DBDM Assignment SoSe 2021

```
% Author: Mehmet Batu Özmeteler
% Matriculation Number: 230306
```

Initialize workspace and load the data

```
clear all; close all; clc;

load step_response.mat;
load experiment_1.mat;
load experiment_2.mat;

figure_raw_vs_clean_vs_smooth = figure;
figure_schwarze_vs_numerical = figure;
figure_parameter_estimation_vs_smooth_data = figure;
figure_g_wb_vs_smooth_data = figure;
figure_experiment1_vs_experiment2 = figure;
figure_input1_vs_input2 = figure;
figure_arx = figure;
figure_armax = figure;
figure_arx_vs_armax = figure;
figure_all = figure;
```

a) Remove the outliers caused by the failure of the measurement system. Describe and justify your procedure.

B = rmoutliers(A) detects and removes outliers from data. A can be a vector, matrix, table, or timetable. If A is a vector, rmoutliers removes the entries detected as outliers.

```
%clean_data = rmoutliers(data_set_step_temperature);

% Due to the failure of the measurement system in certain timesteps,
% reactor temperature shows to be 0. Removing the zeros (outliers)
% from the data cleans the data.

clean_data = [];
for i = 1:length(data_set_step_temperature)
   if data_set_step_temperature(i) ~= 0
       clean_data = [clean_data; data_set_step_temperature(i)];
   end
end

time = (0:5:(5*length(clean_data)) - 1)';
```

b) Choose and apply an appropriate method for smoothing the data after the outlier removal from a). Why did you choose this specific method?

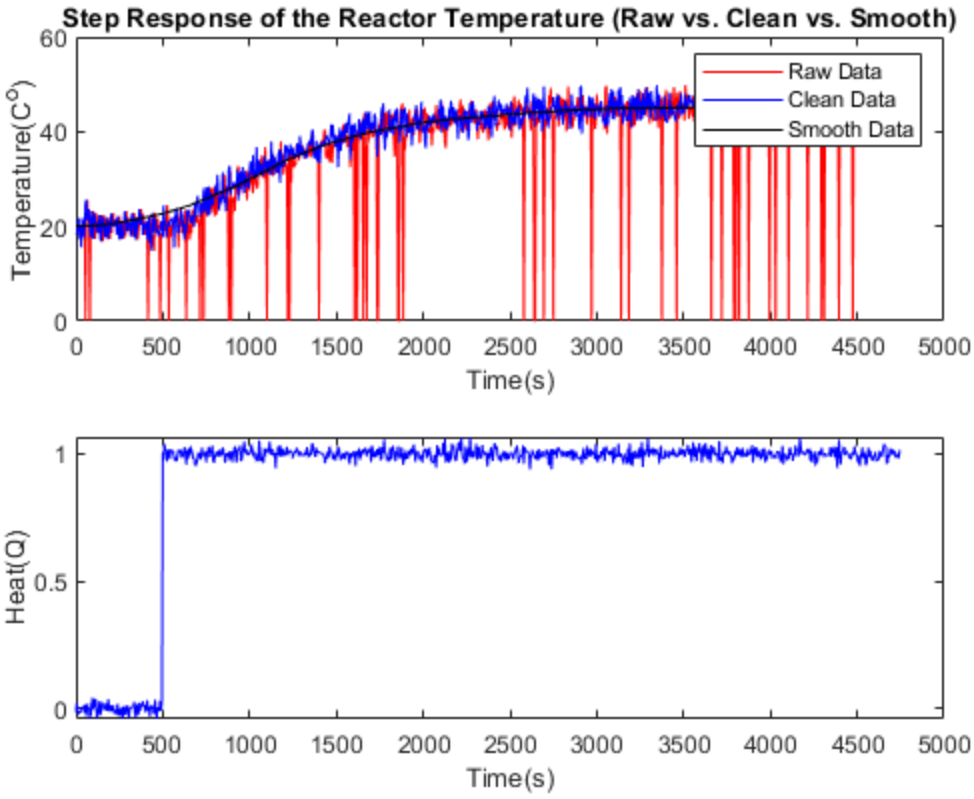
B = smoothdata(A) for a vector A returns a smoothed version of A using a moving average with a fixed window length. The length of the moving average is determined based on the values of A.

```
smooth_data = smoothdata(clean_data);
```

```
save my_data.mat data_set_step_timestamp smooth_data

% Also, instead of using a MATLAB command, a discrete-time first-order
% linear system (a recursive filter with one step delay) can be used
% to smoothen the data.

% a1 = 0.90; % smoothing effect gets stronger as a1 approaches 1
% smoothing_filter = tf(1-a1,[1 -a1], 5);
% smooth_data = lsim(smoothing_filter, clean_data, time);
```

c) Use MATLAB to create a plot of the raw data. In the same plot, add the data without outliers, and the smoothed data.

```
raw_data = data_set_step_temperature(1:length(time));
raw_input = data_set_step_input(1:length(time));
% step_input = [zeros(1, 100) ones(1, length(time) - 100)]';

figure(figure_raw_vs_clean_vs_smooth);
subplot(2,1,1)
plot(time, raw_data, 'r'); hold on;
plot(time, clean_data, 'b'); hold on;
plot(time, smooth_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('Step Response of the Reactor Temperature (Raw vs. Clean vs.
 Smooth)')
legend('Raw Data', 'Clean Data', 'Smooth Data')

subplot(2,1,2)
plot(time, raw_input, 'b')
xlabel('Time(s)')
ylabel('Heat(Q)')
```



d) Use the method of Schwarze with the smoothed data to identify a transfer function G_Schwarze(s). Write down the necessary steps.

Steps for Simple Model Identification 1) Choose what to model 2) Define I/Os 3) Excite the system with known inputs 4) Measure the output 5) Prepare the data 6) Choose a system structure 7) Determine parameters to get the transfer function 8) Approximate the system 9) Validate the model

```
G_schwarze = schwarze_approx(smooth_data, time);
```

e) Choose one of the prepared data sets for numerical transfer function estimation. Justify your choice. Identify a transfer function G_tfest(s) using the MATLAB commands iddata and tfest based on the chosen data set. Hint: The MATLAB documentation can be used for reference.

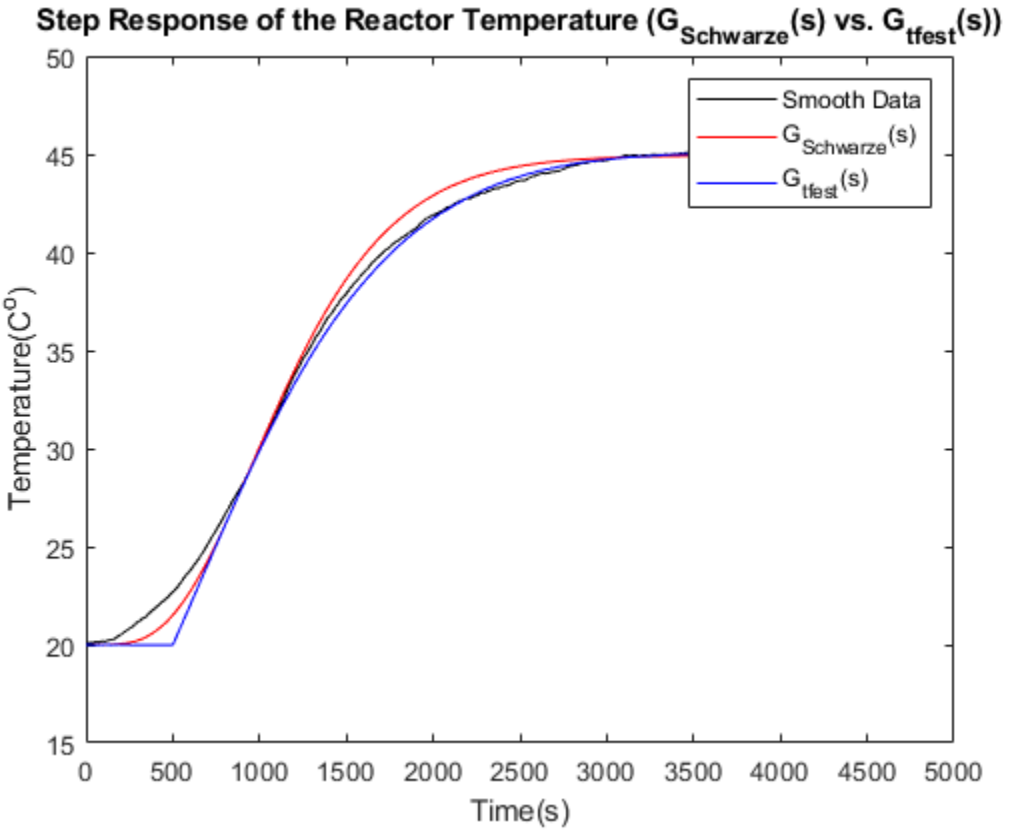
The smooth data will be used for the numerical etimation.

```
y_op = 20;
u_op = 0;

num_estim_data = iddata(smooth_data - y_op, raw_input - u_op, 5);
G_tfest = tfest(num_estim_data, 2); % fit to estimation data: 95.7%
```

f) Plot the data and the response from the transfer functions G_Schwarze(s) and G_tfest(s). Comment on the results.

```
figure(figure_schwarze_vs_numerical);

H_schwarze = step(G_schwarze, time) + y_op;
H_tfest = lsim(G_tfest, raw_input, time) + y_op;
```

```
plot(time, smooth_data, 'k'); hold on;
plot(time, H_schwarze, 'r'); hold on;
plot(time, H_tfest, 'b');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('Step Response of the Reactor Temperature (G_{Schwarze}(s) vs.
 G_{tfest}(s))')
legend('Smooth Data', 'G_{Schwarze}(s)', 'G_{tfest}(s)')

% As one can see from the plots, the numerical transfer function
% estimation approximates the smooth data better than
% G_Schwarze(s). G_Schwarze(s) is still relevant as it captures
% the initial behaviour better than G_tfest(s) but overall,
% G_tfest(s) is a better approximation.
%
```



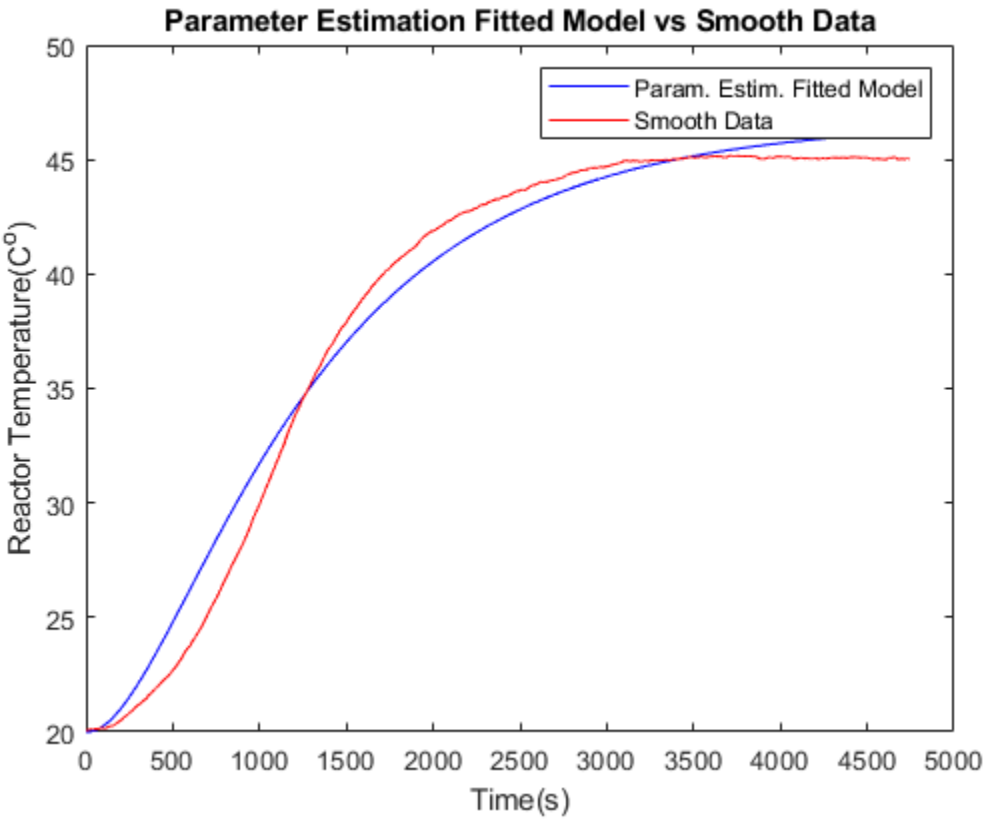
g) Set up the 3 differential balance equations for the system using the variables listed in Table 1.

For the derivation of the differential balance equations, check the handwritten solutions. For the implementation of the differential balance equations, check the cstr_model_ode_rhs.m file.

h) Apply the parameter estimation method from the lecture to estimate the values of k1 and k2. Plot the simulation results of the fitted model and the real measurements without outliers in one graph. Hint: Use the MATLAB command ode45 for the integration of the derived ODE system.

```
initialGuess = [1; 1];  %[k1; k2]
```

```
[result] = fminsearch(@(p) myfunc(p), initialGuess);
k1 = result(1);
k2 = result(2);

initial_temperatures = [293; 293; 293];
[timesteps , temperatures] = ode45(@(t, x)cstr_model_ode_rhs(t, x,
 result), time, initial_temperatures);

figure(figure_parameter_estimation_vs_smooth_data)
plot(timesteps, temperatures(:,1) - 273, 'b'); hold on;
plot(timesteps, smooth_data, 'r');
xlabel('Time(s)')
ylabel('Reactor Temperature(C^o)')
title('Parameter Estimation Fitted Model vs Smooth Data')
legend('Param. Estim. Fitted Model', 'Smooth Data')
```



i) Apply the Laplace transform to each individual equation to determine the white box model transfer function G_WB(s), using the heat transfer coefficients estimated in Task(h). Explain your steps.

Please check the handwritten solutions for the white box model transfer function G_WB(s).

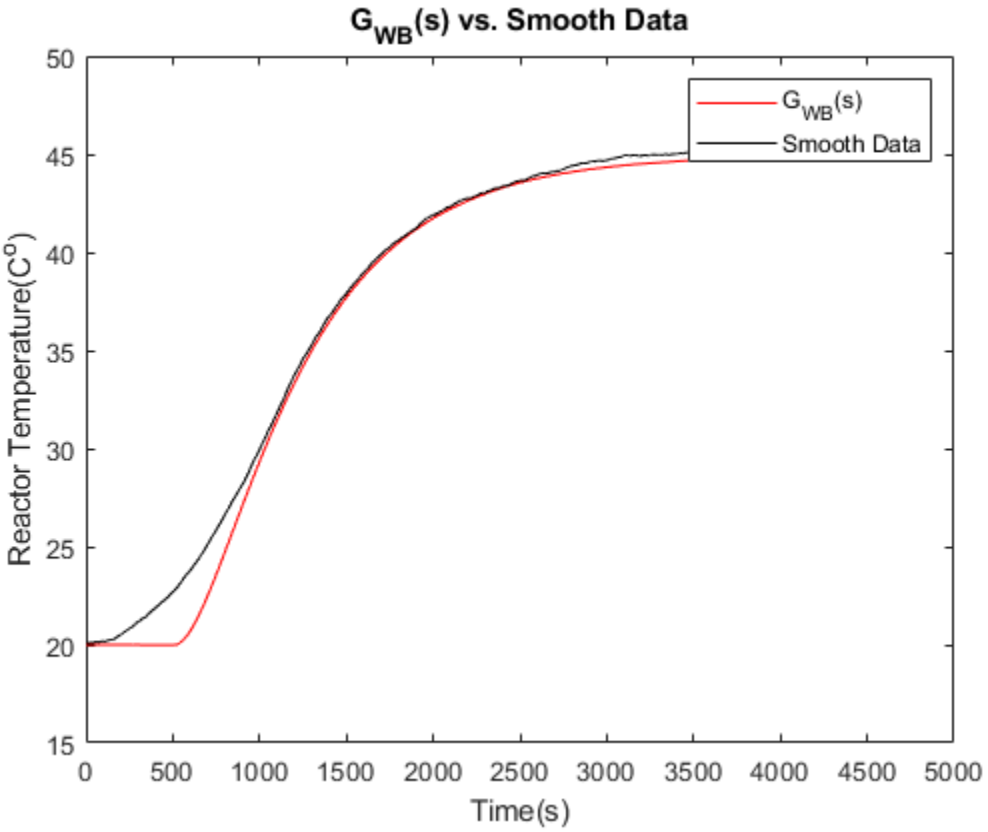
```
T_E = 293;
m_R = 2;
m_J = 0.5;
m_T = 4;
cp_T = 2;
```

```
cp_R = 4;
m_dot_T = 0.02;
A1 = 0.4;
A2 = 0.2;
```

j) Calculate the z-domain transfer function G_WB(z) manually. Write down your steps.

Please check the handwritten solutions for the z-domain transfer function G_WB(z).

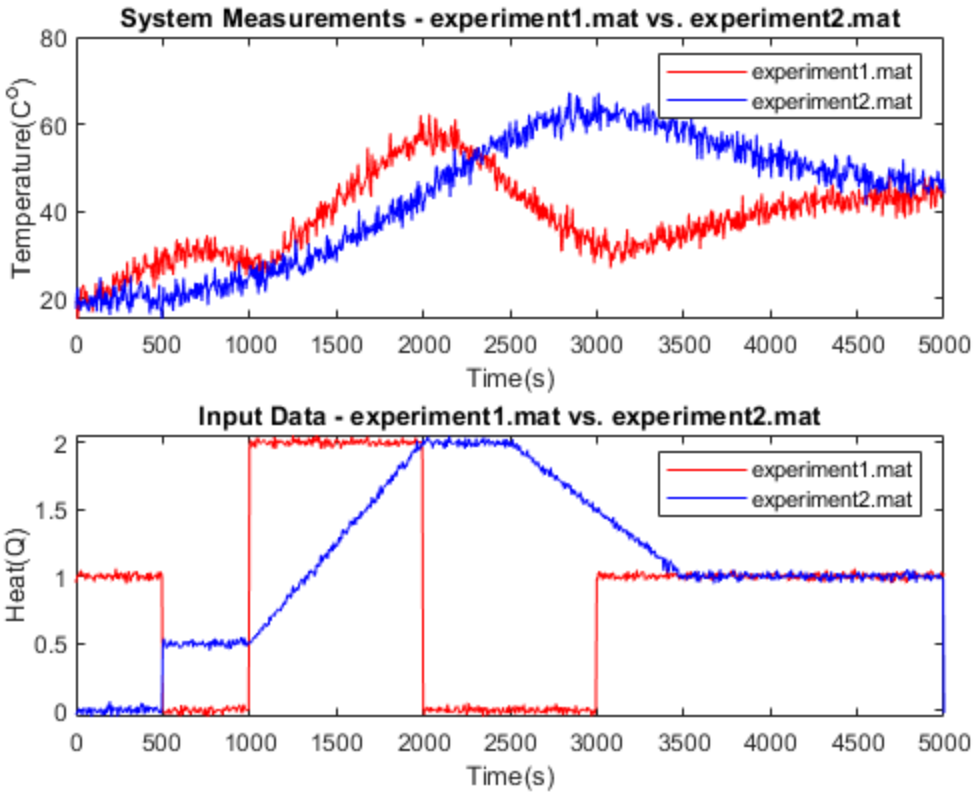
```
G_wb_s = tf(25, [ 10^6, 13*10^4, 825, 1 ]);
G_wb_z = c2d(G_wb_s, 5, 'zoh');

H_wb = lsim(G_wb_s, raw_input, time) + y_op;

figure(figure_g_wb_vs_smooth_data)
plot(time, H_wb, 'r'); hold on;
plot(time, smooth_data, 'k'); hold on;
xlabel('Time(s)')
ylabel('Reactor Temperature(C^o)')
title('G_{WB}(s) vs. Smooth Data')
legend('G_{WB}(s)', 'Smooth Data')
```



k) Plot the inputs and outputs of the data set used in the previous tasks and of the new data sets for parameter estimation (experiment_1.mat) and validation (experiment_2.mat) in one figure. Qualitatively discuss the features of the input signals.

```
figure(figure_experiment1_vs_experiment2);
subplot(2,1,1)
```

```
plot(data_set_arx_estimation_timestamp,
 data_set_arx_estimation_temperature, 'r'); hold on;
plot(data_set_arx_validation_timestamp,
 data_set_arx_validation_temperature, 'b')
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('System Measurements - experiment1.mat vs. experiment2.mat')
legend('experiment1.mat', 'experiment2.mat')

subplot(2,1,2)
plot(data_set_arx_estimation_timestamp,
 data_set_arx_estimation_input, 'r'); hold on;
plot(data_set_arx_validation_timestamp,
 data_set_arx_validation_input, 'b');
xlabel('Time(s)')
ylabel('Heat(Q)')
title('Input Data - experiment1.mat vs. experiment2.mat')
legend('experiment1.mat', 'experiment2.mat')

% The input signal to choose for model identification must be
% dynamically sufficiently rich. The input should provide
% enough excitation to observe the system dynamics better.
% For linear systems, it is better to have a series of steps
% (not a single one) with different durations between steps.
% A generalized binary noise is a great type of input for
% model identification since it is a series of steps jumping
% in between the maximum and minimum value. This provides
% necessarily more information than the values in between
% due to noise relative to the input becoming smaller.
% Considering this criterion, experiment1.mat seems to be
% the better data for the estimation.
```

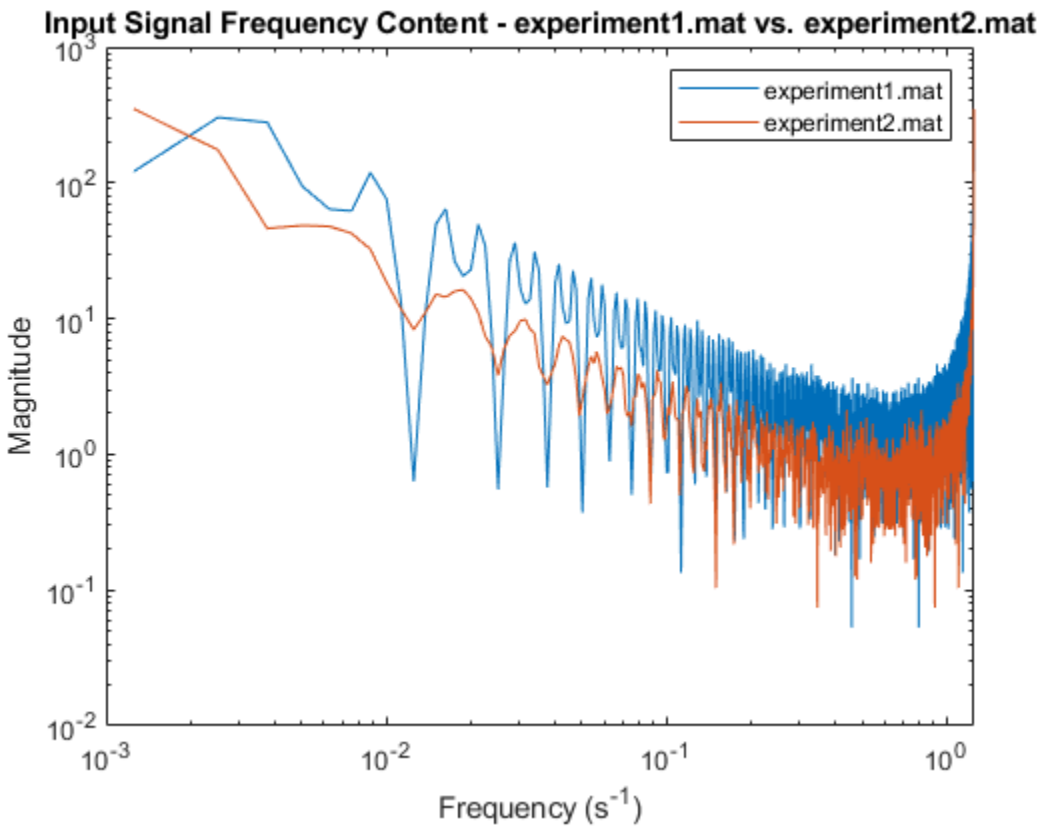


l) Using frequency response techniques, sketch a method to quantitatively compare different input sequences with the goal of choosing the most suitable for model identification. Assume that the noise is white and additive on the output.

A good signal is a signal that stimulates many frequencies meaning it has signals at different frequencies in its composition.(Frequency response of a system shows how much a sinusoidal with a certain frequency is attenuated/amplified and positively/negatively phase shifted if it's applied to this system as an input.) A method to quantitatively compare different input sequences for model identification is DFT. Applying DFT to the input data transfers the sampled data to the frequency domain, providing us with the information of its frequency content. The input signal with richer frequency content will be our candidate. DFT can be calculated numerically in a more efficient manner if FFT is used.

```
figure(figure_input1_vs_input2);
dft_exp1 = fft(data_set_arx_estimation_input);
dft_exp2 = fft(data_set_arx_validation_input);
L = length(data_set_arx_estimation_timestamp);
T = 5;

omega = 2*pi*(0:L-1)'/L/T;
loglog(omega, abs(dft_exp1), omega, abs(dft_exp2))
xlabel('Frequency (s^{-1})')
ylabel('Magnitude')
title('Input Signal Frequency Content - experiment1.mat vs.
 experiment2.mat')
legend('experiment1.mat', 'experiment2.mat')

% From the plot, one can say that experiment1.mat input data set has
```

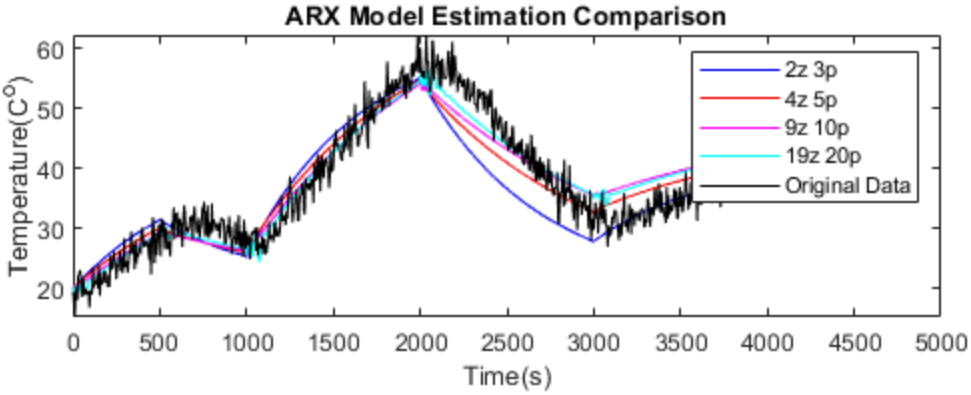
```
% higher requency content than experiment2.mat data set.
% experiment1.mat will be used for the estimation of the model.
```

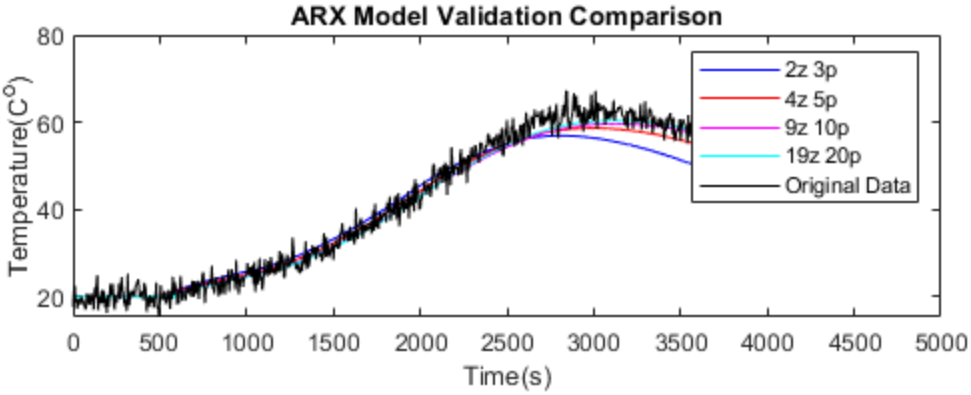


m) Identify 4 ARX models on the experiment_1.mat dataset and plot the prediction results together with the fitting data.

```
figure(figure_arx);

exp1_raw_data = data_set_arx_estimation_temperature(1:length(time));
exp1_raw_input = data_set_arx_estimation_input(1:length(time));

exp2_raw_data = data_set_arx_validation_temperature(1:length(time));
exp2_raw_input = data_set_arx_validation_input(1:length(time));

exp1_data = iddata(data_set_arx_estimation_temperature - y_op,
 data_set_arx_estimation_input - u_op, 5);

G_arx1 = arx(exp1_data,[3, 2, 0]);      % MSE : 5.215, Fit to
 Estimation Data: 75.5%
H_arx1_1 = lsim(G_arx1, exp1_raw_input, time) + y_op;
H_arx1_2 = lsim(G_arx1, exp2_raw_input, time) + y_op;

G_arx2 = arx(exp1_data,[5, 4, 0]);      % MSE : 4.644, Fit to
 Estimation Data: 76.88%
H_arx2_1 = lsim(G_arx2, exp1_raw_input, time) + y_op;
H_arx2_2 = lsim(G_arx2, exp2_raw_input, time) + y_op;
```

```
G_arx3 = arx(exp1_data,[10, 9, 0]);     % MSE : 4.172, Fit to
 Estimation Data: 78.08%
H_arx3_1 = lsim(G_arx3, exp1_raw_input, time) + y_op;
H_arx3_2 = lsim(G_arx3, exp2_raw_input, time) + y_op;

G_arx4 = arx(exp1_data,[20, 19, 0]);    % MSE : 3.868, Fit to
 Estimation Data: 78.9%
H_arx4_1 = lsim(G_arx4, exp1_raw_input, time) + y_op;
H_arx4_2 = lsim(G_arx4, exp2_raw_input, time) + y_op;

subplot(2,1,1)
plot(time, H_arx1_1, 'b'); hold on;
plot(time, H_arx2_1, 'r'); hold on;
plot(time, H_arx3_1, 'm'); hold on;
plot(time, H_arx4_1, 'c'); hold on;
plot(time, exp1_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('ARX Model Estimation Comparison')
legend('2z 3p', '4z 5p', '9z 10p', '19z 20p', 'Original Data')

subplot(2,1,2)
plot(time, H_arx1_2, 'b'); hold on;
plot(time, H_arx2_2, 'r'); hold on;
plot(time, H_arx3_2, 'm'); hold on;
plot(time, H_arx4_2, 'c'); hold on;
plot(time, exp2_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('ARX Model Validation Comparison')
legend('2z 3p', '4z 5p', '9z 10p', '19z 20p', 'Original Data')

% G_arx2 is the best fit.
```





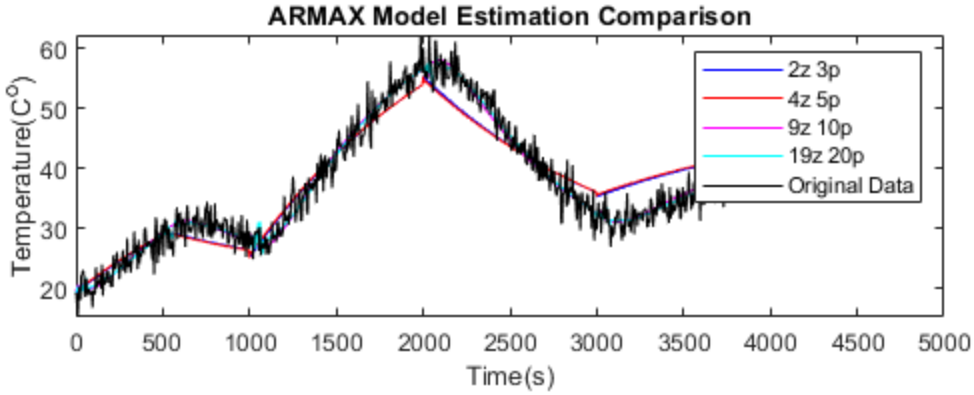
n) Compare the prediction performance of the identified models G_ARX,i(z) on the validation dataset using the error metric. Plot the results. Is the performance of the ARX model satisfactory?

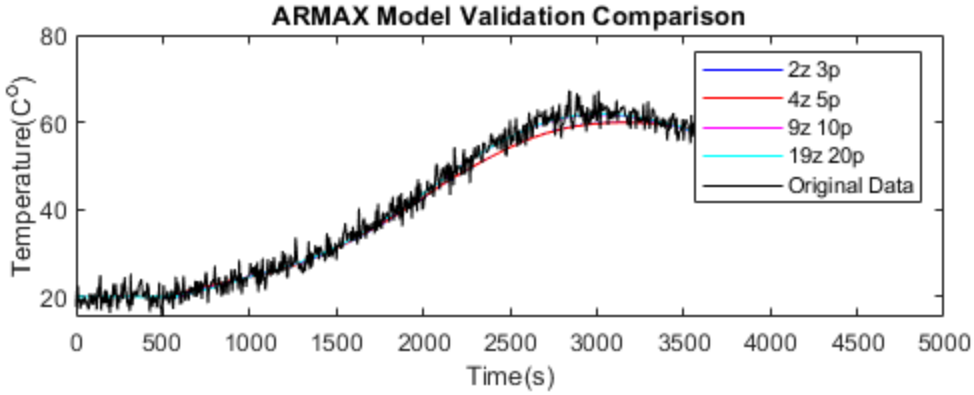
MSE goes down as the model order increases but the model starts to track noise instead of the system's behaviour. Comparing the plots, we can see that the best fit is G_arx2 with 5 poles and 4 zeros. However, the performance of the ARX model isn't satisfactory, it can be better.

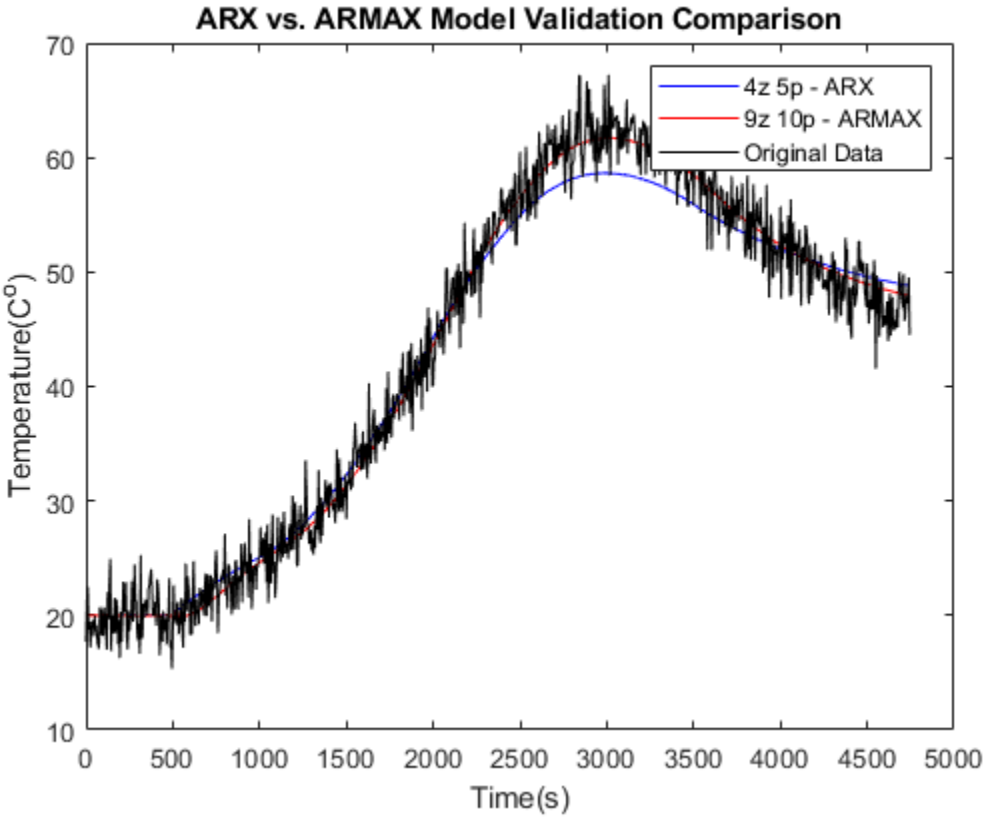
o) Identify 4 ARMAX models on the experiment_1.mat data set and repeat the comparison of the previous point on the validation data set experiment_2.mat using the error metric. Plot the validation results. Which model structure is more suitable for the supplied data between ARX and ARMAX? Provide suggestions as to why is this the case.

```
figure(figure_armax);

G_armax1 = armax(exp1_data,[3, 2, 1, 0]);      % MSE : 4.235, Fit to
 Estimation Data: 77.92%
H_armax1_1 = lsim(G_armax1, exp1_raw_input, time) + y_op;
H_armax1_2 = lsim(G_armax1, exp2_raw_input, time) + y_op;

G_armax2 = armax(exp1_data,[5, 4, 1, 0]);      % MSE : 4.178, Fit to
 Estimation Data: 78.07%
H_armax2_1 = lsim(G_armax2, exp1_raw_input, time) + y_op;
H_armax2_2 = lsim(G_armax2, exp2_raw_input, time) + y_op;

G_armax3 = armax(exp1_data,[10, 9, 1, 0]);     % MSE : 4.061, Fit to
 Estimation Data: 78.38%
H_armax3_1 = lsim(G_armax3, exp1_raw_input, time) + y_op;
```

```
H_armax3_2 = lsim(G_armax3, exp2_raw_input, time) + y_op;

G_armax4 = armax(exp1_data,[20, 19, 1, 0]);    % MSE : 3.693, Fit to
 Estimation Data: 79.38%
H_armax4_1 = lsim(G_armax4, exp1_raw_input, time) + y_op;
H_armax4_2 = lsim(G_armax4, exp2_raw_input, time) + y_op;

subplot(2,1,1)
plot(time, H_armax1_1, 'b'); hold on;
plot(time, H_armax2_1, 'r'); hold on;
plot(time, H_armax3_1, 'm'); hold on;
plot(time, H_armax4_1, 'c'); hold on;
plot(time, exp1_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('ARMAX Model Estimation Comparison')
legend('2z 3p', '4z 5p', '9z 10p', '19z 20p', 'Original Data')

subplot(2,1,2)
plot(time, H_armax1_2, 'b'); hold on;
plot(time, H_armax2_2, 'r'); hold on;
plot(time, H_armax3_2, 'm'); hold on;
plot(time, H_armax4_2, 'c'); hold on;
plot(time, exp2_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('ARMAX Model Validation Comparison')
legend('2z 3p', '4z 5p', '9z 10p', '19z 20p', 'Original Data')

% G_armax3 is the best fit.

% It becomes difficult to distinguish plots when there are many on the
% same plot. Hence, only the best performing models are plotted
% together with the original data.

figure(figure_arx_vs_armax);
plot(time, H_arx2_2, 'b'); hold on;
plot(time, H_armax3_2, 'r'); hold on;
plot(time, exp2_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('ARX vs. ARMAX Model Validation Comparison')
legend('4z 5p - ARX', '9z 10p - ARMAX', 'Original Data')

% As one can see from the figure that ARMAX performs better than ARX.
% ARMAX is a more suitable model when the noise effecting the output
% is different than white noise. In the presence of colored noise or
% transient disturbances, ARMAX is more accurate than ARX, but it
% has more degrees of freedom and may converge to a local minima.
```

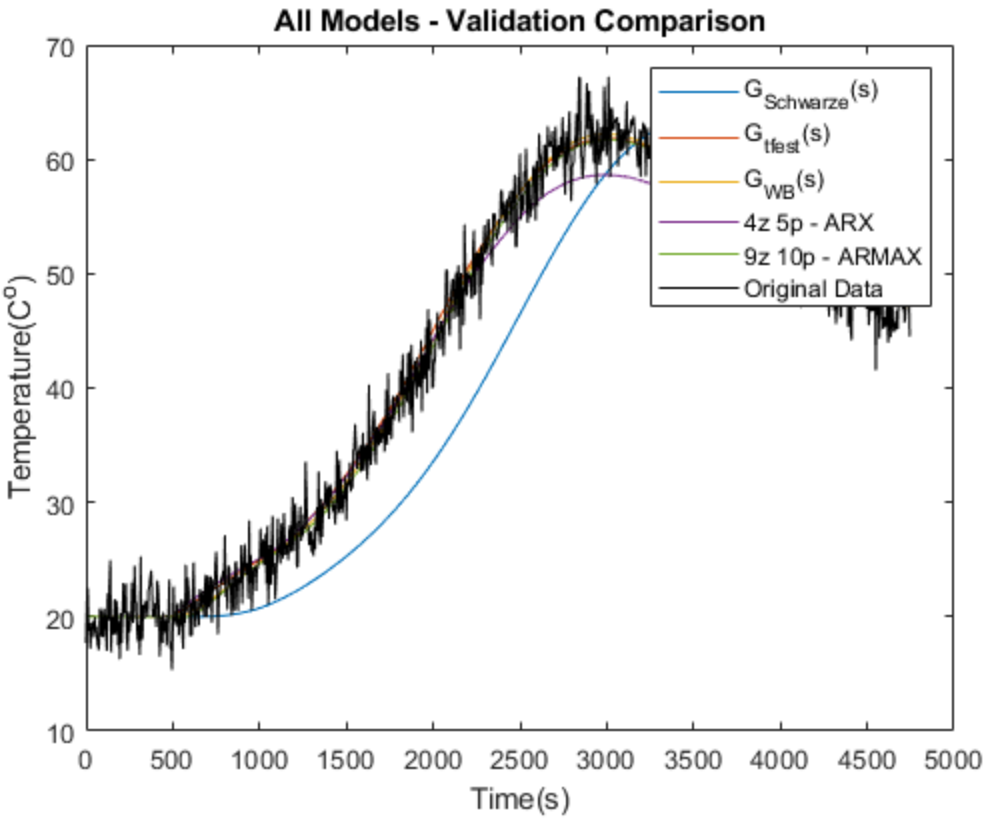

ARMAX Model Estimation Comparison
Temperature(C°)
Time(s)
2z 3p
4z 5p
9z 10p
19z 20p
Original Data


ARMAX Model Validation Comparison
Temperature(C°)
Time(s)
2z 3p
4z 5p
9z 10p
19z 20p
Original Data


ARX vs. ARMAX Model Validation Comparison
Temperature(C°)
Time(s)
4z 5p - ARX
9z 10p - ARMAX
Original Data

p) Plot the validation data experiment_2.mat along with the responses of G_Schwarze(s), G_tfest(s), G_W-B(s), and the best model identified among ARX and ARMAX G_AR(MA)X(z). Compare the different simulations using the error metric. How can you explain the differences? Tip: Consider the whole model identification procedure.

```
figure(figure_all);

exp2_H_schwarze = lsim(G_schwarze, exp2_raw_input, time) + y_op;
exp2_H_tfest = lsim(G_tfest, exp2_raw_input, time) + y_op;
exp2_H_wb = lsim(G_wb_s, exp2_raw_input, time) + y_op;

plot(time, exp2_H_schwarze, time, exp2_H_tfest, time, exp2_H_wb, ...
    time, H_arx2_2, time, H_armax3_2); hold on;
plot(time, exp2_raw_data, 'k');
xlabel('Time(s)')
ylabel('Temperature(C^o)')
title('All Models - Validation Comparison')
legend('G_{Schwarze}(s)', 'G_{tfest}(s)', 'G_{WB}(s)', '4z 5p -
 ARX',...
    '9z 10p - ARMAX', 'Original Data')

% The best result among them are G_WB(s) and G_armax3. The differences
% can be explained by mentioning the noise presence. As mentioned
% above, in the presence of colored noise or transient disturbances,
% ARMAX is more accurate than ARX, but it has more degrees of freedom
% and may converge to a local minima. G_Schwarze(s) is way-off since
% it's just a simple approximation from the values of the step
% function. G_tfest(s) fits the data better than the ARX model since
% it uses numerical optimization.
```



q) Rank the different modelling techniques (graphical, first principles and automatic model identification) from most favorable to least favorable based on the ease of implementation, accuracy and ease of use for the subsequent controller design process.

Ranking of Modelling Techniques (Ease of Implementation, Accuracy and Subsequent Controller Design)

1) tfest: It has great accuracy with easy implementation. Controller design is easy to do as well.

2) ARMAX: It has good accuracy even if there exists colored noise. However, since it has more degrees of freedom, it may converge to a local minima. Subsequent controller design isn't hard.

3) ARX: It has decent accuracy however, it doesn't perform that well when there are transient disturbances or colored noise. It's relatively easy to implement controller and the model.

4) First Principles: It has high accuracy and it's possible to extrapolate. Controller design isn't difficult, however it takes a lot of time and effort to get good parameters for the equations.

5) Schwarze: It has good accuracy for estimation but low accuracy for validation which means that overall the model isn't efficient. The implementation isn't too difficult but the model can't be used for controller design.

Helper Functions

```
function G = schwarze_approx(data, time)

% The method of Schwarze can be used to approximate a system which can
% be fitted using a n-th order lag element G(s) = K / (1 + sT)^n
```

```
% First, the times where the step response h(t) reaches 10, 30, 50, 70
% and 90 % of its value are calculated as t1, t3, t5, t7 and t9.

y_ss = data(end);
y_init = data(1);

delta_y = y_ss - y_init;
delta_u = 1; % assumed to be 1

y1 = find(data >= y_init + 0.1*delta_y);
y3 = find(data >= y_init + 0.3*delta_y);
y5 = find(data >= y_init + 0.5*delta_y);
y7 = find(data >= y_init + 0.7*delta_y);
y9 = find(data >= y_init + 0.9*delta_y);

t1 = time(y1(1)); t3 = time(y3(1)); t5 = time(y5(1));
t7 = time(y7(1)); t9 = time(y9(1));

% Then, these times are used to calculate certain proportions to find
% the degree of the transfer function.

t1_t9 = t1/t9; % = 0.3101
t1_t7 = t1/t7; % = 0.4481
t1_t5 = t1/t5; % = 0.5702
t1_t3 = t1/t3; % = 0.7113
t3_t7 = t3/t7; % = 0.6299
t3_t5 = t3/t5; % = 0.8017

% Looking at the corresponding order n in the table of proportions,
% the order of the tf is n = 5.
n = 5;

% Checking the left-half of the table, we calculate the appropriate
% time constant T.
T = ((t9 / 7.99) + (t7 / 5.89) + (t5 / 4.67) + (t3 / 3.63) + (t1 /
 2.43)) / 5;

% The gain K is calculated from K = delta_y / delta_u.
K = delta_y / delta_u;

% The transfer function has the following form:
G = K*((tf(1, [T 1]))^n);
end

function f = myfunc(p)

load my_data.mat
x0 = [293; 293; 293]; % all temperatures are 20 C^o if
 {Q^{dot}}_{heat} = 0

tspan = data_set_step_timestamp(1:950);

options = odeset('RelTol', 1e-8, 'AbsTol', 1e-6);
```

```
[time, h] = ode45(@(t, x)cstr_model_ode_rhs(t, x, p), tspan, x0,
 options);

f = 0;
    for i = 1:length(time)
        f = f + (h(i, 1) - (smooth_data(i) + 273))^2;
    end

end

function f = cstr_model_ode_rhs(t, x, p)

%
 ------------------------------------------------------------------------

% States
T_R    = x(1);
T_J    = x(2);
T_Heat = x(3);

% Inputs
% step_input = [zeros(1, 100) ones(1, 850)]';

Q_Heat = 1;

%
 ------------------------------------------------------------------------

% Parameters

T_E = 293;      % temperature of the environment(K)
m_R = 2;        % mass of the content in the reactor(kg)
m_J = 0.5;      % mass of the heating medium in the jacket(kg)
m_T = 4;        % mass of the heating medium in the thermostat(kg)
cp_T = 2;       % heat capacity of the content of the jacket
                % and thermostat(kJ/kg*K)
cp_R = 4;       % heat capacity of the content of the reactor(kJ/kg*K)
m_dot_T = 0.02; % in/outlet mass flow of the thermostat(kg/s)
A1 = 0.4;       % heat tf. area between the jacket and
 environment(m^2)
A2 = 0.2;       % heat tf. coef. area between the jacket and
 reactor(m^2)
k1 = p(1);      % heat tf. coef. of jacket-environment(kW/m^2 * K)
k2 = p(2);      % heat tf. coef. of jacket-reactor(kW/m^2 * K)

%
 ------------------------------------------------------------------------

% Differential Equations

f(1,1) = ((-k2*A2)*(T_R - T_J)/(m_R*cp_R)); % T_R_dot
f(2,1) = (((T_Heat - T_J)*m_dot_T)/m_J) - ((k1*A1)*(T_J - T_E)/
(m_J*cp_T)) - ...
    ((k2*A2)*(T_J - T_R)/(m_J*cp_T)); % T_J_dot
```

```
f(3,1) = (Q_Heat/(m_T*cp_T)) + (((T_J - T_Heat)*m_dot_T)/m_T); %
 T_Heat

end
```

Published with MATLAB® R2021a