MCEN4043 - Systems Dynamics
University of Colorado at Boulder

Lab #1: Simulate (Numerically)

Lab Objectives

In this class, simulate means to solve for the output of a system given an input and a model of that system. This could be done by hand, think equations on paper, or could be done using software. For the lab portion of this course, we will differentiate between the two simulation options by calling one "theoretical simulation" (or just "theory") and the other "numerical simulation". Today, in this first lab session, you will familiarize yourself with numerically simulating differential equations (which model the dynamics of systems) using Matlab and Simulink. Although most of the other labs in this course will ask you to simulate both theoretically and numerically, today we will only simulate numerically.

Pre-Lab

Watch the following two videos (in that order) before you begin this lab:

1. `https://www.youtube.com/watch?v=02nCpic_Zb8`
2. `https://www.youtube.com/watch?v=eznRtmTbqc4`

Post-Lab

Watch the following video entitled "Making Nice Plots using Matlab" so you can present the result you get in this lab:

1. `https://youtu.be/kwoPE1eWApM?si=demShYGOE1I8exDP`

Background on Simple RC circuit

One of the systems you will test in the course is the RC circuit shown in Fig. **??**. The input to the circuit is the voltage V_{in} and the output (the part of the circuit we are interested in measuring) is the voltage across the capacitor V_{out}. In real-life, this voltage can be supplied by a function-generator (aka waveform-generator), or battery (a constant voltage), or even an Arduino. In real-life, the output voltage can be measured with an oscilloscope,

or multimeter (although it's not a great tool for viewing time-varying signals), or even an Arduino again. For the majority of these Labs, we will be using the function-generator and the oscilloscope rather than something like an Arduino UNO which has more limitations. Again, today we will only be simulating and will not be doing any real-life experiments... that will start in the next lab.

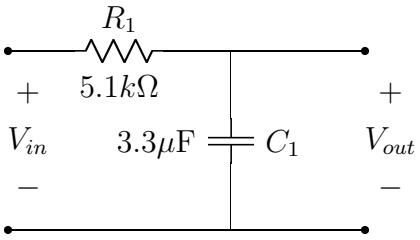

Figure 1: An RC circuit is one of the systems you will test in this course. It's model is shown in this figure.

The differential equation relating the "input" $V_{in}(t)$ to the "output" $V_{out}(t)$ for the ideal circuit model in Fig **??**. is

$$RC\dot{V}_{\text{out}}(t) + V_{\text{out}}(t) = V_{\text{in}}(t), \tag{1}$$

where $R = 5.1k\Omega$ and $C = 3.3\mu F$. We will study several methods for solving this differential equation numerically. For all this methods you will need to solve for the highest-order derivative, of the output variable, in Eqn. **??** and re-write the equation as:

$$\dot{V}_{\text{out}}(t) = -\frac{1}{RC}V_{\text{out}}(t) + \frac{1}{RC}V_{\text{in}}(t), \tag{2}$$

Solve by MATLAB ode45 function

In this section, you will use MATLAB `ode45` function to solve the differential equation (**??**). What you need to do is as follows:

1. Remember that to use `ode45` you need to turn all the higher-order differential equations into a system of first-order differential equations. For this lab, it is already a first-order ODE so you don't have anything else to do and Eqn. **??** is all you'll need.

2. Type "`help ode45`" in the Matlab command-line. Read through it and refresh your memory on the different ways that you can call `ode45`

3. Copy the following code into a new function called ode_RC:

```
function dydt = ode_RC(t,y,t_g,g)
%note that t is scalar the ode45 provides
%note that y is value of the solution at the previous time step
```

```
%note that t_g is time vector for the forcing function
%note that g is the vector of data representing the forcing function

R = 4.99e3;
C = 3.35e-6;
k = 1/(R*C);

g_at_t = interp1(t_g,g,t); % Interpolate the data set (gt,g) at time t
dydt = -k*y + k*g_at_t; % Evalute ODE at time t
```

4. In Matlab, select `File`, `New`, `M-File`. It will create a new M-script file. Create this file by entering the commands that follow from Step 3 to 6.

5. Define time horizon for solving ODE by using the following Matlab code:

```
time_start = 0;
time_final = 2;
t_step = 5000;
```

6. Define the input function $V_{in}(t)$:

```
vin_t = linspace(time_start,time_final,t_step);
freq = 1*2*pi;
vin=sin(vin_t*freq);
```

7. Call the ode45 function to solve ODE:

```
Tspan = [time_start time_final];
IC = 0;
options=odeset('RelTol',1e-4);
[vout_t,vout]=ode45(@(t,y)ode_RC(t,y,vin_t,vin),Tspan,IC,options);
```

8. Plot the solution $V_{in}(t)$ and $V_{out}(t)$: See the Post-Lab on plotting.

9. Run this m-file

Some experimental tasks you should do:

1. Set the "Sine Wave" frequency to 1Hz (note: the frequency is specified in radians per second)in the code. Run the simulation and plot your output signal along with the input signal on a figure.

2. Set the "Sine Wave" frequency to 10Hz and run the same code to generate another plot.

3. Set the "Sine Wave" frequency to 50Hz and run the same code to generate another plot except change `axis([0 0.1 -1 1])` in order to see more clearly the details of the signals.

Solve by Simulink using Derivative Blocks

In this section, you will use Simulink to numerically generate an approximate solution. One Simulink block diagram that implements this differential equation is shown in Fig. **??**.

You need to build this diagram yourself in order to simulate the differential equation. We need to change a few simulation parameters/options... so In your Simulink diagram window, click on the "Modeling" tab at the top and the click on the "Model Settings" button (Gear Button) and change the following options:

- Under "Solver" and then "Solver Details", set the "Max step size" parameter under to `0.001` (instead of "auto").
- Under "Data Import/Export", uncheck "Single Simulation Out". I personally don't like it when it's checked, but you could leave it checked and see if you works for you.
- Under "Data Import/Export" and then "Additional Parameters", make sure that "Limit data points to last" is unchecked. This used to be a problem when memory was an issue, but in modern Simulink it won't be checked and therefore data lengths won't be limited. I leave this bullet point just for history.

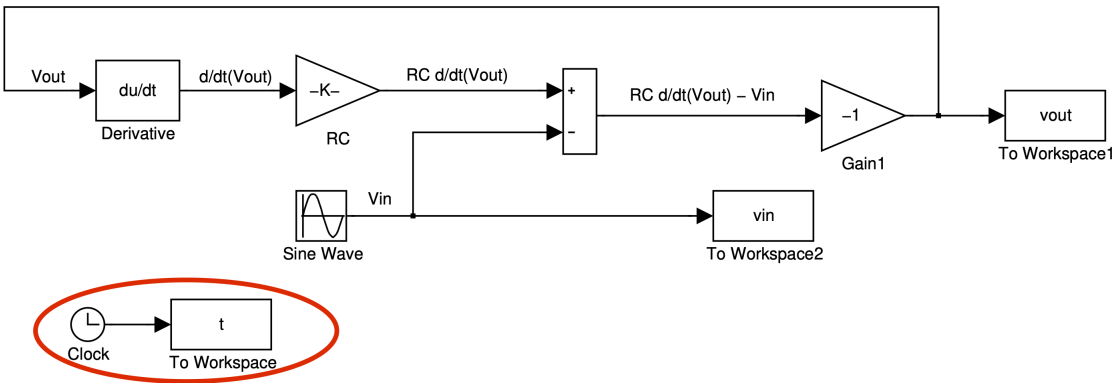

Figure 2: Simulink block diagram that simulates the differential equation (**??**). **Note:** the code in the red oval is no longer need in current versions of Simulink. Simulink now sends the time vector to the workspace automatically

Experimental Tasks:

1. Set the "Sine Wave" block frequency to 1Hz (note: the frequency is specified in radians per second in the parameter box). Also set the "Phase" to be `pi/2` (this means the

V_{in} looks like a cosine function at $t = 0$). Run the simulation and plot your data using Matlab (see Post-Lab): Make sure that V_{in} and V_{out} is clearly labeled in the plot.

2. Set the "Sine Wave" block frequency to 10Hz and run the same code to generate another plot.

3. Set the "Sine Wave" block frequency to 50Hz and run the same code to generate another plot except change `axis([0 0.1 -1 1])` in order to see more clearly the details of the signals.

Solve by Simulink using Integrator Blocks

There is another block diagram formulation of (**??**) that uses an *integrator* instead of a *differentiator*. One way to obtain this alternative block diagram is to integrate each side of (**??**) from 0 to t:

$$RC \int_0^t \dot{V}_{out}(\tau)d\tau + \int_0^t V_{out}(\tau)d\tau = \int_0^t V_{in}(\tau)d\tau,$$

which upon rearrangement yields,

$$V_{out}(t) = V_{out}(0) + \int_0^t \frac{1}{RC}\left(V_{in}(\tau) - V_{out}(\tau)\right)d\tau. \tag{3}$$

The Simulink block diagram in Fig. **??** implements this representation of the differential equation. It seems like the *initial condition* for V_{out} (denoted $V_{out}(0)$) is missing from this

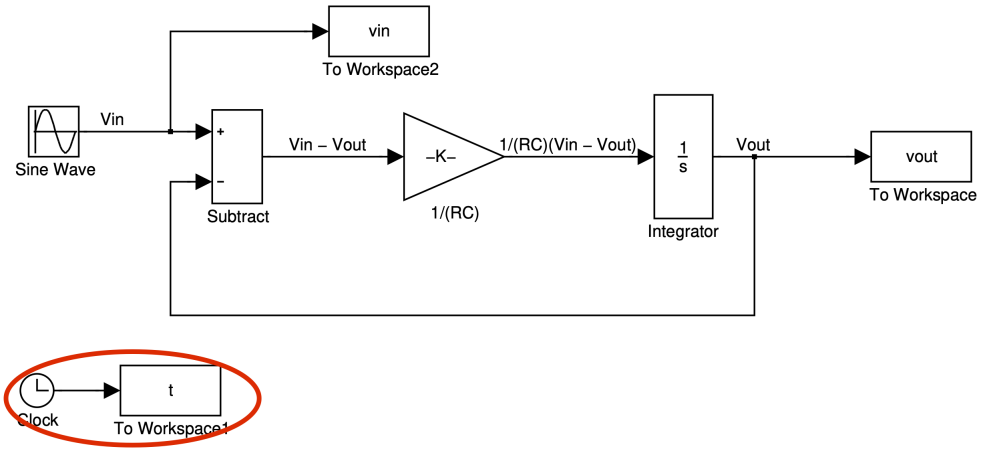

Figure 3: Simulink block diagram that simulates the differential equation (**??**) using an *integrator* instead of a *differentiator*. **Note:** the code in the red oval is no longer need in current versions of Simulink. Simulink now sends the time vector to the workspace automatically

block diagram but it is actually specified in the integrator block parameters. The default value is 0. Also, do note that "$\frac{1}{s}$" text in the integrate block represents an integrator in Laplace which you will see in my Laplace Transform notes. The different simulation formulations (*integrator* block versus *differentiator* block) yield the same response for V_{out} with the exception of the initial "transient". It is easy to specify the correct initial condition for V_{out} in the integrator block – this is just the "Initial condition" parameter. The initial condition for V_{out} in the differentiator block case can not be specified directly. In practice, the *integrator* approach is the superior approach and you **SHOULD NOT** try the *differentiator* approach beyond this lab. Only use the Integrator method from now on.