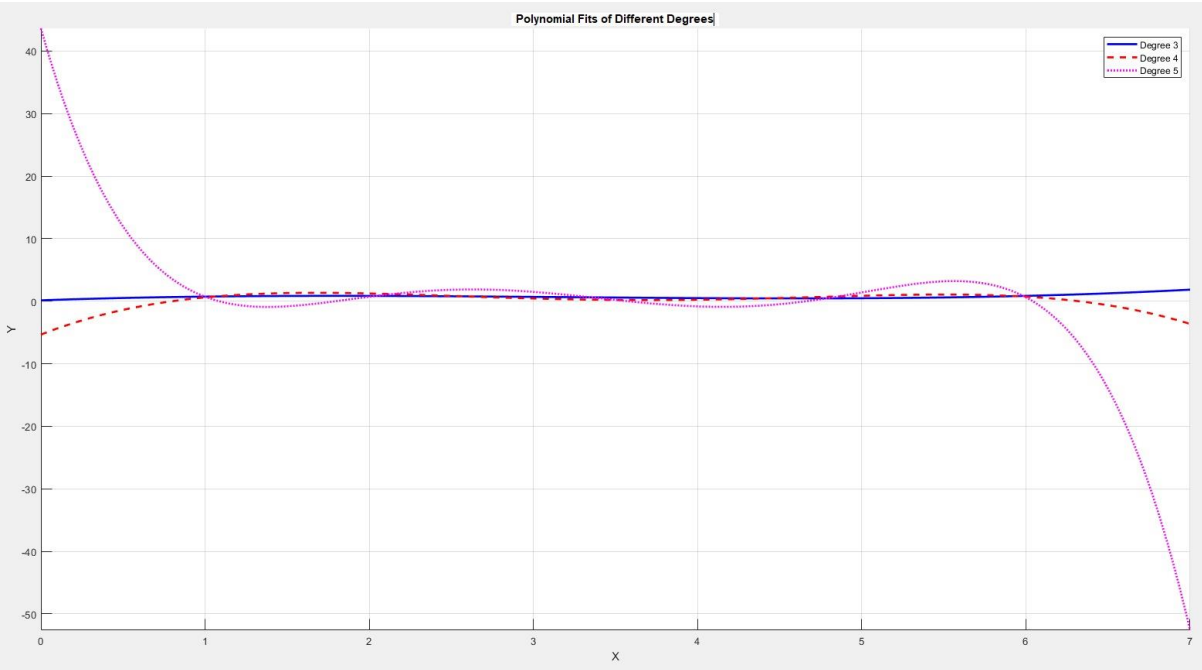
Advanced Control Lab – Matlab Simulink Exercise

Submitters:

Or Shaul

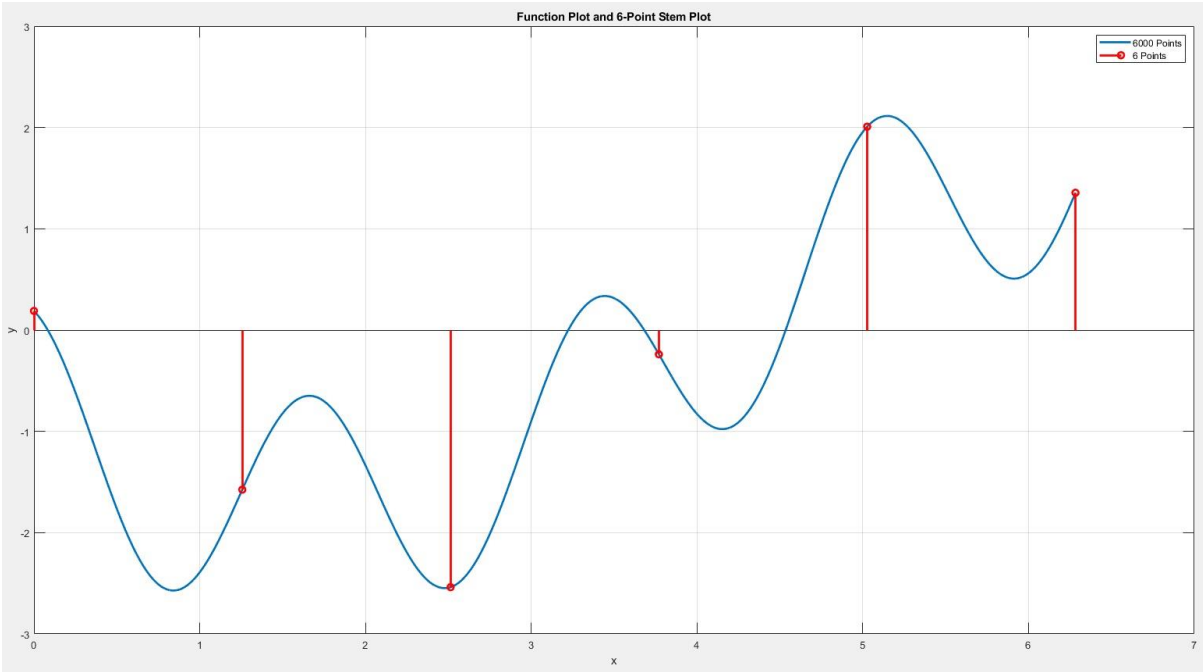
Saray Sokolovsky

1.




```
% Generate a vector of 1x6 random values with normal distribution
vector1 = randn(1, 6);

% Generate a second vector Y which elements are the inverses of the first vector elements
Y = 1./vector1;

% Generate a third vector X which consists of natural numbers 1 to 6
X = 1:6;

% Fit the Y vector to the X vector using polynomial fitting of degrees 3, 4, and 5
p3 = polyfit(X, Y, 3);
p4 = polyfit(X, Y, 4);
p5 = polyfit(X, Y, 5);

% Define the range for plotting
x_range = 0:0.01:7;

% Compute the fitted values
y_fit3 = polyval(p3, x_range);
y_fit4 = polyval(p4, x_range);
y_fit5 = polyval(p5, x_range);

% Plotting the curves
figure;
hold on;

% Plot the third-degree polynomial fit
plot(x_range, y_fit3, 'b-', 'LineWidth', 2);

% Plot the fourth-degree polynomial fit
plot(x_range, y_fit4, 'r--', 'LineWidth', 2);

% Plot the fifth-degree polynomial fit
plot(x_range, y_fit5, 'm:', 'LineWidth', 2);

% Set the axis to fit the graph
axis([0 7 min([y_fit3 y_fit4 y_fit5]) max([y_fit3 y_fit4 y_fit5])]);

% Add labels and title
xlabel('X');
ylabel('Y');
title('Polynomial Fits of Different Degrees');
legend('Degree 3', 'Degree 4', 'Degree 5');

% Display the grid
grid on;

hold off;
```

2.

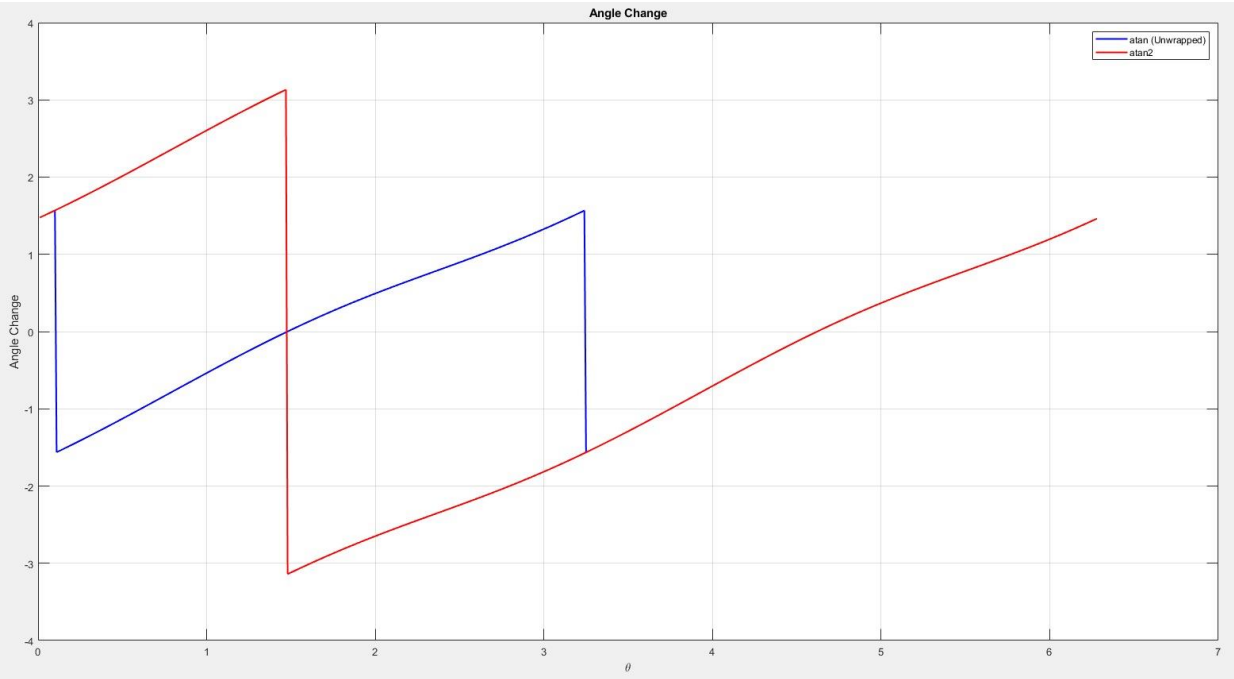


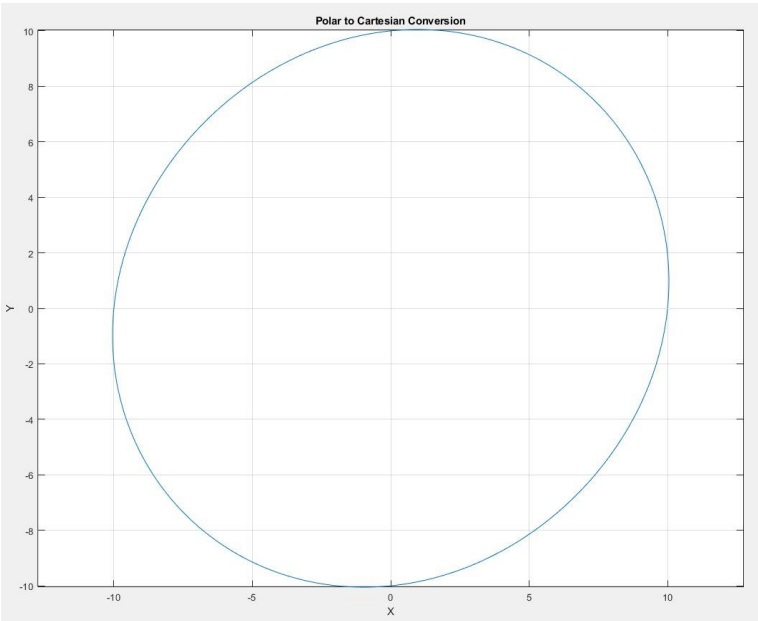

```
% Generate a vector of 1x6 random values with normal distribution
rng('default'); % For reproducibility
A = randn(1, 6);

% Multiply the vector by 2
B = 2 * A;

% Extract the coefficients
a = B(1);
b = B(2);
c = B(3);
d = B(4);
e = B(5);
f = B(6);

% Define the x range
x_range = linspace(0, 2*pi, 6000);

% Define the function y
y = a * sin(b * x_range + c) + d * sin(e * x_range + f);

% Plot the function with 6000 points
figure;
plot(x_range, y, 'LineWidth', 2);
hold on;

% Plot the same function for 6 equally spaced points
x_points = linspace(0, 2*pi, 6);
y_points = a * sin(b * x_points + c) + d * sin(e * x_points + f);
stem(x_points, y_points, 'ro', 'LineWidth', 2);

% Manually adjust the figure using plot tools
xlabel('x');
ylabel('y');
title('Function Plot and 6-Point Stem Plot');
legend('6000 Points', '6 Points');

% Show the grid
grid on;
hold off;
```

By visually comparing the plots from the 6000-point data and the 6-point data using ‘cftool’, we can assess the fitting accuracy and any differences between the detailed data and the sparse data. Typically, the 6000-point data will provide a more accurate representation of the function, while the 6-point data might lead to a less accurate fit, especially in capturing the nuances of the function's behavior.

3.

Using Or's ID: a=1, b=9, c=2, d=0:





```
% Define values
a = 1;
b = 9;
c = 2;
d = 0;

% Define theta vector
theta = 0:0.01:2*pi;

% Calculate rho
rho = (a + b) + 0.5*sin((c + d)*theta);

% Convert polar coordinates to Cartesian coordinates
x = rho .* cos(theta);
y = rho .* sin(theta);

% Plot the graph
figure;
plot(x, y);
title('Polar to Cartesian Conversion');
xlabel('X');
ylabel('Y');
grid on;
axis equal;

% Calculate delta values (dx, dy)
dx = diff(x);
dy = diff(y);

% Calculate angle change using atan and atan2
angle_change_atan = atan(dy ./ dx);
angle_change_atan2 = atan2(dy, dx);

% Calculate angle change using atan and unwrap
angle_change_atan_unwrapped = unwrap(angle_change_atan);

% Plot the angle change results
figure;
plot(theta(2:end), angle_change_atan_unwrapped, 'b', 'LineWidth', 1.5);
hold on;
plot(theta(2:end), angle_change_atan2, 'r', 'LineWidth', 1.5);
title('Angle Change');
xlabel('\theta');
ylabel('Angle Change');
legend('atan (Unwrapped)', 'atan2');
grid on;
```

The Polar to Cartesian Conversion Graph:

This graph shows the conversion of polar coordinates to Cartesian coordinates. The x-axis represents the Cartesian x-coordinate, and the y-axis represents the Cartesian y-coordinate. Each point on the graph corresponds to a pair of Cartesian coordinates (x, y) coming from the polar coordinates (rho, theta). The shape of the graph will depend on the formula used to calculate rho and the range of theta. In this case, rho is calculated using the given formula, and theta ranges from 0 to 2π.

The Angle Change Graph:

This graph shows the change in angle as theta varies, calculated using both the atan function (after unwrapping) and the atan2 function. The x-axis represents the values of theta (from 0 to 2π). The y-axis represents the change in angle. The blue curve represents the change in angle calculated using the atan function after applying the unwrap operation to make the graph continuous and the red curve represents the change in angle calculated using the atan2 function (which handles discontinuities and doesn't require an unwrap operation).

4.

Convert the equation to the state space representation by decreasing it`s order:

$$\ddot{\alpha} + 4\dot{\alpha} + 3\alpha = 2u - \dot{u}$$

$$X_1 = \alpha, X_2 = \dot{\alpha} = \dot{X_1}, \dot{X_2} = \ddot{\alpha}$$

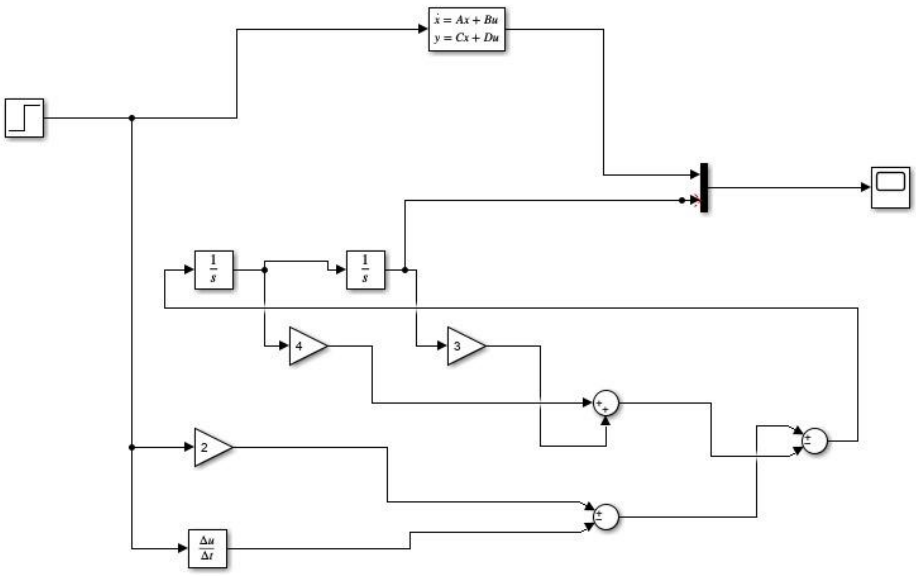
$$\dot{X_2} = 4X_2 + 3X_1 = 2u - \dot{u}$$

$$\begin{cases} \dot{X_1} = X_2 \\ \dot{X_2} = 2u - u - 4\dot{X_2} - 3X_1 \end{cases}$$

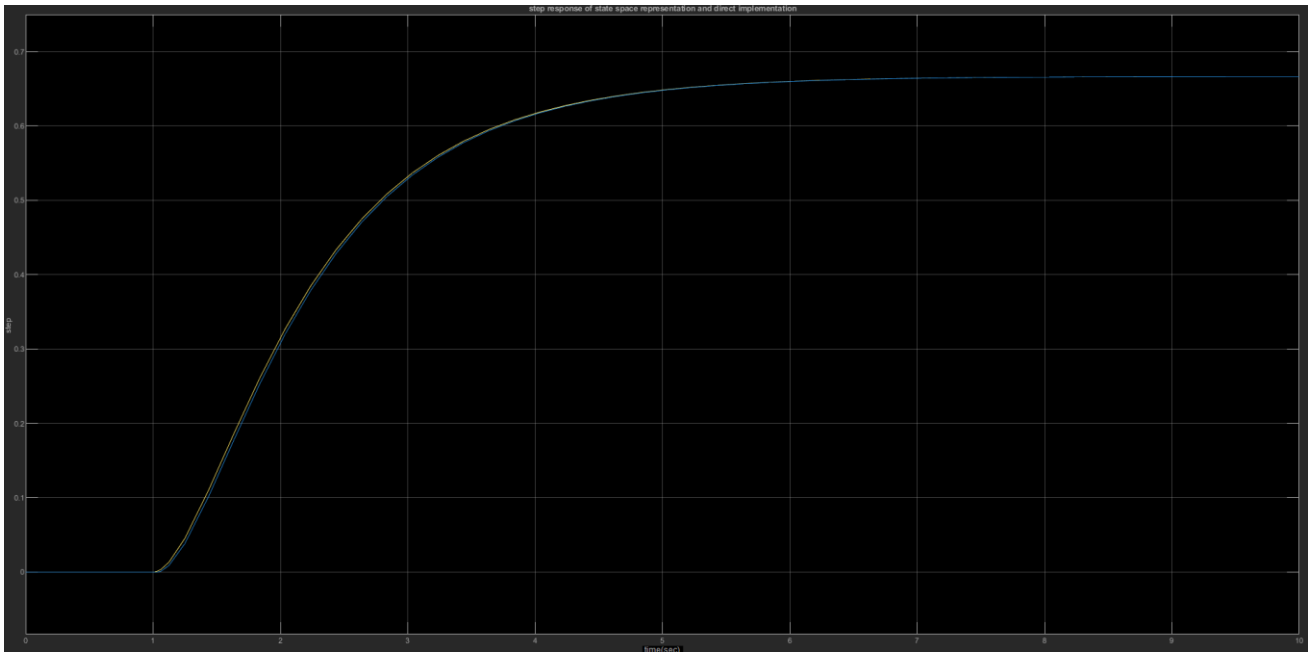
$$\begin{bmatrix} \dot{X_1} \\ \dot{X_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 2u - \dot{u} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, D = 0, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, A = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix}$$

In Simulink:

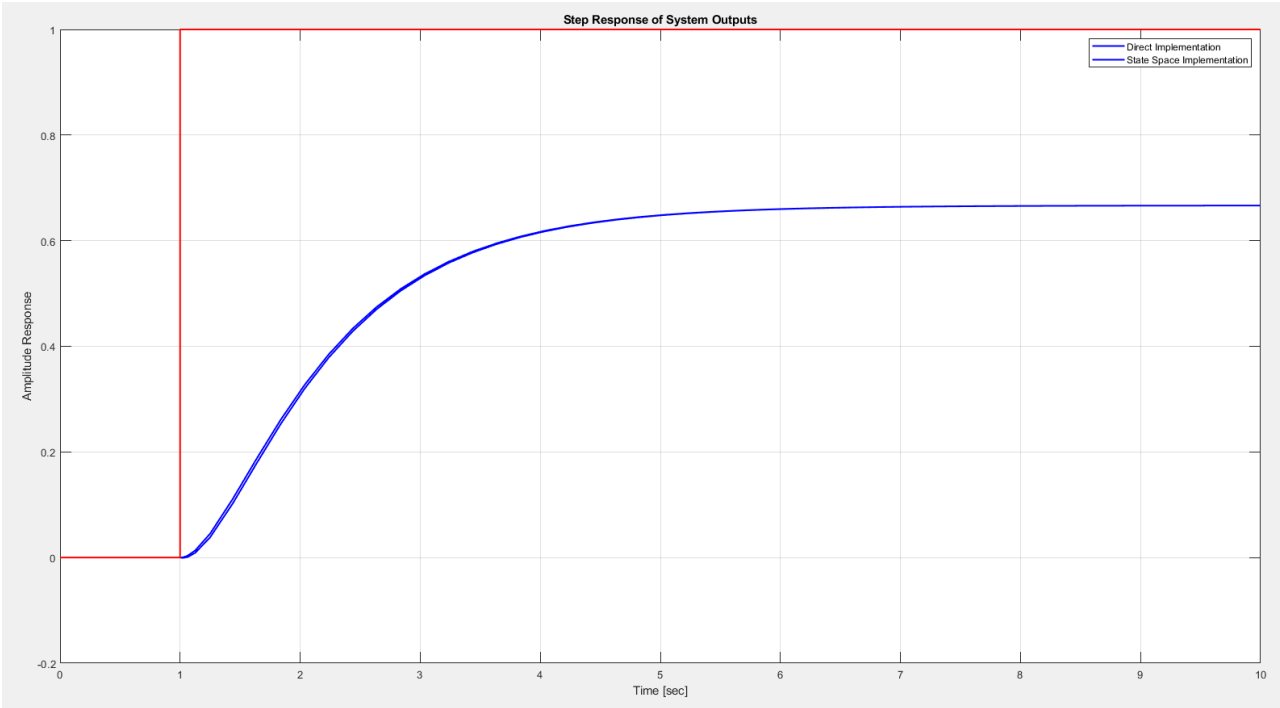


The scope:



Set the simulation for 10 seconds and record the output from the scopes using an array variable, Write a Matlab script which runs the simulation and plots the responses on the plot:

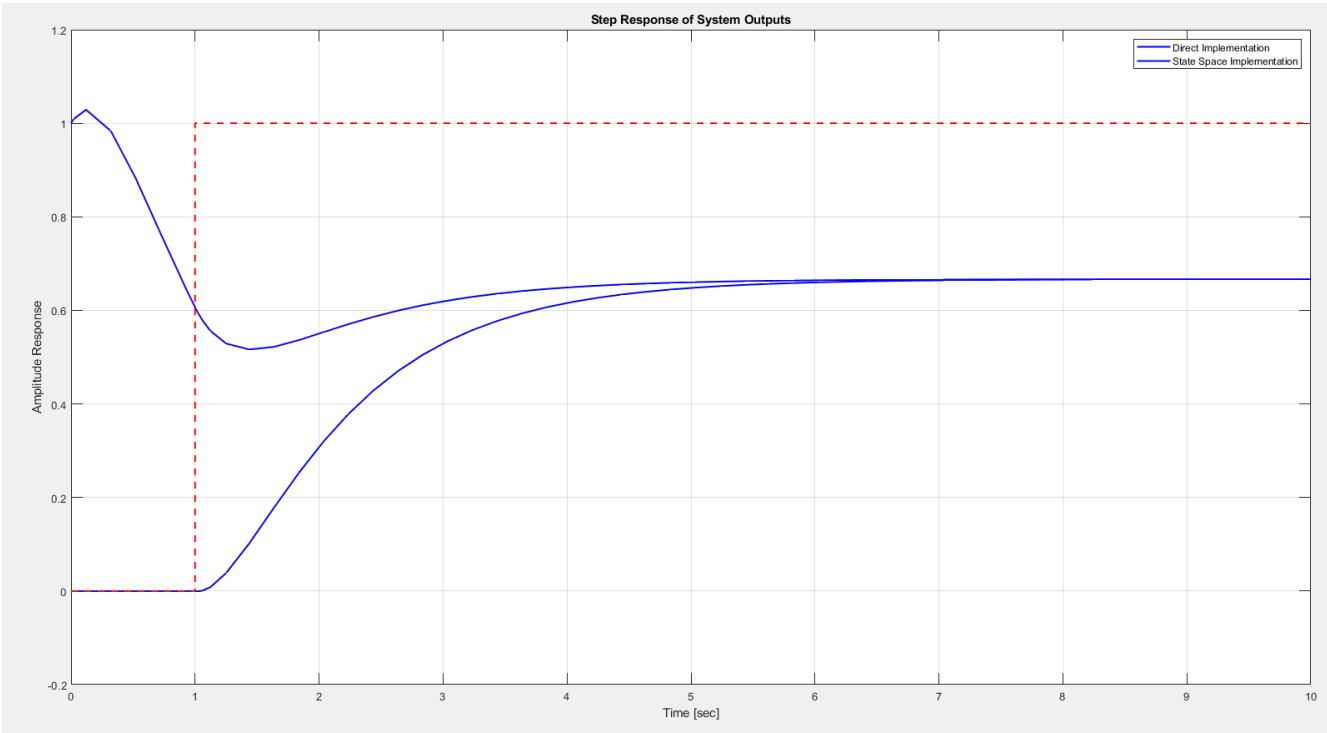
```
% Simulate the Simulink model
simOut = sim('Matlab_itro');

% Extract time and output data from the simulation
t = simOut.tout; % Time vector

% Extract signals directly from simOut structure
alpha = simOut.get('simout'); % Timeseries object for alpha
alpha_dot = simOut.get('simin'); % Timeseries object for alpha_dot

% Extract the data from the timeseries objects
alpha_data = alpha.Data;
alpha_dot_data = alpha_dot.Data;
time = alpha.Time; % Assuming both timeseries objects have the same time vector

% Plotting the responses
figure;
plot(time, alpha_data, 'b', 'LineWidth', 1.5);
hold on;
plot(time, alpha_dot_data, 'r--', 'LineWidth', 1.5);
xlabel('Time [sec]');
ylabel('Amplitude Response');
title('Step Response of System Outputs');
legend('Direct Implementation', 'State Space Implementation');
grid on;
hold off;

% Check if responses are identical
if isequal(alpha_data, alpha_dot_data)
    disp('Responses are identical.');
else
    disp('Responses are not identical. Check your implementation.');
end
```



We can see the responses identical (there are two blue plots almost identical).

The phenomenon present at the beginning of the step response is - This is usually referred to as "transient response," where the system adjusts to changes before reaching a steady state.

The state space implementation adding the initial conditions $\alpha(0) = 0.5, \alpha'(0) = 1$:



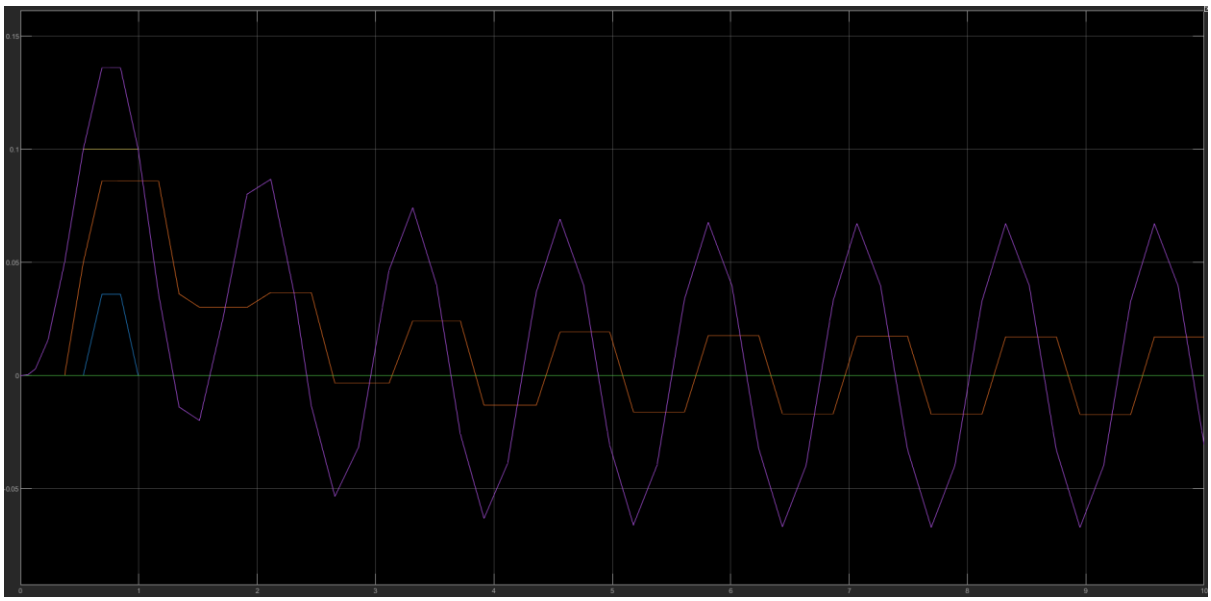
We can see that the steady state is the same for the state space and the direct implementation.

The matrixes C, D for the case of $y = [\alpha \quad \dot{\alpha}]^T$:

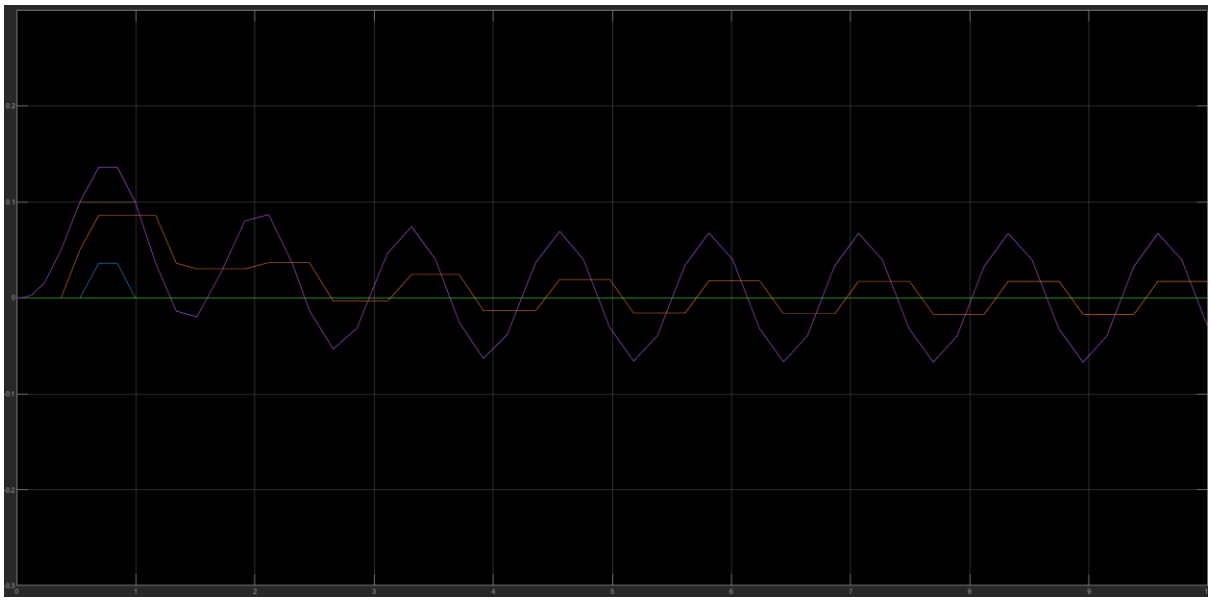
$$c = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

5. Calculate the transfer function of $\frac{\alpha(s)}{u(s)}$

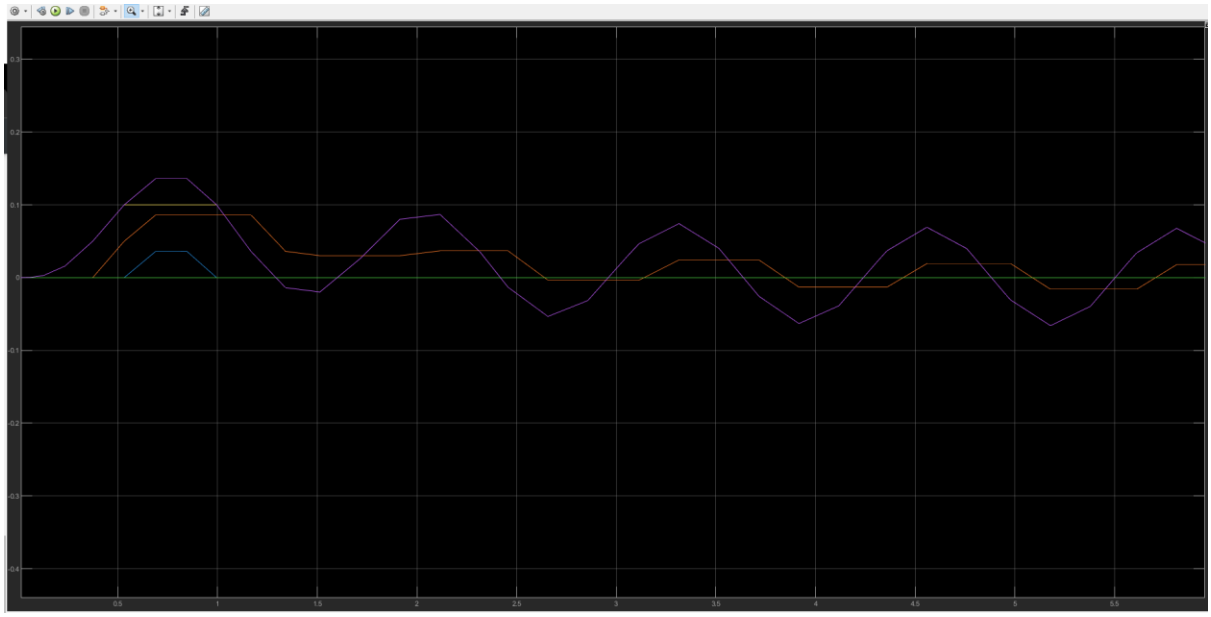
With Auto range:



With y range:

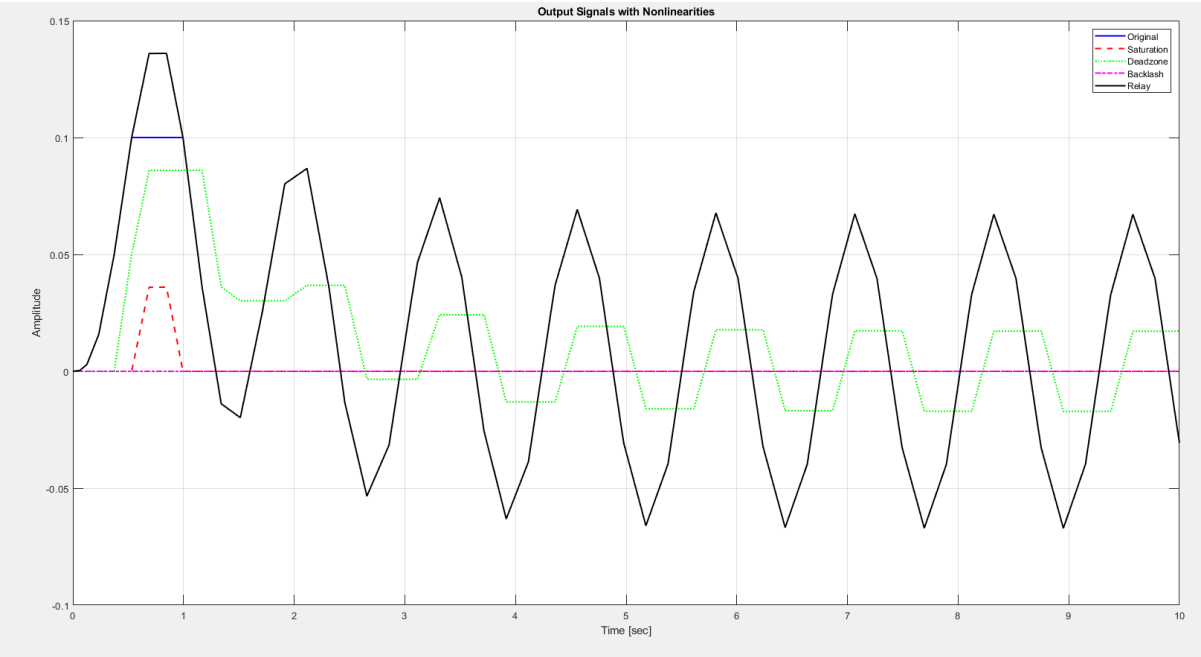


With magnifying glass:



Using Matlab plot the outputs from the scope:

```
% Simulate the model
simOut = sim('MATLAB_INTRO_Q5');

% Extract data from the simulation
time = simOut.tout; % Time vector
outputs = simOut.get('simout'); % Assuming 'simout' is the name used in 'To Workspace' block

% Extract individual signals from the timeseries object
original_signal = outputs.Data(:, 1);
saturation_signal = outputs.Data(:, 2);
deadzone_signal = outputs.Data(:, 3);
backlash_signal = outputs.Data(:, 4);
relay_signal = outputs.Data(:, 5);

% Plotting the signals
figure;
plot(time, original_signal, 'b', 'LineWidth', 1.5); hold on;
plot(time, saturation_signal, 'r--', 'LineWidth', 1.5);
plot(time, deadzone_signal, 'g:', 'LineWidth', 1.5);
plot(time, backlash_signal, 'm-.', 'LineWidth', 1.5);
plot(time, relay_signal, 'k', 'LineWidth', 1.5);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Output Signals with Nonlinearities');
legend('Original', 'Saturation', 'Deadzone', 'Backlash', 'Relay');
grid on;
hold off;
```



Describe the Nonlinearities:

Saturation: The Saturation block limits the output to a specified range. In the plot, you will see that the signal is clipped at the upper and lower limits, making it look flat at those levels.

Dead Zone: The Dead Zone block ignores input changes within a specified range. This results in the output being zero when the input is within the deadzone range, which will be visible as flat regions around zero in the plot.

Backlash: The Backlash block models mechanical play by allowing a difference between input and output within a specified range. The output will show a delayed or smoothed response when compared to the input, which can be seen as a lag in the signal.

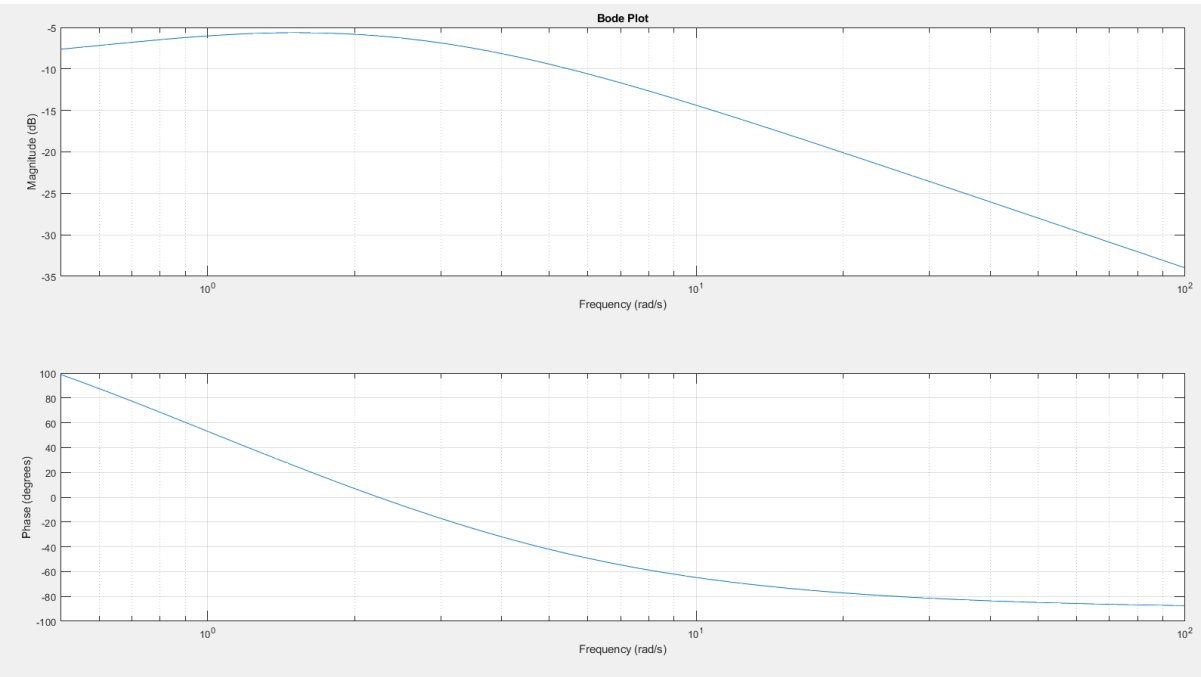
Relay: The Relay block switches the output between two levels based on input thresholds. This results in a square wave-like response, where the signal jumps between the two specified levels.

6. Plot the bode response for the previous transfer function

Sample the plot 5 times for frequencies in the range of 0.5 to 100 rad/sec:

```
freq = logspace(log10(0.5), log10(100), 5); % Frequencies from 0.5 to 100 rad/s
omega = freq; % Already in rad/s

% Calculate gain
s = 1i * omega;
gain = abs((2*s - 1) ./ (m*s.^2 + c*s + k));

% Convert gain to dB
gain_db = 20 * log10(gain);
gain_linear = 10.^(gain_db / 20);

% Display results
disp('Frequency (rad/s):');
disp(freq);
disp('Gain (linear):');
disp(gain);
disp('Gain (dB):');
disp(gain_db);
```

```
Frequency (rad/s):
    0.5000    1.8803    7.0711   26.5915  100.0000

Gain (linear):
    0.4159    0.5161    0.2585    0.0747    0.0200

Gain (dB):
   -7.6202   -5.7458  -11.7523  -22.5338  -33.9836
```

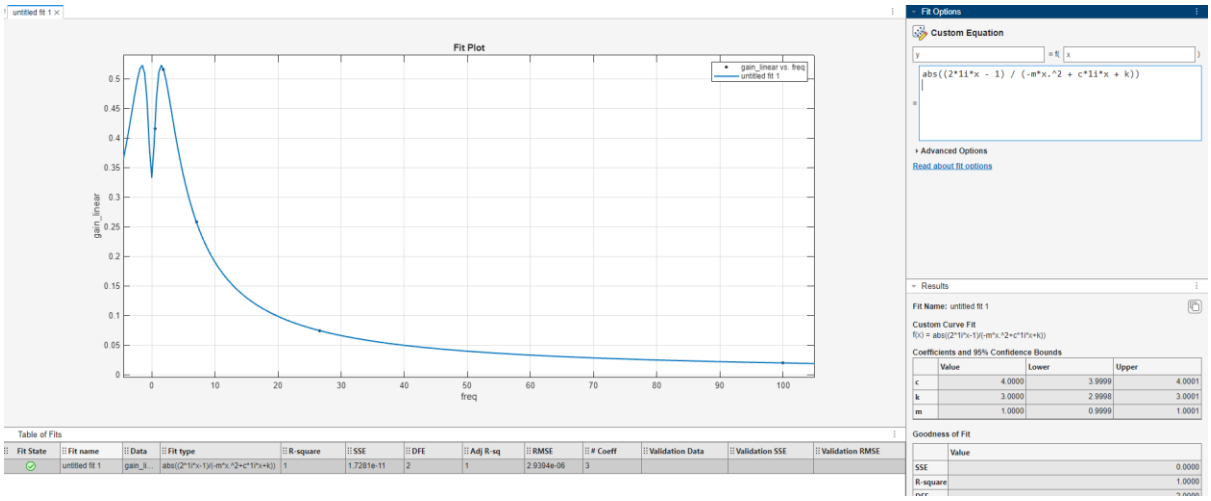


Calculate the gain function:

$$\frac{\alpha(s)}{u(s)} = \frac{2s-1}{ms^2+cs+k} = \frac{2s-1}{s^2+4s+3}$$

$$\frac{\alpha(jw)}{u(jw)} = \frac{2jw - 1}{-w^2 + 4jw + 3}$$

Using cftool with custom equation fitting find the coefficients using the calculated gain function and the sampled points taken before:



We can see we got the coefficients :c, k, m.