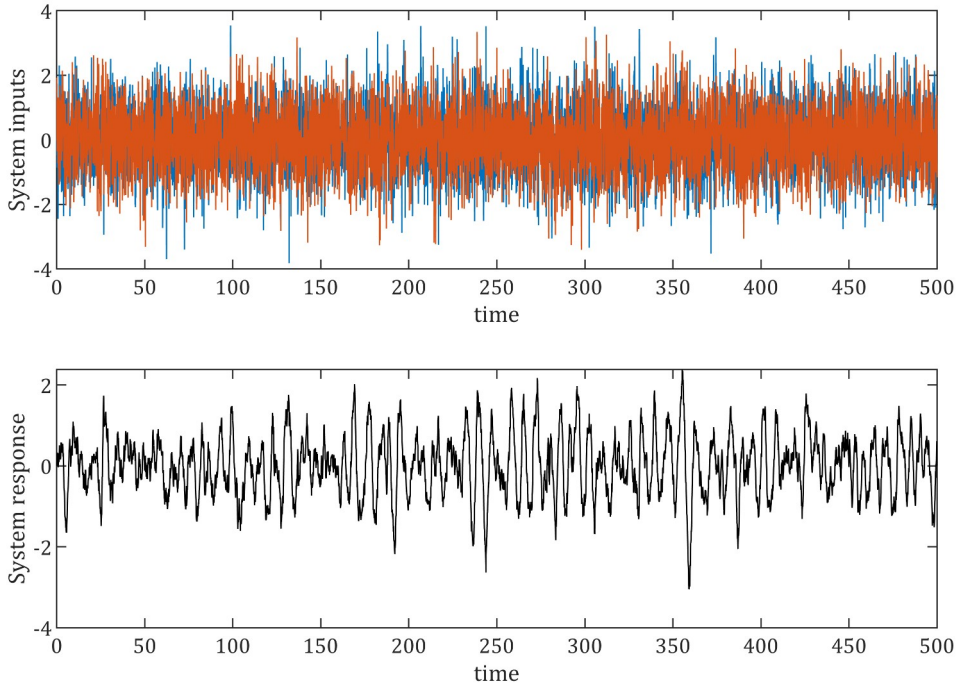
Niech dany będzie wektor czasowy oraz macierz wejść. Początkowo oba wejścia (zakłóceń oraz sterowane) wypełnione są szumem losowym:

```
% Time vector and inputs definition:
TimeStep = 0.1;
T = [0:TimeStep:1000];
Inputs = randn(length(T),2);
```

Ze względu na fakt, że system, który docelowo będziemy budować, będzie wymagał sterowania iteracyjnie w każdej chwili czasowej, zmuszeni jesteśmy przygotować symulację "ręcznie" – bez korzystania z funkcji *lsim*:

```
x = zeros(rank(A),length(Inputs)); % State vector
y = zeros(1,length(Inputs));       % System output

for k = 1:length(T)-1
    y(:,k) = C*x(:,k);
    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')
```

Zachowanie naszego obiektu może wyglądać następująco (Rys 1):



Rys 1 – (a) sygnały wejściowe do obiektu oraz (b) zachowanie obiektu w czasie

Zastosujmy teraz uzyskane w wyniku symulacji dane do uczenia sieci neuronowej. Jako wejście do sieci będziemy chcieli przekazać wejścia obiektu oraz jego stany. Jako wyjście – reakcję obiektu w kolejnej chwili czasowej. Po przygotowaniu wektorów nauczymy prostą sieć neuronową:

```
TrainingInputs = [Inputs(1:length(T)-1,:)';x(:,1:length(T)-1)];
TrainingTargets = y(2:length(T));

NetworkSize = [10]; % Number of neurons in all the hidden layers
OurNet = newff(TrainingInputs,TrainingTargets,NetworkSize); % Net initialization
OurNet.TrainParam.time = 10; % Max time allowed [in sec]
% view(OurNet)  % Comment-it-out if you want to see the network's structure
OurNet = train(OurNet,TrainingInputs,TrainingTargets);
```

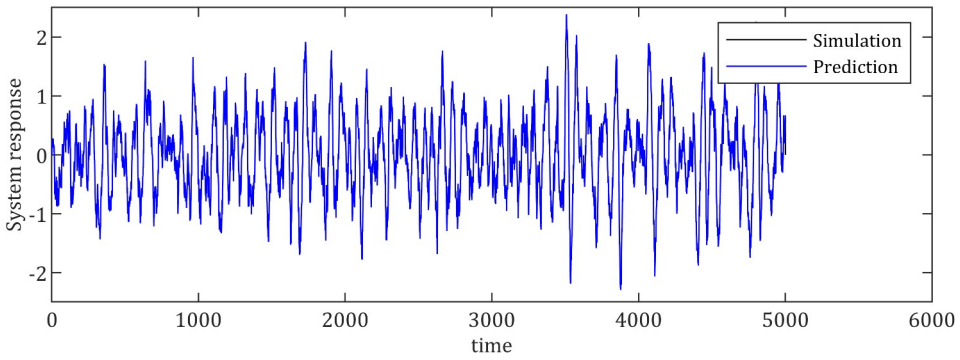
W celu sprawdzenia poprawności działania sieci wykonajmy kolejną symulację zachowania obiektu, tym razem wymuszanego innym sygnałem losowym. Uzyskane dane porównajmy z danymi przewidywanymi przez sieć. Zauważmy, że oczekiwane wyjścia są do sieci przekazywane z wyprzedzeniem – tzn. na podstawie danych z chwili czasowej k chcemy przewidzieć wyjście w chwili k+1.

```
clear Inputs x y dot_X
Inputs = randn(length(T),2);
x = zeros(rank(A),length(Inputs));
y = zeros(1,length(Inputs));

for k = 1:length(T)-1
    dot_X(:,k) = A*x(:,k) + B*Inputs(k,:)';
    y(:,k) = C*x(:,k);
    x(:,k+1) = x(:,k) + TimeStep*(dot_X(:,k));
end

InputData = [Inputs(1:length(T)-1,:)';x(:,1:length(T)-1)];
Targets = y(2:length(T));
```

```
Prediction = sim(OurNet,InputData);

figure(2)
plot(y','k'); hold on
plot([0,Prediction],'b');
xlabel('time'); ylabel('System response')
legend('Simulation','Prediction')
```

Uzyskane wyniki przedstawiają się obiecująco – zachowanie rzeczywiste i przewidywane są nie do odróżnienia :

\\



Rys 2 – Nałożone na siebie sygnały symulowane oraz przewidywane przez sieć

Również średni błąd nie jest duży. Po zastosowaniu poniższego kodu okazało się, że średni błąd to zaledwie 1.4100e-04

```
PredictionErrorValue = mean(abs(y(2:end)-Prediction))
```

Zadanie 3.1: Wykonaj kroki zdefiniowanej wyżej w instrukcji dla obiektu zdefiniowanego w swoim ***indywidualnym zadaniu o rzędzie 4*** Przetestuj działanie sieci i oceń jej skuteczność krzyżowo (sprawdzając wszystkie możliwości, razem 12 testów) ze względu na:

- Ilość danych wejściowych (przetestuj 100 i 1000 punktów danych)
- Krok symulacji (Przetestuj 0.1 oraz 0.01)
- Strukturę sieci (przetestuj struktury [5], [15], [5 5])

Co ma większy wpływ na poprawność wyników? Struktura sieci, ilość danych wejściowych czy krok symulacji?

Sterowanie z wykorzystaniem MPC

Nauczoną sieć neuronową wykorzystać można do odpowiadania na pytanie *Co by było, gdyby...?* ze względu na różne możliwe sterowania w każdej kolejnej iteracji symulacji. Dodajmy więc do kodu zakres, z którego generowane będą możliwości. Fragment zaznaczony na żółto powinien być aktualizowany w każdej iteracji pętli sterowania (tzn. W każdej iteracji powinniśmy generować nowy zestaw potencjalnych sterowań). 0 zawarte jest w wektorze po to, aby kontroler mógł zawsze zrezygnować ze sterowania w danej iteracji, jeśli nie jest potrzebne.

```
RangeOfInputs = 0.5;
InputsNumber = 10;

PossibleControlsVector = [0,[RangeOfInputs*randn(1,InputsNumber-1)]];
```

Dla każdej iteracji sterowania przygotujmy zatem wektor wejść, w którym sygnał na wejście zakłócane zdefiniowany jest losowo, a sygnał na wejście sterujące jest nadpisywany w taki sposób, aby każdorazowo wybierać sterowanie które minimalizuje poziom drgań:

```
    % We build input data for the net for one paticular moment in time:
    NetData =     [ones(size(PossibleControlsVector))*Inputs(k,1); ...
                  PossibleControlsVector; ...
                  x(:,k)*ones(size(PossibleControlsVector))];
    PossibleResults = sim(OurNet,NetData);
    % We minimize error in the next step:
    [mi in] = min(abs(PossibleResults));
    Inputs(k,2) = PossibleControlsVector(in);
```

A następnie zastosujmy to konkretne sterowanie w pętli:

```
    dot_X(:,k) = A*x(:,k) + B*Inputs(k,:)';
    y_con(:,k) = C*x(:,k);
    x(:,k+1) = x(:,k) + TimeStep*(dot_X(:,k));
```

W warunkach rzeczywistych, aby ocenić skuteczność sterowania należałoby wykonać testy statystyczne, zapewniając w miarę możliwości podobne wymuszenia ze sterowaniem aktywnym i nieaktywnym. W naszym przypadku, aby uprościć nieco procedurę wykonamy niewielkie oszustwo i wymusimy nasz obiekt ponownie, tym samym sygnałem losowym lecz bez sterowania:

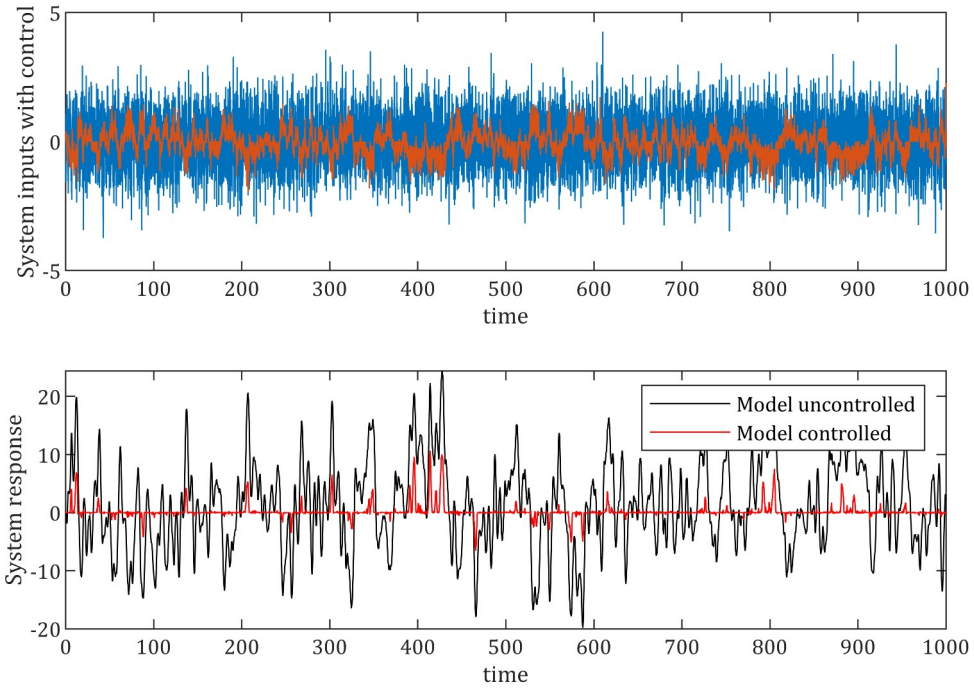
```
NewInputs = Inputs;
for k = 1:length(T)
    NewInputs(k,2) = 0;
    ref_dot_X(:,k) = A*ref_x(:,k) + B*NewInputs(k,:)';
    ref_y(:,k) = C*ref_x(:,k);
    ref_x(:,k+1) = ref_x(:,k) + TimeStep*(ref_dot_X(:,k));
end

VibrationReduction = ((rms(ref_y)-rms(y_con))/rms(ref_y))*100
```

Porównajmy zatem wyniki obu symulacji. W moim przypadku, po zastosowaniu poniższego fragmentu kodu uzyskałem rezultat widoczny na rys. 3 a stopień redukcji drgań wyniósł ok. 33%:

```
figure(3)
subplot(2,1,1);
plot(T,Inputs);
xlabel('time'); ylabel('System inputs with control')
subplot(2,1,2);
plot(T,ref_y','k'); hold on
plot(T,y_con','r'); hold on
legend('Model uncontrolled','Model controlled')
xlabel('time'); ylabel('System response')
```

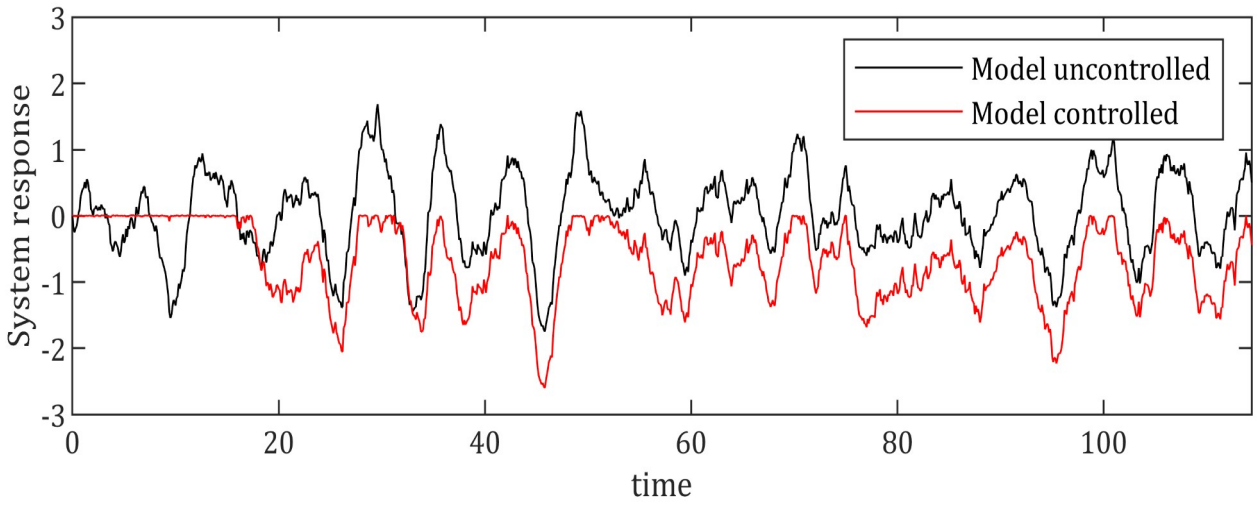
Rys 3

–



Sterowanie z zastosowaniem metody ANN – MPC dla układu o rzędzie 4.

Uwaga! Nie za każdym razem sterowanie "się uda". Częstym problemem, który w tym zadaniu można będzie zaobserwować, będzie fenomen zilustrowany na Rys. 4. Sterowanie okazuje się bardzo skuteczne przez kilka(dziesiąt) pierwszych iteracji, a następnie następuje wydarzenie, które powoduje znaczne pogorszenie wyników w stosunku do układu niesterowanego. Fenomen ten będzie się pojawiał częściej zwłaszcza w przypadku zbyt dużego zakresu wektora potencjalnych sterowań, przy czym wartość graniczna od której problem będzie wyraźnie obserwowalny będzie zależała od złożoności układu sterowanego oraz od zakresu zmienności wejścia na etapie uczenia sieci



Rys 4 – Zbliżenie na początek symulacji – system do pewnego punktu działa idealnie a potem przestaje działać. Dlaczego?

Zadanie 3.2: Zaimplementuj sterowanie MPC obiektem zdefiniowanym za pomocą swojego

indywidualnego zadania, o rzędzie 4. Porównaj wyniki z aktywnym i nieaktywnym kontrolerem dla krótkiego czasu symulacji (ok. 100 iteracji).

Zadanie 3.3: Znajdź wartości graniczne metaparametrów, których przekroczenie powoduje znaczące pogorszenie sterowania. Jakie musi być minimalne okno treningowe? Jaki może być dopuszczalny zakres wejść sterujących? Jaka może być minimalna wielkość sieci neuronowej zapewniająca poprawne działanie systemu? Wykonaj powyższe czynności dla zadań ***indywidualnych*** o rzędzie 2 i 4.

Zadania dodatkowe i rozszerzające

Zadanie 3.5: Zmodyfikuj sieć neuronową w taki sposób, aby przewidywała nie tylko wartości wektora *y* (wyjścia), ale również wektor `x(:,k+1)` (stany w kolejnej chwili czasowej). Następnie zaimplementuj sterowanie w ten sposób, że sieć dostaje na swoim wejściu zakłócenia, możliwe sterowania oraz wektor x przewidziany przez sieć w poprzedniej iteracji. Przetestuj następnie tak zbudowaną sieć rekurencyjną i porównaj ją z rozwiązaniem podstawowym (z zadania 3.2)