Week 2 Tutorial Notes: Mathematical Modeling and Simulation

Zeyad Manaa, Mohammed Elbalshy

Sep 4, 2023

Supplementary document for AE 315
Systems and Control course
Fall 2023

Aerospace Department
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

Abstract

This document provides an introduction to mathematical modeling, specifically focusing on differential equations and their numerical integration schemes. The Euler method and MATLAB's ode45 solver are discussed as techniques for approximating solutions to differential equations. Reduction of order is explored as a method for simplifying differential equations. The application of mathematical modeling in electrical systems is also covered, with a focus on RL circuits and RLC circuits. Simulation techniques using different state variables are demonstrated. Additionally, the document delves into the modeling of mechanical systems, specifically the mass-spring-damper system. The content includes figures illustrating numerical integration, circuit simulations, and mechanical system simulations. This resource serves as a comprehensive guide to mathematical modeling and simulation techniques and their application in electrical and mechanical systems.

Contents

List of Figures

Listings

1 Introduction to Mathematical Modeling

The developed mathematical models of physical systems are a key factor for control systems design and analysis. Mathematical models are typically ordinary differential equations [1, 2]. Most physical systems in reality exhibits high nonlinearity. The vast and most completed body of the control theory has been developed for linear systems. But, the fact that most physical systems are nonlinear is kind of limitation to this complete theory [3, 4]. However, one can employ any linearization technique to linearize the nonlinear dynamics (mathematical model) in order to leverage the very solid linear control theory. This introduces the trade of between the accuracy of the mathematical model used in control systems. One can accept a linear less accurate model in favour of the applicable linear control technique, other may weigh the accuracy over the simplicity.

For most of of this course's material, we will be focusing on the linear control strategies as one pillar, more advanced techniques may be covered as we go on.

2 Differential Equations

Differential equations play a crucial role in describing the behavior of various physical, biological, and engineering systems. They are mathematical equations that involve derivatives and represent the rate of change of a function with respect to one or more independent variables.

2.1 Numerical Integration Schemes

The aim is to integrate a first-order ordinary differential equation (ODE) on the form

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t)) \tag{1}$$

over a finite time interval Δt, to convert them to a difference equation,

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \int_{t}^{t+\Delta t} f(\tau, \mathbf{x}(\tau))d\tau \tag{2}$$

or alternatively, if we assume that $t_n = n\Delta t$ and $\mathbf{x}_n \triangleq \mathbf{x}_n(t_n)$,

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \int_{t}^{t+\Delta t} f(\tau, \mathbf{x}(\tau))d\tau \tag{3}$$

Here, $\frac{d\boldsymbol{x}(t)}{dt}$ represents the derivative of the unknown function $\boldsymbol{x}$ with respect to the independent variable t, and $\boldsymbol{f}(t, \boldsymbol{x})$ is a known function that determines the rate of change. Analytically solving such equations involves finding an expression for $\boldsymbol{x}$ in terms of t by integrating both sides. For instance, if

$$\boldsymbol{f}(t, \boldsymbol{x}) = 2t \tag{4}$$

the solution becomes

$$\boldsymbol{x}(t) = t^2 + C \tag{5}$$

where C is the constant of integration. The solution is obtained by the separation of variables method. In a discrete manner, we can write $\boldsymbol{x}_n$

As discussed earlier, most of the differential equations governing a real physical systems are nonlinear. Obtaining an analytical solution for nonlinear differential equation can be tricky in most of the cases. Instead, we tend to utilize numerical integration methods.

2.1.1 Euler method - RK1

Numerically solving differential equations involves approximating the solution at discrete points. The simplest scheme is the first-order Runge-Kutta method (RK1), also known as the Euler method. Utilizing the discrete formulation of the derivative we can write,

$$\frac{d\boldsymbol{x}(t)}{dt} \approx \frac{\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t)}{\Delta t} \approx \boldsymbol{f}(t, \boldsymbol{x}(t)) \tag{6}$$

The goal is to solve for the next step. So,

$$\boldsymbol{x}(t+\Delta t) \approx \boldsymbol{x}(t) + \Delta t \boldsymbol{f}(t, \boldsymbol{x}(t)) \tag{7}$$

In a discrete format we can approximate the next step by using the forward difference formula:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \boldsymbol{f}(t_n, \boldsymbol{x}_n) \tag{8}$$

Here, Δt is the step size, $\boldsymbol{x}_n$ represents the numerical approximation of $\boldsymbol{x}(t_n)$ at the n^{th} step, and t_n denotes the n^{th} time point. MATLAB code for this scheme would look like:

Listing 1: Numerical Euler method vs. the analytical solution

```
% Define the step size and time points
stepSize = 0.05; % Step size
timePoints = 0:stepSize:1; % Time points

% Initialize the solution arrays
numericalSolution = zeros(size(timePoints)); % Initialize numerical solution array
analyticalSolution = timePoints.^2 + 1; % Analytical solution

% Solve the differential equation using RK1 (Euler's method)
numericalSolution(1) = 1; % Initial condition
for n = 1:length(timePoints)-1
    numericalSolution(n+1) = numericalSolution(n) + ...
    stepSize * (2 * timePoints(n)); % Update solution using RK1
end

% Plot the numerical and analytical solutions
fig = figure(); % Initialize a figure
set(fig, 'color', 'w') % Set the background color to be white
plot(timePoints, numericalSolution, ...
    'b-', 'LineWidth', 1.5); % Plot numerical solution
hold on;
plot(timePoints, analyticalSolution, ...
    'r--', 'LineWidth', 1.5); % Plot analytical solution
hold off;

% Set plot properties
xlabel('$t$', 'Interpreter', 'latex', 'FontSize', 18);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize', 18);
legend('Numerical Solution (RK1)', 'Analytical Solution', ...
    'Interpreter', 'latex', 'FontSize', 16, 'location', 'best');
set(gca, 'FontSize', 16, 'TickLabelInterpreter', 'latex');
```

And the code result can be found in figure 1.

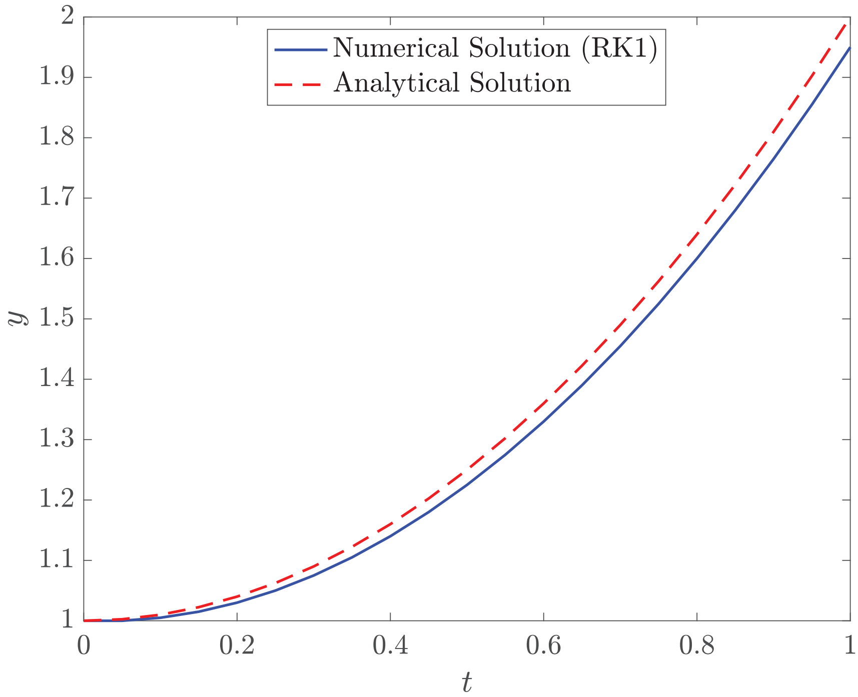

Figure 1: Euler method integration using a step size of 0.05

2.1.2 MATLAB ode45

It is worth noting that the Euler method accuracy and stability are step size dependent. MATLAB has a robust and accurate ODE solver, called `ode45` that is built based on the Dormand and Prince paper [5]. Listing 2 provides a MATLAB code that compares the Euler method against ode45 solver. The code result in figure 2 shows how ode45 solver is kind of robust against the step size as compared to the Euler method.

Listing 2: Comparison between Euler and ode45 solvers with different step sizes

```
% Define the analytical solution
analyticalSolution = @(t) t.^2 + 1;

% Define the step sizes to be compared
stepSizes = [0.1, 0.05, 0.01];

% Initialize the figure
fig = figure('Position', [100, 100, 800, 600]);
set(fig, 'color', 'w')

% Loop over the step sizes
for i = 1:length(stepSizes)
    stepSize = stepSizes(i);

    % Solve the differential equation using Euler's method
    tEuler = 0:stepSize:1;
    yEuler = zeros(size(tEuler));
    yEuler(1) = 1;
    for n = 1:length(tEuler)-1
```

```
        yEuler(n+1) = yEuler(n) + stepSize * (2 * tEuler(n));
    end

    % Solve the differential equation using ode45
    [tODE45, yODE45] = ode45(@myODE, [0:stepSize:1], 1);

    % Plot the numerical solutions
    subplot(length(stepSizes), 1, i);
    plot(tEuler, yEuler, 'b-', 'LineWidth', 1.5);
    hold on;
    plot(tODE45, yODE45, 'r-', 'LineWidth', 1.5);
    hold on;
    % Plot the analytical solution
    tAnalytical = 0:0.001:1;
    yAnalytical = analyticalSolution(tAnalytical);
    plot(tAnalytical, yAnalytical, 'k--', 'LineWidth', 1.5);
    hold off;

    % Set plot properties
    xlabel('$t$', 'Interpreter', 'latex', 'FontSize', 18);
    ylabel('$y$', 'Interpreter', 'latex', 'FontSize', 18);
    title(sprintf('Step Size = %.2f', stepSize), ...
        'Interpreter', 'latex', 'FontSize', 18);
    legend('Euler Method', 'ode45', 'Analytical', ...
        'Interpreter', 'latex', 'FontSize', 16, ...
        'location', 'west');
    set(gca, 'FontSize', 16, 'TickLabelInterpreter', 'latex');
end


% Define the differential equation as a separate function
function dydt = myODE(t, y)
    dydt = 2 * t; % Define the derivative of y with respect to t
end
```

There are a number of extensions available for numerically solving ODEs, the Euler technique being just one of them. In fact, this is an independent area of study focused on identifying numerical approximations to ODE solutions. However, we will utilize practical MATLAB libraries like ode45 and other variations like ode11, ode4, and so on. These libraries offer reliable and effective ODE-solving methods. Interested readers may refer to the paper by Sola et al. [6]. The exploration of various numerical systems is discussed in a good way in this paper, and also provides insightful analysis.

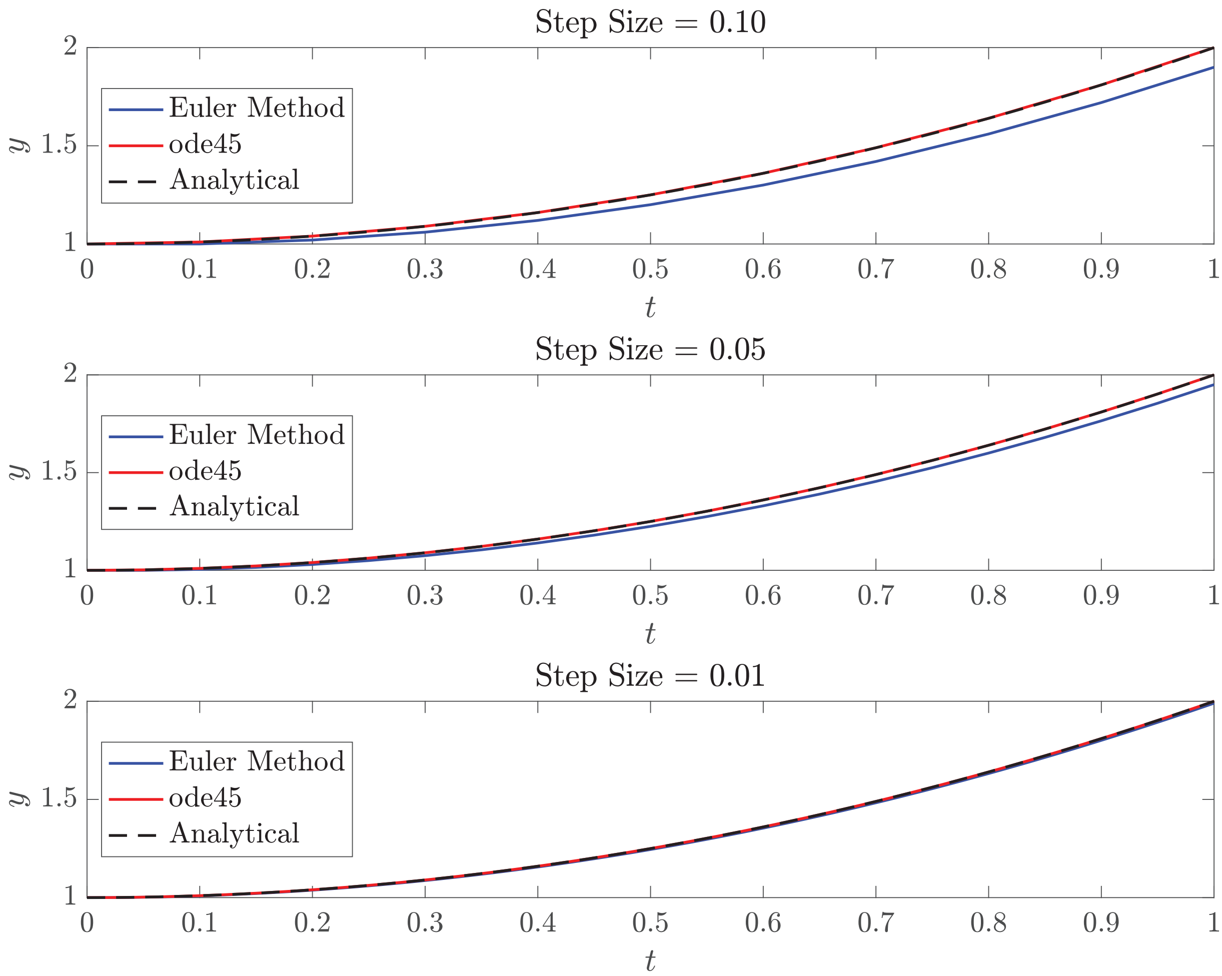

Figure 2: Euler, ode45, and analytical solution comparison

2.2 Reduction of Order

Most of the cases in real world employ higher order differential equations. In the previous sub-section we introduced how a first order differential equation can be solved using different numerical techniques. A higher-order ODE can be reduced in order to create a system of first-order ODEs. This allows us to use numerical techniques developed for first-order ODE systems to solve higher-order ODEs. When utilizing numerical solvers designed to handle systems of first-order ODEs, such as MATLAB's `ode45` function, this strategy is especially helpful.

Let's use a second-order ODE as an example to demonstrate reduction of order:

$$\ddot{y}(t) + p(t)\dot{y}(t) + q(t)y(t) = r(t) \tag{9}$$

For short, we will drop the (t) argument and only write $\dot{y}$, where the $\dot{()}$ is a symbol for the time derivative. To reduce the order we introduce an auxiliary variable $x_1 = y$, and another variable $x_2 = \dot{y}$, to convert this second-order ODE to a system of first-order ODEs. As a result, we can reformat the equation as follows:

$$\begin{aligned} x_1 &= y \quad \longrightarrow \dot{x}_1 = \dot{y} = x_2 \\ x_2 &= \dot{y} \quad \longrightarrow \dot{x}_2 = \ddot{y} = r - px_2 - qx_1 \end{aligned} \tag{10}$$

In terms of x_1 and x_2, a system of two first-order ODEs has been established. The system can then be solved using numerical techniques created for first-order ODE systems.

2.2.1 Reduction of order example

Let's use MATLAB to numerically solve a specific scenario to illustrate reduction of order. Think about the subsequent second-order ODE:

$$\ddot{y}(t) + 4\dot{y}(t) + 3y(t) = \cos t$$

We can rewrite this equation as a system of first-order ODEs:

$$\begin{aligned} y'(t) &= v(t) \\ v'(t) &= \cos(t) - 4v(t) - 3y(t) \end{aligned}$$

Now, we can solve this system numerically in MATLAB using the `ode45` function. Here's the MATLAB code:

Listing 3: The solution of the system of first-order ODE reduced from a second order ODE

```
% Init. timeSpan
timeSpan = [0, 5];

% Init. initial conditions
ICs = [0, 0];

% Solve the system of ODEs using the ode45 solver
[t, y] = ode45(@myODE, timeSpan, ICs);

% Plot the numerical solution
fig = figure(); % Initialize a figure
set(fig, 'color', 'w') % Set the background color to be white
plot(t, y(:, 1), 'b-', 'LineWidth', 1.5);
hold on
plot(t, y(:, 2), 'r-', 'LineWidth', 1.5);
xlabel('$t$', 'Interpreter', 'latex', 'FontSize', 18);  % Label for the x-axis
ylabel('$y(t)$', 'Interpreter', 'latex', 'FontSize', 18);  % Label for the y-axis
legend('$y$', '$v$', ...
       'Interpreter', 'latex', 'FontSize', 16, ...
       'location', 'best')
set(gca, 'FontSize', 16, 'TickLabelInterpreter', 'latex');


function dydt = myODE(t, y)
    % Function representing the system of first-order ODEs
    % Inputs:
    %   t: time
    %   y: vector containing the variables y(t) and v(t)
    % Output:
    %   dydt: vector of derivatives of y(t) and v(t)

    dydt = zeros(2, 1);
    dydt(1) = y(2);  % Derivative of y(t) is v(t)
    dydt(2) = cos(t) - 4 * y(2) - 3 * y(1);  % Derivative of v(t) is given by the second-order ODE
end
```

The code result can be found in figure 3.

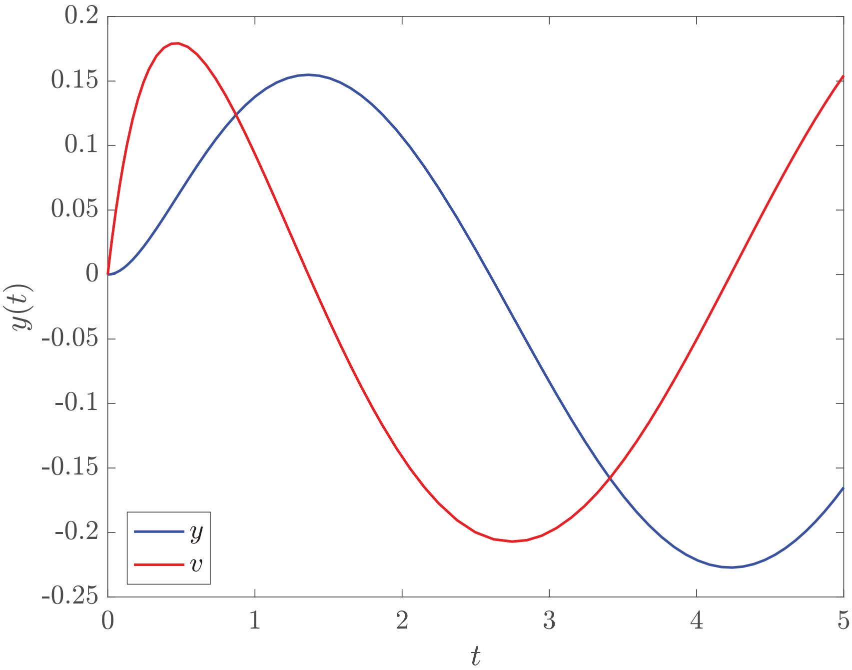

Figure 3: Solution of the system of first order ODEs

3 Modeling Electrical Systems

By using one or both of Kirchhoff's rules, it is possible to obtain a mathematical representation for an electrical circuit. Our main concern in this section is the analysis of straightforward electrical circuits after developing a governing mathematical modelling.

3.1 RL Circuit

It is frequently required to use numerical simulations to evaluate and comprehend the behavior of circuits. We will examine two distinct methods for simulating RL circuits in this section: one using current i as the *state variable* and the other using *charge* q. We will walk through the mathematical derivations and the process of executing these simulations in MATLAB using the well-known numerical solver `ode45`. We will be able to examine the time-dependent reactions of RL circuits through these simulations, gaining important knowledge about their behavior and assisting in the study of real-world applications.

Consider the circuit in figure 4, along with Kirchhoff's voltage law (KVL) to derive the differential equation for the current.

$$V = Ri(t) + L\frac{di(t)}{dt} \tag{11}$$

Another way to look at the problem in hand is to introduce the modeling using the charge q instead of the current i. The goal here is to get used to the reduction of order, as the modeling using q will result in second order ODE instead of the first order ODE produced from the current i model. Knowing that,

$$i(t) = \frac{dq(t)}{dt} \tag{12}$$

Substituting by equation 12 in equation 11:

$$V = R\frac{dq(t)}{dt} + L\frac{d^2q(t)}{dt2} \tag{13}$$

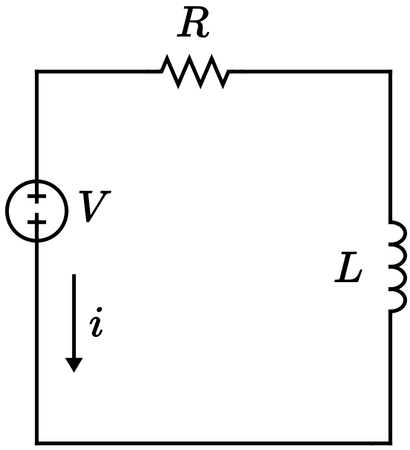

Figure 4: RL circuit

3.1.1 Simulating the model using the current as the state variable

Recalling equation 11 and rearranging,

$$\frac{d(i)}{dt} = \frac{1}{L}\left(V - Ri(t)\right) \tag{14}$$

Fortunately, we have a first order ODE, so we can directly use `ode45` and get the evolution of the current as a function of time. The MATLAB code in listing 4 simulate the behaviour of the circuit with the current being a state variable.

Listing 4: RL Circuit Simulation - Current as State Variable

```
% Initial condition
i0 = 0; % Initial current (in Amperes)

% Time span
tstart = 0; % Start time (in seconds)
tend = 1;   % End time (in seconds)

% Solve the differential equation
[t, i] = ode45(@rl_circuit_current, [tstart, tend], i0);

% Plotting
fig = figure(); % Initialize a figure
set(fig, 'color', 'w') % Set the background color to be white
plot(t, i, '-b','LineWidth', 1.5);
xlabel('Time (s)', 'Interpreter', 'latex', 'FontSize', 18);
ylabel('Current (A)', 'Interpreter', 'latex', 'FontSize', 18);
title('RL Circuit Simulation - Current as State Variable', '
    Interpreter', 'latex', 'FontSize', 16);
set(gca, 'FontSize', 16, 'TickLabelInterpreter', 'latex');

function di_dt = rl_circuit_current(t, i)
    R = 10; % Resistance (in Ohms)
    L = 1;  % Inductance (in Henrys)
    V = 2;
```

```
    di_dt = (1/L)*(V - R/L * i);
end
```

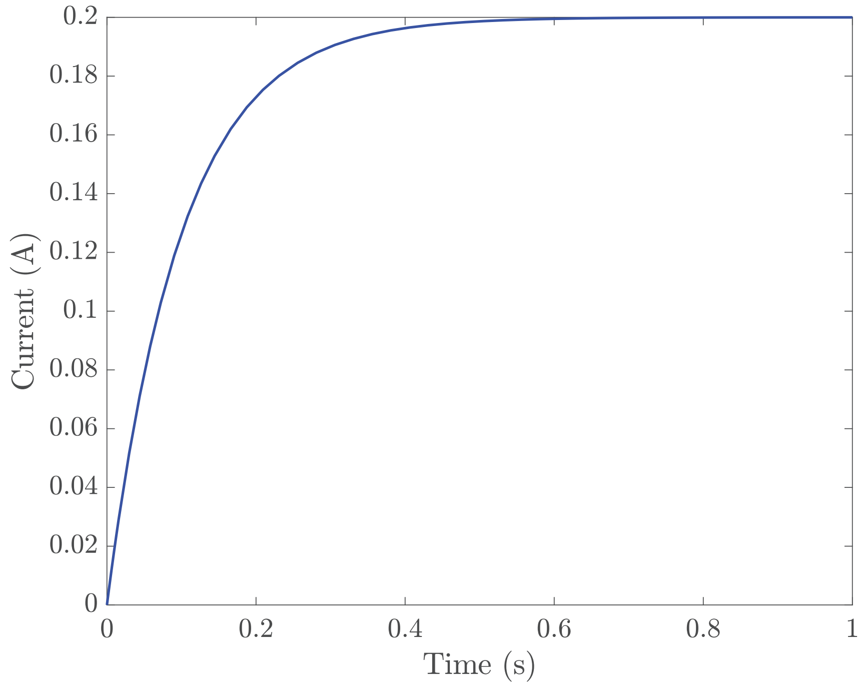

Figure 5: RL Circuit Simulation - Current as State Variable

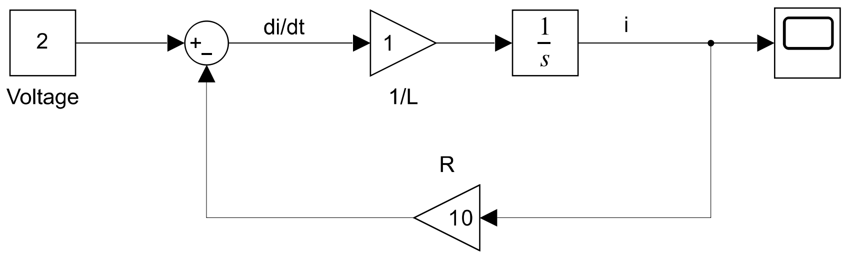

Figure 6: RL circuit modelling on SIMULINK

Figure 6 shows how to model the RL circuit on SIMULINK using the current as the state variable.

3.1.2 Simulating the model using the charge as the state variable

Recalling equation 13,

$$V = R\frac{dq}{dt} + L\frac{d^2q}{dt^2} \tag{15}$$

Let $x_1 = q$ and $x_2 = \dot{q}$, then

$$\begin{aligned} \dot{x}_1 &= \dot{q} = x_2 \\ \dot{x}_2 &= \ddot{q} = \frac{V}{L} - \frac{R}{L}\dot{x}_1 \end{aligned} \tag{16}$$

We now have a system of first order ODEs that can be solved using `ode45` directly as in listing 5.

Listing 5: RL Circuit Simulation - Charge as State Variable

```
% Define the parameters
resistance = 10;  % R
inductance = 1;  % L
appliedVoltage = 2;  % V

% Define the initial conditions
initialConditions = [0; 0];  % q(0) = 0, dq(0)/dt = 0

% Define the time span
timeSpan = [0 1];  % Simulation time from 0 to 10 seconds

% Solve the integro-differential equation using ode45
[time, state] = ode45(@(t, y) rlcEquation(t, y, ...
    resistance, inductance, appliedVoltage), timeSpan,
       initialConditions);

% Plot the charge on the capacitor as a function of time
fig = figure;
set(fig, 'Color', 'w');  % Set figure background to white
plot(time, state(:, 1), 'b-', 'LineWidth', 1.5);
hold on
plot(time, state(:, 2), 'r-', 'LineWidth', 1.5);
xlabel('Time $s$', 'Interpreter', ...
    'latex', 'FontSize', 18);  % Use LaTeX for x-axis label
legend('$q$', '$i$', ...
    'Interpreter', 'latex', 'FontSize', 16, ...
    'location', 'best');
set(gca, 'TickLabelInterpreter', ...
    'latex', 'FontSize', 16);  % Use LaTeX for axis ticks and
       increase font size

function dstates = rlcEquation(t, x, R, L, V)
    % RLC circuit integro-differential equation

    % Variables
    q = x(1);  % Charge on capacitor
    dqdt = x(2);  % Derivative of charge

    % Compute derivatives
    ddqdt = (1 / (L )) * (V - (R * dqdt));

    % Return derivatives
    dstates = [dqdt; ddqdt];
end
```

The code output can be found in figure 7.

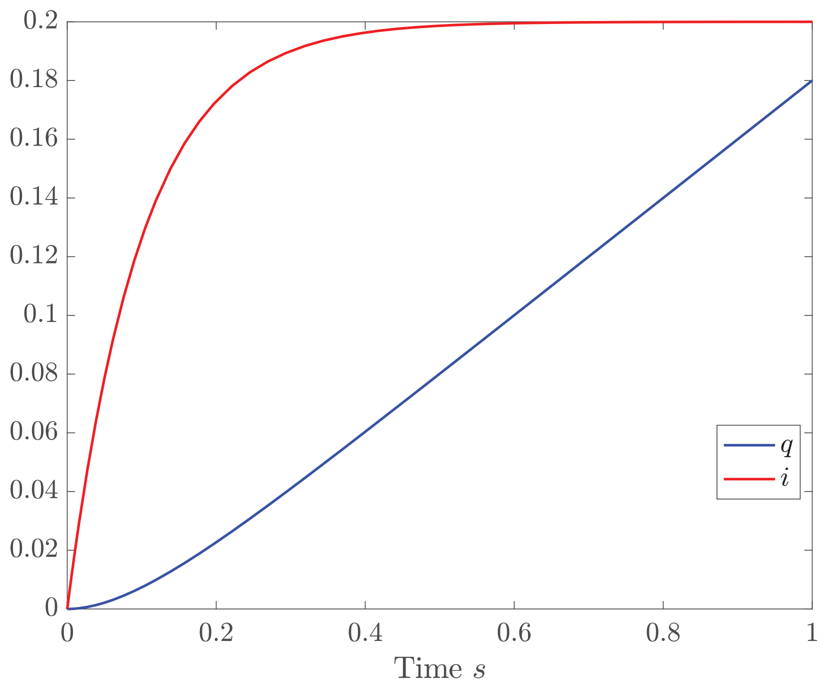

Figure 7: RL Circuit Simulation - Charge as State Variable

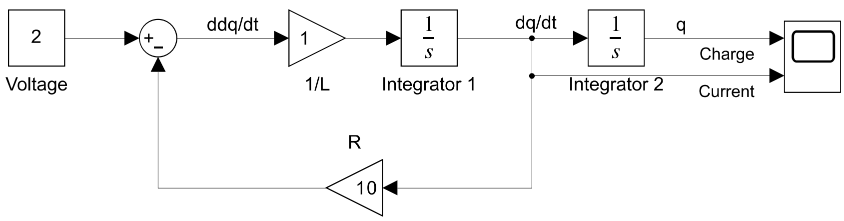

Figure 8: RL circuit modelling on SIMULINK

Figure 8 shows how to model the RL circuit on SIMULINK using the charge as the state variable.

3.2 RLC Circuit

Here, in this section, we will examine the time-dependent reactions of RLC circuits through MATLAB simulations. Consider the circuit in figure 9.

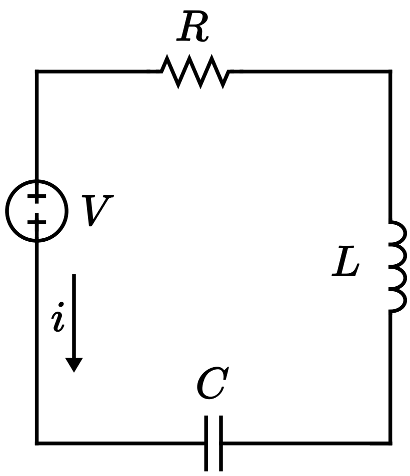

Figure 9: RLC circuit

The RLC circuit mathematical model with current as the main variable can be represented as follows

$$V = Ri + L\frac{di}{dt} + \frac{1}{C}\int i\,dt \tag{17}$$

Here we end up with what so called the integro-differential equation. In order to avoid complicated solution type, we can just utilize the fact stated in equation 12 and the equation can be written as,

$$V = L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} \tag{18}$$

After reducing the order of equation 18, we get

$$\begin{aligned} \dot{x}_1 &= \dot{q} = x_2 \\ \dot{x}_2 &= \ddot{q} = \frac{1}{L}\left(V - R\dot{x}_1 - \frac{x_1}{C}\right) \end{aligned} \tag{19}$$

The system can be simulated via MATLAB code as following,

Listing 6: RLC Circuit Simulation

```
% Define the parameters
resistance = 10;  % R
inductance = 1e-3;  % L
capacitance = 1e-6; % C
appliedVoltage = 10;  % V

% Define the initial conditions
initialConditions = [0; 1];  % q(0) = 1, dq(0)/dt = 3

% Define the time span
timeSpan = [0 1e-3];  % Simulation time from 0 to 1 ms

% Solve the integro-differential equation using ode45
[time, state] = ode45(@(t, y) rlcEquation(t, y, ...
    resistance, inductance, capacitance, appliedVoltage), ...
```

```
    timeSpan, initialConditions);

% Plot the charge on the capacitor as a function of time
fig = figure;
set(fig, 'Color', 'w');  % Set figure background to white
subplot(211);
plot(time, state(:, 1), 'b-', 'LineWidth', 1.5);
xlabel('Time $s$', 'Interpreter', ...
    'latex', 'FontSize', 18);
ylabel('Charge $c$', 'Interpreter', ...
    'latex', 'FontSize', 18);
subplot(212);
plot(time, state(:, 2), 'r-', 'LineWidth', 1.5);
xlabel('Time $s$', 'Interpreter', ...
    'latex', 'FontSize', 18);
ylabel('Current $A$', 'Interpreter', ...
    'latex', 'FontSize', 18);

set(gca, 'TickLabelInterpreter', ...
    'latex', 'FontSize', 16);  % Use LaTeX for axis ticks and increase font size

function dstates = rlcEquation(t, x, R, L, C, V)
    % RLC circuit integro-differential equation
    % x = [x1 = q
    %      x2 = dqdt]

    dqdt = x(2);  % Charge on capacitor

    % Compute derivatives
    ddqdt = (1 / (L)) * (V - (R * x(2)) - (x(1)/C));

    % Return derivatives
    dstates = [dqdt; ddqdt]; % (q, i) after integration
end
```

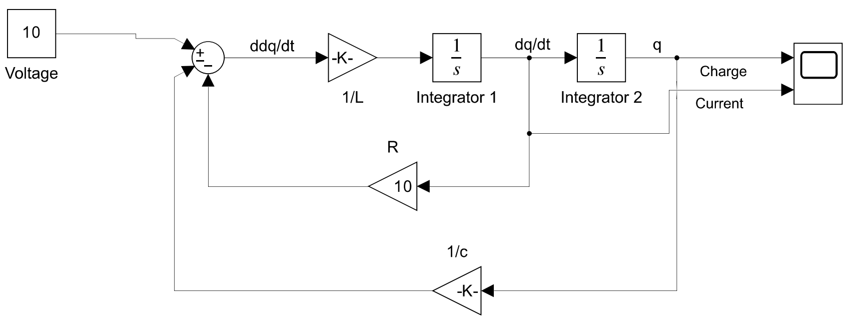

Figure 10: RLC circuit modelling on SIMULINK

Figure 10 shows how to model the RLC circuit on SIMULINK.

Figure 12: Mass-spring-damper system free body diagram

4 Modeling Mechanical Systems

By applying Newoton's second law, we can also obtain a mathematical representa- tion for a mass-spring-damper mechanical system. Our main concern in this section is the analysis of straightforward this mechanical system after developing a governing mathematical modelling.

4.1 Mass-Spring-Damper Mechanical System

From Newton's second law of motion in the dynamical systems, the acceleration of a mass caused by an external force acting on the system is directly proportional to the magnitude of the net force, in the same direction as the net force, and inversely proportional to the mass of the object. This can be expressed mathematically in equation 20

$$\sum \boldsymbol{F} = m\boldsymbol{a} = m\ddot{\boldsymbol{x}} \tag{20}$$

Where m is the mass of the body, $\sum \boldsymbol{F}$ is the net force and $\boldsymbol{a}$ is the acceleration.

Consider the mass spring damper system in figure 11, by applying Newoton's Laws to get the differential equations for the displacement.

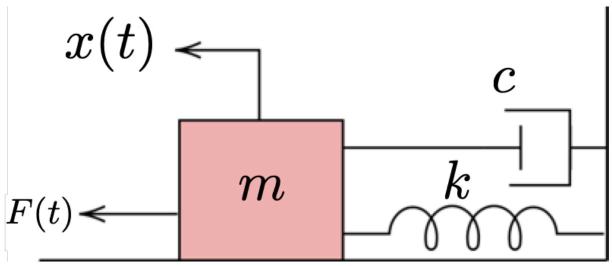

Figure 11: Mass-spring-damper system

The equation of motion of the system can be represented as equation 21

$$\sum F = F_e - F_s - F_d = m\ddot{x} \tag{21}$$

Where F_e, F_s, and F_d are applied or external force, spring force and damping force respectively. The spring and damping force is represented mathematically as equation 22

$$\begin{aligned} F_s &= kx \\ F_d &= c\dot{x} \end{aligned} \tag{22}$$

Where k and d is the spring and damping coefficient respectively. Figure 12 represents the free body diagram for the system. The differential equation for the system can be represented as shown in equation 23 assuming that there is no external force.

$$m\ddot{x} + kx + c\dot{x} = 0 \tag{23}$$

This can be rearranged into

$$\ddot{x} = \frac{1}{m}(-kx - c\ddot{x}) \tag{24}$$

By applying reduction of order, let $z_1 = x$ and $z_2 = \dot{x}$, then

$$\begin{aligned} \dot{z}_1 &= \dot{x} = z_2 \\ \dot{z}_2 &= \ddot{x} = \frac{1}{m}(-kx - c\ddot{x}) \end{aligned} \tag{25}$$

We now have a system of first order ODEs that can be solved using `ode45` directly as in listing 7.

Listing 7: Mass-spring-damper system

```
clear all; clc; close all;


% Define system parameters
m = 1;       % Mass (kg)
k = 10;      % Spring constant (N/m)
c = 0.5;     % Damping coefficient (Ns/m)

% Define initial conditions
x0 = 0;      % Initial displacement (m)
v0 = 1;      % Initial velocity (m/s)

% Define simulation time span
tspan = [0 10];      % Simulation time from 0 to 10 seconds

% Solve the differential equation using ode45
[t, y] = ode45(@(t, y)mass_spring_damper_equation(t, y, m, k, c),
    ...
    tspan, [x0; v0]);

% Extract displacement and velocity from the solution
x = y(:, 1);
v = y(:, 2);

% Plot the charge on the capacitor as a function of time
fig = figure;
set(fig, 'Color', 'w');  % Set figure background to white
subplot(2, 1, 1);
plot(t, x, 'b-', 'LineWidth', 1.5);
xlabel('Time $s$', 'Interpreter', ...
    'latex', 'FontSize', 18);
ylabel('Displacement $m$', 'Interpreter', ...
    'latex', 'FontSize', 18);
subplot(2, 1, 2);
plot(t, v, 'r-', 'LineWidth', 1.5);
xlabel('Time $s$', 'Interpreter', ...
    'latex', 'FontSize', 18);
ylabel('Velocity $m/s$', 'Interpreter', ...
    'latex', 'FontSize', 18);

set(gca, 'TickLabelInterpreter', ...
    'latex', 'FontSize', 16);

```

```
function dydt = mass_spring_damper_equation(t, y, m, k, c)
    % Extract state variables
    z1 = y(1);
    z2 = y(2);

    % Define the differential equations
    z1_dot = z2;
    z2_dot = (-k*z1 - c*z2) / m;

    % Pack the derivatives into a column vector
    dydt = [z1_dot; z2_dot];
end
```

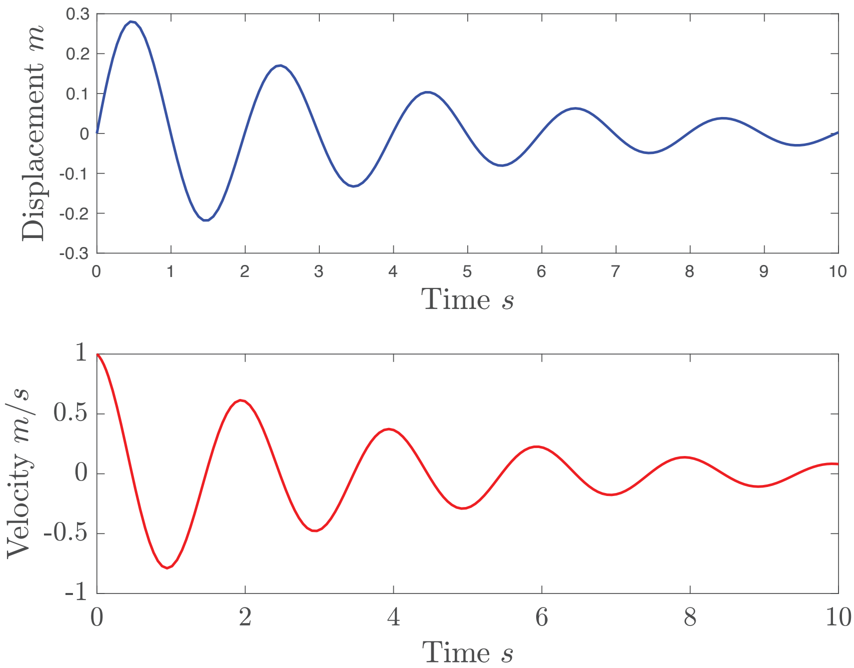

Figure 13: Mass-spring-damper system simulation

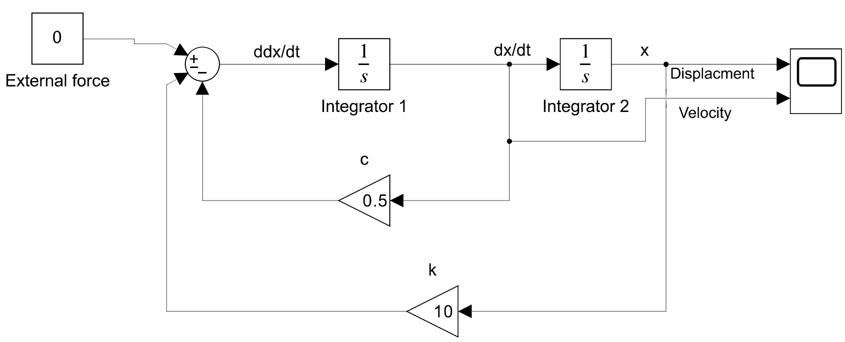

Figure 14: Mass spring damper system modelling on SIMULINK

Figure 14 shows how to model the mass spring damper system on SIMULINK.

References

[1] R. Dorf and R. Bishop, *Modern control Systems.* 2017.

[2] K. Ogata, *Modern control engineering fifth edition.* 2010.

[3] H. Kwakernaak and R. Sivan, *Linear optimal control systems*, vol. 1. Wiley-interscience New York, 1972.

[4] K. J. Åström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers.* Princeton university press, 2021.

[5] J. R. Dormand and P. J. Prince, "A family of embedded runge-kutta formulae," *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[6] J. Sola, "Quaternion kinematics for the error-state kalman filter," *arXiv preprint arXiv:1711.02508*, 2017.