ECE 1050: Lab M8
Transient Analysis with Capacitors

Learning Objectives:

- Model/simulate the step response of a series RC circuit.
- Utilize concatenation and logical indexing in MATLAB.
- Use loops and conditionals in MATLAB.
- Simulate transients in LTspice.

Overview: For the majority of this class, you have primarily focused on the static, or DC, behavior of electrical circuits. One of the main assumptions with a DC circuit is that any changes in the sources will instantaneously propagate throughout the rest of the network until immediately settling into a new state. With the introduction of capacitors, however, we can begin to study circuits where sudden changes in the sources do not necessarily result in a sudden change in the rest of the network. Instead, there is a transition period where currents and voltages slowly adjust to the new drive conditions (hence the term *transients*).

Part 1: RC Circuit Rise Time (10 points)

One of the simplest circuits involving a capacitor is the *series RC* arrangement depicted in Fig. 1. The source voltage V_s is assumed to have a value of $V_s = 0$ with no other currents or voltages present in the circuit. Then, at time $t = 0$, the source suddenly jumps to a value of $V_s = 1.0$ V, thereby exciting the circuit and charging the capacitor. The voltage across the capacitor then follows a simple exponential function given by

$$V_c(t) = V_i + (V_f - V_i)\left(1 - e^{-t/\tau}\right) \ , \tag{1}$$

where $V_i = 0$ is the *initial* voltage across the capacitor and $V_f = 1.0$ V is the *final* voltage after infinite time has passed. The *time constant* $\tau = RC$ then determines how quickly the capacitor charges.

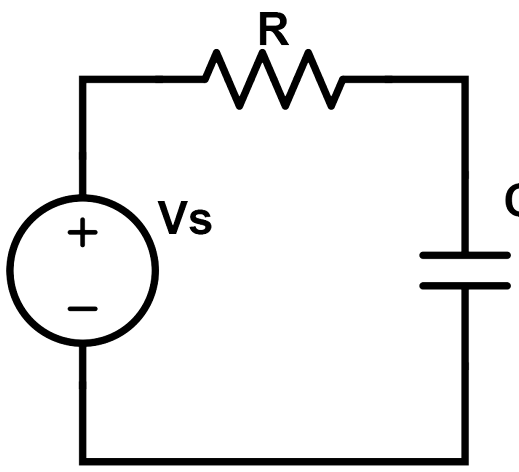

Figure 1: *A simple series RC circuit.*

Let us now use MATLAB to model the behavior of our circuit. Start by creating a new script file and naming it *RC_Circuit.m*. Define a series resistance of $R = 1.0$ kΩ with a capacitance of $C = 10$ μF. Next, create an array called "t" to represent time samples across the range of $[0, 0.1]$ seconds with $N = 1001$ samples in between (HINT: Remember the "linspace" function). Calculate the voltage $V_c(t)$ across the capacitor and then plot it. For clarity, scale the time axis to milliseconds (ms). If all goes well, you should observe a simple exponential curve that begins at 0.0 V (V_i) and then slowly rises up towards a value of 1.0 V (V_f). Calculate the rise time $\tau = RC$ and compare it against your plot. You should observe a voltage of about 0.63 V when $t = \tau$.

$\tau =$ ____________________ ms .

Part 2: FOR Loops (10 pts)

One of the main advantages to a language like MATLAB is that it will automatically perform simple calculations across every element in an array. For example, when you calculated $V_c(t)$ in the previous section, all you had to do was plug the "t" array into Eq. (1), and then sit back as MATLAB crunched all the numbers in one simple step. In many other programming languages, however, you would have been required to carefully instruct the computer to iterate over each element in the array. More generally, however, there are situations where you would like the computer to repeat some specific task over and over. To that end, all programming languages offer some sort of command structure for *looping*, with the simplest loop structure being the FOR loop.

To demonstrate how a FOR loop works in MATLAB, let us calculate the series current I_c through your RC circuit. To begin, you will need to set aside a block of memory that will eventually accept our calculations. In MATLAB, this is called *preallocation*, and it is usually accomplished by defining an array of all zeros. A great way to do this is with the "zeros" command as follows:

```
% Preallocate memory for series current (A)
Ic = zeros(1,N);
```

Next, you will need to construct your FOR loop according to the following template:

```
for n = 1:N
    % Instruction 1...
    % Instruction 2...
    % etc...
end
```

The variable "n" in this context is called the *loop index*. Although it looks like we have assigned it to an array of integers from 1:N, it will only take on a single value at a time. For the first iteration of the loop, n will take a single, scalar value of n=1. MATLAB will then execute all the instructions that follow until arriving at the "end" command. It will then reassign the loop index to n=2 and repeat everything over again until reaching n=N (or, equivalently, n=1001 since that is how we defined N).

Finally, you will need to apply Ohm's law to calculate the current at the "nth" time step, or simply $I_c(t_n)$. If we assign the source voltage a value of $V_s = 1.0$ V, then this is accomplished by writing

```
for n = 1:N
    % Calculate nth current using Ohm's law (A).
    Ic(n) = (Vs - Vc(n))/R;
end
```

Use the FOR loop above to calculate the series current I_c and then plot it. Use units of milliamperes (mA) for the current and milliseconds (ms) for time.

Part 3: Logical Indexing & Concatenation (15 pts)

Let us now consider a slightly different scenario with our RC circuit. Instead of turning on and remaining on forever, imagine what would happen if the voltage source V_s turned back off again before the capacitor was finished charging. Mathematically speaking, very little has to change, but we do need to be careful in how we arrange our terms. Begin by defining three variables called t1, t2, and t3, where

```
t1 = 0;      % Start time (s)
t2 = 0.03;   % ON/OFF transition time (s)
t3 = 0.1;    % Stop time (s)
```

Next, you will need to create two arrays of time samples. The first array will sample the interval from [t1, t2], and the second array will sample along [t2, t3]. Since you already have an array called "t" with all the time samples, we simply need to find a way to break it apart into two separate sub-arrays. One simple way to accomplish this is through a process called *logical indexing*. To illustrate how this works, remember that you can pluck out individual values from an array by simply calling the index. For instance, the first three indexes of the time array are:

```
t(1)         % First time sample
t(2)         % Second time sample
t(3)         % Third time sample
```

Likewise, you can pluck out an entire range of indexes by using the colon operator, as with the following command:

```
t(1:3)       % First through third time samples
t([1,2,3])   % Alternative first through third time samples
```

Finally, you can use a relational operator to extract every sample that is less than 100 ms. To demonstrate, try the following command:

```
x = t( t < 0.01 );  % Logical indexing
```

Notice how the command "t < 0.1" returns an array of logical ones and zeros. For every entry with a logical 1, MATLAB will return that value and then ignore all of the zeros. The new array "x" will then contain all samples from t that were less than 0.01 s.

Create two new arrays called "tON" and "tOFF." Using logical indexing, have tON sample all the values in t that are less than t2. This will unpack all of the time samples where $V_s = 1.0$ V. Likewise, have tOFF extract all time samples that are greater than or equal to t2, thereby indicating all the time samples where $V_S = 0.0$ V.

To calculate $V_c(t)$, you will need to break it apart into two segments. Let us call these "Von" and "Voff." To calculate Von, you simply need to apply Eq. (1), but with tON in place of t. To calculate Voff, however, you will need to make two changes. First, V_f and V_i need to be modified. When the capacitor was charging, we used the values $V_f = 1.0$ V and $V_i = 0.0$ V. As it discharges, however, you will need to set $V_f = 0.0$ V and V_i to whatever the final value was in Von.

The second change you will need to make is a shift in the time samples defined by tOFF. Looking at Eq. (1), it is important to remember that the time variable t is referenced against the moment of transition. That is to say, a time of $t = 0$ is defined as the moment when the source turned on. When the source turns back off again, you need to ensure that $t = 0$ is redefined accordingly. The simplest way to do this is by adding an *offset* to tOFF. For example:

```
tOFF = tOFF - t2;   % Offset first index to zero.
```

You can now calculate Voff by applying Eq. (1) again, but using tOFF in place of t.

Once you have calculated Von and Voff, you will need to put them back together again into a single array. In computer science, the act of connecting two arrays together into one is called *concatenation*. Fortunately, MATLAB makes this very easy through the use of brackets. Simply define a new array called "Vc" using

```
Vc = [Von,Voff];    % Concatenate Von and Voff into single array.
```

As long as Von and Voff have the same dimensions (i.e., are both row vectors), then the new array in Vc will be a concatenation of the two. More generally, you can also try using the "cat" command to perform a similar operation.

Create a plot of $V_c(t)$ that accounts for the complete ON/OFF state of the source voltage V_s over the interval $[0, 0.1]$ s.

Part 4: Conditionals (15 pts)

For the final part of this exercise, we shall calculate the series current I_c through the capacitor during both the ON and OFF states of the source voltage. As we found earlier, Ohm's law dictates that the series current at time t satisfies

$$I_c(t) = \frac{V_s(t) - V_c(t)}{R} \ .$$

We have already calculated $V_c(t)$ in the previous section, but we have not yet created an array of source voltages to represent $V_s(t)$. To accomplish this, we shall make use of a special tool known as the *conditional.*

In MATLAB, the most basic conditional is the classic IF/ELSE structure depicted below:

```
if CONDITION
    % Instruction block A
else
    % Instruction block B
end
```

The CONDITION term is typically some sort of logical relation, like "a == b" or "x <= y." If the relation returns a value of 1, then the instructions in block A are executed while block B gets ignored. Otherwise, block A gets ignored and block B gets executed.

To demonstrate the conditional, create an array of source voltages as follows:

```
Vs = zeros(1,N);    % Source voltages (V)
```

Next, create a FOR loop like the one in Part 2 that scans across each sample in time:

```
for n = 1:N
    % Instructions go here...
end
```

Finally, insert a conditional (IF/ELSE) statement within the FOR loop. Have the conditional check the nth time index with the ON/OFF transition time as follows:

```
if t(n) < t2
    % Set nth Vs sample to 1.0 V
else
    % Set nth Vs sample to 0.0 V
end
```

This should fill the array of voltage samples to their appropriate values. When finished, calculate the series current I_c and then plot both $V_s(t)$ and $I_c(t)$.

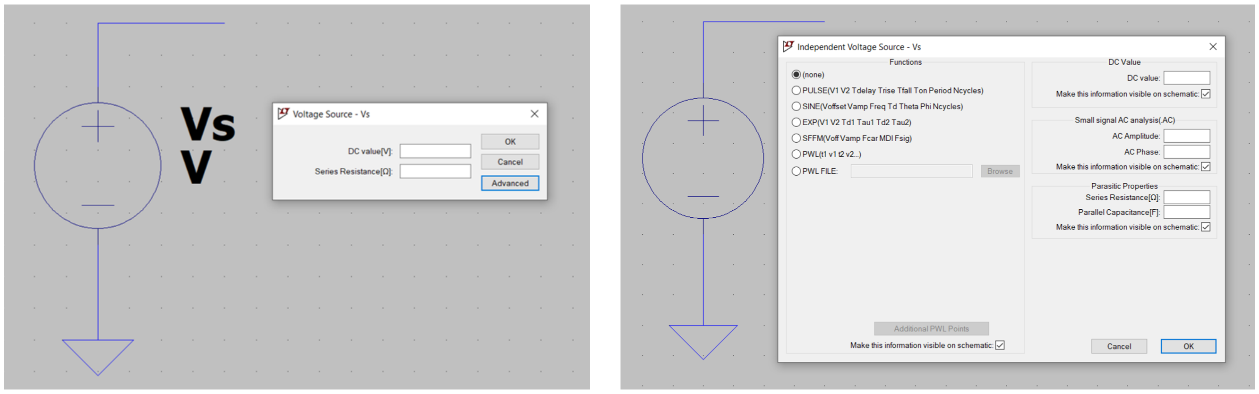
Vs
V
Voltage Source - Vs
DC value[V]:
Series Resistance[Ω]:
OK
Cancel
Advanced
Independent Voltage Source - Vs
Functions
(none)
PULSE(V1 V2 Tdelay Trise Tfall Ton Period Ncycles)
SINE(Voffset Vamp Freq Td Theta Phi Ncycles)
EXP(V1 V2 Td1 Tau1 Td2 Tau2)
SFFM(Voff Vamp Fcar MDI Fsig)
PWL(t1 v1 t2 v2...)
PWL FILE:
Browse
Additional PWL Points
Make this information visible on schematic: ✓
DC Value
DC value:
Make this information visible on schematic: ✓
Small signal AC analysis(.AC)
AC Amplitude:
AC Phase:
Make this information visible on schematic: ✓
Parasitic Properties
Series Resistance[Ω]:
Parallel Capacitance[F]:
Make this information visible on schematic: ✓
Cancel
OK

Figure 2: *How to modify a voltage source's behavior in the "advanced" menu.*

EXTRA CREDIT: Transients in LTspice (25 pts)

Now that you are familiar with the mathematical theory of RC circuits, we are ready to simulate the same circuit in LTspice. Unlike your previous simulations, however, we will not calculate a DC operating point. Instead, we are simulating a *transient* model, meaning the sources are changing over time, and so various voltages/currents in the circuit are also changing in time.

Begin by creating a simple voltage source as shown in Fig. 2. Be sure to include a ground reference as well as a little piece of wire from the positive terminal. If you right-click on the source, you should see a button called "advanced" from the window. Select this button to open a new menu of excitation functions. Notice the huge variety of possibilities, including pulse functions, sine waves, and even exponentials.

Select the "PULSE" option from the list to indicate a rectangular pulse that turns on and off over time. You should notice a series of new parameters to fill in, thereby defining the pulse function. Define your voltage source to an initial value of 0.0 V and an "on" value of 1.0 V. Set the total "on time" (Ton) to a value of 30 ms with one cycle of the pulse.

Although most of these parameters are pretty self-explanatory, there are some tricks to be mindful of. In particular, the rise and fall time cannot be left at zero, or else the simulation seems to default to a value of 10% of the pulse width. You must therefore insert something very tiny, like 1.0 ns, to override this behavior and mimic an instantaneous transition.

Once the pulse function has been defined, the next step is tell LTspice to perform a transient simulation. Up until now, you have only been running DC operation point simulations, so we will need to modify our command. Begin by clicking the "Simulate" drop-down menu and selecting "Edit Simulation Command." Chose the tab marked "Transient" and insert the appropriate parameters. Specifically, select a stop time of 100 ms with a maximum time step of 1.0 ms. When finished, you should be able to run the simulation and probe the output voltage of the source. Take a moment to verify that the voltage source is behaving as you expect before moving on to the next step.

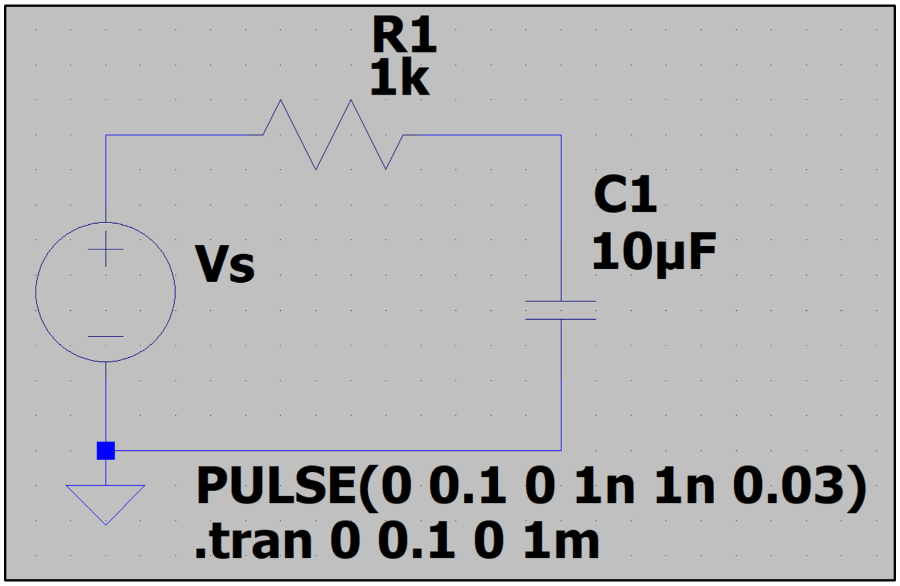

Figure 3: *Transient simulation of a simple RC circuit in LTspice.*

Now that your voltage source is creating a simple on/off pulse, you are ready to simulate a basic RC circuit. All you need to do is add a 1.0 kΩ resistor with a 10 μF capacitor in series. When finished, your circuit should look similar to the one in Fig. 3. Rerun your simulation again and then plot the voltage across the capacitor along with the current. Take a screen capture of the signals and compare them against your analytic calculations from Part 1. If everything is in agreement, then turn in your images along with your source code.