Lab 1 Report

Pre-lab Exercise: Sinusoidal Data Generation and Plotting

1. Introduction

The objective of this pre-lab exercise is to refresh MATLAB skills by generating sinusoidal data (sine and cosine waves) and plotting the results, both original and after data truncation. This report details the steps to achieve the intended plots using MATLAB.

2. Objective

The goal is to generate two sinusoidal datasets, introduce noise, and then truncate part of the data. Finally, the original and truncated datasets will be plotted to visualize the differences.

3. Procedure

Step 1: Generate Time Vector

A time vector t was created, ranging from 0 to 100 seconds with a step size of 0.1 seconds.

```
% Step1
t = (0:0.1:100)'; % Time column vector
```

Step 2: Generate Sinusoidal Data

Using the sin and cos functions, two sinusoidal waveforms, y1 and y2, were generated. A uniformly distributed random noise was added to both signals.

```
% Step2
y1 = sin(0.02*pi*t)+0.4*rand(size(t))-0.2; % y1
y2 = cos(0.02*pi*t)+0.4*rand(size(t))-0.2; % y2
```

Step 3: Truncate Data

The first 200 samples, corresponding to the first 20 seconds of data, were removed from the dataset. The time vector was adjusted accordingly.

```
%Step3
t_new = t(201:end)-t(201); % Cut-off first 20s
y1_new = y1(201:end); %Cut-off sin wave
y2_new = y2(201:end); % Cut-off cos wave
```

Step 4: Create New Matrix

A new matrix data_new was created, containing the truncated time, sine, and cosine values.

```
%Step 4
data_new = [t_new y1_new y2_new];  % nx3 matrix
```

Step 5: Plot the Original Data

The original sinusoidal data was plotted on a figure with two subplots. The first subplot shows the original y1 and y2 data.

```
%Step 5
figure;
subplot(2,1,1); %First plot in a 2-row, 1-column layout, 1st fig
plot(t, y1, 'r');
hold on;
plot(t, y2, 'b');
hold on; %blue & red
title('Original Data');
xlabel('Time (sec)');
ylabel('Data');
xlim([0 140]);
ylim([-1.5 1.5]);
grid on;
legend('sin(0.02*pi*t)', 'cos(0.02*pi*t)');
```

Step 6: Plot the Truncated Data

The truncated y1_new and y2_new data were plotted on the second subplot using different colors.

```
%Step 6
subplot(2,1,2);%Second plot
plot(t_new, y1_new, 'g');
hold on;
plot(t_new, y2_new, 'color', [0.6 0.7 0.8]);
hold on;
title('Cut-off Data');
xlabel('Time (sec)');
ylabel('Data');
xlim([0 140]);
ylim([-1.5 1.5]);
grid on;
legend('sin(0.02*pi*t)', 'cos(0.02*pi*t)');
```

4. Results and Analysis

Here is the plot this code generated.

Original Data: In the first subplot, the red curve represents the noisy sine wave (sin(0.02*pi*t)), and the blue curve represents the noisy cosine wave (cos(0.02*pi*t)).

Cut-off Data: The second subplot shows the truncated data. The green and gray lines represent the sine and cosine waves after the first 20 seconds of data have been removed.

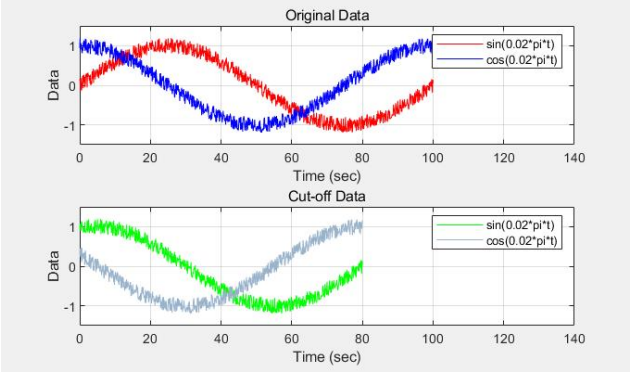

Fig 1

5. Conclusion

This pre-lab exercise successfully demonstrated the generation of sinusoidal data in MATLAB, the introduction of noise, and data truncation techniques. Additionally, plotting the results using MATLAB's plotting functions was accomplished, showcasing both original and truncated datasets.

Lab Exercise: Identifying a Second-Order Discrete-Time Linear Model

1. Introduction

This lab focuses on identifying a second-order discrete-time linear model using pre-collected input-output data from a water tank (W-T) setup. The exercise involves extracting the data, removing offsets, and applying system identification techniques to develop a second-order model, which is then verified by comparing the simulated and actual output.

2. Data Extraction and Preprocessing

We first load the pre-collected data (SysIdenData_StudentVersion.mat) into MATLAB, which contains time (t), the actual output (y_act), and input signal (u_act). Both the noise-reduced output (y_act) and measured output (y_actm) are plotted for comparison, along with the actual input signal.

```
% Step 1
load('SysIdenData_StudentVersion.mat');
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;
```

```
% Step 2
figure;
subplot(2,1,1);
plot(t, y_act, 'b', t, y_actm, 'r');
title('Actual Output Signal');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([1 4]);
legend('Noise-Reduced Output', 'Measured Output');
grid on;

subplot(2,1,2);
plot(t, u_act, 'b');
title('Actual Input Signal');
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
legend('Actual Input');
xlim([0 700]);
ylim([1 3]);
grid on;
```

Here is the figure generated.

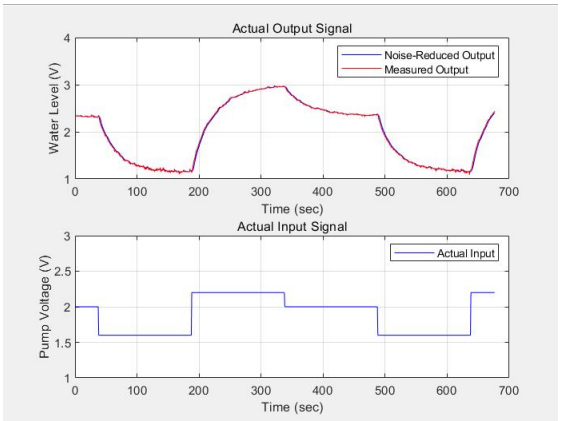

Fig 2

3. Removing Offset from Data

Offsets are detected and removed from both input and output signals to create offset-free data.

This ensures that the model identifies the correct system dynamics.

```
% Step 3
i = 1;
while u_act(i) == u_act(1)
    i = i + 1;
end
% Output offset
u_offset = u_act(1);
u = u_act - u_offset;
% Input offset
y_offset = mean(y_act(1:i-1));
y = y_act - y_offset;
```

After removing the offsets, we plot the offset-free signals.

```
% Step 4
figure;
subplot(2,1,1);
plot(t, y, 'r');
title('Actual Offset-Free Output Signal');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-2 1]);
legend('Actual Output');
grid on;

subplot(2,1,2);
plot(t, u, 'b');
title('Actual Offset-Free Input Signal');
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
xlim([0 700]);
ylim([-0.5 0.5]);
legend('Actual Input');
grid on;
```

Here is the figure generated.

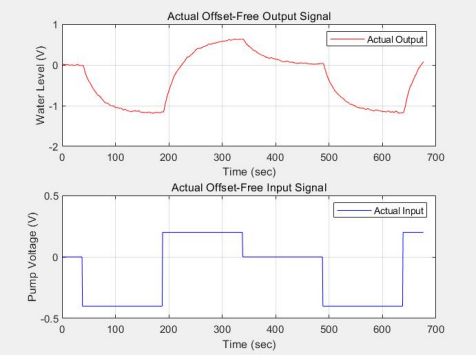

Fig 3

5. Second-Order Model Identification

Using the first half of the offset-free data, a second-order model is identified by constructing the matrix Φ and solving for the model parameters a_1, a_2, b_1 and b_2.

```
% Part 2 step a
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];

% Part 2 step b
theta = inv((Phi' * Phi)) * (Phi' * Y);
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2
```

The second-order transfer function and state-space model are derived based on the identified parameters:

```
% Part 2 step c
% transfer function
h = 0.01;
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1

% state space
G = [0 1; -a2 -a1];  % matrix G
H = [0; 1];  % matrix H
C = [b2 b1];  % matrix C
D = 0;  % matrix D
sys_ss = ss(G, H, C, D, h);
```

5. Model Validation

To validate the model, the second half of the data is used for simulation, and the simulated output is compared to the actual offset-free output.

```
% Part3 Step a
t_half = t(N+1:end);
y_half = y(N+1:end);
t_half_shifted = t_half - t_half(1);
simulated_output = filter(numerator, denominator, u(N+1:end));

% Step b
figure;
subplot(2,1,1);
plot(t_half_shifted, y_half(),'r', t_half_shifted, simulated_output, 'b--');
title('Offset-Free Model Verification (2nd Half)');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 350]);
ylim([-2 2]);
legend('Actual Output', 'Simulated Output');
grid on;
```

The entire dataset is then used to simulate the model and compare it with the actual output across the full time period.

```
simulated_output2 = filter(numerator, denominator, u(k_start:end));
t_simulated = t(k_start:end);

subplot(2,1,2);
plot(t_simulated, y(k_start:end), 'r', t_simulated, simulated_output2, 'b--');
title('Offset-Free Model Verification (Entire)');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-2 2]);
legend('Actual Output', 'Simulated Output');
grid on;
```

Here is the generated figure.

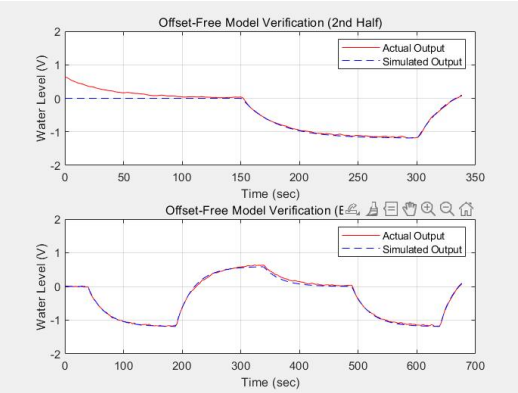

Fig 4

6. Conclusion

This lab exercise demonstrated how to extract data, remove offsets, and identify a second-order discrete-time linear model from input-output data. The model was validated using both the second half and the entire dataset, showing close agreement between the simulated and actual outputs. This suggests the model effectively captures the system dynamics of the water tank.

Post-lab Exercise: Comparison of First and Second-Order Models

1. Introduction

The goal of this post-lab exercise is to compare the accuracy of first-order and second-order discrete-time linear models in system identification. The comparison is based on the Mean Squared Error (MSE) between the simulated model responses and the actual system output.

2. Model Identification

We follow the same procedure used in the previous lab exercise to identify both first-order and second-order models using the offset-free input (u) and output (y). The two models are compared by simulating their outputs and comparing them to the actual system response.

Step 1: Data Extraction

The first step involves loading the pre-collected data (SysIdenData_StudentVersion.mat), extracting the time, input, and output signals, and removing the offset from the signals.

```
% Step 1
load('SysIdenData_StudentVersion.mat');
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;

i = 1;
while u_act(i) == u_act(1)
    i = i + 1;
end
% Output offset
u_offset = u_act(1);
u = u_act - u_offset;
% Input offset
y_offset = mean(y_act(1:i-1));
y = y_act - y_offset;
```

Step 2: Second-Order Model Identification

Using the offset-free input-output data, the second-order model parameters are identified by creating the matrix Φ and solving the system of equations. The transfer function and state-space models are then derived.

```
% Step 2 second order model
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];

theta = inv((Phi' * Phi)) * (Phi' * Y);
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2

% transfer function
h = 0.01;
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1
```

Step 3: First-Order Model Identification

Similarly, a first-order model is identified using the same dataset, but with a simplified equation structure.

```
% Step 3 first order model
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start_2 = 3;
Y_2 = y(k_start:N);
Phi_2 = [y(k_start-1:N-1), u(k_start-1:N-1)];

theta_2 = inv((Phi_2' * Phi_2)) * (Phi_2' * Y_2);
a1_2 = -theta_2(1);  % a1
b1_2 = theta_2(2);  % b1

% transfer function
numerator_2 = b1_2;
denominator_2 = [1 a1_2];
Gz_2 = tf(numerator_2, denominator_2, h);% Eq1
```

3. Model Simulation and Comparison

After identifying both models, we simulate the system response using the entire dataset for both first and second-order models. The simulated responses are then compared with the actual output, and the MSE for each model is calculated.

```
% Simulating both models
simulated_output_1st = filter(numerator, denominator, u(k_start:end));
simulated_output_2nd = filter(numerator_2, denominator_2, u(k_start:end));
t_simulated = t(k_start:end);

% First Order MSE
MSE_1st = mean((y(k_start:end) - simulated_output_1st).^2);

% Second Order MSE
MSE_2nd = mean((y(k_start:end) - simulated_output_2nd).^2);
mse_str_1 = sprintf('%.8f', MSE_1st);
mse_str_2 = sprintf('%.8f', MSE_2nd);

figure;
subplot(1,1,1);
plot(t_simulated, y(k_start:end), 'r', t_simulated, simulated_output_1st, 'g--',t_simulated, simulated_output_2nd, 'b--');
title('Comparison of Different Offset-Free Order Models','FontWeight', 'bold');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-1.5 1]);
legend('Actual Output', '1st Order Model Response', '2nd Order Model Response');
grid on;

text(10, 0.56, ['MSE1 = ', num2str(mse_str_1)], 'FontSize', 9.5);
text(10, 0.44, ['MSE2 = ', num2str(mse_str_2)], 'FontSize', 9.5);
```

4. Results and Discussion

The comparison between the two models is visualized in the plot, with the MSE values for both models displayed on the graph.

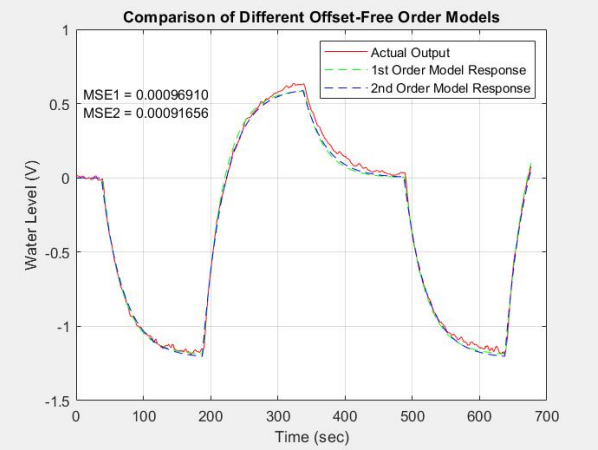

Fig 5

First-order model MSE: 0.00096910

Second-order model MSE: 0.00091656

The second-order model demonstrates a slightly lower MSE compared to the first-order model, indicating a marginally better fit to the actual data. Both models track the actual output with reasonable accuracy, but the second-order model captures the system dynamics with more precision due to the inclusion of an additional parameter. The smaller MSE for the second-order model suggests that it provides a more accurate representation of the system's behavior, especially during more complex transitions.

5. Conclusion

The second-order model, with a lower MSE (0.00091656), offers better accuracy compared to the first-order model (0.00096910). While the difference is small, the second-order model is preferable for applications requiring precision. However, the first-order model remains a viable option when simplicity and computational efficiency are prioritized.