SPRAWOZDANIE 9

Modelowanie Systemów Dynamicznych

Temat ćwiczenia: Linearyzacja układów nieliniowych

Michał Midor, gr. 5, 10.12.2025 r. - Środa 13.15

Cel ćwiczenia:

Niniejsze ćwiczenia koncentrują się na praktycznym zastosowaniu linearyzacji systemów dynamicznych. Proces ten zostanie przeprowadzony na bazie opracowanego wcześniej nieliniowego modelu zbiornika z grzaniem i stałym odpływem. Do realizacji zadania wykorzystane zostaną natywne narzędzia środowiska MATLAB, takie jak polecenia `trim` (do wyznaczania punktu równowagi) oraz `lsim` (do symulacji odpowiedzi modelu liniowego). Kluczowym etapem będzie implementacja modelu nieliniowego w formie S-funkcji, co umożliwi jego poprawną integrację ze środowiskiem SIMULINK i efektywne wykorzystanie algorytmów linearyzujących.

Rozwiązanie zadań:

Rzeczywistość fizyczna jest z natury nieliniowa, co objawia się zarówno w charakterystykach komponentów (np. rezystancja w elektronice), jak i w samych równaniach opisujących dynamikę procesów. Choć modele nieliniowe najwierniej oddają zachowanie obiektów, większość klasycznych metod syntezy układów regulacji oraz narzędzi analitycznych opiera się na teorii układów liniowych. Wymusza to dokonania aproksymacji zachowania obiektu nieliniowego za pomocą modelu liniowego w procesie linearyzacji.

O ile do samej symulacji wcześniej wspomanianiego zbiornika wystarczały solvery równań różniczkowych, o tyle projektowanie zaawansowanych algorytmów sterowania wymaga wyznaczenia liniowego przybliżenia dynamiki tego procesu.

Należy pamiętać, że linearyzacja jest poprawna jedynie w bliskim sąsiedztwie konkretnego punktu pracy. Definiujemy go jako stan ustalony obiektu, w którym zmienne stanu (objętość i temperatura) oraz sygnały wejściowe (w szczególności moc grzałki) przyjmują stałe, pożądane wartości.

Proces uzyskiwania przybliżenia liniowego przebiega w następujących krokach:

1. Definicja dynamiki nieliniowej, czyli opracowanie opisu matematycznego w formie funkcji stanu (zgodnie z metodami z poprzednich zajęć), umożliwiającej integrację numeryczną. Kod źródłowy zawarty jest w pliku `zbiornik_stan.m` i wygląda następująco:

```
function dx = zbiornik_stan(t,x,wi,w,Ti,Q)
% Argumenty wejściowe:
% t - czas
% x - wektor stanu układu
% wi - dopływ
% w - wypływ
% Ti - temperatura cieczy dopływającej
% Q - moc dostarczana (grzanie)
%----------------------------- zmienne stanu ---------
x1 = x(1); % objętość
x2 = x(2); % temperatura
%------------------------------- parametry -----------
C = 4200; % ciepło właściwe [J/(Kg*K)]
ro = 1000; % gęstość [kg/m3]
%%----------------------------- równania stanu --------
dx1 = 1 / ro * (wi-w);
dx2 = wi * (Ti - x2) / (ro * x1) + Q / (ro * x1 * C);
dx = [dx1;dx2]; % pochodne stanu
```

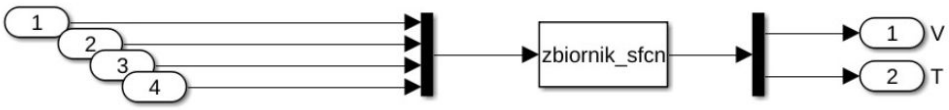
W tej funkcji tworzone są 2 równanie różniczkowe, które zostaną rozwiązane przez solver.

2. Przygotowanie S-funkcji adekwatnej dla modeli nieliniowego zapisanej w pliku `zbiornik_sfcn.m` oraz modelu w simulinku na jej podstawie w pliku `zbiornik_sys.mld`.

Wnętrze funkcji wygląda następująco:

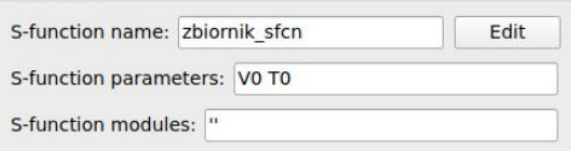
```
function [sys,x0,str,ts]=zbiornik_sfcn(t,x,u,flag,V0,T0)
switch flag % ustawiane na początku przez simulink na 0
case 0 % inicjalizacja
str = [];
ts = [0 0];
s = simsizes;
s.NumContStates = 2; % liczba stanów ciągłych
s.NumDiscStates = 0; % liczba stanów dyskretnych
s.NumOutputs = 2; % liczba wyjść
s.NumInputs = 4; % liczba wejść
s.DirFeedthrough = 0; % wejście nie przenosi się bezpośrednio na wyjście
s.NumSampleTimes = 1; % czas próbkowania, simulink próbkuje automatycznie
sys = simsizes(s);
x0 = [V0, T0]; % stan początkowy
case 1 % pochodne, stan normalnej pracy
wi = u(1);
w = u(2);
Ti = u(3);
Q = u(4);
sys = zbiornik_stan(t,x,wi,w,Ti,Q);
case 3 % wyjście
sys = x;
case {2 4 9}
sys =[];
otherwise
error(['unhandled flag =',num2str(flag)]);
end
```

Model w simulinku wygląda w ten sposób:



Sygnały sterujące (bloczki import) przekazywane przez multiplekser odpowiadają tym zdefiniowanym w S-funkcji: 1 - wi, 2 - w, 3 - Ti, 4 - Q i są rozdzielane przez demultiplekser do bloczków wyjściowych (outport).

Należy również ustawić następujące parametry początkowe:

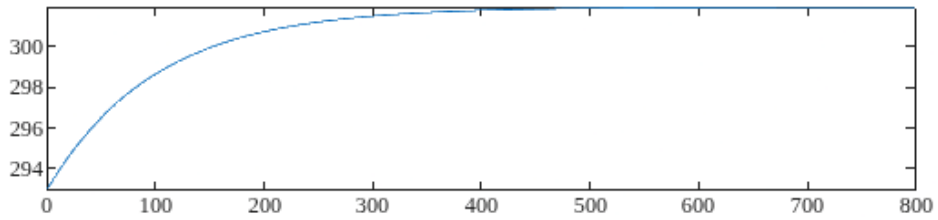


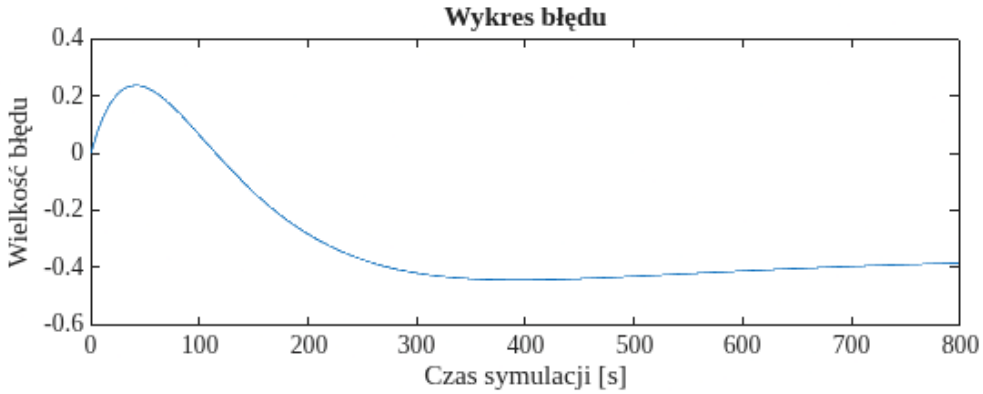
Przed użyciem funkcji trim() deklarowane są stałe oraz parametry modelu nielinowego, a także przeprowadzona została wizualizacja rozwiązań solverów ode45.

```
clear;
Q = 15000; %[W], dla 12000 nie działa poprawnie
w = 0.4; %[kg/s] wypływ
wi = 0.4; %[kg/s] dopływ
Ti = 293; %[K]

T0 = 293; %[K]
V0 = 0.04; %[m3]

T_zad = 303; % zadana temperatura [K]
V_zad = 0.04; % zadana objętość [m3]

display("Dopływ i wypływ równe")
```

```
"Dopływ i wypływ równe"
```

```
[t,x] = ode45(@zbiornik_stan, [0:799], [V0,T0], [], wi, w, Ti, Q);
x(:,2); % objętość | temperatura wyjścia
subplot(3,1,1)
plot(t, x(:,2))
```






```
display("Dopływ większy od wypływu")
```

```
    "Dopływ większy od wypływu"
```

```
w = 0.4; %[kg/s]
wi = 0.6; %[kg/s]
[t,x] = ode45(@zbiornik_stan, [0:799], [V0,T0], [], wi, w, Ti, Q);
subplot(3,1,2)
plot(t, x(:,2))

display("Dopływ mniejszy od wypływu")
```

```
    "Dopływ mniejszy od wypływu"
```

```
w = 0.4; %[kg/s]
wi = 0.3; %[kg/s]
[t,x] = ode45(@zbiornik_stan, [0:799], [V0,T0], [], wi, w, Ti, Q);
x(:, 1)
```

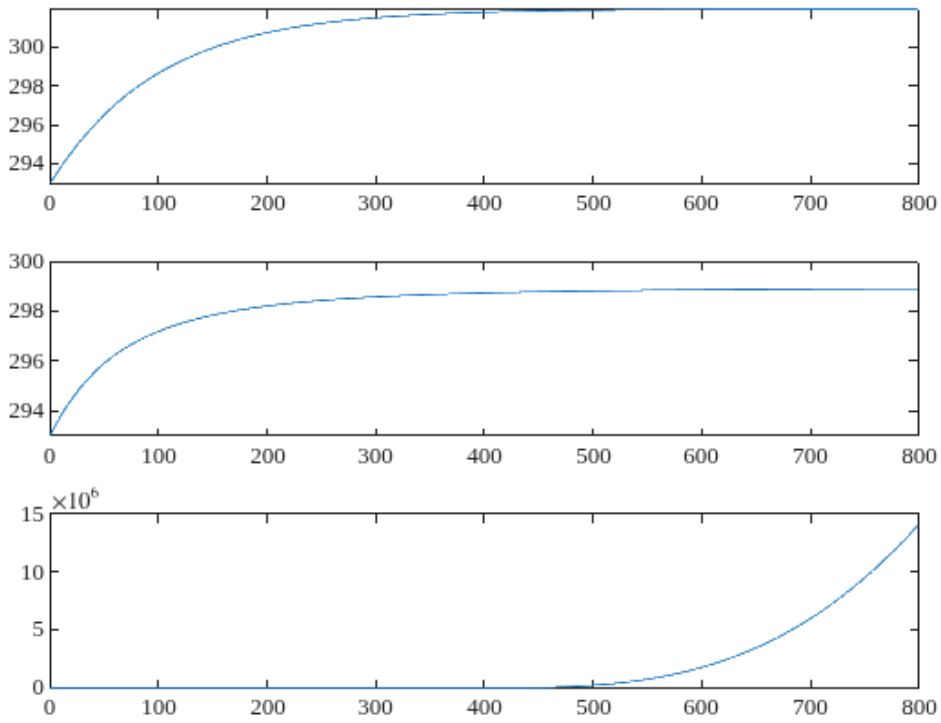
```
ans = 800×1
    0.0400
    0.0399
    0.0398
    0.0397
    0.0396
    0.0395
    0.0394
    0.0393
```

```
    0.0392
    0.0391
    0.0390
    0.0389
    0.0388
    0.0387
    0.0386
     ⋮
```

```
x(:,2)
```

```
ans = 800×1
10^7 ×
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
     ⋮
```

```
subplot(3,1,3)
plot(t, x(:,2))
```



Model dla ostatniego warunku matematycznie jest poprawny, ale fizycznie nie ma już sensu, ponieważ pojemność cieczy w zbiorniku staje się ujemna i prowadzi do błędnych wyników temperatury.

3. Obliczenie zestawu parametrów dla punktu pracy za pomocą funkcji trim().

Kluczowym etapem przygotowania modelu do linearyzacji jest odnalezienie stanu równowagi systemu przy użyciu polecenia `trim`. Algorytm ten, będący integralną częścią środowiska Simulink, wykonuje obliczenia optymalizacyjne, dążąc do zminimalizowania pochodnych stanu.

Pierwszym argumentem jest nazwa pliku `.slx` zawierającego strukturę blokową badanego zbiornika.

Kolejne parametry definiują wartości początkowe oraz restrykcje narzucone na stany (X), wejścia (U) i wyjścia (Y).

Szczególną uwagę należy zwrócić na ostatni argument wywołania. Jest to wektor sterujący zachowaniem solvera optymalizacyjnego. Jego drugi element określa dopuszczalny błąd poszukiwania rozwiązania. Standardowa wartość (10^-4) okazała się niewystarczająca dla dynamiki badanego obiektu, co wymusiło zmniejszenie wartości parametru.

```
% Wokół tych punktów trim będzie szukać stanu ustalonego
X0=[0.04;303]; % wektor stanu (w stanie ustalonym)
U0=[0.4;0.4;293;Q]; % wektor wejść [wi,w,Ti,Q]
Y0=[0.04;303]; % wektor wyjść jaki chcemy otrzymać

% Blokada wejść, wyjść i stanu
```

```
IX=[]; % wartości stanu nie są blokowane
IU=[1;2;3]; % pierwsza (wi), druga (w) i trzecia zmienna wejściowa (Ti) jest
zablokowana (na wartościach odpowiednio: 0.4,0.4,293)
IY=[1;2]; % pierwsza (V=0.04) i druga (T=303) zmienna wyjściowa jest
zablokowana
% Takie blokady, aby tylko Q się zmieniało.

dx0 = [];
idx = [];
options = [1, 1e-6];
display("Wywołanie funkcji trim().")
```

```
"Wywołanie funkcji trim()."
```

```
[x,u,y,dx,options] = trim('zbiornik_sys',X0,U0,Y0,IX,IU,IY,dx0,idx,options)
```

```
Warning: S-function block 'zbiornik_sys/S-Function' references obsolete level-1 MATLAB S-function
'zbiornik_sfcn'. Manually review the code and convert to level-2 MATLAB S-function if necessary.
For more information, see Convert Level-1 MATLAB S-Functions to Level-2.
f-COUNT     MAX{g}         STEP  Procedures
    8    0.0107143            1
   16    0.0234955            1  Hessian modified twice
   26    0.0264011         0.25  Hessian modified twice
   44    0.0264111     0.000977  Hessian modified twice
   55    0.0276964        0.125  Hessian modified twice
   70    0.0277711      0.00781  Hessian modified twice
   81    0.0289678        0.125  Hessian modified twice
   96    0.0290369      0.00781  Hessian modified twice
  105    0.0335164          0.5  Hessian modified twice
  125    0.0335178     0.000244  Hessian modified twice
  133    0.0394223            1  Hessian modified twice
  142    0.0396946          0.5  Hessian modified twice
  151    0.0398351          0.5  Hessian modified twice
  160    0.0399064          0.5  Hessian modified twice
  168    0.0399783            1  Hessian modified twice
  176    0.0399785            1  Hessian modified twice
  184    0.0399775            1  Hessian modified twice
  192    7.991e-05            1  Hessian modified
  200  3.16677e-13            1  Hessian modified
  201  3.16677e-13            1  Hessian modified
Optimization Converged Successfully
Active Constraints:
     1
     2
     3
     6
     7
     8
x = 2×1
    0.0400
  303.0000
u = 4×1
10^4 ×
    0.0000
    0.0000
    0.0293
    1.6800
y = 2×1
    0.0400
  303.0000
dx = 2×1
```

```
10^-12 ×
         0
    0.3167
options = 1×18
    1.0000    0.0000    0.0001    0.0000         0         0    1.0000         0 ···
```

```
display(u(4)); % czwarte wejście to moc Q
```

```
   1.6800e+04
```

```
Q = u(4);
```

Czwartym elementem wektora wejść jest zoptymalizowana przez algorytm moc grzałki Q.

4. Linearyzacja układu poprzez wywołanie funkcję 'linmod', która na podstawie otrzymanych powyżej parametrów 'x' oraz 'u' zwraca 4 macierze opisujące w przestrzeni stanów model liniowy badanego obiektu wokół punktu pracy. Najpierw przeprowadzona zostanie linearyzacja w stanie ustalonym.

```
display("Linearyzacja układu w stanie ustalonym")
```

```
    "Linearyzacja układu w stanie ustalonym"
```

```
[A,B,C,D] = linmod('zbiornik_sys', x, u) % linearyzacja układu w stanie
ustalonym
```

```
Warning: S-function block 'zbiornik_sys/S-Function' references obsolete level-1 MATLAB S-function
'zbiornik_sfcn'. Manually review the code and convert to level-2 MATLAB S-function if necessary.
For more information, see Convert Level-1 MATLAB S-Functions to Level-2.
A = 2×2
         0         0
   -0.0000   -0.0100
B = 2×4
    0.0010   -0.0010         0         0
   -0.2500         0    0.0100    0.0000
C = 2×2
    1.0000         0
         0    1.0000
D = 2×4
     0     0     0     0
     0     0     0     0
```

Korzystając z przestrzeni stanów można wygenerować po 2 transmitancje dla każdego wejścia, czyli zestaw 8 transmitancji.

```
display("Zestaw 8 transmitancji układu.")
```

```
    "Zestaw 8 transmitancji układu."
```

```
iu = [1, 2, 3, 4];
for i = iu
 [licz,mian] = ss2tf(A,B,C,D,i);
 printsys(licz,mian)
end
```

```
num(1)/den =
```

```
   0.001 s + 1e-05
   ---------------
     s^2 + 0.01 s

num(2)/den =

   -0.25 s - 8.0064e-15
   --------------------
       s^2 + 0.01 s

num(1)/den =

   -0.001 s - 1e-05
   ----------------
     s^2 + 0.01 s

num(2)/den =

    8.0064e-15
   ------------
   s^2 + 0.01 s

num(1)/den =

         0
   ------------
   s^2 + 0.01 s

num(2)/den =

      0.01 s
   ------------
   s^2 + 0.01 s

num(1)/den =

         0
   ------------
   s^2 + 0.01 s

num(2)/den =

   5.9524e-06 s
   ------------
   s^2 + 0.01 s
```

Kluczowym aspektem symulacji modelu liniowego jest fakt, że obliczenia są realizowane w dziedzinie zmiennych odchyłkowych. Reprezentują one różnicę pomiędzy wartością aktualną a wartością przyjętą w punkcie pracy.

Aby poprawnie zainicjalizować funkcję lsim(), konieczne jest zdefiniowanie wymuszeń w formie odchyłek. Rozmiar macierzy sterowań U ściśle zależy od wektora czasu t (liczba wierszy) oraz liczby sygnałów wejściowych (liczba kolumn). Ponieważ w rozważanym scenariuszu sygnały sterujące pozostają stałe i równe wartościom z punktu pracy, macierz odchyłek sterowań składa się wyłącznie z zer.

```
U = zeros(length(t), 4); % same zera bo sterowanie takie same jak w stanie
zerowym, czyli odchyłki w porównaniu do początku są zerowe
x_pocz = [0.04; 293];
x_ust = [0.04; 303];
x0 = x_pocz - x_ust; % warunek początkowy w zmiennych odchyłkowych
```

```
y = lsim(A,B,C,D,U,t,x0);

display('Model liniowy w stanie ustalonym')
```

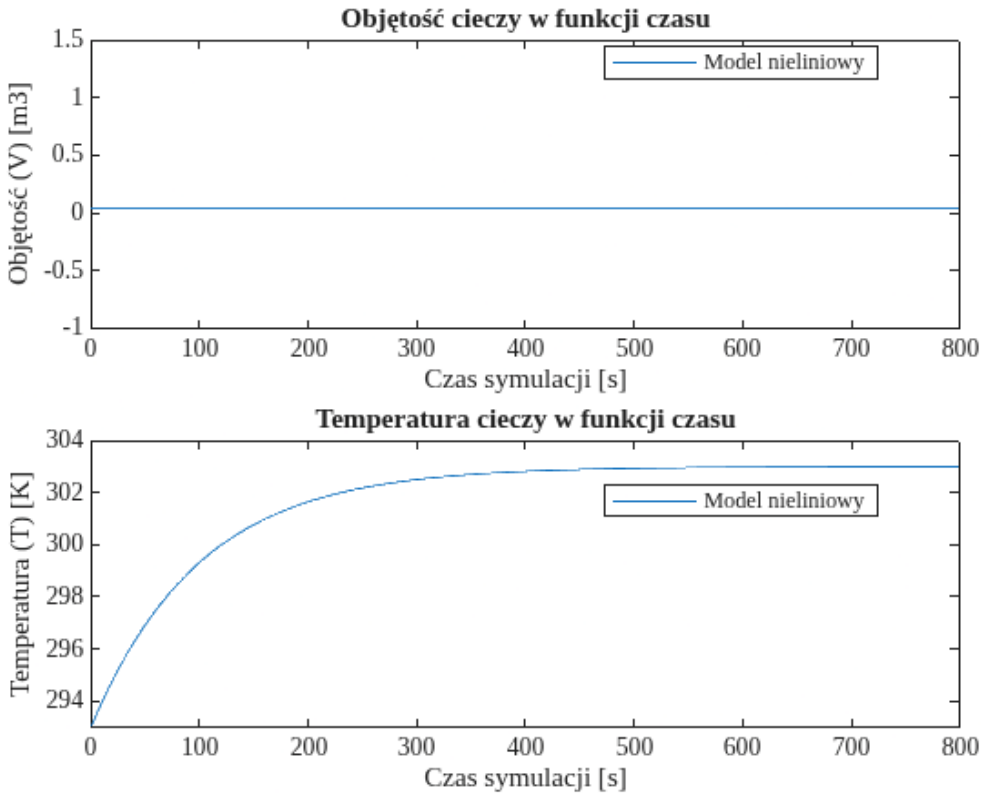
```
Model liniowy w stanie ustalonym
```

```
figure();

subplot(2,1,1)
plot(t,y(:,1) + x_ust(1))
title("Objętość cieczy w funkcji czasu")
xlabel("Czas symulacji [s]")
ylabel("Objętość (V) [m3]")
legend('Model nieliniowy', 'Model liniowy', 'Location', 'best')
```

```
Warning: Ignoring extra legend entries.
```

```

subplot(2,1,2)
plot(t,y(:,2) + x_ust(2))
title("Temperatura cieczy w funkcji czasu")
xlabel("Czas symulacji [s]")
ylabel("Temperatura (T) [K]")
legend('Model nieliniowy', 'Model liniowy', 'Location', 'best')
```

```
Warning: Ignoring extra legend entries.
```



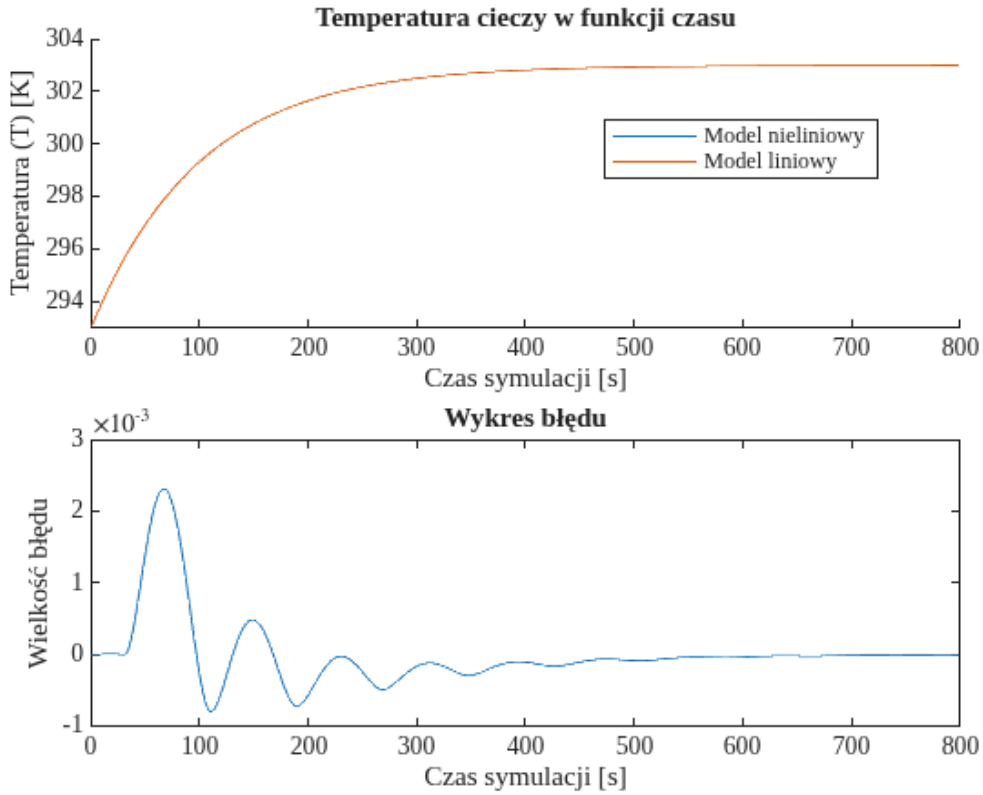
Przebieg funkcji liniowej i nieliniowej będzie taki sam w stanie ustalonym, ponieważ obie funkcje w punkcie pracy są tak naprawdę liniowe.

By zauważyć różnicę, należy wykreślić wykres błędu.

```
wi = 0.4;
w = 0.4;
[t_n_lin_2, x_n_lin_2] = ode45(@zbiornik_stan, [0:799], [V0,T0], [], wi, w,
Ti, Q);

display('Zmiana temperatury dla obu modeli. Stan ustalony.')
```

```
Zmiana temperatury dla obu modeli. Stan ustalony.
```

```
figure();
subplot(2, 1, 1);
hold on
plot(t, x_n_lin_2(:, 2)); % nieliniowy
plot(t,y(:,2) + x_ust(2)); % liniowy
hold off
title("Temperatura cieczy w funkcji czasu")
xlabel("Czas symulacji [s]")
ylabel("Temperatura (T) [K]")
legend('Model nieliniowy', 'Model liniowy', 'Location', 'best')

subplot(2, 1, 2)
plot(t, x_n_lin_2(:, 2) - (y(:,2) + x_ust(2)))
title("Wykres błędu")
xlabel("Czas symulacji [s]")
ylabel("Wielkość błędu")
```



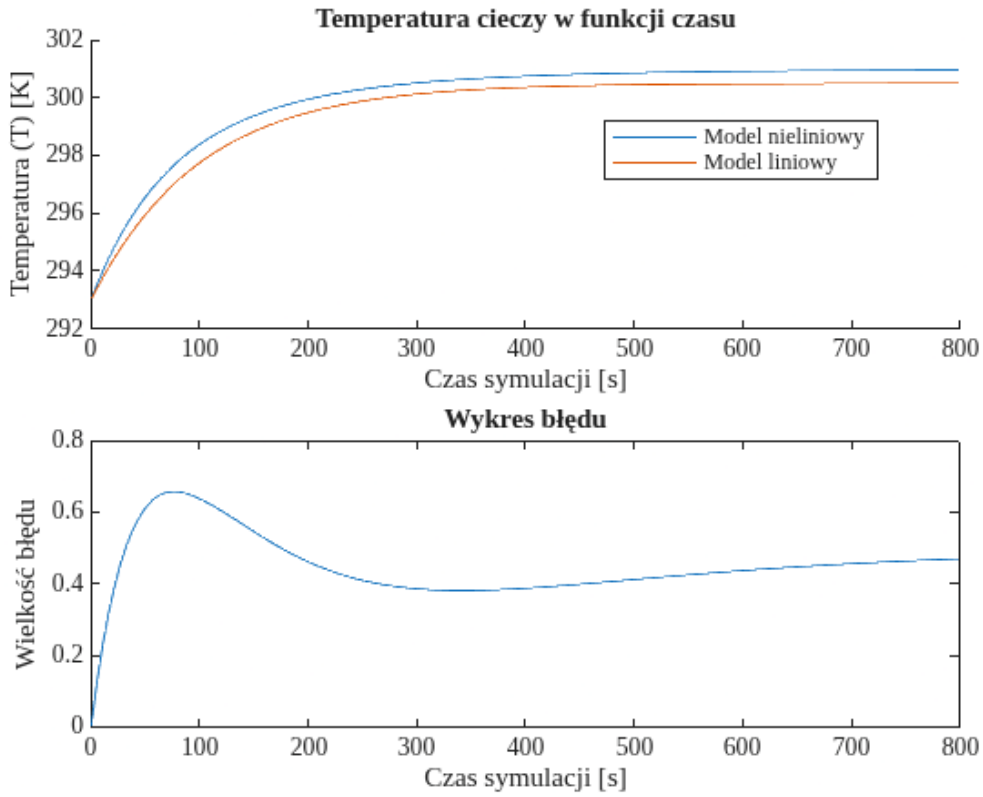
Błąd występuje, lecz jest kilka rzędów wartości mniejszy od wartości, do której się odnosi, więc takie przybliżenie jest wystarczające. Znaczna różnica będzie widoczna dopiero po wyjściu ze stanu ustalonego (oddalenie się od punktu pracy). W celu zasymulowania tego zdarzenia dopływ zostanie zwiększony o 0.1 kg/s, zatem wartość odchyłki w pierwszej kolumnie sterowań wyniesie 0.1

```
% stan nieustalony
wi = 0.5;
w = 0.4;

U(:, 1) = 0.1;

x_pocz = [V0, T0];
x_ust = [V_zad, T_zad];

x0 = x_pocz-x_ust;

y = lsim(A,B,C,D,U,t,x0);
[t_n_lin_2, x_n_lin_2] = ode45(@zbiornik_stan, [0:799], [V0,T0], [], wi, w,
Ti, Q);

display('Zmiana temperatury dla obu modeli. Stan nieustalony.')
```

```
Zmiana temperatury dla obu modeli. Stan nieustalony.
```

```
figure();
```

```
subplot(2, 1, 1)
hold on
plot(t, x_n_lin_2(:, 2))
plot(t, y(:,2) + x_ust(2))
hold off
title("Temperatura cieczy w funkcji czasu")
xlabel("Czas symulacji [s]")
ylabel("Temperatura (T) [K]")
legend('Model nieliniowy', 'Model liniowy', 'Location', 'best')

subplot(2, 1, 2)
plot(t, x_n_lin_2(:, 2) - (y(:,2) + x_ust(2)))
title("Wykres błędu")
xlabel("Czas symulacji [s]")
ylabel("Wielkość błędu")
```



Modele już na pierwszy rzut oka nie zachowują się identycznie. Ciecz w zbiorniku nie osiąga zadanej temperatury, lecz model liniowy zbliża się do niej bardziej. Sama wartość błędu znacząco wzrosła, nie maleje do zera, utrzymuje się na ujemnym poziomie. Wniosek jest taki, że korzystając z modelów zlinearyzowanych trzeba trzymać się punktu pracy i ewentualnie przełączać między różnymi modelami symulującymi różne punkty pracy.