ME 417 Control of Mechanical Systems

Summer 2020 - Numerical Assignment #1

Complete the numbered tasks and submit your work as a MATLAB Livescript in addition to an exported PDF report

PART I: Simulating A System Response

In this part you will be simulating a general second-order dynamic system using different numerical methods, and you will also be exploring the effect of changing the system parameters on the response.

The general form of a second-order dynamic system represented by Figure 1, is given by the transfer functionIn this part you will be simulating a general second-order dynamic system using different numerical methods, and you will also be exploring the effect of changing the system parameters on the response.

The general form of a second-order dynamic system represented by Figure 1, is given by the transfer function

$$\frac{X(s)}{U(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad (1)$$

Where ω_n and ζ are given by the definitions: $\omega_n = \sqrt{\frac{K}{M}}, 2\zeta\omega_n = \frac{f_v}{M}$

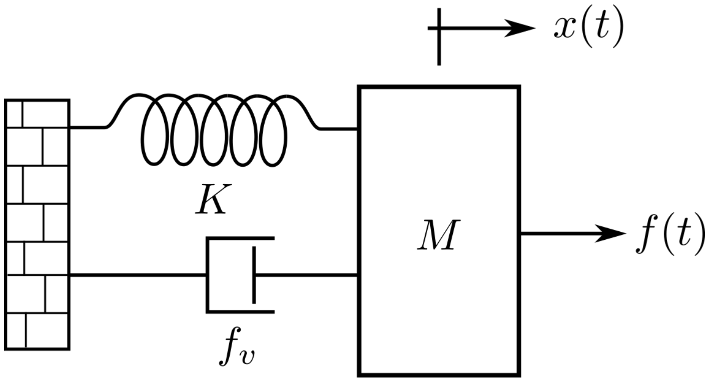


Figure 1 - General Second Order (Spring-Mass-Damper) System

Given the following parameters: $M = 7\text{kg}, K = 2.8\frac{N}{m}, f_v = 1.5N - \frac{s}{m}$

1. Simulate the response of the system to a step input using the command *step(G)*. Repeat the step simulation with $f_v = 5$, and $f_v = 20$. Plot the three responses on the same plot. Explain the observed difference between the three responses.

Using the command ***step()*** is useful for analyzing a system's response to a simple and constant input to a system, but what if the input varies? We can instead use the command ***lsim()***

2. Repeat task 1 above, but to an input force defined by the piecewise function below, using the command *lsim()*, from $t = 0s$ to $t = 100s$, and plot the result in a new figure.

$$f(t) = \begin{cases} 0 & 0 < t \le 2.5s \\ 6 & 2.5s < t \le 9s \\ -6 & 9s < t \le 16s \\ 0 & 16s < t \end{cases}$$

Hint: typing >> help lsim in MATLAB's command window will guide you on how to use the function.

Now, what if we don't know the input function in advance, more specifically, what if the input is a function of the output of the system $f(t, y, y_{\text{sp}})$, as in the case of any feedback control system. In this case we must use a different method to simulate the response. Also note that the methods above require the system to be linear and time-invariant. The next methods can be used to simulate any dynamic system (nonlinear and/or time-variant) if the differential equations and initial conditions are defined.

Given the following differential equation

$$a\ddot{x}(t) + b\dot{x}(t) + \text{c}x(t) = u(t) \quad (2)$$

We can separate it into two equations that can then be used in MATLAB to simulate the response. Let's re-write the equation variables as $x_1 = x(t), x_2 = \dot{x}(t)$. If we define a vector $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ then $\begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$. In other words

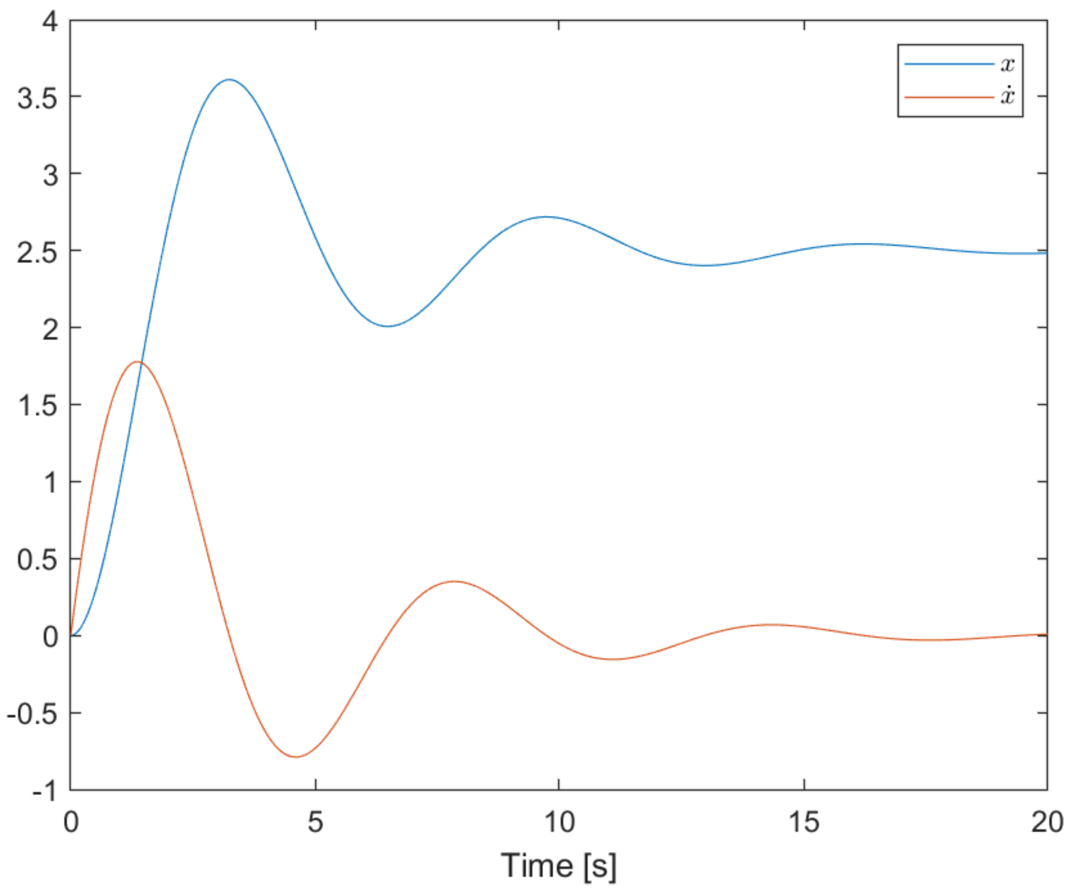
$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \dfrac{u(t) - \text{c}x_1 - \text{b}x_2}{a} \end{bmatrix}$$

If we start off with some given intial conditions for x_1, x_2, we can numericall integrate $\dot{\boldsymbol{x}}$ at consecutive timesteps to obtain $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$ over a given time period $t \in [0, t_f]$. In the In the following code snippet is an example on how to simulate the above equation in MATLAB using ***ODE45()***. ***ODE45()*** is MATLAB's Ordinary Differential Equation solver using the Runge-Kutta integration method with a variable time-step for efficient computation.

Here is a code xample for simulating the above system using **ode45()**

```
clear all; close all;
% System Parameters
a = 2;
b = 1;
c = 2;
u = 5; % This is the input, it can also be a variable

% Initial Conditions
x0 = [0; 0];

% funtion for xdot vector - using an anonymous function. You can also
% define a separate function in a separate file or at the end of the
% script
dxdt = @(t,x) [x(2); (u - b * x(2) - c * x(1)) / a];
```

```
%Time vector
t = 0:0.05:20;

% Simulate using ODE45
[t_sim, x_sim] = ode45(dxdt,t,x0);

% And plot
plot(t_sim, x_sim)
legend('$x$','$\dot{x}$', 'Interpreter','latex')
xlabel('Time [s]')
```

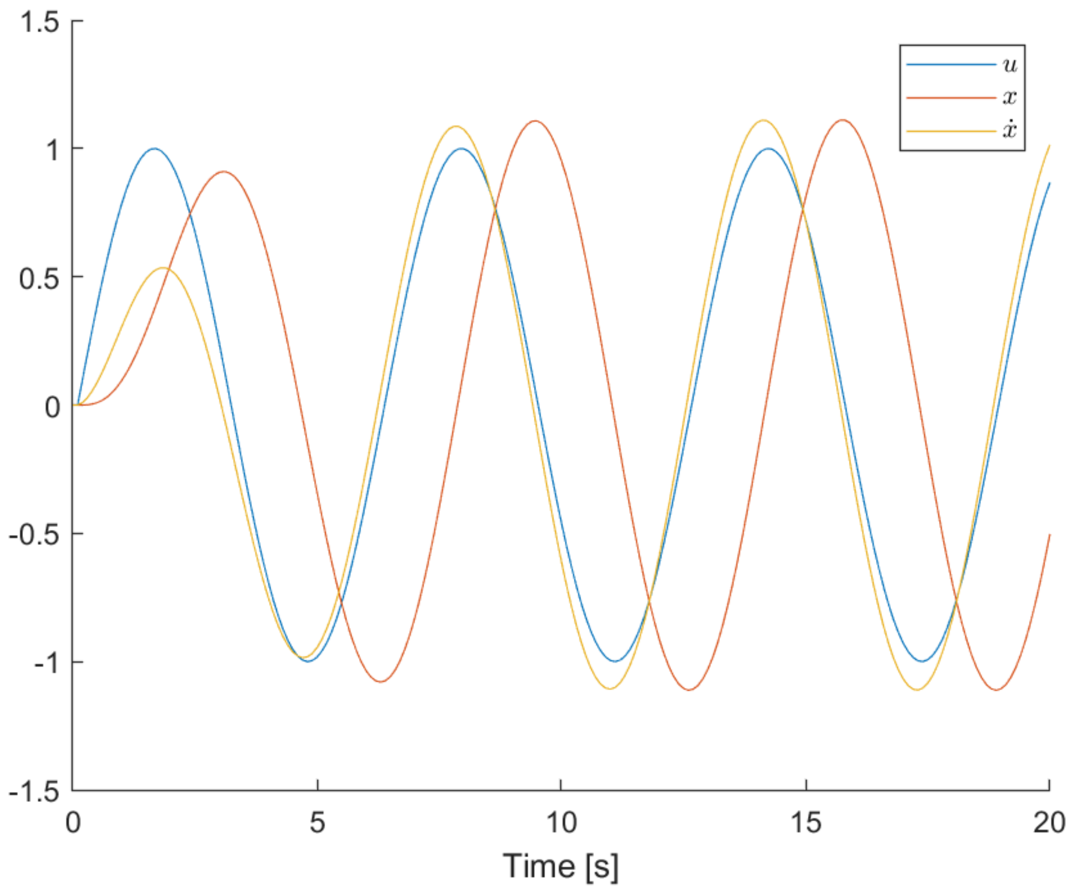


3. Obtain the differential equation from the transfer function on the first page from equation (1), into the form shown in equation (2), then simulate the response to a sinusoidal inputof unity magnitude and $f = 0.2\text{Hz}$**,** $u(t) = \sin(2\pi \text{ft})$ **using *ode45()*, also change the value of the damping coefficient** f_v **to** $f_v = 0.1\dot{x}$**. You now have a nonlinear system. Simulate for** $t = [0, 100s]$ **and plot both** $x, \dot{x}$ **on the figure. Note that the differential equation for the spring-mass-damper system above can also be expressed as:**

$$M\frac{d^2x}{\text{dt}^2} + f_v\frac{\text{dx}}{\text{dt}} + \text{Kx} = \text{Ku}$$

Another method for solving a differential equation is using basic numerical integration with a fixed time-step. This is generally less efficient for higher order systems, but is more flexible in simulating dynamic systems, specifically, in simulating systems in real-time (simulating the dynamics of a car in a video-game is an example). Moreover, this method is sufficient for simulating many mechanical dynamic systems. The code snippet below shows an example of basic forward integration to simulate a second-order system response; the same system from equation (*3*).

```
clear all
close all
% System Parameters
a = 1;
b = 1;
c = 1;
% Initial Conditions
x0 = [0; 0];

% funtion for xdot vector - using an anonymous function. You can also
% define a separate function in a separate file or at the end of the
% script

dxdt = @(t,x,u_) [x(2); (u_ - b * x(2) - c * x(1)) / a];

% Integration Time step
dt = 0.1;
% Time vector
t = 0:dt:20;

% Initialize x
x_sim = zeros(2,length(t)); % Empty 2xn array
u = zeros(1,length(t)); % Empty 1xn vector
x_sim(:,1)= x0;

hold on
for ix = 1:length(t)-1
    u(1,ix+1) = sin(t(ix)); % Calculate the next input value
    xdot = dxdt(t(ix), x_sim(:,ix), u(1,ix+1)); % Grab the derivative vector
    x_sim(1, ix+1) = x_sim(1, ix) + xdot(1) * dt; % Integrate x1
    x_sim(2, ix+1) = x_sim(2, ix) + xdot(2) * dt; % Integrate x2
end

% And plot
plot(t,u(1,:))
plot(t,x_sim(1,:))
plot(t,x_sim(2,:))
xlabel('Time [s]')
legend('$u$','$x$','$\dot{x}$', 'Interpreter','latex')
```



4. Repeate the previous task 3, but this time, using basic numerical integration, with the following time steps $\Delta t = 0.01s$, $\Delta t = 0.001s$. Then plot the responses from the *ode45()* method and the basic integration method, as three subplots on the same figure (Three: Ode45(), $\Delta t = 0.01s$, $\Delta t = 0.001s$) . Do you notice a difference in the response between $\Delta t = 0.01s$ and $\Delta t = 0.001s$?

Part II: Modeling and Simulating Real-World Systems

Model 1. Satellite Attitude

We wish to model the 2-D attitude of a space satellite body. The 2-D satellite body can be modeled as a rigid-body mass with a moment of inertia I, rotating about its center of gravity $C.G.$ If we neglect the reaction moment from the satellite solar panels attached to the body, the remaining forces acting on the satellite are the thrust forces from the on-board gas thrusters in addition to disturbances that can be modeled as a moment M_D, as shown on Figure 2.

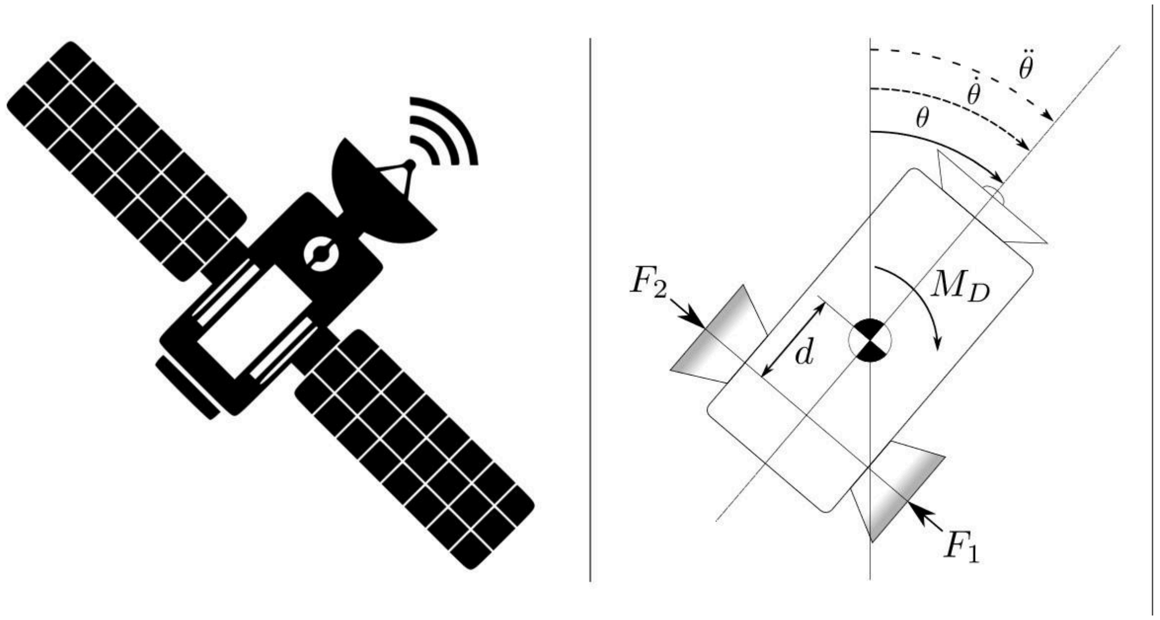


Figure 2 - Space Satellite Body 2-D Attitude Control

Given $d = 2m$, $I = 3.5\text{kg} \cdot m^2$

5. Derive the equation of motion for the satellite body shown, then derive the transfer function relating the resultant thrust force $F_R = F_1 - F_2$ to the angular position of the body θ, that is, find $G(s) = \frac{\Theta(s)}{F_R(s)}$. Neglect M_D

6. Given the system equation of motion, simulate the response to the following inputs, assuming $M_D(t) = 0$. Simulate the response using basic numerical integration.

$$F_1(t) = \begin{cases} 0 & 0 < t < 3s \\ 4 & 3s \le t \le 5s \\ 0 & 5s < t \end{cases}, \quad F_2(t) = \begin{cases} 0 & 0 < t < 1s \\ 4 & 1s \le t \le 3s \\ 0 & 3s < t \end{cases}$$

The thruster input function above is termed a bang-bang command. The idea is to apply an equal but opposite set of moments in order to roll the satellite body over a defined angle. The gas thrusters can only act in one direction, so a pair is needed to move-then-stop. This input is termed an open-loop input, meaning it doesn't consider the actual response in the system (it's blind toward the system's actual response). In reality it's quite impossible to control a satellite body in this fashion, we will need to employ a feedback controller to provide just the right amount of thruster force to perform a rotation. Next we will see the effect of adding just a tiny amount of disturbance noise to the model.

7. Repeat the simulation but this time assume that the disturbance moment is modeled as a zero-mean Gaussian noise with $\sigma = 0.005$. Hint: to generate this random noise value use: $\sigma \times \text{randn}(1)$

Model 2. Simple Pendulum

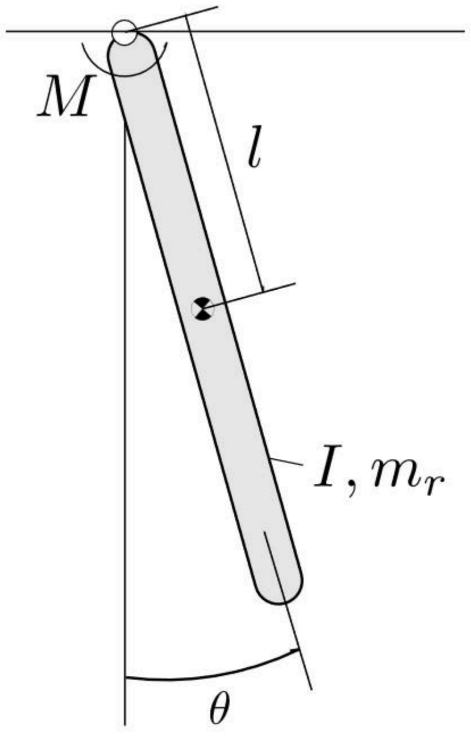


Figure 3 - Simple Pendulum

Given $m_r = 3, l = 19\text{cm}$

8. Derive the equations of motion and simulate the response of the above <u>nonlinear</u> system given the following input function and plot the following responses:$\theta, \dot{\theta}$. Simulate using basic numerical integration. If the response is unstable (grows out of bounds) try to reduce the integration time step Δt

$$M = \begin{cases} 5N \cdot m & 0 < t < 3s \\ 0 & 3s \le t \end{cases}$$

In order to design a controller for this system using the tools we introduce in this course; the system must be linearized. In this case we can linearize the system using the small angle approximation assumption. We can do this because we generally intend to keep (control) the inverted pendulum close to $\theta \approx 0^o$

9. Linearize the above equations using the small angle approximation. That is,assume $\sin\theta = \theta$

10. Repeat task 7 using the linearized model and compare the nonlinear versus the nonlinear responses, in two subplots, one for each of $\theta, \dot{\theta}$. Briefly explain the observed differences. At what value of θ does the approximation become noticeably inaccurate?

Note that when experimenting with a controller design via simulation, we design the controller using the linearized model if we wish, but it is important to simulate the response as accurate as possible. That is, simulate the nonlinear response.

11. Briefly explain why it is better to simulate the nonlinear model and not the linearized model, when testing a controller?