ME C134 Lab1 Report

Rui Wang*
3033461836

Contents

*wangrui@berkeley.edu

1 General Problem

1.1 Problem Description

Given the simple RLC circuit as shown below, we can derive its transfer function as well as state space representation.

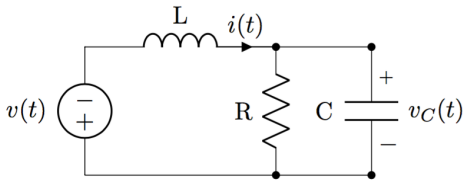

Figure 1: The original problem: a simple RLC circuit

The problem defines $v(t)$ as the input, $i(t)$ and $v_c(t)$ as system states, and $v_c(t)$ as the output.

1.2 Solution

Applying Kirchoff's law (specifically, KVL), we obtain the following equations:

$$\begin{cases} (i(t) - C\dfrac{dv_c}{dt})R = v_c & (1) \\ L\dfrac{di(t)}{dt} + v_c(t) + v(t) = 0 & (2) \end{cases}$$

Differentiate both sides of (1), and we get

$$\dot{i} = \frac{\dot{v_c}}{R} + C\ddot{v_c} \tag{3}$$

Substitute (3) into (1), and we obtain

$$L(\frac{\dot{v_c}}{R} + C\dot{v_c}) + v_c - v = 0 \tag{4}$$

Therefore, the transfer function turns out to be

$$\boxed{F(s) = \frac{1}{LCs^2 + \frac{L}{R}s + 1}} \tag{5}$$

Also, rearrange the equations (1) and (4) in the form of $\dot{X} = AX + Bu$, with $X = [i, v_c]^T$ and $u = v$, and $Y = CX + D$, with $Y = v_c$, we obtain the following values for parameters A, B, C and D. Eventually, we have $\boxed{A = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} B = [\frac{1}{L}, 0]^T,\ C = [0, 1],\ D = 0}$

2 Simulation

The simulation is a specific implementation of the general problem proposed above. In our implementation, the value of R, L and C are given and fixed as

$$R = 2, L = 10^{-3}, C = 10^{-3}$$

2.1 Using MATLAB Built-in ODE Solver

2.1.1 Usage

In this simulation method, we use MATLAB's ode45 numerical solution to calculate the evolution of X with time. The usage of the function is `[t, x] = ode45(@f, tspan, x0)`, where `f` is a differential function following a certain format and `tspan` is a vector with critical time stamp, `x0` is the initial condition of X in the differential equation.

`ode45` requires `f` to be defined in this format:

```
xdot = f(t, x)
```

where `t, x` are time t and state variable X, respectively.

2.1.2 Code

In our implementation, we name `f` as `RLCdynamics`. The code is a simple implementation of the differential equation (4). Code of the function is attached below:

Listing 1: RLCdynamics.m

```
function xdot = RLCdynamics(t, x)
  L = 0.001;
  C = 0.001;
  R = 2;
  xdot = [0, -1/L; 1/C, -1/R/C] * x + [1/L, 0]'*1;
end
```

In MATLAB command window, type

Listing 2: Plotting X with regard to t

```
  x0 = [0; 0]; tspan = [0, 0.02]; % initial setting
  [t, x] = ode45(@RLCdynamics, tspan, x0); % solve function
  [hAx, hLine1, hLine2] = plotyy(t, x(:, 1), t, x(:, 2));
  hLine2.LineStyle = '--'; % change line style
  title('Graph of x - t') % title, label, legend
  xlabel('t / s')
  ylabel(hAx(1),'i / A') % left y-axis
  ylabel(hAx(2),'v_c / V') % right y-axis
  legend('i','v_c')
```

we will obtain the following plot2:

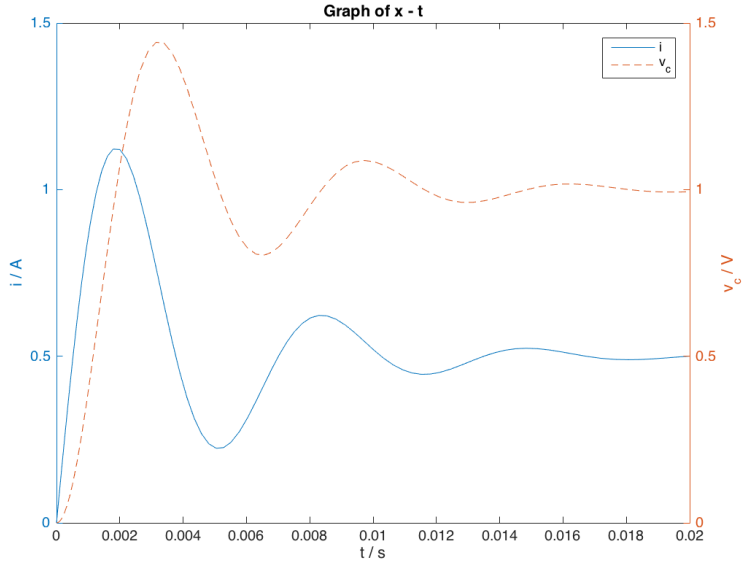

Figure 2: The evolution of X relative to t.

From the plot we can see that both v_c and i follows an underdamped pattern, with their final value stabilized at $1V$ and $0.5A$, respectively. This is due to the initial leap by the step input and the later correction(damping) by the circuit(system) itself.

2.2 State-space Model

2.2.1 Method

This method uses MATLAB's built-in `ss` function to create a state-space model out of the standard format

$$\dot{X} = AX + Bu, Y = CX + Du$$

2.2.2 Code

Listing 3: Plotting with state-space model

```
A = [0, -1/L; 1/C, -1/R/C]; B = [1/L; 0]; C = [0, 1]; D = 0;
% parameter configuration
RLCss = ss(A, B, C, D);
step(RLCss);
```

The result is shown below, which is the same as the previous plot, which should be the case as we are modeling the same system, only with different methods.

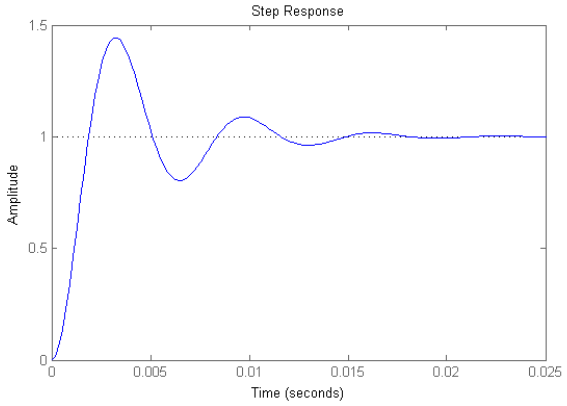

Figure 3: The evolution of v_c with regard to t.

2.3 Tranfer Function Model

2.3.1 Method

In this method, we construct a transfer function and make use of the `step` function to plot the response of v_c.

2.3.2 Code

Listing 4: Plotting with transfer function model

```
L = 0.001; C = 0.001; R = 2;
num = [1]; den = [L*C, L/R, 1];
sys = tf(num, den);
step(sys)
```

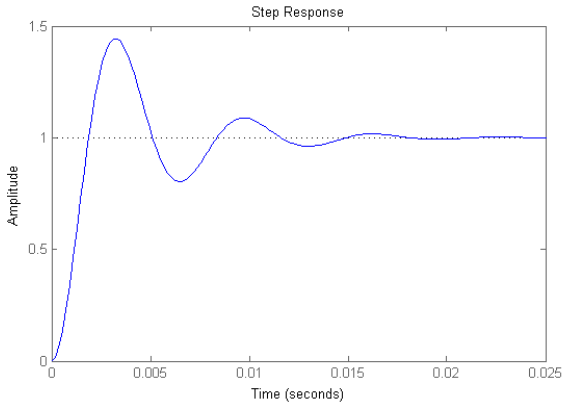

Figure 4: The evolution of v_c with regard to t.

Basically the three methods above are equivalent, and it is not difficult to understand that their results are the same. In other words, they validate each other.

2.4 Simulink Model

2.4.1 Method

In this method, we construct a Simulink model representing the RLC circuit. As the model for the circuit is not unique, we are given a model architecture and are in charge of the calculation for specific parameters.

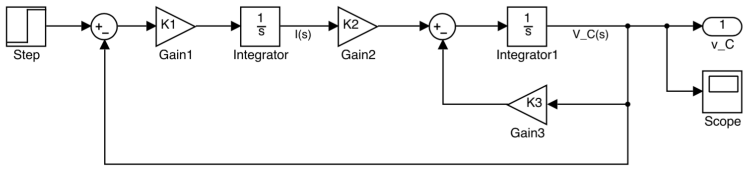

Figure 5: The given model.

We then calculate the parameters $K1, K2$ and $K3$ in the reverse manner:

1. As the output is v_c, the value before integration is $\dot{v_c}$, and the value before the first integration is $\ddot{v_c}$ with some coefficient.
2. According to equation (1), we have $\dot{v_c} = \frac{i}{C} - \frac{v_c}{RC}$, which enables us to decide that $K3 = \frac{1}{RC}$.
3. The element directly before and after the first integral can also be decided according to $\dot{v_c} = \frac{i}{C} - \frac{v_c}{RC}$, which is $\dot{i}$ and i, respectively.
4. Furthermore, still with the same equation, we obtain $K2 = \frac{1}{C}$.
5. According to equation (2), we have $\dot{i} = \frac{v - v_c}{L}$, which indicates $K1 = \frac{1}{L}$.
6. Finally, we check whether our derivation is correct by deriving the relationship between variables in the diagram.

Note that the model does not contain differentiation blocks. *The reason is, instead of using differentiation to obtain values such as $\dot{i}$, we treat them as input and use* ***integration*** *to obtain values such as v_c. In the calculation of Simulink model, however, we perform differentiation in order to solve the parameters in the reverse order.*

After running the model, we can obtain the evolution of output from data collected by **scope**. As data in the scope is also stored in MATLAB workspace as `yout` by default, we can therefore plot the data in MATLAB command window.

2.4.2 Model

The actual model I constructed looks like this in figure6

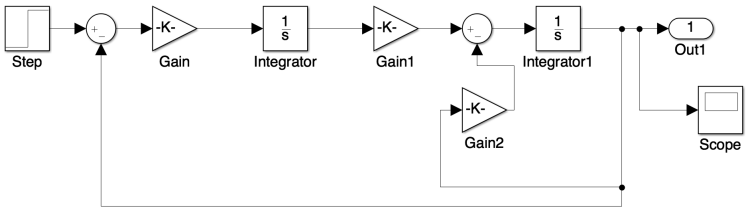

Figure 6: The final model.

Note that on applying the actual data to $K1, K2$ and $K3$, we have $K1 = K2 = 10^3$ and $K3 = 500$.

2.4.3 Normal Case

After configuring Simulink(which is a pain) correctly, we get the following result:

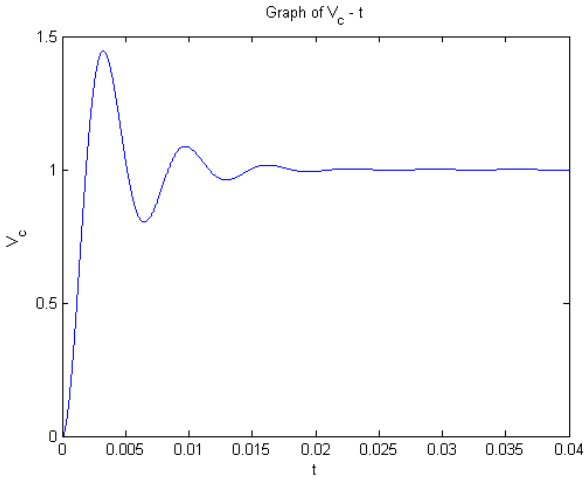

Figure 7: The Simulink model running result.

Again the plot accords with our previous plots, since they are describing the same system.

Following the lab guide, we modify the model input in several ways to see and compare the results.

2.4.4 Changing Initial Condition

Changing the initial condition to $1V$, which is to change the initial value of the last integral to $1V$, yields the following result:

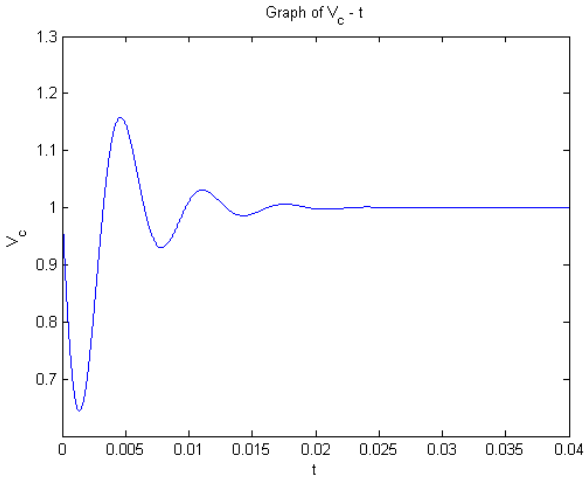

Figure 8: Initial condition changed to $1V$.

In this case, although the system is at its final state in the beginning, the initial difference at time $0-$ between v and v_c will cause the system to start a dramatic oscillation. However, once oscillation begins, and v is changed to 1 V, the oscillation will be damped. By and by, the system will reach its stability.

2.4.5 Changing Source to Square Wave

To change the input to a square wave, we will need to use the block **Waveform Generator** and select square wave within the block properties. **Note** that we also need to select frequency instead of the default `rad/s` as unit.

With a frequency of 25Hz, v_c acts in a periodical pattern, with time interval between the square wave rising/descending trim edge large enough for the output to be damped to stability. This is natural as the square wave for the system is just like a connection of lots of steps, for each step the circuit will act in the same pattern. **In order to display this periodicity clearly, a** $0.12s$ **time span is chosen,** instead of the initial $0.04s$.

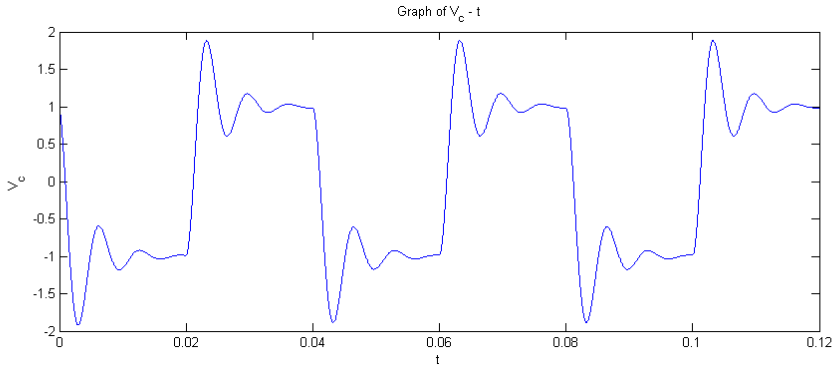

Figure 9: Input changed to 25Hz square wave.

With a frequency of 250Hz, however, input changes too fast for the system to reach stability within a single period. Therefore the system reaches a quasi-stability where it oscillates with the same frequency as the input, which is also called `undamped`.

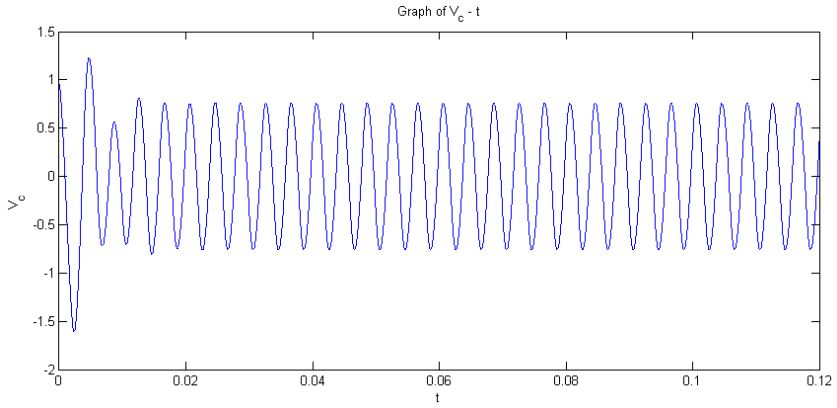

Figure 10: Input changed to 250Hz square wave.

3 Comparison of Different Methods

3.1 Which...Non-linear?

Which of the model representations above can model non-linear dynamic systems?

Basically `ode45` and **Simulink** are both able to deal with non-linear differential equations. However, in Simulink we have to manually assign the largest step, which is not always easy, especially for very complicated equations. But this does not occur often and Simulink is mostly good enough. The **state-space model** is designed for linear systems, making it impossible to represent non-linear systems. Modeling using **transfer function** is also not always viable as for non-linear systems, $X(s)$ and $Y(s)$ can not be factored out from the equation, disabling us from getting a transfer function.

Generally, I think Simulink is the best tool for complicated cases as we do not need to bother about the choice of state variables and the rearrangement of equations.

3.2 Which...Initial States?

Which of them is well-suited to account for different initial states?

Again, both the **simulate by foot** and **Simulink** are good, while the other two are not. Because for the former two, initial conditions are treated as variables while for the latter two, initial conditions are not passed in. Instead, we need special functions for each different initial state, which makes life difficult.

3.3 Additional Advantages and Disadvantages

Table 1: Advantages and Disadvantages of Four Methods

Method	Advantages	Disadvantages
Simulate by foot	Easy to use in simple cases; optimized and professional, solves quickly	Some models can not be solved; needs different solver function for different kinds of DE's; requires written down equations and well-defined variables
Simulink	Can model almost anything; graphical and intuitive; does not need writing down equations	Needs assigning maximum step; is often not optimized; setting-up can be troublesome
State-space Model	Clear and general; easily processed by computer	Only linear; needs rearrangement and manually performing lots of calculations; matrix can be bulky
Transfer Function Model	Can be used to reconstruct system equation easily	Only linear; very limited