EEEE2055: Modelling Methods and Tools - Coursework 2

Numerical Integration and Techniques

Name: [Kexin Yu]

Student id: [20320941]

Date: [2023.03.21]

Contents

1. Initial model setup and Initial simulation (5%)

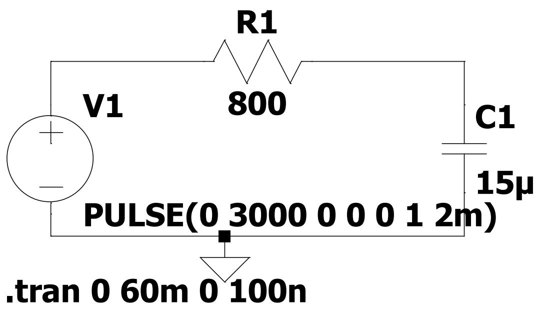


Figure 1.1: LTSpice model of the RC circuit

Figure 1.1 shows the RC circuit model designed as required. It can be observed from the information in the figure that the simulation stop time is set to 60 ms and the maximum time step is set to 100 ns. The voltage source of the circuit is a voltage pulse, T_{rise} and T_{fall} are set at 0s to make it like a stepped input. At t=0, its step voltage V_{on} is 3 KV.

Figures 1.2 and 1.3 show the results of the simulation based on the model in Figure 1.1. Figure 1.2 shows mainly the simulated values for the capacitor $C_1$and Figure 1.3 shows the simulated values for the resistor R_1. In the following, the simulation results are discussed in detail for each of the two figures based on capacitor and resistor.

Capacitor C_1:

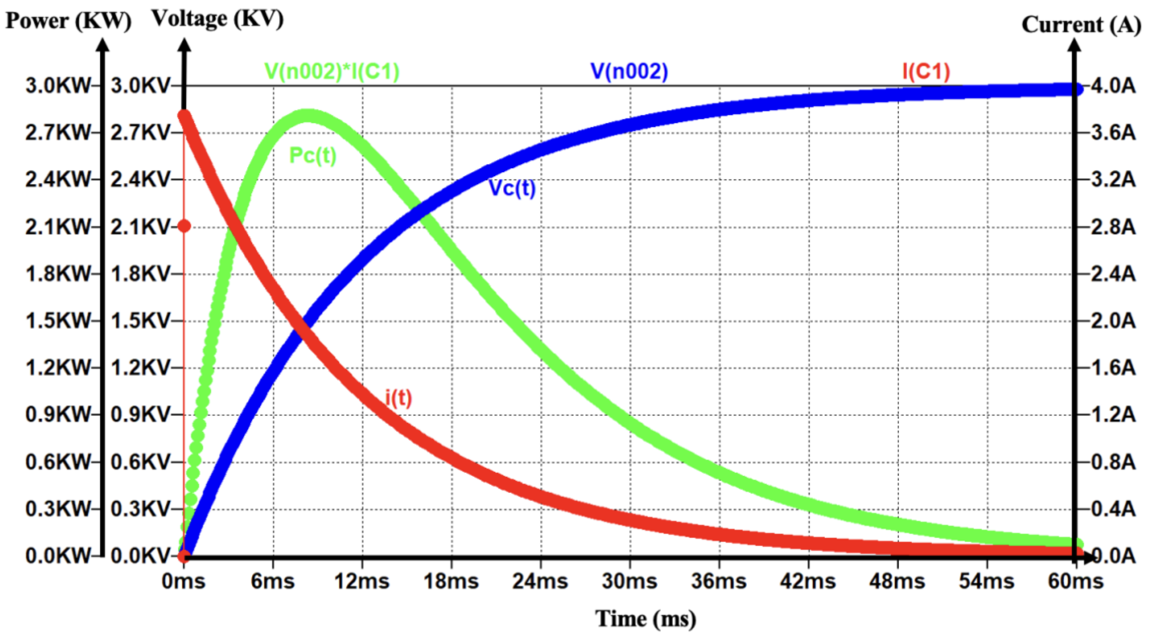


Figure 1.2: Simulation results for i(t), V_C(t) and P_C(t) of the capacitor C_1

The red line I(C1) in this figure is the current through the capacitor i(t), whose vertical coordinate is on the right side of the figure in A. The blue line V(n002) is the voltage across the capacitor V_C(t), whose vertical coordinate is on the left side of the figure in KV. The green line V(n002)*I(C1) is the instantaneous capacitor power $P_C(t)$, whose vertical coordinate is on the left side of the figure in KW. The horizontal coordinate in the Figure is the time in ms.

Equation 1 is the function of the current i(t). Based on this, the current through the capacitor $V_C(t)$ can be calculated as:

$$\mathrm{i(t)} = I_{pk}e^{-t/\tau} \qquad \textbf{Eqn.1}$$

$$\mathrm{i(t)} = \frac{V_{on}}{R}e^{-t/RC} = \frac{3000}{800}e^{-t/800\times15\times10^{-6}} = 3.75e^{-83.333t}$$

Equation 2 is the function of the voltage $V_C(t)$. The voltage across the capacitor (t) can be calculated as:

$$\mathrm{V}_C(\mathrm{t}) = \mathrm{E} - \mathrm{V}_R(\mathrm{t}) \qquad \textbf{Eqn.2}$$

$$\mathrm{V}_C(\mathrm{t}) = \mathrm{E} - \mathrm{Ri(t)} = 3000 - 3000e^{-83.333t} = 3000(1 - e^{-83.333t})$$

Based on the function of the $V_C(t)$ and i(t), instantaneous capacitor power $P_C(t)$ is:

$$\mathrm{P}_C(\mathrm{t}) = \mathrm{V}_C(\mathrm{t})\mathrm{i(t)} = 3.75e^{-83.333t}(3000 - 3000e^{-83.333t}) = 11250(e^{-83.333t} - e^{-166.667t}) \quad \textbf{Eqn.3}$$

When the derivative of the above equation is 0, its t can be calculated as follows. At this point the power reaches its maximum.

$$\frac{d\mathrm{P}_C(\mathrm{t})}{dt} = 11250(-83.333e^{-83.333t} + 166.667e^{-166.667t}) = 0$$

$$t = \frac{ln_2}{83.333} = 8.32\ ms$$

Based on the above equation, the current i(t) should start out at 3.75 A, and as the charging time increases, the current drops exponentially from a peak of 3.75 A to 0 A. For the voltage $V_C(t)$, the initial value should be 0 V, and as time increases, the voltage increases exponentially to a peak of 3 KV. The power $P_C(t)$ should start out at 0 W and reach a maximum of 2.81 KW at a time of 8.32 ms. These values can be verified in Figure 1.2.

Resistor R_1:

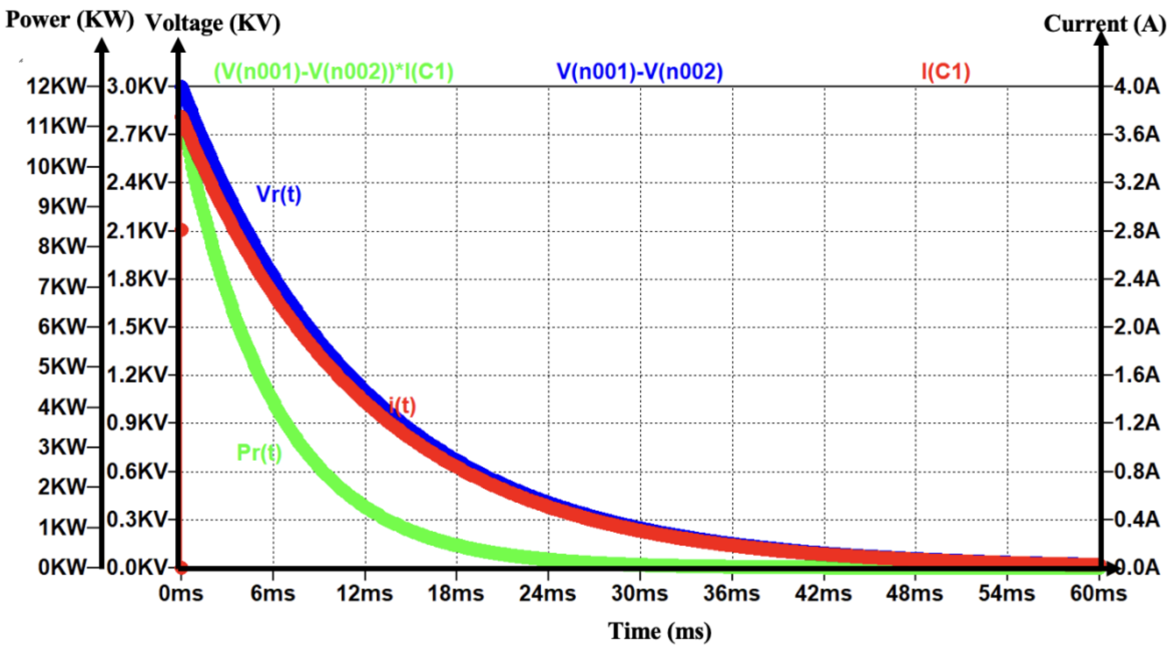


Figure 1.3: Simulation results for i(t), V_R(t) and P_R(t) of the resistor R_1

The red line I(C1) in this figure is the current through the resistor i(t), whose vertical coordinate is on the right side of the figure in A. The blue line V(n001)-V(n002) is the voltage across the capacitor V_R(t), whose vertical coordinate is on the left side of the figure in KV. The green line (V(n001)-V(n002))*I(C1) is the instantaneous capacitor power P_R(t), whose vertical coordinate is on the left side of the figure in KW. The horizontal coordinate in the figure is the time in ms.

Equation 4 is the function of the voltage. The voltage across the resistor V_R(t) can be calculated as:

$$V_R(\mathrm{t}) = \mathrm{i(t)}R \qquad \textbf{Eqn.4}$$

$$V_R(\mathrm{t}) = 3.75e^{-83.333t} \times 800 = 3000e^{-83.333t}$$

Based on the function of the V_R(t) and i(t), instantaneous resistor power P_R(t) is:

$$P_R(\mathrm{t}) = V_R(\mathrm{t})\mathrm{i(t)} = 3.75e^{-83.333t} \times 3000e^{-83.333t} = 11250e^{-166.667t} \qquad \textbf{Eqn.5}$$

According to the above equation, the current i(t) changes as discussed in the capacitor section. The voltage V_R(t) should start at 3 KV and as the charging time increases, the voltage is transferred to the capacitor, whose value drops exponentially from a peak of 3 KV to 0 V. The power P_R(t) is maximum at the beginning, at 11250 W. These values can be verified in Figure 1.3.

2. Comparison of results with varying timesteps (10%)

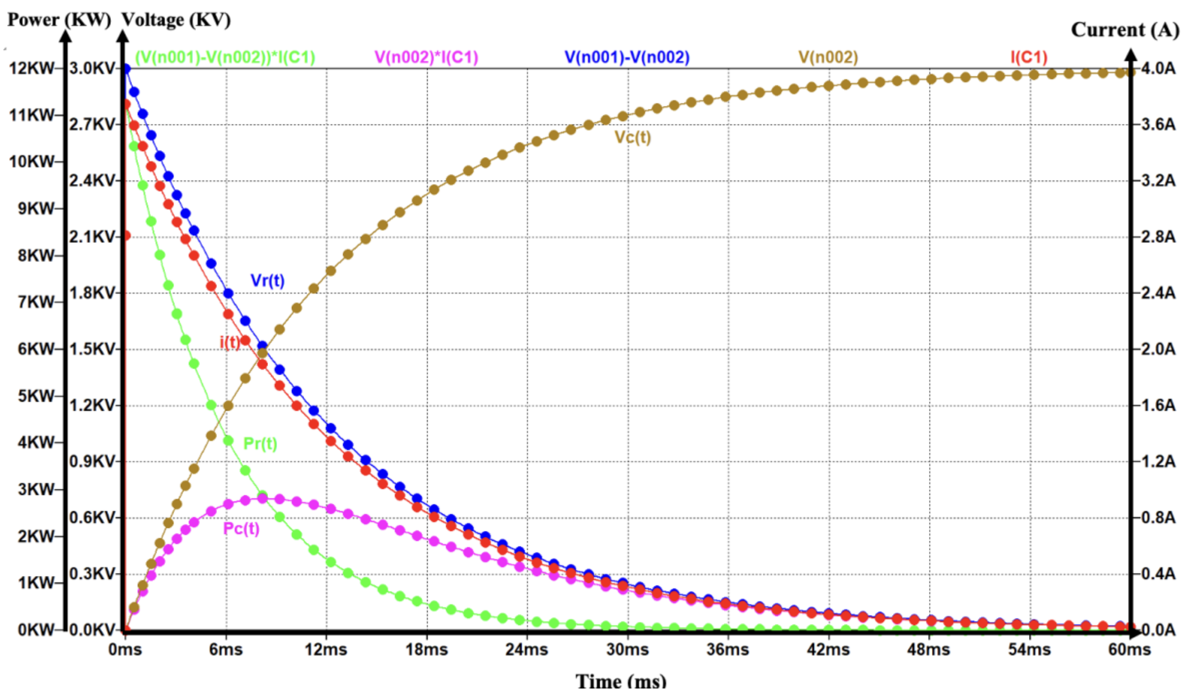


Figure 2.1: Simulation results for i(t), V_C(t), V_R(t), P_C(t) and P_R(t) (Maximum timestep = 1μs)

Figure 2.1 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 1 μs. The marked points in this plot are the most numerous of the six different timestep results.

In Figure 2.1, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor V_C(t) is represented by the brown line V(n002), and the voltage across the resistor V_R(t) is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor P_C(t) and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor P_R(t), the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

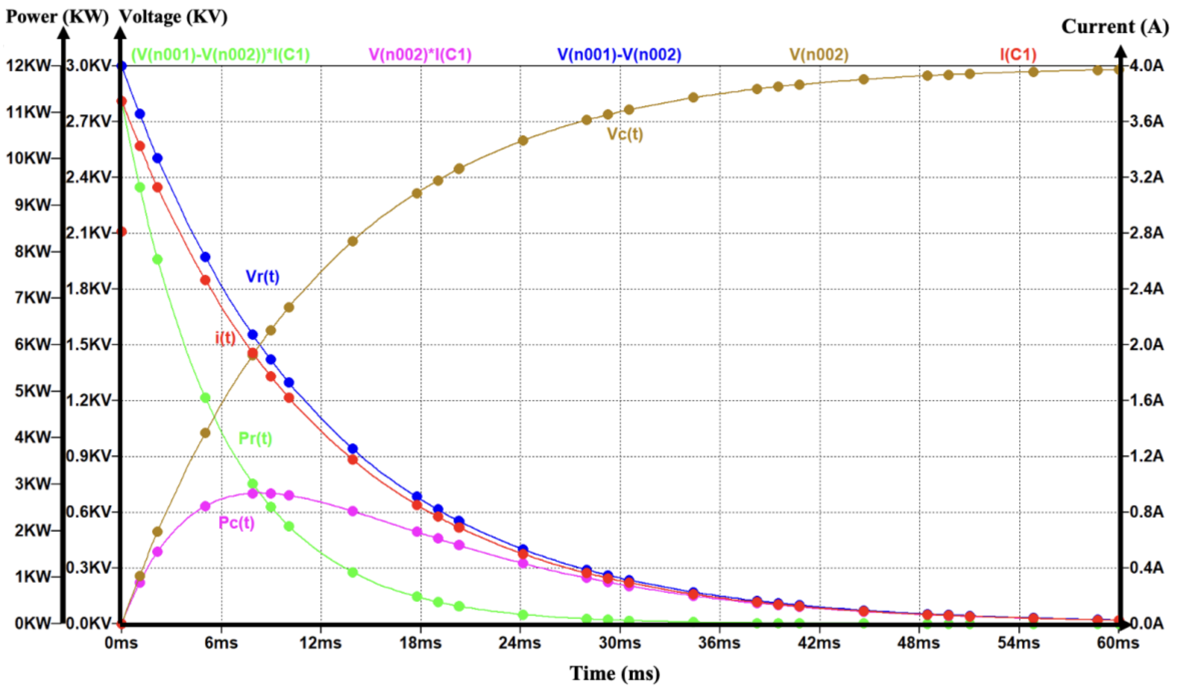


Figure 2.2: Simulation results for i(t), V_C(t), V_R(t), P_C(t) and P_R(t) (Maximum timestep = 10µs)

Figures 2.2 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 10 µs. The marked points in this plot are the second numerous of the six different timestep results.

In Figure 2.2, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor V_C(t) is represented by the brown line V(n002), and the voltage across the resistor V_R(t) is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor P_C(t) and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor P_R(t), the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

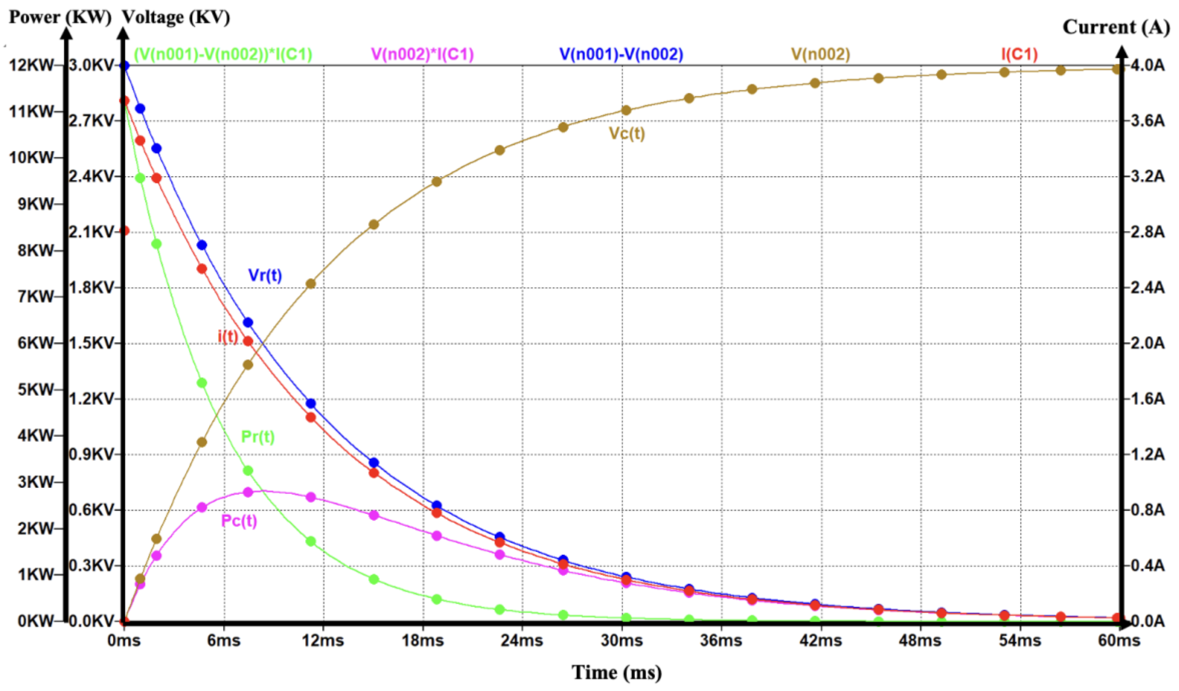


Figure 2.3: Simulation results for i(t), V_C(t), V_R(t), P_C(t) and P_R(t) (Maximum timestep = 100µs)

Figures 2.3 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 100 µs. The marked points in this plot are the third numerous of the six different timestep results.

In Figure 2.3, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor V_C(t) is represented by the brown line V(n002), and the voltage across the resistor V_R(t) is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor P_C(t) and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor P_R(t), the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

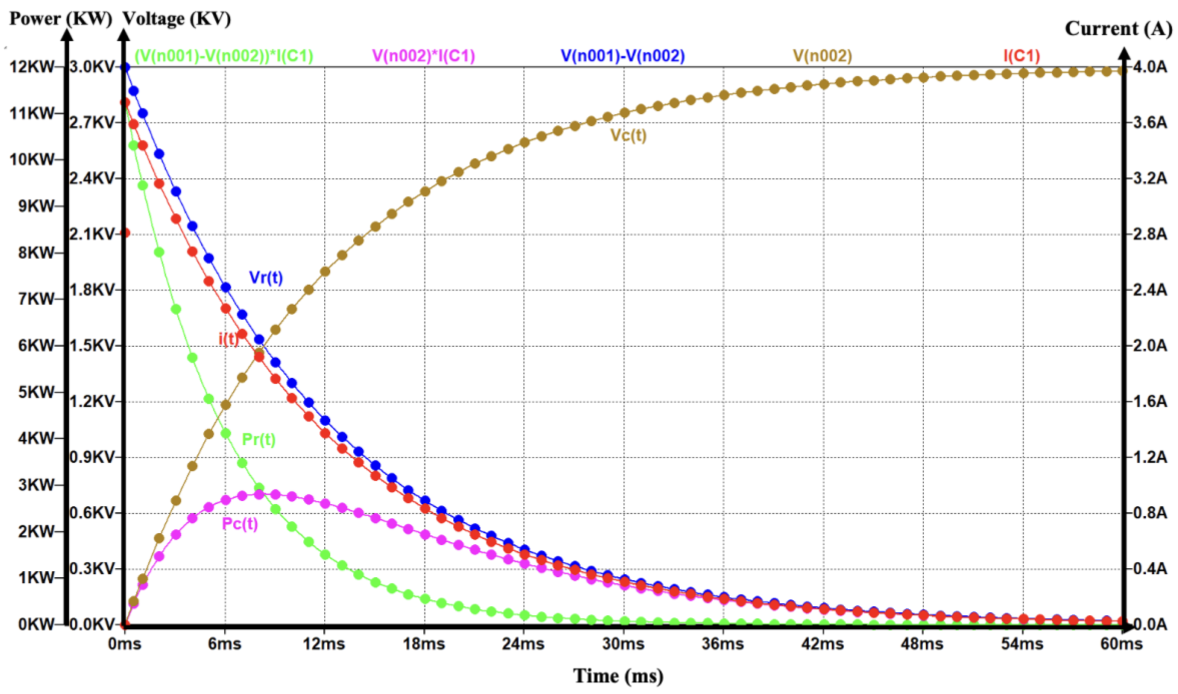


Figure 2.4: Simulation results for i(t), V_C(t), V_R(t), P_C(t) and P_R(t) (Maximum timestep = 1ms)

Figure 2.4 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 1 ms. The marked points in this plot are the fourth numerous of the six different timestep results.

In Figure 2.4, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor V_C(t) is represented by the brown line V(n002), and the voltage across the resistor V_R(t) is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor P_C(t) and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor P_R(t), the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

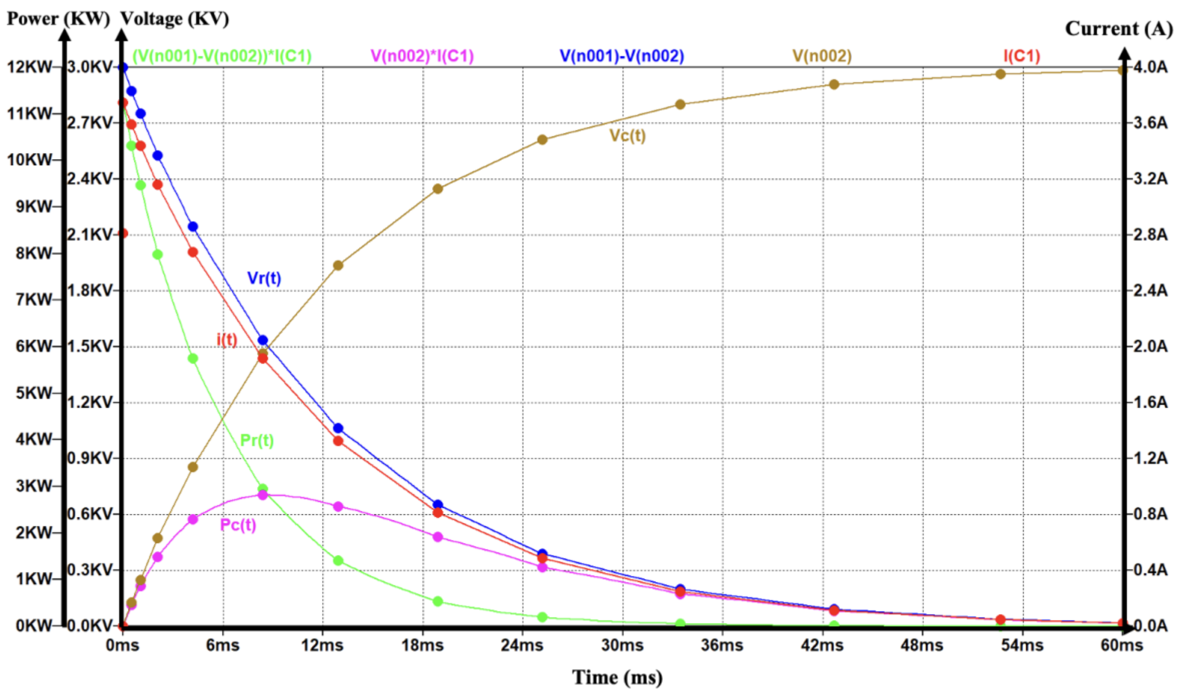


Figure 2.5: Simulation results for i(t), $V_C(t)$, $V_R(t)$, $P_C(t)$ and $P_R(t)$ (Maximum timestep = 10ms)

Figures 2.5 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 10 ms. The marked points in this plot are the fifth numerous of the six different timestep results.

In Figure 2.5, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor $V_C(t)$ is represented by the brown line V(n002), and the voltage across the resistor $V_R(t)$ is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor $P_C(t)$ and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor $P_R(t)$, the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

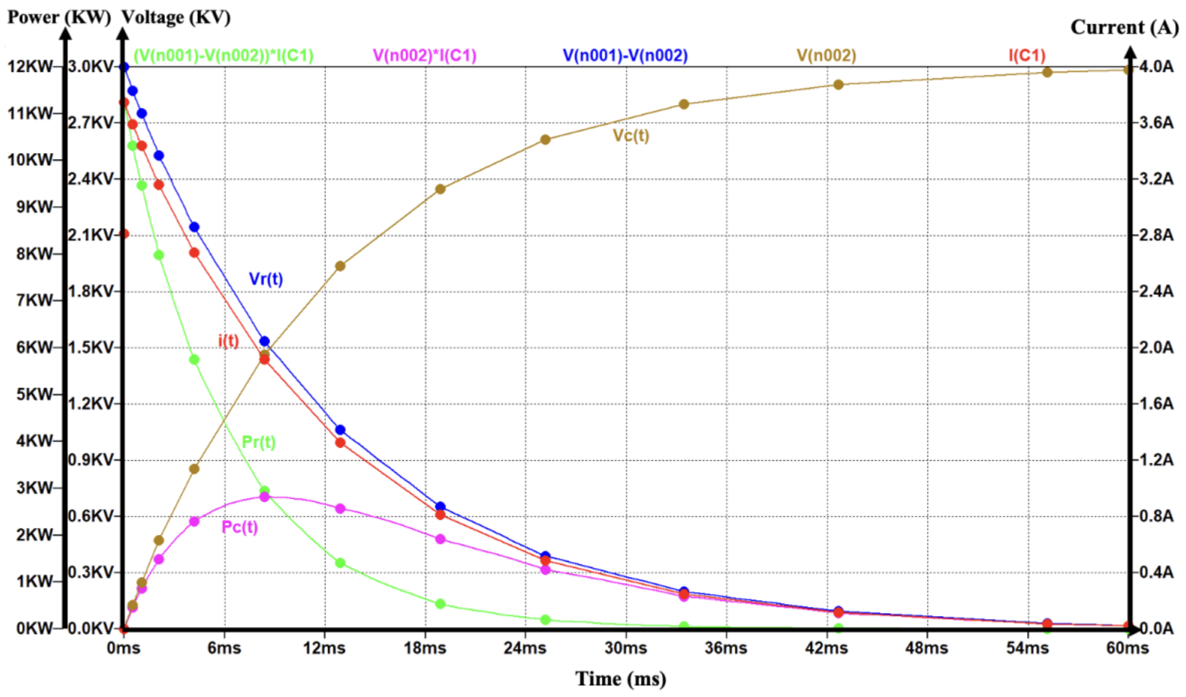


Figure 2.6: Simulation results for i(t), V_C(t), V_R(t), P_C(t) and P_R(t) (Maximum timestep = 100ms)

Figure 2.6 shows the simulation results for the capacitance and resistance related quantities at a stop time of 60 ms and a maximum time step of 100 ms. The marked points in this plot are the least numerous of the six different timestep results.

In Figure 2.6, the red line I(C1) is the current through capacitor i(t). Its vertical coordinate is on the right side of the figure, and the unit is A. The voltage across the capacitor V_C(t) is represented by the brown line V(n002), and the voltage across the resistor V_R(t) is represented by the blue line V(n001)-V(n002). The vertical coordinates of both lines are on the left side of the figure and their units are KV. The purple line V(n002)*I(C1) is the instantaneous power of the capacitor P_C(t) and the green line (V(n001)-V(n002))*I(C1) is the instantaneous power of the capacitor P_R(t), the left side of the figure shows their vertical coordinates, which are in KW. The horizontal coordinate in the figure is the time and its unit is ms.

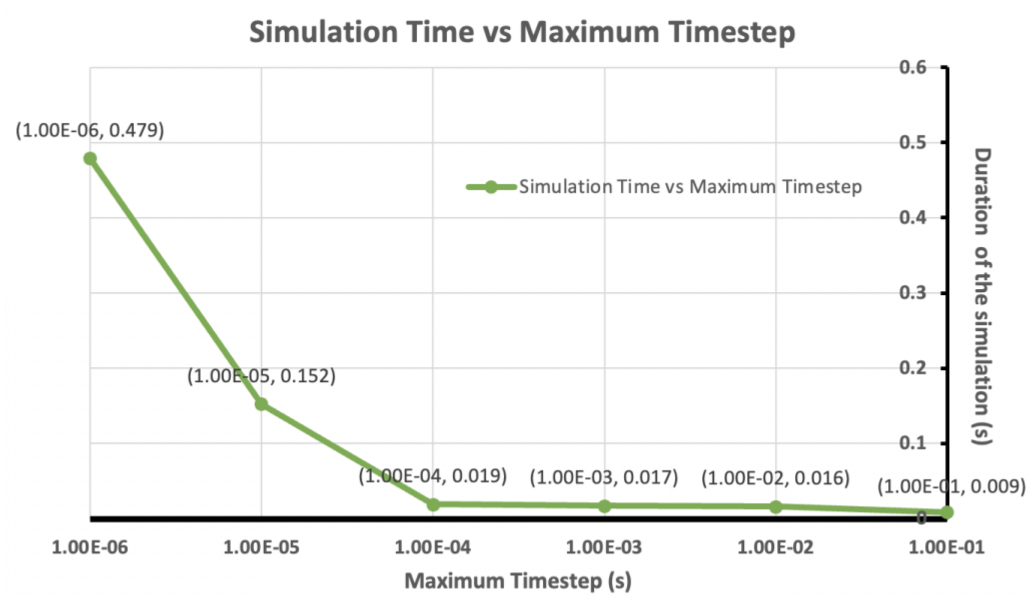


Figure 2.7: Simulation time at different Maximum timestep

Figure 2.7 shows the simulation time versus maximum timestep for runs with different maximum timesteps of 1 μs, 10 μs, 100 μs, 1 ms, 10 ms, and 100 ms.

Accuracy:

It can be found that as the maximum time step increases from 1us to 100ms, there are fewer and fewer marker points in the figure. One strange phenomenon is that more marker points are displayed when timestep =1 ms than when timestep =100 μs and 10 μs. This is because LTspice cannot display too many marker points at once and can only show a trend. By running the log file for each simulation, we can see that there are approximately 60000 marker points at timestep =1 μs, 6000 marker points at timestep =10 μs, 600 marker points at timestep =100 μs, and 60 marker points at timestep =1 ms. This is still consistent with the feature that the larger the timestep the fewer the markers. The reason why this happened is that the time step increases but the stop time remains the same resulting in an increase in the interval between each marker point. As the interval between two marker points increases, the detail of the interval variation is affected, producing some spikes or variations, which leads to a decrease in accuracy.

Duration of the simulation:

It can be observed that as the maximum time step increases from 1us to 100ms, the time needed for the simulation decreases. This is because, with the same stopping time, an increase in the step size means that fewer data points need to be processed during this period, and thus less simulation time is spent. Although the increase in time step leads to a reduction in simulation time, the accuracy of the simulation results is reduced.

It can be concluded from the above analysis that a larger time step can reduce the time required for the simulation, but it also leads to a reduction in its accuracy. It is therefore important to choose a suitable timestep.

3. Analytic calculation: Energy dissipation in the resistor and capacitor (5%)

Equations 6 and 7 are the formula to calculate the energy dissipated in the resistor $E_R(t)$ and the energy stored in the capacitor $E_C(t)$. From the requirements of the question, R is 800 Ω, C is 15 μF, Von is 3 KV, and only the calculation procedure for t = 0.5 ms is listed here. The calculation results for t = 0.5 ms, 5 ms, 10 ms, 30 ms and 60 ms are written in Table 1.

$$E_R = \frac{1}{2} \times C{V_{on}}^2 (1 - e^{-\frac{2t}{RC}}) \qquad \textbf{Eqn.6}$$

$$E_C = \frac{1}{2} \times C{V_{on}}^2 (1 - e^{-\frac{t}{RC}})^2 \qquad \textbf{Eqn.7}$$

$$E_R(0.5\,ms) = \frac{1}{2} \times 15 \times 10^{-6} \times 3000^2 \left(1 - e^{-\frac{2\times0.5\times10^{-3}}{800\times15\times10^{-6}}}\right) = 5.3970\,J$$

$$E_C(0.5\,ms) = \frac{1}{2} \times 15 \times 10^{-6} \times 3000^2 \left(1 - e^{-\frac{0.5\times10^{-3}}{800\times15\times10^{-6}}}\right)^2 = 0.1124J$$

Table 1: Energy derived from analytic calculations

	0.5ms	**5ms**	**10ms**	**30ms**	**60ms**
energy stored in the capacitor (E_C) [J]	0.1124	7.8379	21.5783	56.8733	66.5934
energy dissipated in the resistor (E_R) [J]	5.3970	38.1646	54.7508	67.0452	67.4969

4. Estimation of Energy at Specified Time Intervals using LT Spice (15%)

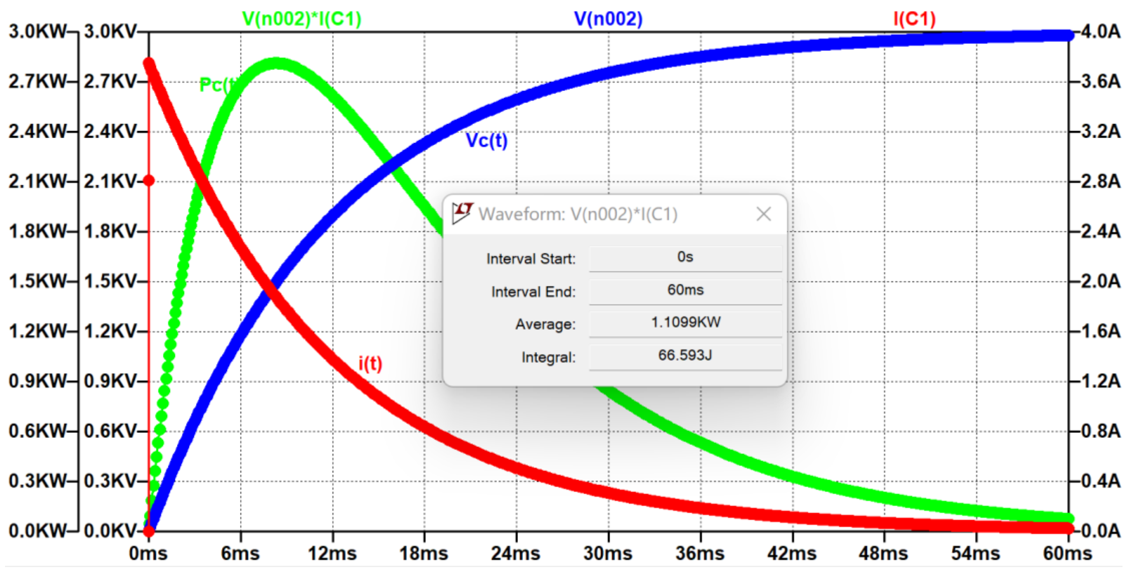
Three integration techniques are available on the control panel: trapezoidal, modified trap, and gear.

Trapezoidal method is a simple numerical integration method that approximates the integration of a function by dividing the area under the curve into trapezoids. The voltage and current values for each time step are calculated by averaging the values of the current and previous time steps. Since the trapezoidal technique is a first-order approach, its error is proportional to the step size.

Modified trap method is a tweaked version of the trapezoidal method that lowers the error by including an extra term that relies on the second derivative of the voltage or current. Being a second-order approach, its error is proportional to the square of the step size. For a given step size, this method is more accurate than the trapezoidal method, but takes longer to calculate.

Gear method is a variable order method designed to deal with rigid equations. This method gives the most accurate results, but it may require more calculation time than the two methods above. The gearing method is used to derive the voltage and current values for each time step from the previous time steps by using a variable order polynomial. This method is suitable for simulating complex circuits.

Modified Trap method was used here. When analysing simple RC circuits, this method is more applicable as it is more accurate than the trapezoidal method and faster than the Gear integral. It is unique among SPICE programs and is the best technique for integrating the differential equations of a circuit. [1] The specific steps to integrate via LTSpice are: First, zoom in on the image to the area to be integrated. Afterward, click on the label of the region to be integrated while holding down the "Ctrl" key. And finally, the integration value for that region is displayed. As shown in Figure 4.1, the magnitude of the capacitor energy in the region of 0 to 60 ms at a timestep of 100 ns is 66.593 J.



Waveform: V(n002)*I(C1)

Interval Start:	0s
Interval End:	60ms
Average:	1.1099KW
Integral:	66.593J

Figure 4.1: Capacitor energy integration values from 0 to 60ms (Maximum timestep = 100 ns)

According to the method mentioned above, the energy stored in capacitor E_C, and the energy dissipated in resistor E_R during the first 0.5ms, 5ms, 10ms, 30ms and 60ms of the voltage step are ¥

Table 2: Estimation of Energy using LT Spice when the maximum step size is set to 10ms

	0.5ms	**5ms**	**10ms**	**30ms**	**60ms**
energy stored in the capacitor (E_C) [J]	0.112	7.664	21.297	58.223	68.926
energy dissipated in the resistor (E_R) [J]	5.398	38.689	56.450	70.272	70.836

Table 3: Estimation of Energy using LT Spice when the maximum step size is set to 100ns

	0.5ms	**5ms**	**10ms**	**30ms**	**60ms**
energy stored in the capacitor (E_C) [J]	0.112	7.838	21.578	56.873	66.593
energy dissipated in the resistor (E_R) [J]	5.397	38.165	54.752	67.046	67.498

Equation 8 and Equation 9 calculate the errors in the theoretically calculated and LTSpice simulated values of the energy stored in the capacitor E_C and the energy consumed by the resistor E_R respectively. With these two equations and the values in Table 1,2,3, the energy errors for different timesteps and times can be calculated. The calculation procedure here only shows the energy of the resistor and capacitor in the first 60 ms when timestep = 10 ms, the other cases are similar. All results are presented in Table 4 and Table 5.

$$LTspice\_Error_{E_R} = \frac{LTspice_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} \qquad \textbf{Eqn.8}$$

$$LTspice\_Error_{E_c} = \frac{LTspice_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} \qquad \textbf{Eqn.9}$$

$$LTspice\_Error_{E_R}(60\text{ ms}) = \frac{LTspice_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} = \frac{70.836 - 67.4976}{67.4976} \times 100\% = 4.947\%$$

$$LTspice\_Error_{E_c}(60\text{ ms}) = \frac{LTspice_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} = \frac{68.926 - 66.5934}{66.5934} \times 100\% = 3.503\%$$

Table 4: Error in energy estimation using LT Spice when the maximum step size is set to 10ms

Error %	**0.5ms**	**5ms**	**10ms**	**30ms**	**60ms**
energy stored in the capacitor (E_C)	-0.356	-2.219	-1.304	2.373	3.503
energy dissipated in the resistor (E_R)	0.018	1.374	3.103	4.813	4.947

Table 5: Error in energy estimation using LT Spice when the maximum step size is set to 100ns

Error %	**0.5ms**	**5ms**	**10ms**	**30ms**	**60ms**
energy stored in the capacitor (E_C)	-0.356	0.001	-0.002	-0.001	-0.001
energy dissipated in the resistor (E_R)	0.000	0.001	0.002	0.001	0.002

As can be observed from the results in the table, out of the 20 calculated error values, 1 value is 0.000%, which means that the estimates integrated using LTSpice are equal to the calculated values (means it is correct), 12 values are positive, meaning that the estimates are larger than the calculated values, which is an over estimation, and 7 values are negative, meaning that the estimates are smaller than the calculated values, which is an under estimation (7 values are under estimation and 12 values are over estimation).

By comparing the error values in both tables, the errors for the 100ns maximum time step are smaller than the errors for the 10ms maximum time step (all error values are even less than 1% for the 100 ns timestep and 8 error values are greater than 1% in absolute value for the 10 ms timestep), which is in line with the conclusion in the second question that larger time steps result in fewer data points being analysed, reducing the accuracy values and leading to larger errors. Another point worth mentioning is that the error in the energy stored by the capacitor is smaller than the error in the energy consumed by the resistor for most of the maximum step sizes being equal. This is because the change in resistor energy is greater than the change in capacitor energy over the same time frame, which means that the resistor has more product components over the same time frame, resulting in a greater error.

5. Analysis of LTSpice data points (5%)

The name of the dataset file is “Dataset_20320941.txt”. The first 10 lines of the data in this file were recorded in Table 6. The table also includes a space between two adjacent data points, which can be observed as not being the same, indicating that they are not uniform.

Table 6: First 10 lines of the data in “Dataset_20320941.txt” with calculated space time

Time (s)	Space Time (s)	P_C (W)	P_R (W)
0.000000000000000e+000	1.00e-12	0.000000e+000	0.000000e+000
9.999999960041972e-013	1.00e-12	7.031250e-007	8.437500e+003
1.999999992008394e-012	1.44e-05	1.875000e-006	1.125000e+004
1.436158342849684e-005	1.44e-05	1.343979e+001	1.122310e+004
2.872316485699368e-005	1.44e-05	2.683142e+001	1.119627e+004
4.308474628549053e-005	1.44e-05	4.017500e+001	1.116951e+004
5.744632771398737e-005	1.44e-05	5.347066e+001	1.114280e+004
7.180790914248421e-005	1.44e-05	6.671850e+001	1.111616e+004
8.616949057098106e-005	1.44e-05	7.991865e+001	1.108959e+004
1.005310719994779e-004	1.00e-04	9.307121e+001	1.106307e+004

$$Trapezoidal\ Rule: \int_0^T v(t)dt \approx \Delta\left(0.5v_0 + \sum_{j=1}^{N-1} V_j + 0.5v_n\right) \quad \textbf{Eqn.10}$$

$$Simpson's\ Rule: \int_{t_1}^{t_2} v(t)dt \approx \frac{3}{\Delta}\left(v_0 + 4\sum_{j=1,3,5,7,\ldots n-1} v_j + 2\sum_{j=2,4,6,8,\ldots n-2} v_j + v_n\right) + O\Delta^4 \quad \textbf{Eqn.11}$$

Equation 10 and Equation 11 are formulas for the Trapezoidal rule and Simpson's rule respectively, both of which require equidistant data points (Consistent Δ) to accurately approximate the integral. Due to the different time intervals of the derived data, it may not be appropriate to use these methods. Trapezoidal rule assumes that the function being integrated is linear between the data points and approximates the area under the curve using segments of straight lines. Simpson's rule assumes that the function being integrated is quadratic between the data points and uses a quadratic polynomial to approximate the area under the curve. If the data points are not equally spaced, the approximation may not accurately represent the shape of the curve and the results obtained may be less accurate.

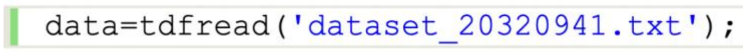
6. Calculation of Energy using Simpson’s Rule and Trapezoidal Rule in Matlab (15%)

To calculate the energy in Matlab using Simpson's rule and the Trapezoidal rule, the data points in the dataset must be equally spaced, so this should first be implemented by code. The task is described in three parts, the first concerns the preparation of the equally spaced dataset. Secondly, Simpson's rule is used to calculate energy. Finally, Trapezoidal rule is used to calculate the energy. The main code for each part is shown below, differences from those in the provided Matlab script are indicated below, and the full code is shown in Appendix A and B.

Prepare datasets:

1. Import data

The first step was to import the dataset obtained in LTspice into Matlab to facilitate further data processing. The code used here is shown in Figure 6.1, where “dfread” can read data from a ‘txt’ file. This command creates a variable in the workspace for each file column and names it based on the value of its first line. The content in brackets is the name of the file, the only difference with the provided script is that here the name of the file had been changed.

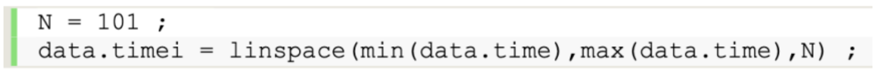


```
data=tdfread('dataset_20320941.txt');
```

Figure 6.1: Code used to import data

2. Create equally spaced data points for x-axis (time)

The second step was to create equidistant points for the x-axis. Figure 6.2 shows the code to achieve this. “N” is the number of data points, and it needs to be an odd number so that an even “N-1” number of equidistant strips can be produced. The "linspace(min(data.time),max(data.time),N)" command can be used in the "min(data.time)" and "max(data.time)" to create “N-1” equidistant points between the two values. Here “N” is 101, indicating that 100 equidistant points can be created between the two values. The difference with the supplied script is that when it requires data points of 10, 100, 1000, and 10000, “N” should also be changed to 11, 101, 1001, and 10001.

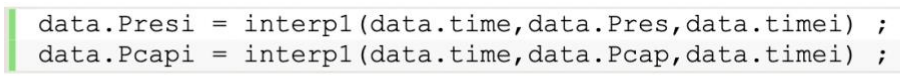


```
N = 101 ;
data.timei = linspace(min(data.time),max(data.time),N) ;
```

Figure 6.2: Code used to create equally spaced data points for x-axis (time)

3. Create equally spaced data points for y-axis

The third step was to create equidistant points for the y-axis. Figure 6.3 shows the code to achieve this. "interp1()" returns the interpolated value for a specific query point, using linear interpolation. With this command, the y-value corresponding to the original x-value becomes the y-value of the new equidistant x point.



```
data.Presi = interp1(data.time,data.Pres,data.timei) ;
data.Pcapi = interp1(data.time,data.Pcap,data.timei) ;
```

Figure 6.3: Code used to create equally spaced data points for y-axis

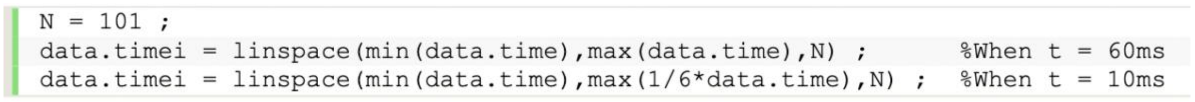
With the above steps, the datasets have equal spacing. Table 7 shows the results of the dataset when the number of intervals is 10, the intervals are equal (= 0.006 s). Such a dataset can further be used using Simpson's rule or the Trapezoidal rule.

Table 7: Dataset with 10 interval points

Time (s)	P_C(W)	P_R (W)
0.000000	0.000000	0.000000
0.006000	2684.742254	4138.787005
0.012000	2616.110442	1522.557755
0.018000	1950.111109	560.117755
0.024000	1316.477159	206.058037
0.030000	847.654654	75.802041
0.036000	532.222516	27.886967
0.042000	329.463468	10.258881
0.048000	202.278274	3.774052
0.054000	123.588857	1.388410
0.060000	75.29115	0.510750

Integrate with Simpson's rule:

Due to the different time requirements when using this method for energy calculations, the code in the second step of "Prepare for datasets" should be changed according to the different times. When the first 10 ms of energy was required, only the first 1/6th of the data points were taken, as the overall simulation time was 60 ms. When the first 60 ms of energy was to be calculated, all the data points were taken. Figure 6.4 shows the difference.



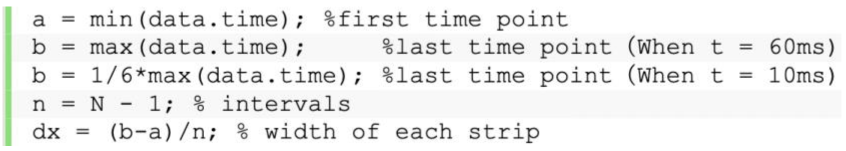
```
N = 101 ;
data.timei = linspace(min(data.time),max(data.time),N) ;        %When t = 60ms
data.timei = linspace(min(data.time),max(1/6*data.time),N) ;  %When t = 10ms
```

Figure 6.4: Code for creating equidistant data points for the x-axis (time) at different times

The implementation of this rule in Matlab is mainly a matter of converting the Simpson mathematical formula shown in Equation 11 into a script in Matlab (The formula is shown again below for ease of explanation). The following is an explanation of the code used to obtain the different values in Equation 11:

$$Simpson's\ Rule\text{: } \int_{t_1}^{t_2} v(t)dt \approx \frac{3}{\Delta}\left(v_0 + 4\sum\nolimits_{j=1,3,5,7,\ldots n-1} v_j + 2\sum\nolimits_{j=2,4,6,8,\ldots n-2} v_j + v_n\right) + O\Delta^4$$

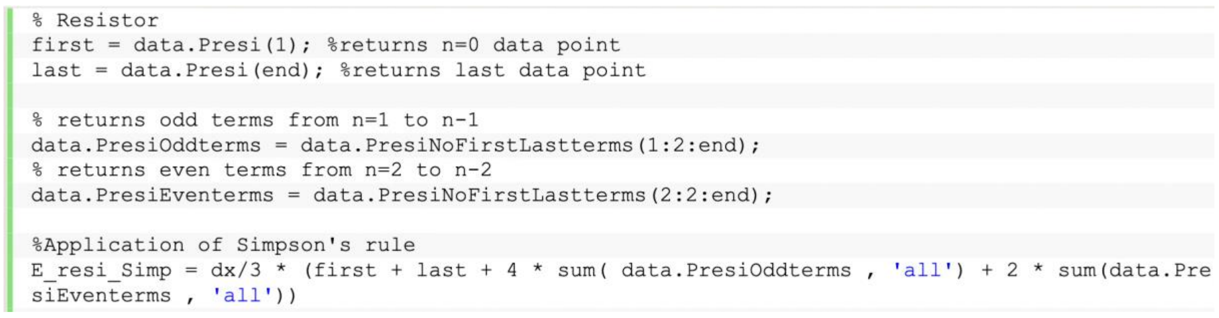
Figure 6.5 shows the code to obtain the width of the stripe “Δ” in equation 11. "dx" is “Δ”. This width is equal to the difference between the maximum and minimum data points divided by the number of intervals. When calculating the first 10 ms of the result, the maximum data point should be 1/6 since the total time is 60 ms. When calculating the first 60 ms of the result, the maximum data point is all.



```
a = min(data.time); %first time point
b = max(data.time);      %last time point (When t = 60ms)
b = 1/6*max(data.time); %last time point (When t = 10ms)
n = N - 1; % intervals
dx = (b-a)/n; % width of each strip
```

Figure 6.5: Code for obtaining the width of stripe “Δ” at different times

The different remaining parts of Equation 11 applied to the Matlab script are shown in Figure 6.6. “first” is “V_0” and it is the first data point. “last” is “V_n” and it is the last data point. “data.PresiOddterms” is “V_j” in “$\sum_{j=1,3,5,7,\ldots n-1} v_j$” and it is the odd terms data point. “data.PresiEventerms” is “V_j” in “$\sum_{j=2,4,6,8,\ldots n-2} v_j$” and it is the even terms data point. All parts of Equation 11 were converted into Matlab script form. Figure 6.6 shows only the code for calculating the resistance energy using Simpson's rule, for calculating the capacitance energy only the variable names need to be changed, the principle and method are the same.



```
% Resistor
first = data.Presi(1); %returns n=0 data point
last = data.Presi(end); %returns last data point

% returns odd terms from n=1 to n-1
data.PresiOddterms = data.PresiNoFirstLastterms(1:2:end);
% returns even terms from n=2 to n-2
data.PresiEventerms = data.PresiNoFirstLastterms(2:2:end);

%Application of Simpson's rule
E_resi_Simp = dx/3 * (first + last + 4 * sum( data.PresiOddterms , 'all') + 2 * sum(data.Pre
siEventerms , 'all'))
```

Figure 6.6: Code for application of Simpson’s rule for calculating the energy of resistor

Based on the above code and the adjustment methods for the different cases, the results of the energy consumed by the resistor E_R and the energy stored by the capacitor E_C during the first 10ms and 60ms of the simulation by using Simpson's rule of 10, 100, 1000 and 10,000 intervals are recorded in Table 8 and Table 9.

Table 8: Capacitor energy during the first 10ms and 60ms of the simulation by using Simpson's rule of 10, 100, 1000 and 10,000 intervals

Energy stored in the capacitor (E_C) [J]		Number of intervals used			
		10	**100**	**1000**	**10000**
Simulation run time	**10ms**	21.5773	21.5775	21.5775	21.5775
	60ms	66.3034	66.5926	66.5926	66.5926

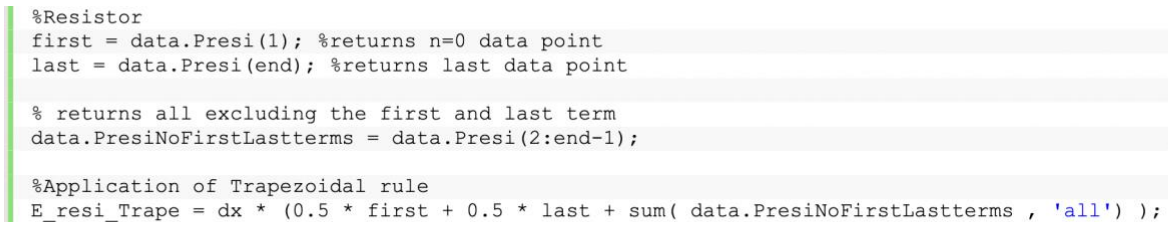
Table 9: Resistor energy during the first 10ms and 60ms of the simulation by using Simpson's rule of 10, 100, 1000 and 10,000 intervals

Energy dissipated in the resistor (E_R) [J]		Number of intervals used			
		10	**100**	**1000**	**10000**
Simulation run time	**10ms**	51.0024	54.3772	54.7147	54.7484
	60ms	45.3330	65.2486	67.2735	67.4760

Integrate with Trapezoidal rule:

Trapezoidal rule follows much the same steps as those using Simpson's rule, obtaining equidistant data points at different intervals, obtaining the stripe width "Δ" from the calculation, and adjusting code for the time required. The main difference is that the two rules apply different mathematical formulas and so require different data points to be obtained. For ease of explanation, the formula for the Trapezoidal rule is again shown below. Figure 6.7 shows the code for calculating the energy of resistor using Trapezoidal rule, "first" is "V_0" and it is the first data point. "last" is "V_n" and it is the last data point. "data.PresiNoFirstLastterms" is "V_j" in "$\sum_{j=1}^{N-1} V_j$" and it is a data point except for the first and last. Figure 6.7 shows only the code for calculating the resistance energy using Trapezoidal rule, for calculating the capacitance energy only the variable names need to be changed, the principle and method are the same.

$$Trapezoidal\ Rule: \int_0^T v(t)dt \approx \Delta\left(0.5v_0 + \sum_{j=1}^{N-1} V_j + 0.5v_n\right)$$



```
%Resistor
first = data.Presi(1); %returns n=0 data point
last = data.Presi(end); %returns last data point

% returns all excluding the first and last term
data.PresiNoFirstLastterms = data.Presi(2:end-1);

%Application of Trapezoidal rule
E_resi_Trape = dx * (0.5 * first + 0.5 * last + sum( data.PresiNoFirstLastterms , 'all') );
```

Figure 6.7: Code for application of Trapezoidal rule for calculating the energy of resistor

Based on the above code and the adjustment methods for the different cases, the results of the energy consumed by the resistor E$_R$ and the energy stored by the capacitor E$_C$ during the first 10ms and 60ms of the simulation by using Trapezoidal rule of 10, 100, 1000 and 10,000 intervals are recorded in Table 10 and Table 11.

Table 10: Capacitor energy during the first 10ms and 60ms of the simulation by using Trapezoidal rule of 10, 100, 1000 and 10,000 intervals

Energy stored in the capacitor (E_C) [J]		Number of intervals used			
		10	**100**	**1000**	**10000**
Simulation run time	**10ms**	21.4950	21.5767	21.5775	21.5777
	60ms	63.8418	66.5643	66.5924	66.5926

Table 11: Resistor energy during the first 10ms and 60ms of the simulation by using Trapezoidal rule of 10, 100, 1000 and 10,000 intervals

Energy dissipated in the resistor (E_R) [J]		Number of intervals used			
		10	**100**	**1000**	**10000**
Simulation run time	**10ms**	49.2539	54.1909	54.6960	54.7466
	60ms	39.2813	64.1798	67.1616	67.4648

7. Error in numerical Integration Techniques (20%)

Equations 12 and 13 are used to calculate the errors in obtaining capacitor and resistor energy using Simpson's rule. Equations 14 and 15 are used to calculate the errors in obtaining capacitor and resistor energy using the Trapezoidal rule. Based on the exact calculated values obtained in Table 1 and the values obtained with the Matlab script in Tables 8, 9, 10, 11, the errors at different numbers of intervals and times can be calculated. Here only the energy error calculation process is shown for the first 60 ms at an interval number of 10. All error results are presented in Tables 12 and 13.

$$Simpson\_Error_{E_R} = \frac{Simpson's\,rule_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} \qquad \textbf{Eqn.12}$$

$$Simpson\_Error_{E_c} = \frac{Simpson's\,rule_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} \qquad \textbf{Eqn.13}$$

$$Trapezoidal\_Error_{E_R} = \frac{Simpson's\,rule_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} \qquad \textbf{Eqn.14}$$

$$Trapezoidal\_Error_{E_c} = \frac{Simpson's\,rule_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} \qquad \textbf{Eqn.15}$$

$$Simpson\_Error_{E_R} = \frac{Simpson's\,rule_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} = \frac{45.3330 - 67.4969}{67.4969} \times 100\% = -32.837\%$$

$$Simpson\_Error_{E_c} = \frac{Simpson's\,rule_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} = \frac{66.3034 - 66.5934}{66.5934} \times 100\% = -0.435\%$$

$$Trapezoidal\_Error_{E_R} = \frac{Trapezoidal\,rule_{E_R} - Calculated_{E_R}}{Calculated_{E_R}} = \frac{39.2813 - 67.4969}{67.4969} \times 100\% = -41.803\%$$

$$Trapezoidal\_Error_{E_c} = \frac{Trapezoidal\,rule_{E_c} - Calculated_{E_c}}{Calculated_{E_c}} = \frac{63.8418 - 66.5934}{66.5934} \times 100\% = -4.132\%$$

Table 12: % error in estimation of capacitor energy after 10ms and 60ms

% error in estimation of capacitor energy		Number of intervals used							
		Simpson's				Trapezoidal			
		10	**100**	**1000**	**10,000**	**10**	**100**	**1000**	**10,000**
Run time	**10 ms**	-0.005	-0.004	-0.004	-0.004	-0.386	-0.007	-0.004	-0.003
	60 ms	-0.435	-0.001	-0.001	-0.001	-4.132	-0.044	-0.001	-0.001

Table 13: % error in estimation of resistor energy after 10ms and 60ms

% error in estimation of resistor energy		Number of intervals used							
		Simpson's				Trapezoidal			
		10	**100**	**1000**	**10,000**	**10**	**100**	**1000**	**10,000**
Run time	**10 ms**	-6.846	-0.682	-0.066	-0.004	-10.040	-1.023	-0.100	-0.008
	60 ms	-32.837	-3.331	-0.331	-0.031	-41.803	-4.914	-0.497	-0.048

Table 14: execution time in estimation of capacitor energy after 10ms and 60ms

MATLAB script execution time (s)		Number of intervals used							
		Simpson's				Trapezoidal			
		10	**100**	**1000**	**10,000**	**10**	**100**	**1000**	**10,000**
Run time	**10 ms**	0.007421	0.007832	0.008885	0.010848	0.006573	0.007213	0.008295	0.008338
	60 ms	0.008338	0.008515	0.09977	0.011304	0.007158	0.007395	0.008298	0.009419

Table 15: execution time in estimation of resistor energy after 10ms and 60ms

MATLAB script execution time (s)		Number of intervals used							
		Simpson's				Trapezoidal			
		10	**100**	**1000**	**10,000**	**10**	**100**	**1000**	**10,000**
Run time	**10 ms**	0.006767	0.007561	0.008711	0.01684	0.006382	0.007281	0.007801	0.009256
	60 ms	0.007381	0.007912	0.010663	0.01202	0.007214	0.00774	0.008056	0.011919

Error VS Intervals for Trapezoidal rule and for Simpson's rule:

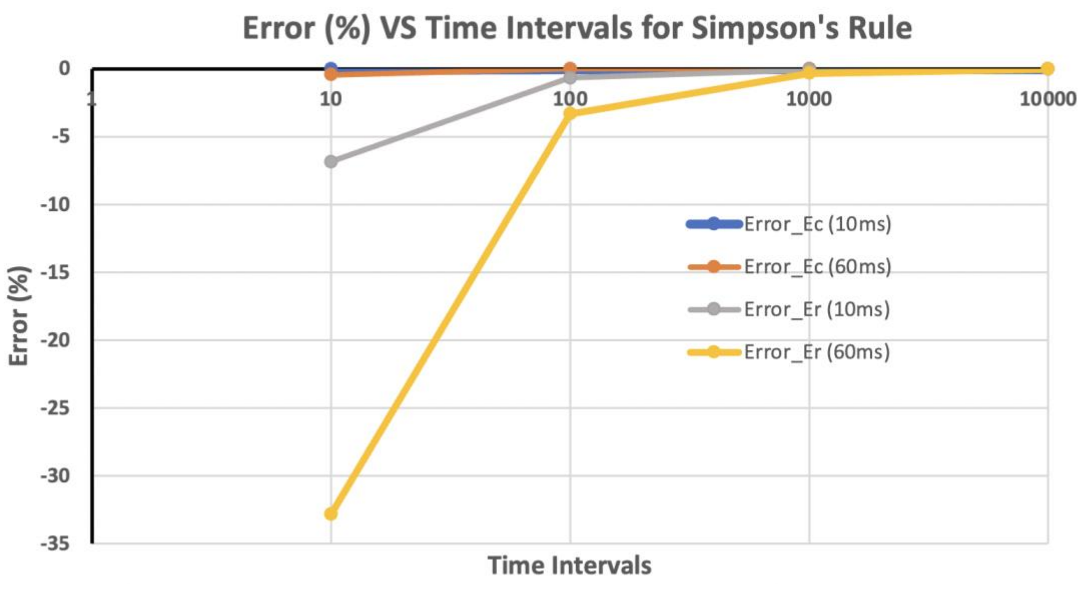


Figure 7.1: Error vs intervals for Simpson's rule

Figure 7.1 shows the energy errors calculated using Simpson's rule vs time intervals based on the results in Tables 12 and 13. There are four types of data in the figure, the blue and orange lines are the errors in capacitor energy calculated by Simpson's rule for simulation times of 10 ms and 60 ms respectively. The grey and yellow lines are the error in the resistor energy calculated by Simpson's rule for simulation times of 10 ms and 60ms respectively. The horizontal coordinate of the figure is the time interval between each data point and the vertical coordinate is the error between the results calculated by Matlab using Simpson's rule and the results calculated in Task 3. Based on the results in the above figure it can be observed that:

1. When the maximum time step is kept at 100 ns, the larger the number of time intervals the smaller the error. At a time of 10ms, the resistance energy error is -6.846% at an interval of 10, and close to 0 at an interval of 10000. This is because when the time is the same, the increase in the number of intervals allows more data points to be processed for a more accurate calculation resulting in a smaller error.
2. When the number of intervals is the same, the error in the first 10 ms of the same component is less than the first 60 ms. When the number of intervals is 10, the resistor energy error is -6.846% for the first 10 ms and -32.837% for 60 ms. This is because when calculations are made using intervals that cause inaccurate data, the longer the time, the more error is accumulated. Since the method is inherently inaccurate, increasing the time can still accumulate some errors even when the number of intervals is large.
3. For the same time and number of intervals, the error of resistor is greater than the error of capacitor. At an interval of 10 and the first 10ms, the capacitor error is -0.005%, and the resistor error is -6.846%. According to the energy equation for capacitance and resistance (They are

equations 3 and 5 from Task 1. For ease of explanation, they are presented again here), it can be found that the resistance energy changes more than the capacitance in the same time interval. When the number of intervals is relatively small, a larger difference between the two data point values means that a larger error can be caused.

$$P_C(t) = 11250(e^{-83.333t} - e^{-166.667t})$$

$$P_R(t) = 11250e^{-166.667t}$$

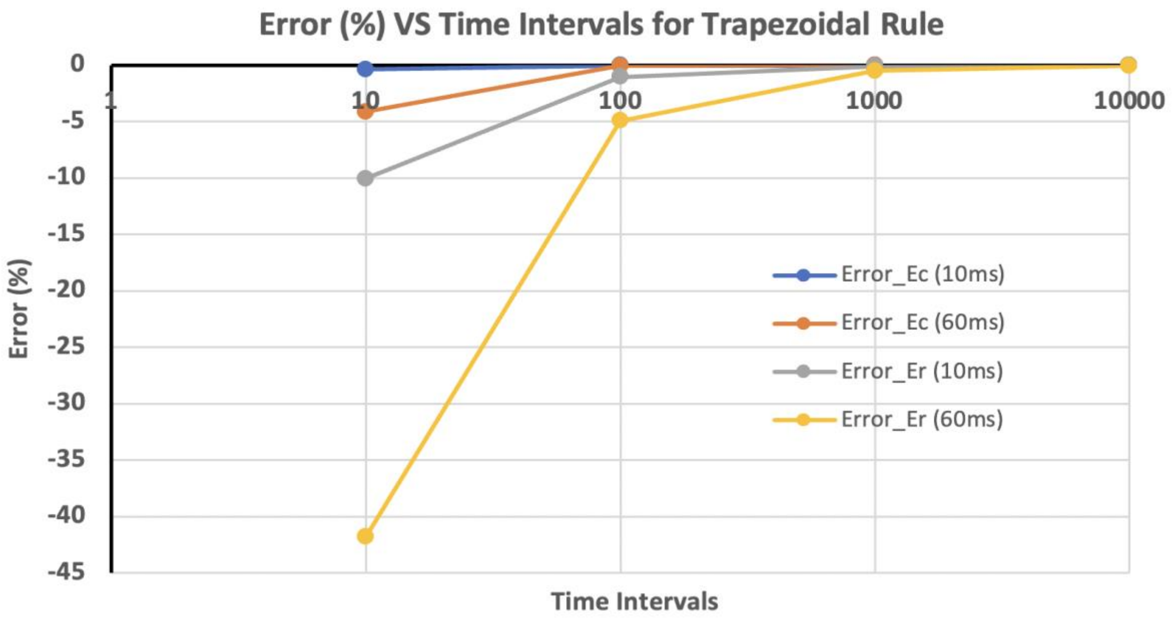


Figure 7.2: Error vs intervals for Trapezoidal rule

Figure 7.2 shows the energy errors calculated using the Trapezoidal rule vs time intervals based on the results in Tables 12 and 13. There are four types of data in the figure, the blue and orange lines are the errors in capacitor energy calculated by Trapezoidal rule for simulation times of 10 ms and 60 ms respectively. The grey and yellow lines are the error in the resistor energy calculated by Trapezoidal rule for simulation times of 10 ms and 60ms respectively. The horizontal coordinate of the figure is the time interval between each data point and the vertical coordinate is the error between the results calculated by Matlab using Trapezoidal rule and the results calculated in Task 3. Figure 7.2 follows a similar trend to Figure 7.1, The reasons for the below points of the phenomenon are the same as those explained in Figure 7.1 part:

1. The larger the number of intervals, the smaller the error for the same component at the same time. At a time of 10ms, the resistance energy error is -10.040% at an interval of 10 and -0.008 at an interval of 10,000.
2. When the number of intervals is the same, the error of the same element in the first 10 ms is less than in the first 60 ms. When the number of intervals is 10, the resistance energy error is -10.040% in the first 10 ms and -41.803% at 60 ms.

3. For the same time and number of intervals, the resistor error is greater than the capacitor error. At an interval of 10 and for the first 10 ms, the error in capacitance is -0.386 while the error in resistance is -10.040%.

Error VS Time for Trapezoidal rule and for Simpson's rule:

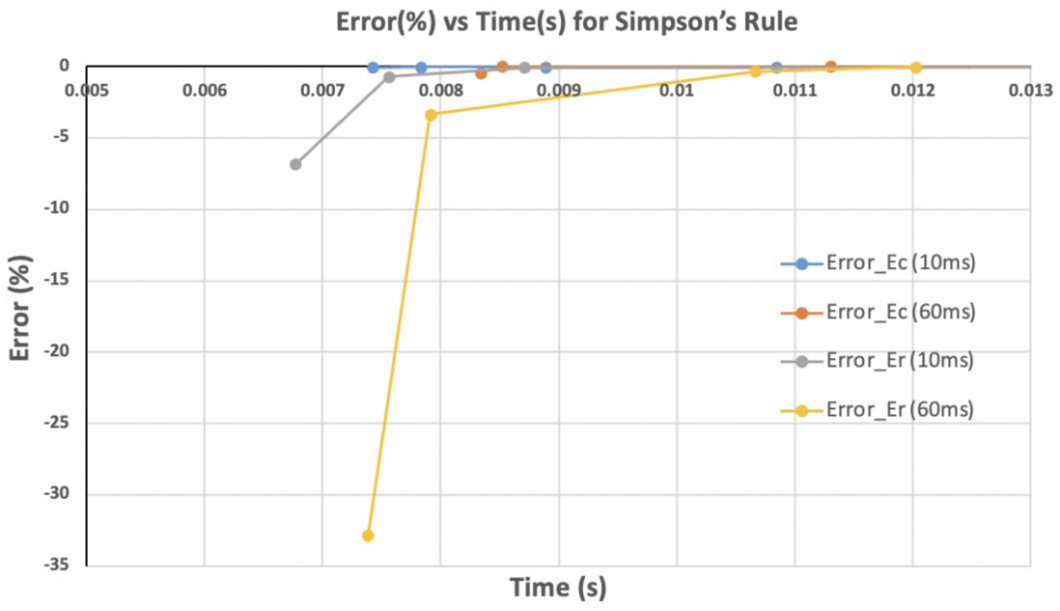


Figure 7.3: Error vs time for Simpson's rule

Figure 7.3 shows the error in energy calculated using Simpson's rule vs execution time based on the results in Tables 14 and 15. There are four types of data in the figure, the blue and orange lines are the errors in capacitor energy calculated using Simpson's rule at simulation times of 10 ms and 60 ms, respectively. The grey and yellow lines are the errors in the resistor energy calculated by Simpson's rule at simulation times of 10 ms and 60 ms respectively. The horizontal coordinates of the figure are the execution time of running the program once and the vertical coordinates are the error between the results calculated by Matlab using Simpson's rule and the results calculated in Task 3. Based on the results in the above figure it can be observed that:

1. The longer the time taken to execute the code the smaller the error. For the first 60 ms, the resistance has an error of -32.837% at an execution time of 0.007381 s and -0.031% at an execution time of 0.01202 s (Yellow Line). This is because the longer execution time means that there are more data points to analyse at this point and the calculated energy is more accurate and its error can be smaller.

2. The same component has a larger error in the first 60ms than in the first 10ms. The resistor in the figure has an error value of -6.846% at the first point in 10ms (Grey Line) compared to -32.837% at the first point in 60ms (Yellow Line). This is because there is some inaccuracy in calculating the

energy using Simpson's rule, which increases the execution time for calculating the error in the first 60ms compared to calculating the error in the first 10ms. This also means that more errors are calculated.

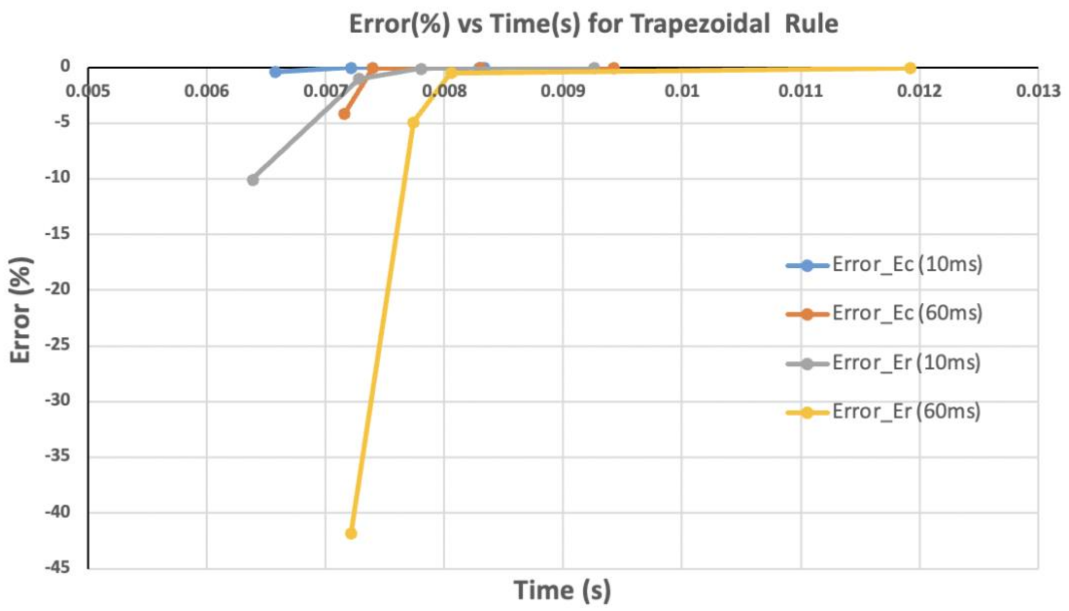


Figure 7.4: Error vs time for Trapezoidal rule

Figure 7.4 shows the error in energy calculated using the trapezoidal rule versus execution time based on the results in Tables 14 and 15. There are four types of data in the figure, the blue and orange lines are the errors in capacitor energy calculated using the trapezoidal rule at simulation times of 10 ms and 60 ms, respectively. The grey and yellow lines are the errors in the resistor energy calculated using the trapezoidal rule at simulation times of 10 ms and 60 ms respectively. The horizontal coordinates of the figure are the execution time for one run of the code and the vertical coordinates are the error between the results calculated by Matlab using the trapezoidal rule and the results calculated by Task 3. Figure 7.4 follows a similar trend to Figure 7.3. The reasons for the above points of the phenomenon are the same as those explained in Figure 7.3 part.

1. The longer the time taken to execute the code the smaller the error. For the first 60 ms, the resistance has an error of -41.803% at an execution time of 0.007214 s and -0.048% at an execution time of 0.011919 s (Yellow Line).

2. The same component has a larger error in the first 60ms than in the first 10ms. The resistor in the figure has an error value of -10.040% at the first point in 10ms (Grey Line) compared to -41.803% at the first point in 60ms (Yellow Line).

Accuracy of the numerical integration:

From the results of the above figures, it can be seen that the error calculated using Simpson's rule is less than the error calculated using the trapezoidal rule. For an interval number of 10 and the first 10ms, the resistance energy is calculated with an error of -6.846% by Simpson's rule compared to -10.040% by the trapezoidal rule. This indicates that Simpson's rule is more accurate and that Simpson's rule approximates the definite integral of the function more accurately than the trapezoidal rule. The reason for this is that:

1. Trapezoidal rule approximates the area under the curve by approximating the curve as a trapezoid and calculating its area. Simpson's rule approximates the area under the curve as a series of parabolas, which provides a more accurate approximation.
2. The error term in the trapezoidal rule is proportional to the second order derivative of the function being integrated, whereas the error term in Simpson's rule is proportional to the fourth order derivative of the function being integrated. Equations 16 and 17 are the formulas for calculating the errors of the trapezoidal rule and Simpson's rule. ("a" and "b" is the endpoint of the closed interval [a, b], "n" is the number of intervals) [2] Since the fourth order derivative is usually smaller than the second order derivative, Simpson's rule provides a more accurate approximation to the integral.
3. On the other hand, each method requires a different number of function evaluations. Simpson's rule requires three function evaluations per interval, while the trapezoidal rule requires two function evaluations per interval. While this may appear to be a drawback of Simpson's rule, it allows Simpson's rule to capture more of the curvature of the function being integrated, which can lead to a more accurate approximation. It is worth noting that, because of this, Simpson's rule requires more computational effort to calculate than the trapezoidal rule and its execution time is longer. From Figure 7.3 and Figure 7.4 it can be seen that at an interval number of 10,000 and the first 10ms, the execution time for calculating the resistance energy using Simpson's rule is 0.01684 s, whereas for the trapezoidal rule, it is 0.009256 s.

$$Error_{Trapezoidal} = \frac{(b-a)^3}{12n^2}[\max|f''(x)|] \qquad \textbf{Eqn.16}$$

$$Error_{Simpson} = \frac{(b-a)^5}{180n^4}[\max|f^4(x)|] \qquad \textbf{Eqn.17}$$

8. Reference

[1] M. Engelhardt, "SPICE differentiation," LT Journal of Analog Innovation, vol. 24, no. 4, pp. 10-16, 2015.

[2] "Errors in the Trapezoidal Rule and Simpson's Rule," Statistics How To. [Online]. Available: https://www.statisticshowto.com/errors-in-the-trapezoidal-rule-and-simpsons-rule/. [Accessed: 03 21, 2023].

Appendix A: Matlab code to create equally spaced data points

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: import data %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data=tdfread('dataset_20320941.txt');
% tdfread can read data from tab-delimited text files with
% .txt, .dat, or .csv file extensions.
% Select a file that has variable names in the first row and
% values separated by tabs in the remaining rows. tdfread
% creates a variable in the workspace for each column of the
% file, and names each variable according to its first row
% value.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: create equally spaced data points for x-axis (time)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N=11; % It needs to be changed depends on different intervals
% N=11(10 intervals); N=101(100 intervals); N=1001(1000 intervals); N=10001(10000 intervals);
data.timei = linspace(min(data.time),max(data.time),N) ;
% x = linspace(x1,x2,n) generates n points.
% The spacing between the points is (x2-x1)/(n-1).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: create equally spaced data points for y-axis. This
% is achieved by interpolating for specific x- values (time).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data.Presi = interp1(data.time,data.Pres,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
data.Pcapi = interp1(data.time,data.Pcap,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
```

Appendix B: Matlab code for Simpson's rule and Trapezoidal rule

Simpson's rule:

```
% "tic" starts the clock to measure how long it takes
% for the script to run
tic;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: import data %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data=tdfread('dataset_20320941.txt');
% tdfread can read data from tab-delimited text files with
% .txt, .dat, or .csv file extensions.
% Select a file that has variable names in the first row and
% values separated by tabs in the remaining rows. tdfread
% creates a variable in the workspace for each column of the
% file, and names each variable according to its first row
% value.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: create equally spaced data points for x-axis (time)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = 11 ; %number of data points
% N needs to be odd so that an even number of equally spaced
% strips are created when Simpson's rule is applied
% N=11(10 intervals); N=101(100 intervals); N=1001(1000 intervals); N=10001(10000 intervals);
data.timei =linspace(min(data.time),max(data.time),N) ;
%data.timei = linspace(min(data.time),max(1/6*data.time),N) ;
% During the first 60ms, linspace(min(data.time),max(data.time),N) is used ;
% During the first 10ms, linspace(min(data.time),max(1/6*data.time),N) is used ;
% x = linspace(x1,x2,n) generates n points.
% The spacing between the points is (x2-x1)/(n-1).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: create equally spaced data points for y-axis. This
% is achieved by interpolating for specific x- values (time).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data.Presi = interp1(data.time,data.Pres,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
```

```
data.Pcapi = interp1(data.time,data.Pcap,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 4: Integrate with Simpson's rule %
% from a to b using n intervals (n+1 points) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
a = min(data.time); %first time point
b = max(data.time); %last time point
% b = 1/6*max(data.time);
% During the first 60ms: b=max(data.time) is used;
% During the first 10ms: b=1/6*max(data.time) is used;
% Intervals need to be an even number,
% so N must be an odd number
n = N - 1; % intervals
dx = (b-a)/n; % width of each strip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
Resistor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
first = data.Presi(1); %returns n=0 data point
last = data.Presi(end); %returns last data point
% returns odd terms from n=1 to n-1
data.PresiOddterms = data.PresiNoFirstLastterms(1:2:end);
% returns even terms from n=2 to n-2
data.PresiEventerms = data.PresiNoFirstLastterms(2:2:end);
%Application of Simpson's rule
E_resi_Simp = dx/3 * (first + last + 4 * sum( data.PresiOddterms , 'all') + 2 *
sum( data.PresiEventerms , 'all'))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
Capacitor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
first = data.Pcapi(1); %returns n=0 data point
last = data.Pcapi(end); %returns last data point
% returns odd terms from n=1 to n-1
data.PcapiOddterms = data.PcapiNoFirstLastterms(1:2:end);
% returns even terms from n=2 to n-2
data.PcapiEventerms = data.PcapiNoFirstLastterms(2:2:end);
%Application of Simpson's rule
E_capi_Simp = dx/3 * (first + last + 4 * sum( data.PcapiOddterms , 'all') + 2 * sum(
data.PcapiEventerms , 'all'))
% "toc" stops the clock and writes how long it
% took for the script to run
toc;
```

Trapezoidal rule:

```
% "tic" starts the clock to measure how long it takes
% for the script to run
tic;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: import data %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data=tdfread('dataset_20320941.txt');
% tdfread can read data from tab-delimited text files with
% .txt, .dat, or .csv file extensions.
% Select a file that has variable names in the first row and
% values separated by tabs in the remaining rows. tdfread
% creates a variable in the workspace for each column of the
% file, and names each variable according to its first row
% value.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: create equally spaced data points for x-axis (time)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = 11 ; %number of data points
% N needs to be odd so that an even number of equally spaced
% strips are created when Simpson's rule is applied
% N=11(10 intervals); N=101(100 intervals); N=1001(1000 intervals); N=10001(10000 intervals);
```

```
data.timei =linspace(min(data.time),max(data.time),N) ;
%data.timei = linspace(min(data.time),max(1/6*data.time),N) ;

% During the first 60ms, linspace(min(data.time),max(data.time),N) is used ;
% During the first 10ms, linspace(min(data.time),max(1/6*data.time),N) is used ;
% x = linspace(x1,x2,n) generates n points.
% The spacing between the points is (x2-x1)/(n-1).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: create equally spaced data points for y-axis. This
% is achieved by interpolating for specific x- values (time).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data.Presi = interp1(data.time,data.Pres,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
data.Pcapi = interp1(data.time,data.Pcap,data.timei) ;
% returns interpolated values at specific query points using
% linear interpolation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 4: Integrate with Simpson's rule %
% from a to b using n intervals (n+1 points) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
a = min(data.time); %first time point
b = max(data.time); %last time point
% b = 1/6*max(data.time);
% During the first 60ms: b=max(data.time) is used;
% During the first 10ms: b=1/6*max(data.time) is used;
% Intervals need to be an even number,
% so N must be an odd number
n = N - 1; % intervals
dx = (b-a)/n; % width of each strip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
Resistor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
```

```
first = data.Presi(1); %returns n=0 data point
last = data.Presi(end); %returns last data point
% returns all excluding the first and last term

% returns all excluding the first and last term
data.PresiNoFirstLastterms = data.Presi(2:end-1);
%Application of Trapezoidal rule
E_resi_Trape = dx * (0.5 * first + 0.5 * last + sum( data.PresiNoFirstLastterms ,
'al') ); %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
%%%%%%%%%%%%
Capacitor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% %%%%%%%%%%%%%
first = data.Pcapi(1); %returns n=0 data point
last = data.Pcapi(end); %returns last data point
% returns all excluding the first and last term
data.PcapiNoFirstLastterms = data.Pcapi(2:end-1);
 %Application of Trapezoidal rule
E_capi_Trape = dx * (0.5 * first + 0.5 * last + sum( data.PcapiNoFirstLastterms , 'all') );
% "toc" stops the clock and writes how long it
% took for the script to run
toc;
```