EE2-08C Numerical Analysis

Group 9

Contents

1 Exercise 1 - RL Circuit

1.1 Background

This simple RL circuit (1) forms a high pass filter circuit which takes an input signal Vin and only allows the high frequency components to pass. Despite the inductor being less convenient than a capacitor, this RL circuit can approach the model of a DC motor. In this specific case, we want to model a DC motor with inertia 250 μsNm/s^2 and $T_{max} = 50mNm/A$. To do so, according to KCL, we know that

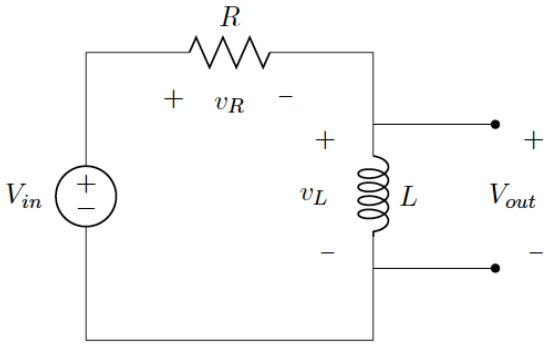

Figure 1: RL Circuit

$$Vin(t) = v_L(t) + v_R(t) \tag{1}$$

and therefore:

$$V_{in}(t) = L\frac{d}{dt}i_L(t) + Ri_L(t) \tag{2}$$

$i_L(t)$ is the state that we will solve following the three different second-order Runge Kutta methods.
All methods follow the same pattern:

$$x_{i+1} = x_i + h \tag{3}$$

h being the step size and

$$y_{i+1} = y_i + h(ak_1 + bk_2) \tag{4}$$

where:

$$k_1 = f(x_i, y_i) \tag{5}$$

$$k_2 = f(x_i + ph, y_i + qk_1h) \tag{6}$$

and the values of a, b, p and q depend on which method we are using:

Parameter	Heun's	Midpoint	Ralston
a	1/2	0	1/4
b	1/2	1	3/4
p	1	1/2	2/3
q	1	1/2	2/3

Each method will approximate the exact solution according to these parameters.
After doing so, the output value will be obtained as follows:

$$V_{out} = V_{in}(t) - Ri_L(t) \tag{7}$$

1.2 Second-Order Runge Kutta Scripts

The code for Heuns.m to solve $i_L(t)$ will be as follows:

```
function [t,y] = heun(func,t0,y0,tf,h) % y = i_L(t), func = equation (2),
% t0 = initial time, tf = final time, h = step size
N = round((tf-t0)/h);   % N = nr. of steps
arrt = zeros(1,N);      % arrt = array of time
arry = zeros(1,N);      % arry = array of y
arrt(1) = t0;           % initial conditions
arry(1) = y0;
for i = 1 : N-1         % evaluate arry and arrt for each step
    k1 = feval(func, arrt(i), arry(i));
    k2 = feval(func, arrt(i) + h, arry(i) + k1*h);
    arry(i+1) = arry(i) + h*0.5*(k1 + k2);
    arrt(i+1) = arrt(i) + h;
end
t= arrt;                % return values
y = arry;
end
```

Similarly, the code for Midpoint.m will be:

```
function [t,y] = midpoint(func, t0, y0, tf, h) % y = i_L(t), func = equation (2),
% t0 = initial time, tf = final time, h = step size
N = round((tf - t0)/h); % N = nr. of steps
arrt = zeros(1,N);      % arrt = array of time
arry = zeros(1,N);      % arry = array of y
arrt(1) = t0;           % initial conditions
arry(1) = y0;
for i = 1:N-1           % evaluate arry and arrt for each step
    k1 = feval(func,arrt(i),arry(i));
    k2 = feval(func, arrt(i) + 0.5*h, arry(i) + 0.5*h*k1);
    arry(i+1) = arry(i) + h*((1/4)*k1 + (3/4)*k2);
    arrt(i+1) = arrt(i) + h;
end
t = arrt; y = arry;     % return values
end
```

Lastly, Ralston.m is similar:

```
function [t, y] = ralston(func,t0,y0,tf,h) % y = i_L(t), func = equation (2),
% t0 = initial time, tf = final time, h = step size
N = round((tf-t0)/h);   % N = nr. of steps
arrt = zeros(1,N);      % arrt = array of time
arry = zeros(1,N);      % arry = array of y
arrt(1) = t0;           % initial conditions
arry(1) = y0;
for i = 1 : N-1         % evaluate arry and arrt for each step
    k1 = feval(func, arrt(i), arry(i));
    k2 = feval(func, arrt(i)+(2/3)*h, arry(i)+(2/3)*h*k1);
    arry(i+1) = arry(i) + (h/4)*(k1 + 3*k2);
    arrt(i+1) = arrt(i) + h;
end
y = arry; t = arrt;     % return values
end
```

As mentioned above, we can appreciate that the three second-order Runge Kutta methods are almost identical with the parameters inside the *for loop* the only difference between them.

1.3 Script to solve the ODE

Given that the values of the components of the RL circuit are:

$$R = 0.5\Omega \quad L = 1.5mH$$

We will find the value of V_{out} according to a script which always follows the same pattern, depending on the values of V_{in} and adjusting the values of h and t_f. The step size h determines the size of the step between successive points/samples. Therefore, the smaller the step size, the more accurate the approximation is to the exact solution of the ODE. t_0 and t_f define the time boundaries for which the ODE is plotted. Using equations 2 and 5 from the background section, a standard script would be as follows:

```
y0 = 0;
t0 = 0; tf = ***;   % to decide
h = ***;            % to decide

Vin =@(t) ***;      % different inputs

R = 0.5; L = 1.5*10^(-3);           % R, L values for this project
func =@(t, y) (Vin(t) - R*y)/L;     % equation 2
[t, y] = ****(func,t0,y0,tf,h);     % heuns, midpoint or ralston
vout = Vin(t)-R*y;                  % equation 5
figure;
plot(t, vout);      % plot V_out against time
title '****'
xlabel 't(s)'
ylabel 'Vout(t)'
```

1.4 Exercise 1 - Graphs

1.4.1 Step signal with amplitude

$Vin = 3.5V$ To study the step signal, we will input a constant voltage of 3.5V throughout the whole t.

```
t0 = 0, tf = 0.02;
h = 0.001;
Vin = @(t) 3.5;
```

In the step signal case, all methods will lead to an equal approximation of the solutions. Figure 2c is applying Heun's method, but if we apply Midpoint (Figure 2b) or Ralston (Figure 2a) with the same step size (Figure 2d) and time conditions, we get similar plots. The resulting output voltage shows a constant exponential decay due to the inductor response as seen in the figures below.

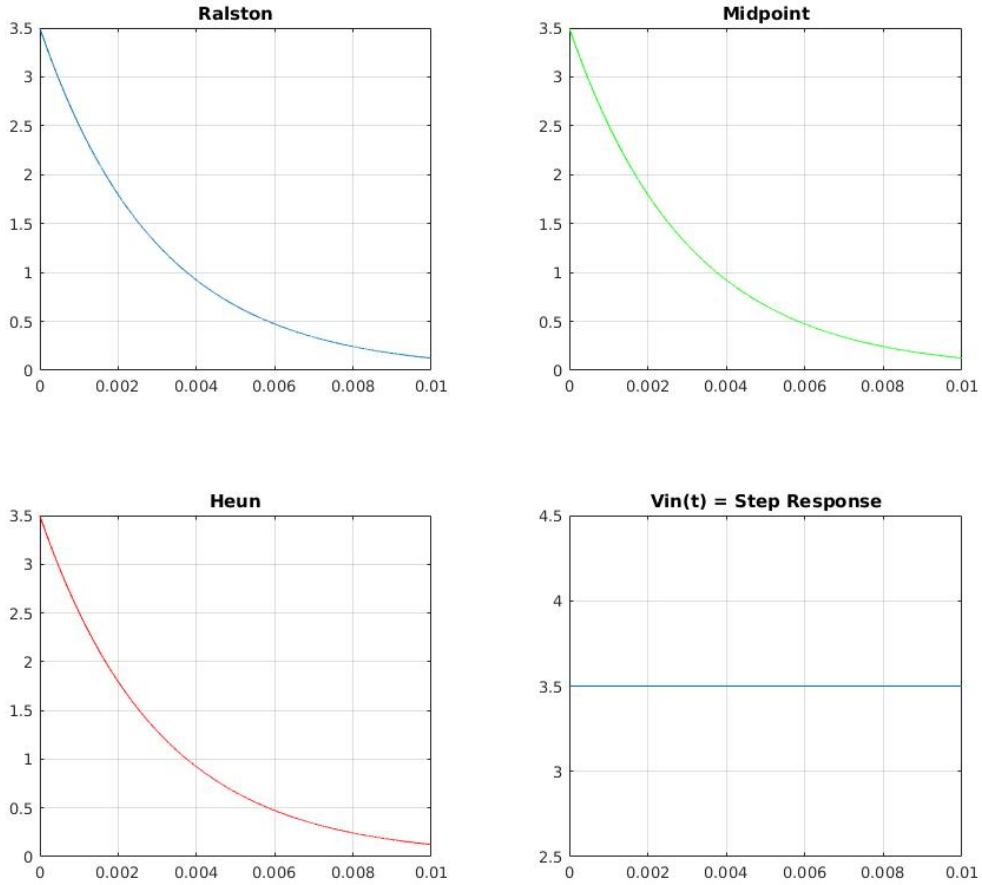

Figure 2: Vin = Step Response

1.4.2 Impulsive signal and decay

In this section, we will examine the 2nd order Runge Kutta Methods using an impulsive signal input given in the equation below:

$$V_{in} = \tilde{V}_{in} exp{-t^2/\tau} \tag{8}$$

```
t0 = 0, tf = 0.05;
h = 0.0005;
vin = 3.5;
z = 150*10^(-12); % z = \tau
t = t0: T : tf;
Vin = @(t) vin*exp(-(t.^2)/z)
```

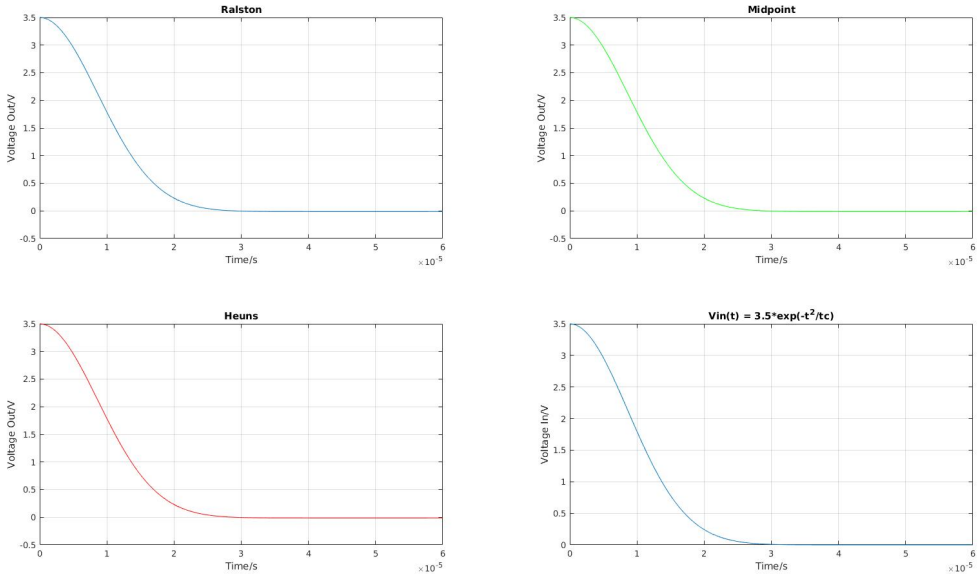

Figure 3: Vin = $3.5exp(\frac{-t^2}{\tau})$

Although small, we will find a minimum point of $V_{out} = -0.01$ around $t = 30\mu s$ which will tend to zero as we increase t.

$$Vin = Vin * exp(-t/\tau)$$

In order to examine the behaviour of the three 2nd order Runge Kutta methods, we will demonstrate their outputs with another impulsive signal given below:
$V_{in} = \tilde{V}_{in} exp{-t/\tau}$

```
t0 = 0; tf = 0.1;
h = 0.001;
vin = 3.5;
z = 150*10^(-6);
t = t0: T : tf;
Vin = @(t) vin*exp(-(t)/z);
```

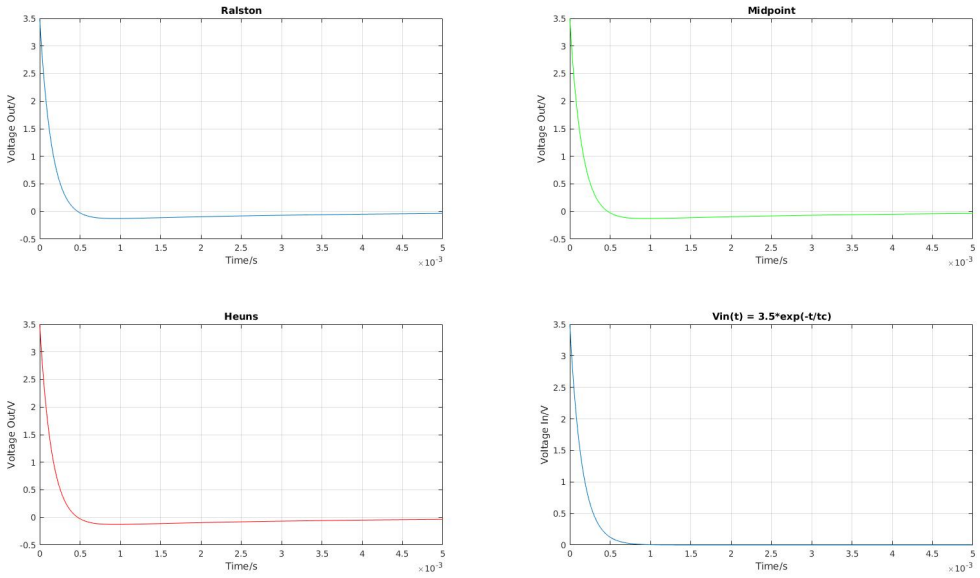

Figure 4: Vin = $3.5exp(\frac{-t}{\tau})$

In this case, the minimum point is easier to see of $V_{out} = -0.1V$ at $t = 0.8ms$.
For both impulsive signals, the difference in the outputs obtained from the three different methods is minimal. This can be seen from the graphs in Figure 4, which plot all three curves obtained for the two impulsive signal input $Vin = Vin * exp(-t/\tau)$.

1.4.3 Sine wave amplitude $\tilde{V}_{in} = 4V$

```
vin = 4;
T = ***10^(-6);
t = t0: T : tf;
f = 1/T;
Vin =@(t) vin*sin(2*pi*f*t);
```

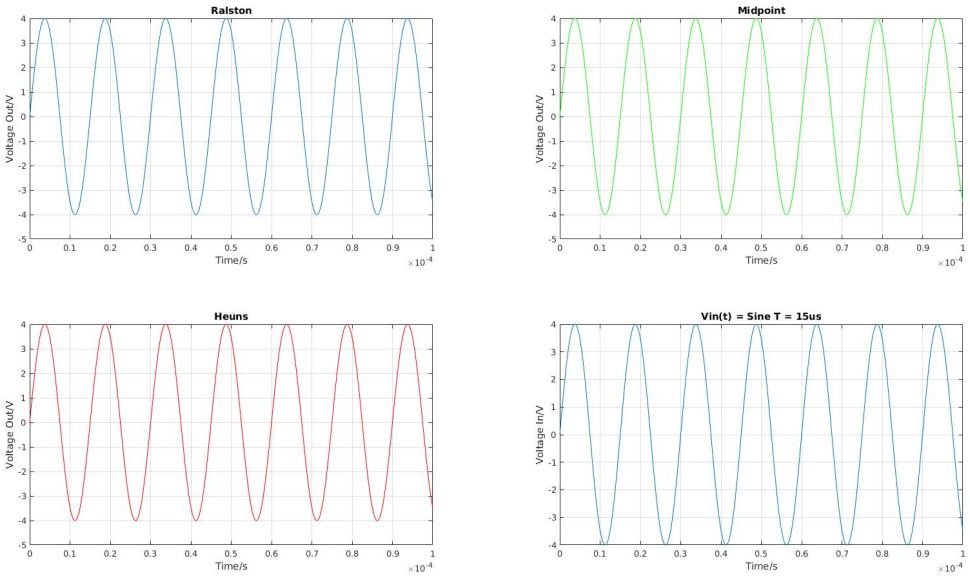

Figure 5: Sine Wave Input $T = 15\mu s$

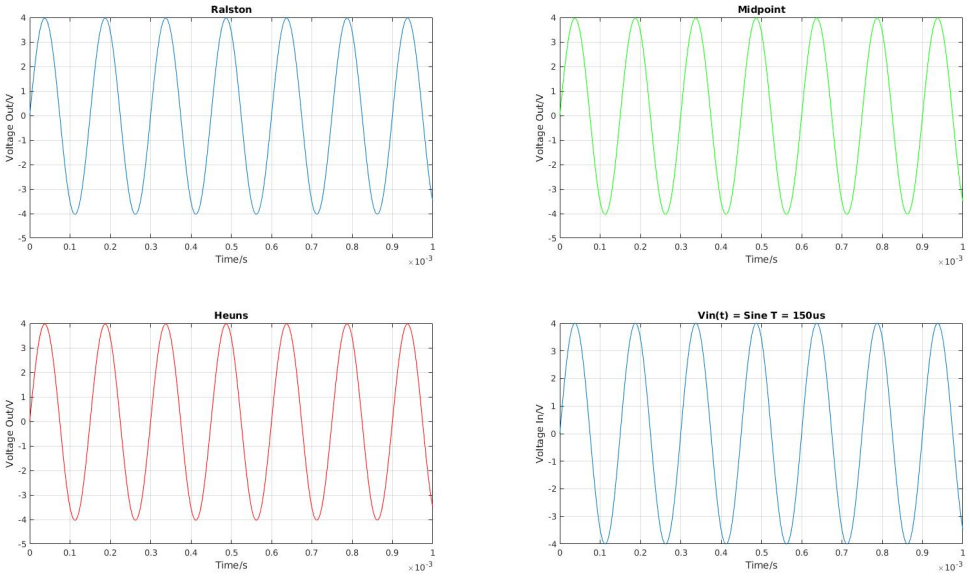

Figure 6: Sine Wave Input $T = 150\mu s$

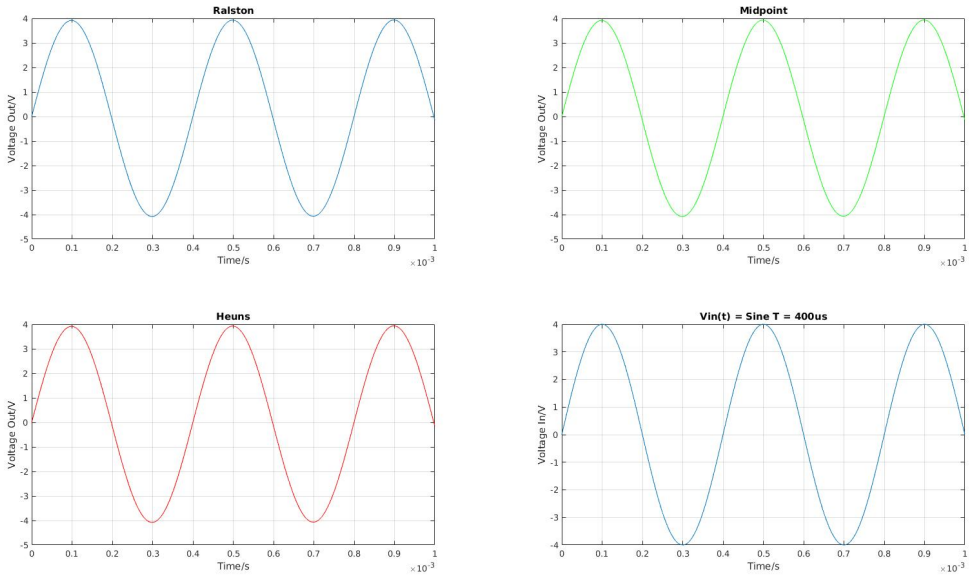

Figure 7: Sine Wave Input $T = 400\mu s$

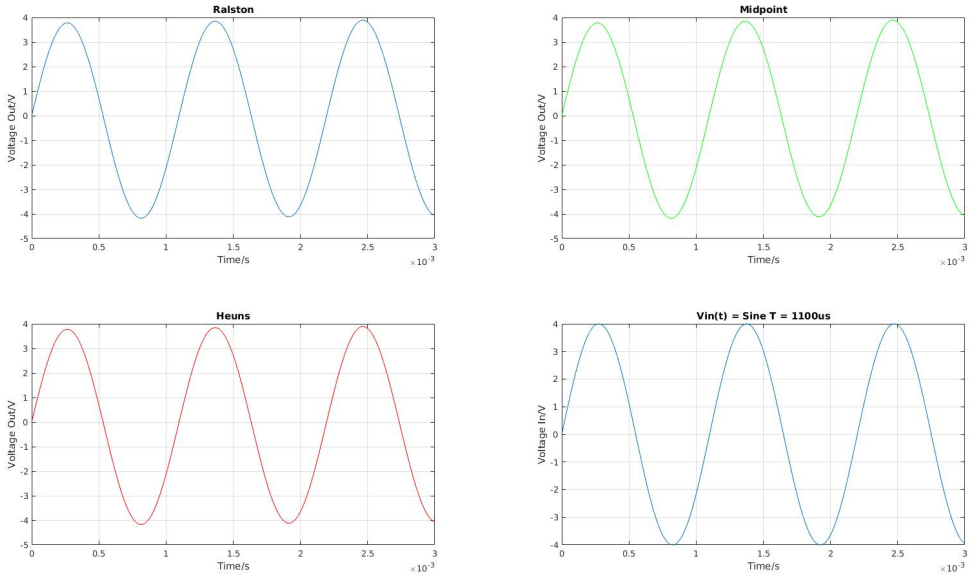

Figure 8: Sine Wave Input $T = 1100\mu s$

For the sine waves, the approximations given by the three methods are very similar and smooth. However, in the error analysis we will see how these methods differ from each other and the small error that the three fo them carry comparing with the exact solution.

1.4.4 Square wave with $\tilde{V}_{in} = 4V$

```
vin = 4;
T = ***10^(-6);
t = t0: T : tf;
f = 1/T;
Vin =@(t) vin*square(2*pi*f*t);
```

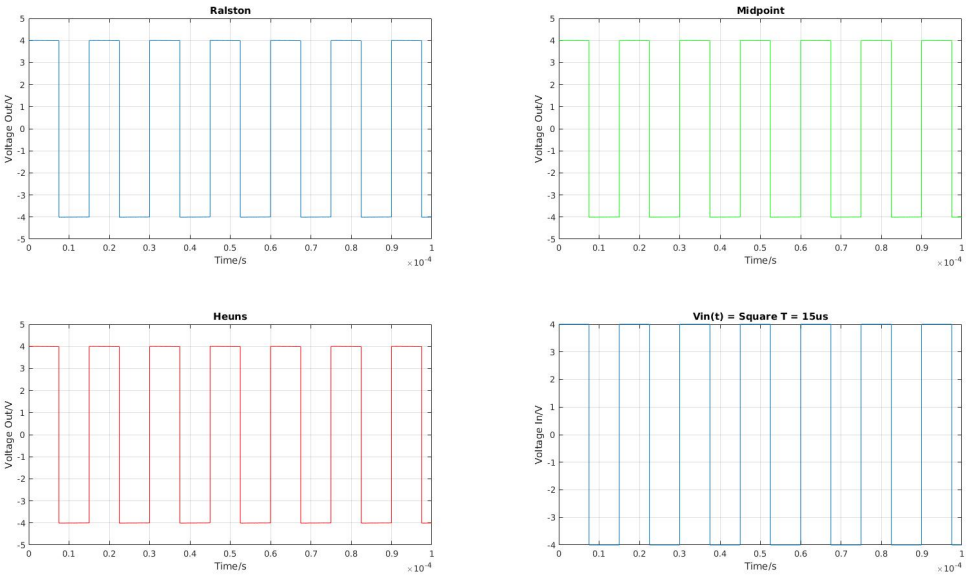

Figure 9: Square Wave Input $T = 15\mu s$

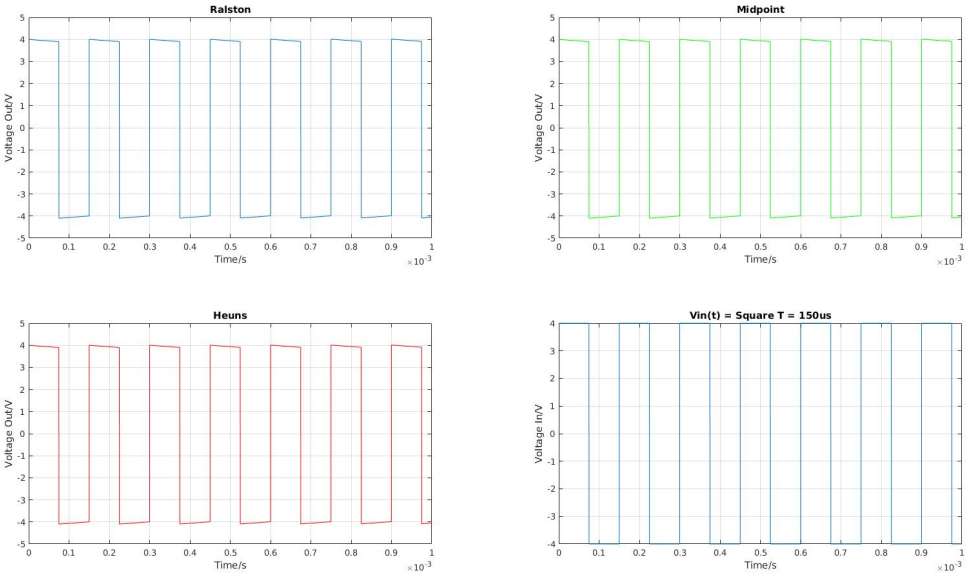

Figure 10: Square Wave Input $T = 150\mu s$

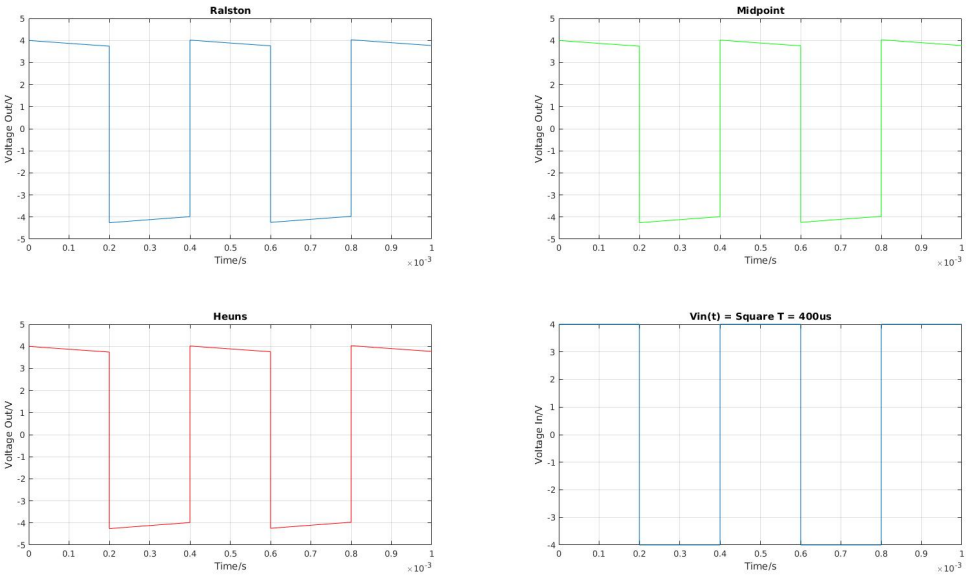

Figure 11: Square Wave Input $T = 400\mu s$

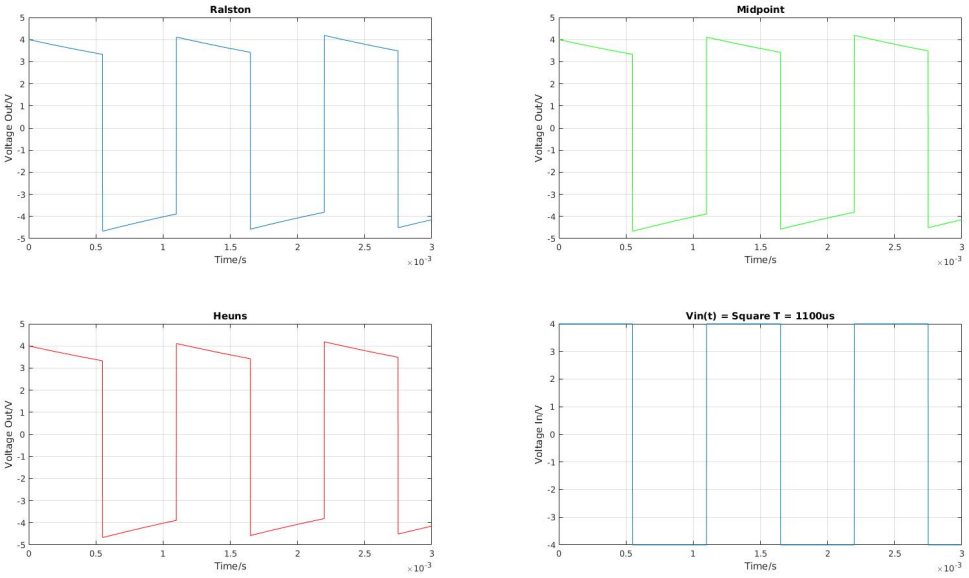

Figure 12: Square Wave Input $T = 1100\mu s$

As seen in the square wave case, when the frequency is smaller, a response of the inductor charging and discharging can be clearly appreciated in the edges of the slopes of $T = 400\mu s$ and $T = 1100\mu s$, this response begins appearing with $T = 150\mu s$.

1.4.5 Sawtooth wave with $\tilde{V}_{in} = 4V$

```
vin = 4;
T = ***10^(-6);
t = t0: T : tf;
f = 1/T;
Vin =@(t) vin*sawtooth(2*pi*f*t);
```

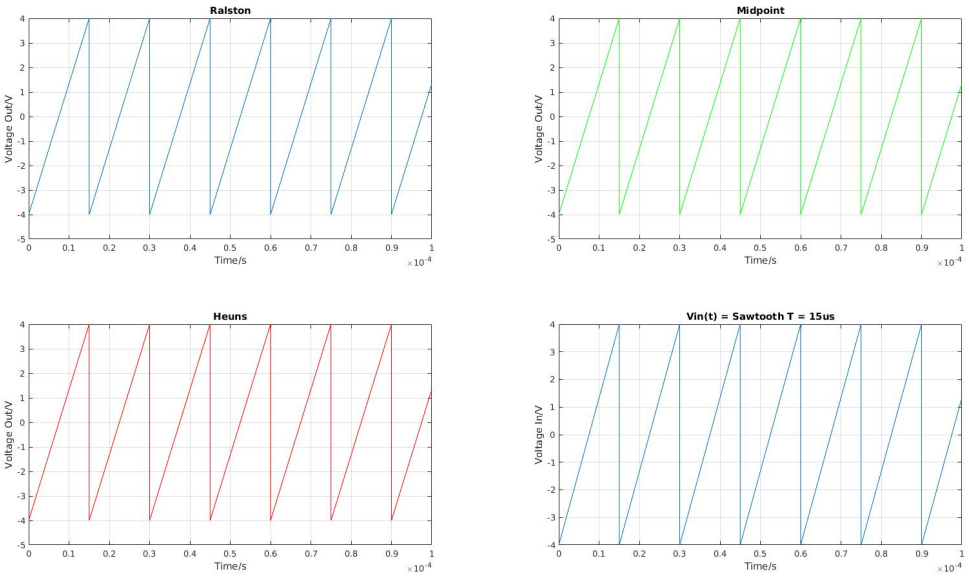

Figure 13: Sawtooth Wave Input $T = 15\mu s$

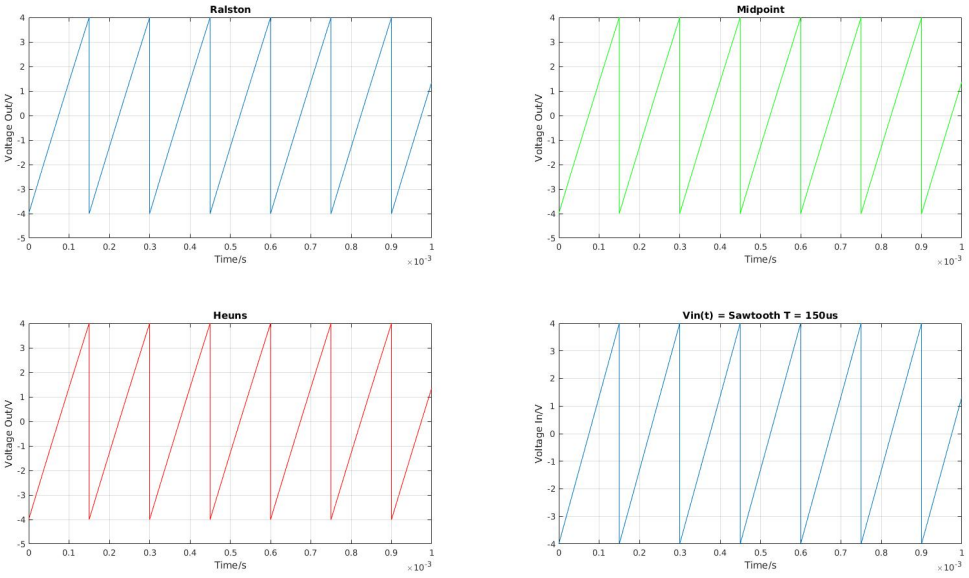

Figure 14: Sawtooth Wave Input $T = 150\mu s$

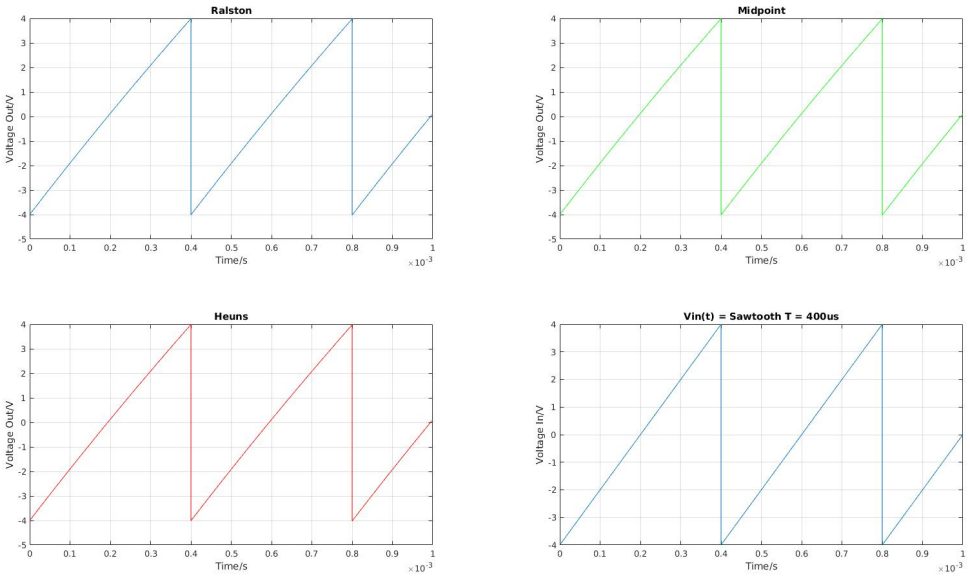

Figure 15: Sawtooth Wave Input $T = 400\mu s$

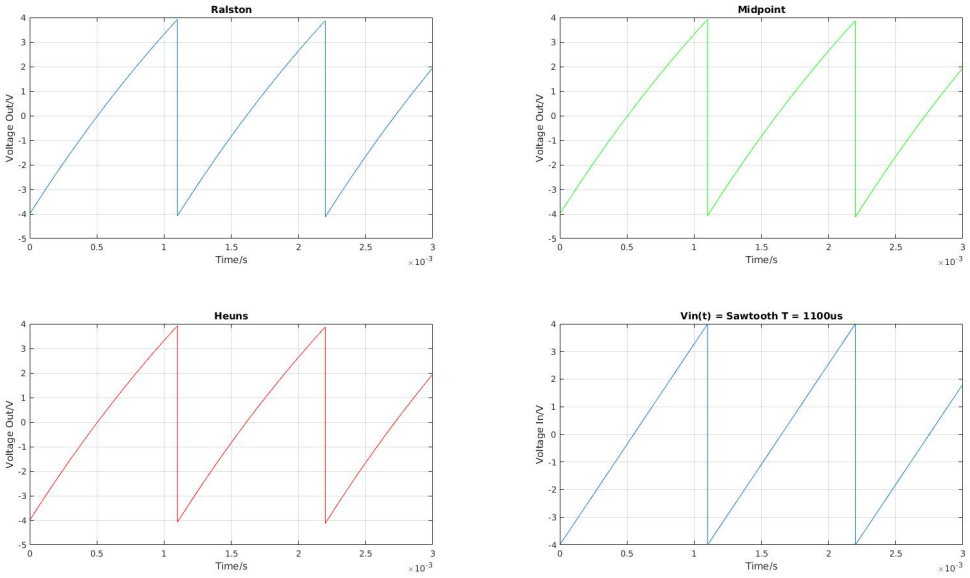

Figure 16: Sawtooth Wave Input $T = 1100\mu s$

Regarding the sawtooth functions, we can see that for all frequencies, there is an abrupt discharge response followed by a gradually increasing charge.

2 Exercise 2 - Error Analysis for RL Circuit

2.1 Background

In Exercise 1, three numerical methods(Heun,Midpoint and Ralston) were used to estimate the value of functions for different inputs. However,these numerical methods have errors and do not give the exact solutions to the ODE. Therefore, in Exercise 2, the exact solutions to the ODEs will be obtained using MATLAB. These solutions will then be compared with those obtained using the numerical methods (in Exercise 1) to estimate the errors associated with the numerical methods.

2.2 Exact Value

For this RL circuit, a linear first order ODE is used to obtain the exact equation.

2.3 Linear first order ODE

$$\frac{dy}{dx} + p(x)y = Q(x) \tag{9}$$

$$y(x) = Ce^{-\int P(x)dx} + e^{-\int P(x)dx} \int Q(x)e^{\int P(x)dx}dx \tag{10}$$

2.4 Linear first order ODE for RL circuit

$$v_L(t) + v_R(t) = V_{in}(t) \tag{11}$$

$$L\frac{di_L(t)}{dt} + Ri_L(t) = V_{in}(t) \tag{12}$$

$$R = 0.5\Omega L = 1.5mH \tag{13}$$

Input is given as:

$$V_{in} = 6cos(\frac{2\pi t}{150 * 10^{-6}}) \tag{14}$$

According to linear first order ODE, we get:

$$1.5 * 10^{-3}\frac{di_L(t)}{dt} + 0.5i_L(t) = 6cos(\frac{2\pi t}{150 * 10^{-6}}) \tag{15}$$

$$\frac{di_L(t)}{dt} + \frac{1}{3} * 10^3 i_L(t) = 4 * 10^3 cos(\frac{2\pi t}{150 * 10^{-6}}) \tag{16}$$

$$e^{\frac{1}{3}*10^3 t}\frac{di_L(t)}{dt} + e^{\frac{1}{3}*10^3 t} * \frac{1}{3} * 10^3 i_L(t) = e^{\frac{1}{3}*10^3 t} * 4 * 10^3 cos(\frac{2\pi t}{150 * 10^{-6}}) \tag{17}$$

$$\frac{d(e^{\frac{1}{3}*10^3 t} i_L(t))}{dt} = e^{\frac{1}{3}*10^3 t} * 4 * 10^3 cos(\frac{2\pi t}{150 * 10^{-6}}) \tag{18}$$

Integrate both sides of the equation:

$$exact = -\frac{12}{1600\pi^2 + 1}exp(-1000/3t) + \frac{480\pi}{1600\pi^2 + 1}sin(\frac{40000t\pi}{3}) + \frac{1}{1600\pi^2 + 1}cos(\frac{40000t\pi}{3}) \tag{19}$$

2.5 Error Functions

Using the same MATLAB scripts in Exercise 1 (heun.m,midpoint.m and ralston.m) the estimated values for the ODEs can be obtained. The result of subtracting these estimated solutions from the exact solutions leads to the error functions. In this section, the error functions will be developed using this technique for the three numerical analysis methods.

2.5.1 Heun's Method

Figure 17 shows the exact solution to the ODE. Figure 17 shows the estimated solution using Heun's method. When the estimated solution is subtracted from the exact solution, the resulting function is the error function for Heun's method which is plotted in Figure 17. In the Heun's method error plot, a small oscillating error can be seen (with amplitude $1.4 * 10^{(} - 5)$). Furthermore, the oscillations are centered around 0 so the poisitve error is equal to the negative error.

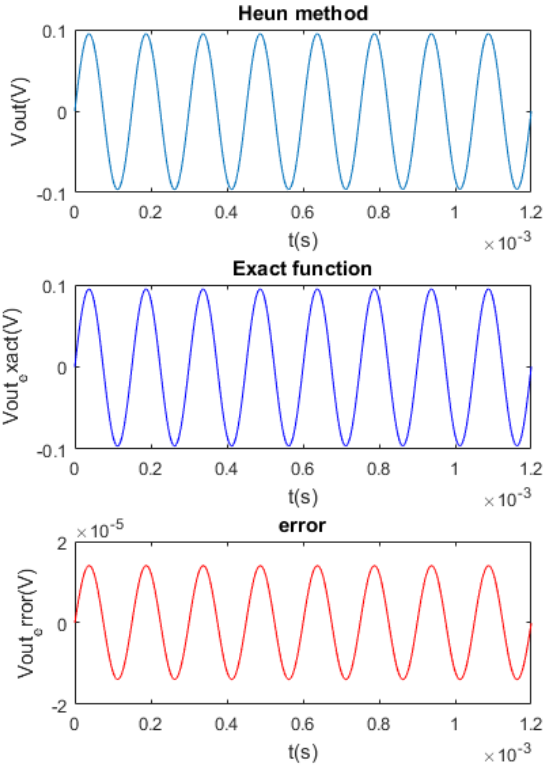

Figure 17: Heun's Method Error

2.5.2 Midpoint method

The same analysis is used to obtain the error function for Midpoint method. The exact solution for the ODE is the same as that for Heun's method (Figure 18). Figure 18 shows the estimated solution obtained by using the midpoint method. Similar to Heun's error analysis, when the estimated solution is subtracted from the exact solution, the resulting function is the error function. This error function is plotted in Figure 18. From the error function for Midpoint, it can be seen that there is a small oscillating error (with amplitude $4.9 * 10^{(} - 4)$). The error is centered around $(-5 * 10^{(} - 4))$ and thus we are underestimating the value of the exact solution when using the Midpoint method.

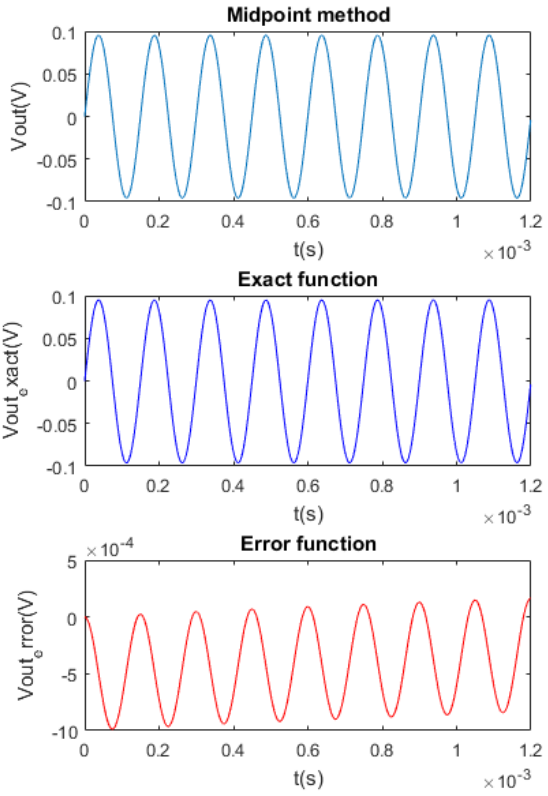

Figure 18: Midpoint Method Error

2.5.3 Ralston method

Making use of the same error analysis technique as above (for Heun's and Midpoint methods), the estimated value obtained using the Ralston method is in Figure 19 and the error function is plotted in Figure 19.
From the error function for Ralston, it can be seen that the error is oscillating with an amplitude of $2.5 * 10^{(} - 7)$. The error function is centered around $2.5 * 10^{(} - 7)$ and thus we can see that the Ralston method overestimates the solution to the ODE.

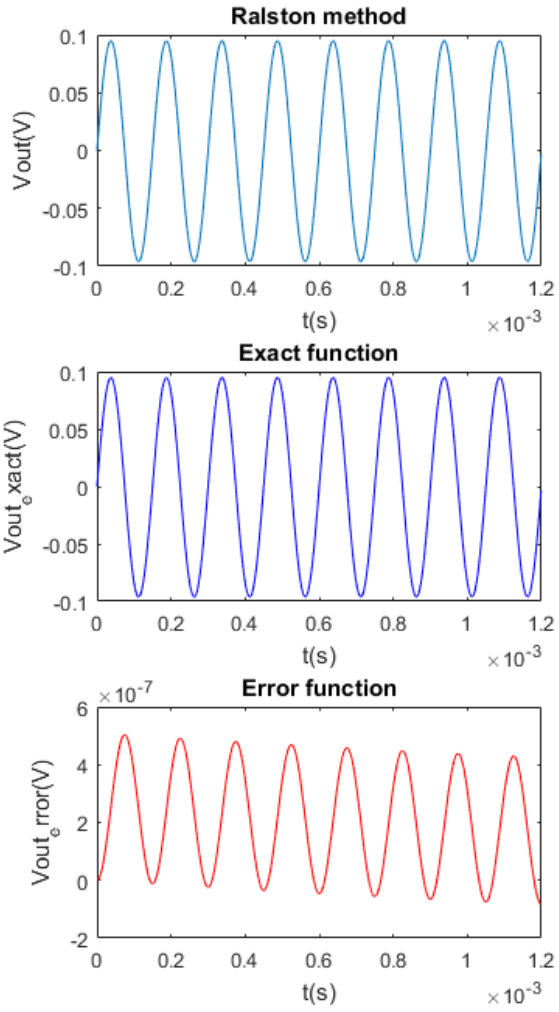

Figure 19: Ralston Method Error

2.6 Comparison

From the analysis above, we found that the error functions associated with the three numerical analysis methods are oscillatory. However, the error functions were centered at different points and had different amplitudes. From the center points of the different methods, we can deduce that Midpoint method underestimates the solution to the ODE, the Ralston method overesitmates the solution to the ODE and Heun's method is centered around 0. Comparing the amplitudes of the error functions, we found that the Midpoint method has the largest error associated with it and the Ralston method has the smallest error. Therefore, we can conclude that the Ralston analysis is the best method for approximating the solution to the linear first order ODE.

2.7 Log-log plot

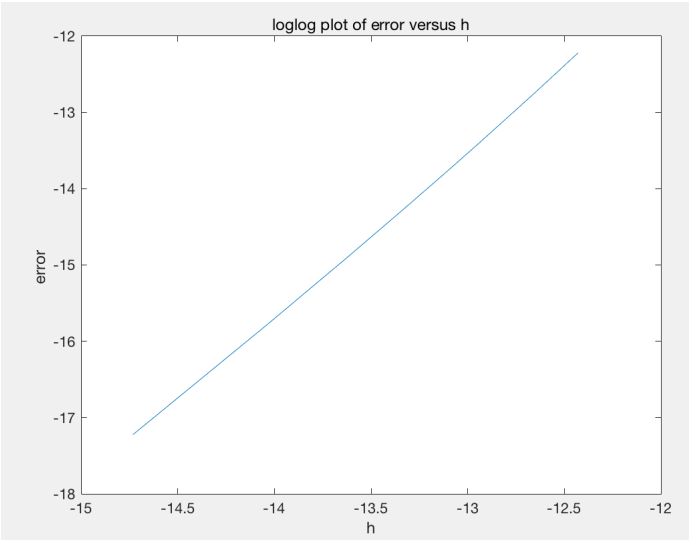

Figure 20: Log-Log plot for Ralston method

From Figure 20,it can be seen that as h increases, the error in the estimated solution using Ralston's method increases. Furthermore, the error has a positive linear relationship with h. The gradient of the straight line is approximately 2. It can be expressed as:

$$logE = 2log(h) + constant \tag{20}$$

Thus, it can be deduced that all of the three methods (Heun's, Midpoint and Ralston) have errors of order h^2. Therefore, all their respective Log-Log plots are similar.

$$E(y, x) = O(h^2); \tag{21}$$

3 Exercise 3 - RLC Circuit

3.1 Background

From Kirchoff's Voltage Law (KVL), the sum of all the voltages in a closed loop must sum to 0 V. From this we can derive the equation $V_{in}(t) = v_L(t) + v_R(t) + v_C(t)$. The voltage across an inductor is the inductance multiplied by the derivative of it's current. the voltage across the resistor is equal to it's resistance multiplied by the current travelling through it. The voltage across the capacitor is the charge held within the capacitor divided by it's capacitance. Substituting these characteristics into the equation above gives us:

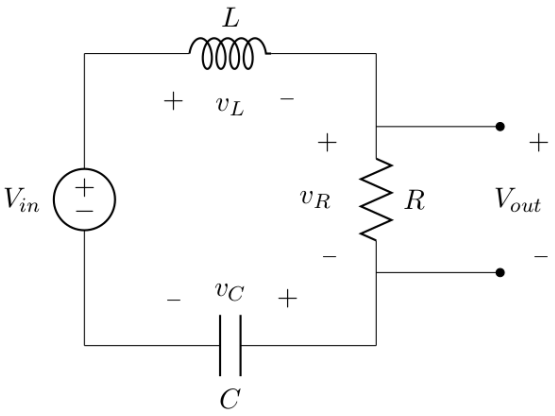

Figure 21: RLC Circuit

$$L\frac{d}{dt}i_L(t) + Ri_L(t) + \frac{1}{C}\int_0^t i_L(t)\ dt = V_{in}(t)$$

Substituting $i_L(t)$ for $\frac{d}{dt}q_C(t)$ gives the equation:

$$L\frac{d^2}{dt^2}q_C(t) + R\frac{d}{dt}q_C(t) + \frac{1}{C}q_C(t) = V_{in}(t)$$

We were then able to turn this second order ODE into a set of two coupled, first order ODEs through equating $q'_C(t) = y(t)$ and substituting $q_C(t)$ for $x(t)$. The coupled equations are:

$$y(t) = x'(t) \quad \text{and} \quad y'(t) = \frac{1}{L}[V_{in}(t) - Ry(t) - \frac{1}{C}x(t)]$$

For our project we have the values:

$$R = 250\Omega, C = 3\mu F, L = 650mH, i_L(0) = 0\ \ (y_0)\ \text{and}\ \ q_C(0) = 500nC\ \ (x_0).$$

3.2 Script Coupled Equations and Constants

Now that we have these two equations we are able to implement them into a MATLAB script called 'RLC_script.m' by defining them as:

```
R = 250; L = 650*10^-3; C = 3*10^-6; %Impedance values for the components
w1 = 2*pi*500; %frequency for the 500 Hz sinusoid
w2 = 2*pi*100; %frequency for the sinusoid
w3 = 2*pi*5; %frequency for the sinusoid
tc = 3*10^-6; %tau for the exponential decay input
y0 = 0; x0 = 500*10^-9; t0 = 0; %Initial conditions y = iL, x =qC and t = time
h = 0.00001; %step size
tf = 0.03; %final condition
Vin = @(t) 5;
func1 = @(x, y, t) y; %y = q'
func2 = @(x, y, t) (Vin(t) - R*y - x/C)/L; %the second coupled equation
```

3.3 4th Order Runge Kutta 3/8 Algorithm

3.3.1 Theory

We use the Runge Kutta 4th order 3/8 algorithm to use these two coupled equations to estimate the next point of the current i_L and the charge q_C. The algorithm is:

$$f_1(x,y,t) = y \quad f_2(x,y,z) = \frac{1}{L}[V_{in}(t) - Ry - \frac{x}{C}]$$

$$k1_x = hf_1(x_i, y_i, t_i) \quad k1_y = hf_2(x_i, y_i, t_i)$$

$$k2_x = hf_1(x_i + \frac{k1_x}{3}, y_i + \frac{k1_y}{3}, t_i + \frac{h}{3}) \quad k2_y = hf_2(x_i + \frac{k1_x}{3}, y_i + \frac{k1_y}{3}, t_i + \frac{h}{3})$$

$$k3_x = hf_1(x_i - \frac{k1_x}{3} + k2_x, y_i - \frac{k1_y}{3} + k2_y, t_i + \frac{2h}{3})$$

$$k3_y = hf_2(x_i - \frac{k1_x}{3} + k2_x, y_i - \frac{k1_y}{3} + k2_y, t_i + \frac{2h}{3})$$

$$k4_x = hf_1(x_i + k1_x - k2_x + k3_x, y_i k1_y - k2_y + k3_y, t_i + h)$$

$$k4_y = hf_2(x_i + k1_x - k2_x + k3_x, y_i k1_y - k2_y + k3_y, t_i + h)$$

$$x_{i+1} = x_i + \frac{k1_x + 3k2_x + 3k3_x + k4_x}{8} \quad y_{i+1} = y_i + \frac{k1_y + 3k2_y + 3k3_y + k4_y}{8}$$

$$t_{i+1} = t_i + h \quad \text{where h is time step}$$

This starts from a single point and calculates the gradient to the next point. Using this, it then creates another slope. This is then repeated twice more. Using the equations above and a weighting is give to each gradient. The weighted gradients are summed and divided by the total weighting, to scale it back to a uniform size. From this we are able to estimate more accurately predict where the next point is going to go for this time step.

3.3.2 Implementation

To implement this into MATLAB we created a function called 'RK4second.m', which would take in the two functions and work out the k coefficients for both x and y and from there, work out the next position of the x (charge) and y (current) coordinates.

```
function [xout, yout] = RK4second (xin, yin, h, tin, func1, func2)

k1x = h*feval(func1, xin ,yin, tin); %approx consts for RK4th ord 3/8 Method
k1y = h*feval(func2, xin ,yin, tin);
k2x = h*feval(func1, xin + k1x/3, yin + k1y/3, tin+h/3);
k2y = h*feval(func2, xin + k1x/3, yin + k1y/3, tin+h/3);
k3x = h*feval(func1, xin  - k1x/3 + k2x, yin - k1y/3 + k2y, tin+2*h/3);
k3y = h*feval(func2, xin  - k1x/3 + k2x, yin - k1y/3 + k2y, tin+2*h/3);
k4x = h*feval(func1, xin + k1x - k2x + k3x, yin + k1y - k2y + k3y, tin+h);
k4y = h*feval(func2, xin + k1x - k2x + k3x, yin + k1y - k2y + k3y, tin+h);

xout = xin + (k1x + 3*k2x + 3*k3x + k4x)/8;
yout = yin + (k1y + 3*k2y + 3*k3y + k4y)/8;
end
```

This takes in the present value of current and charge at a certain time, the time position, the step time and the two functions to evaluate and approximate the next point, which is then output at the end.

3.4 RLC_Script.m Implementation

```
N = round((tf-t0)/h); %calculates number of time steps
ya = zeros(1,N); xa = zeros(1,N);
ta = zeros(1,N); in = zeros(1,N); %set up arrays

xa(1) = x0; ya(1) = y0; ta(1) = t0; in(1) = Vin(t0); %input initial conditions
for i = 1:N-1 %works out next timestep value
   [xa(i+1), ya(i+1)] = RK4second (xa(i), ya(i), h, ta(i),func1, func2);
    ta(i+1) = ta(i) + h; %next time step
    in(i+1) = Vin(ta(i+1)); %next value of the input
end

figure; %sets up the figures
Vout = R*ya; %Vout = R*iL
subplot(2,1,1);
plot(ta, Vout); %plot output Voltage
grid on;
xlabel('Time/s'); ylabel('Voltage Out/V');
title('R*dq/dt with a Step Response')
subplot(2,1,2);
plot(ta, in); %plot input Voltage
grid on;
xlabel('Time/s'); ylabel('Voltage In/V');
title('Step Response')
```

3.5 Exercise 3 Graphs

3.5.1 Step Response

The step response causes a sudden rise of voltage to 5 V. Due to the inductor not being able to change current instantaneously, the voltage out starts from 0 V and, as you can see from Figure 22. It increases up to around 1.8 V and at this point the capacitor starts to reduce the rate of charging due to it saturating in charge. As the voltage out starts to tend to 0, the capacitor emits charge, causing a negative current through the inductor. As this goes on, the voltage across the capacitor tends to 5 V, causing no current to flow through the inductor and the resistor, which causes the output voltage to tend to 0 V.

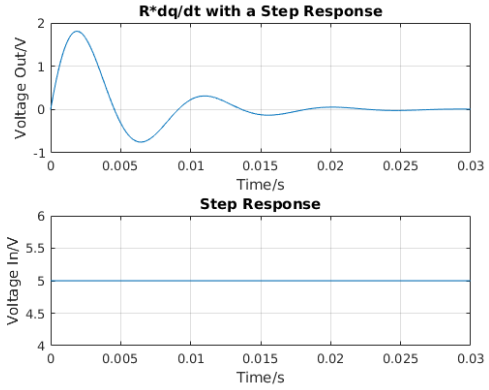

Figure 22: $V_{in}(t) = 5$

3.5.2 Sine Wave Input

In Figure 23 the input is a sine wave of amplitude 5 V and frequency of 5 Hz. As the sine wave moves away from zero it causes a sharp increase in the current going through the inductor, which causes a sharp rise in the voltage out. This then attenuates to form the shape of a sine wave of amplitude ≈ 0.1 V which leads the input voltage by a phase difference of $\frac{\pi}{2}$. This is relative to the frequency of the input and the complex impedance

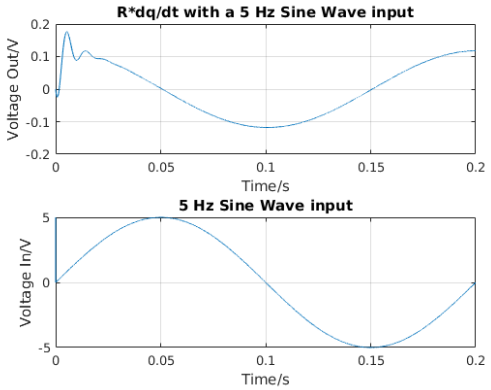

Figure 23: $V_{in}(t) = 5sin(2\pi 5t)$

of the inductor and the capacitor being $j\omega L$ and $\frac{1}{j\omega C}$ respectively. We chose the final time as 0.2 s to show how the output initially changes but then follows the shape of the input eventually.

In Figure 24 the input is a sine wave with an amplitude of 5 V and a frequency of 100 Hz. Upon the start of the wave, the output voltage rises slower than the voltage of the input, due to the fact that the rate of change of the inductor current is not instantaneous. It will eventually equal the frequency of the input. This causes a slight phase difference and the amplitude is far greater than that of the 5 Hz input, resulting in an amplitude of ≈ 4 V. This greater output voltage is due to a band pass filter and at low frequencies, 5 Hz for example, we can see attenuation to a value close to 0 V, whereas at this frequency the output voltage is close to the input voltage and beyond some frequency greater than this, the same result as the 5 Hz wave will be apparent.

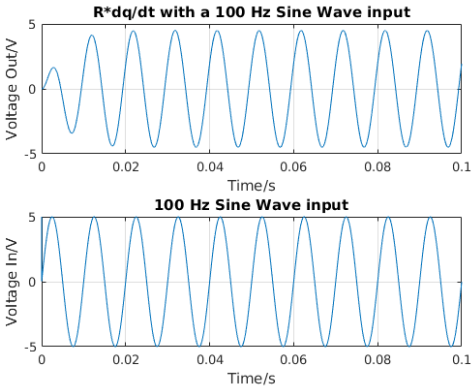

Figure 24: $V_{in}(t) = 5sin(2\pi 100t)$

In Figure 25 the input is a sine wave of amplitude 5 V with a frequency of 500 Hz. The response is very similar to Figure 23's output signal, rising initially to a value that is greater than its steady state amplitude of ≈ 0.5 V, which is 10 times less than the input amplitude, and less than the the amplitude of the Figure 24. This suggests that the center of the band pass filter is based around 100 Hz as is seen in Figure 24 and therefore the outer bounds fall in amplitude at frequencies greater and less than this.

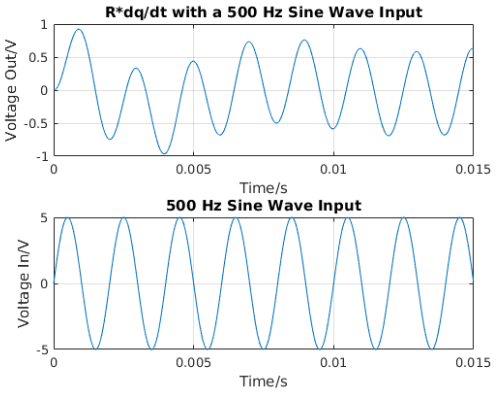

Figure 25: $V_{in}(t) = 5sin(2\pi 500t)$

3.5.3 Square Wave Inputs

In Figure 26 the input is a square wave of frequency 5 Hz and amplitude 5 V. At this frequency, the response is identical to the response of a step response due to there being enough time for the wave to return to 0 V after the change in amplitude. The only difference is that when the voltage drops to -5 V the step response is equal to the positive step response but negative. The time scale was chosen to show 1 and a quarter wavelengths so that it is easy to see that, each time the sign changes, the response is identical.

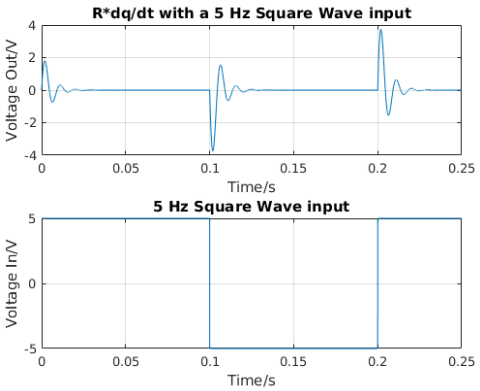

Figure 26: $V_{in}(t) = 5square(2\pi 5t)$

In Figure 27 the input is a square wave of frequency 100 Hz and amplitude 5 V. At this frequency, the output follows the characteristics of a sine wave due to the step response giving a sinusoidal peak. This is repeated but reversed in direction as the sign of the input changes, caus-

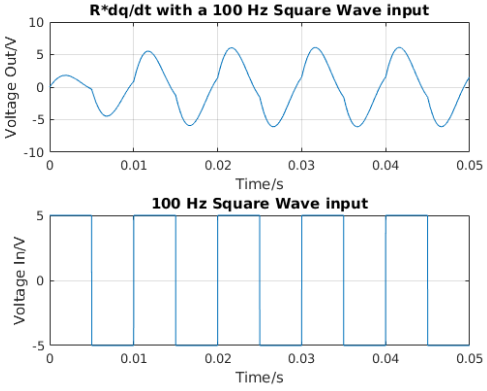

Figure 27: $V_{in}(t) = 5square(2\pi 100t)$

ing a periodic signal on the output respresenting something similar to a sine wave.

In Figure 28, the input is a square wave of frequency 500 Hz and amplitude 5 V. At this frequency, the step responses causes an output signal which is similar to a periodic tent function of exponential charge and discharge. This is due to the fact that the square wave changes sign too rapidly for the signal to smooth off or return to zero before the next half wavelength. As it gets to the 4th period of the input, the output settles to an amplitude of ≈ 1 V. Looking back over Figures 6, 7 and 8, the bandpass filter seems to give similar changes in output amplitude relative to the input amplitude, while at 100 Hz, in Figure 27 the magnitude is greater than both magnitudes. Confirming that the center frequency is based around 100 Hz and the bandwidth is based between 5 Hz and 500 Hz.

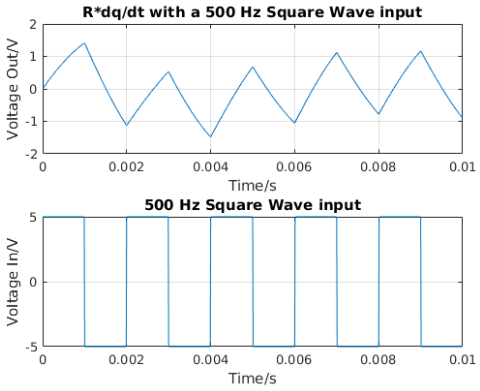

Figure 28: $V_{in}(t) = 5square(2\pi 500t)$

3.5.4 Exponential Decay Input

In Figure 29, the input is a decaying exponential with the time constant $\tau_C = 3(ms)^2$ causing a sharp drop in amplitude towards 0 V in a time of less than 5 ms. This causes an output which is incredibly similar to the output of the step response (Figure 22). The main difference is that the max amplitude of the output of Figure 29 is ≈ 1 V, which is ≈ 0.8 V less than the max amplitude of Figure 22 which was ≈ 1.8 V.

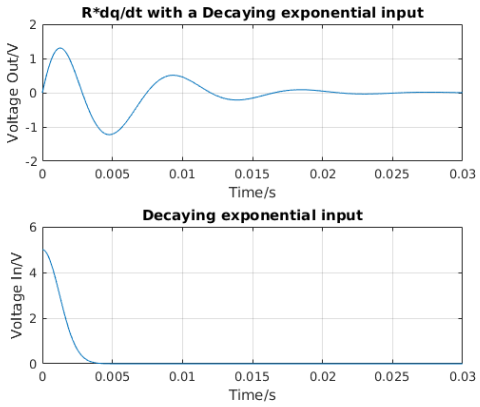

Figure 29: $V_{in}(t) = 5e^{\frac{-t^2}{\tau_C}}$

4 Exercise 4 - Finite Differences for PDE

4.1 Background

For this exercise we were asked to implement the finite difference method for the 1D heat equation which is:

$$\frac{\partial y}{\partial t} = \frac{\partial^2 x}{\partial t^2} \quad \text{in which} \quad 0 < x < 1, \quad t > 0 \quad \text{and} \quad y(0,t) = y(1,t) = 0 \ \forall \ t$$

4.2 Script Equations and Constants

```
x0 = 0;
h = 0.005; %Step size in the X direction
Tin = @(x) sin(2*pi*x);
Bc = @(t) 0;
xf = 1 + h; %Final value of X plus one more step to complete the graph
t0 = 0; %Initial time value
k = 0.45*h^2; %v max of 1/2 we use 0.45 work out k relatively, steps in time
tf = 200*k; %the end time is 250 steps in the k direction
N = round((xf-x0)/h); %number of steps in the x direction
M = round((tf - t0)/k); %number of steps in the t direction
v = k/(h^2); %recalculating the value of v
b = 1 - 2*v; %create the value of beta
ya = zeros(1,N); xa = zeros(1,N); %create empty arrays
xa(1) = x0; xa(N) = xf-h;  ya(1) = Bc(0); ya(N) = Bc(0); %initial conditions
```

In the script above, we define the step size in the x direction, and from this we are then able to calculate k, the step in time. Due to the formula to calculate finite difference:

$$U_j^{m+1} = vU_{j-1}^m + (1-2v)U_j^m + vU_{j+1}^m \quad \text{where for stability} \quad v < \frac{1}{2}$$

By making v = 0.45, the system never becomes unstable, and if the x direction step size is changed, we can calculate k by multiplying it by h^2 due to $v = \frac{k}{h^2}$ we also replaced $1 - 2v$ with β.

4.3 Implementation of the Finite Difference Method

```
for i = 1:N-2
    xa(i+1) = xa(i) + h; %next step in x direction
    ya(i+1) = feval(Tin, xa(i+1)); %calculate heat inital points
end
t = t0; %initial time value
plot(xa, ya, '.', 'markersize', 4); %plot graph at t = 0
hold on; %plots them all on the same graph
yn = ya; %to take the next time values
for a = 1:15 %plots the graph 15 times
    for i = 1:M %cycle through every time step
        ya(1) = Bc(t); ya(N) = Bc(t); %set boundary conditions
       for j = 2:N-1 %work out temp for next time step
           yn(j) = v*ya(j-1) + b*ya(j) + v*ya(j+1);
       end
        t = t+k; %change the time one step forward
        ya = yn; %replace ya with the next time array
    end
    plot(xa, ya, '.', 'markersize', 4); %plot heat
```

```
    title 'Finite Difference of PDE 1D Heat Equation - '
end
xlabel 'Distance/cm'
ylabel 'Temperature/C'
```

The way this works is that we use a for loop to calculate the initial temperature value at each spot on the x axis and place them into the array "ya", this is then plotted on a graph to show the temperature at time 0. After that we use a *for loop* which runs 15 times, so that we are able to plot each graph whilst being able to visualise how it changes as time increases. There is then another *for loop*, which cycles through every step, and recalculates the boundary conditions (for the sake of having time variable boundary conditions). Finally, another *for loop* cycles through all the points between the first and last values of temperature and applies the finite difference equation stated above. After this "ya" is replaced with the newly calculated array and then the values are plotted on the graph.

4.4 Finite Difference Graphs

As is shown in Figure 30, for the tent function the temperature decreases as it tends towards 0 and 1, from the centre of 0.5. As it steps through time, the heat tries to diffuse evenly within the object. As time goes on, the heat decreases within the body and if we were to go further into time, the heat would altogether tend to zero across the body.

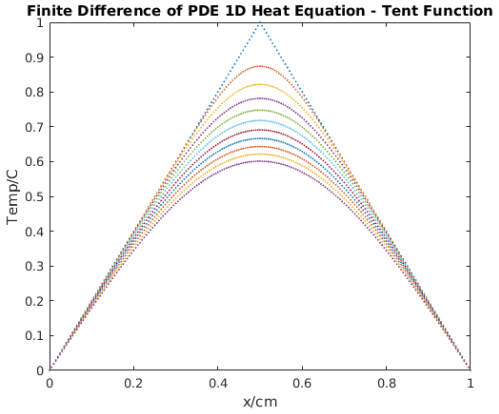

Figure 30: $T_{in}(x) = tent(x)$

In Figure 31 you can see the sine function in which meets the boundary conditions. Having 'positive' and 'negative' temperatures, they have identical characterisics determining how the temperature is dissipated. The central point at x = 0.5 remains equal to 0 due to the equation taking both points at either side of it, and multiplying with v. They both have the same magnitude, but different signs, so when added together they cancel and cause the temperature to remain at 0 for the time being.

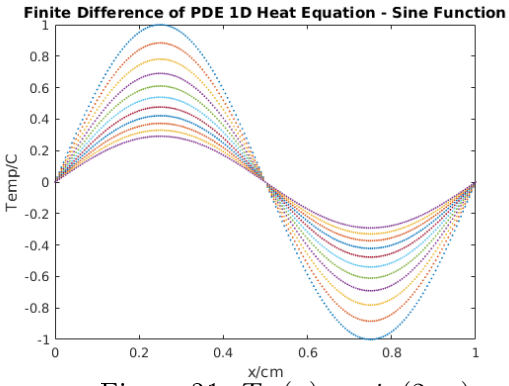

Figure 31: $T_{in}(x) = sin(2\pi x)$

In Figure 32, the absolute value of the sine wave was taken. The reason it differs with the standard sine curve is that because now, across the centre, on either side of the point they are no longer opposite in magnitude, therefore the point at the centre increases in magnitude as the heat disperses across the body, trying to uniformally distribute it.

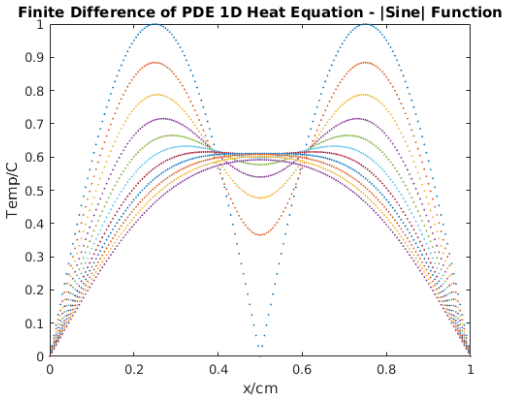

Figure 32: $T_{in}(x) = |sin(2\pi x)|$

In Figure 33, the input is half of a sine wave in which you can simply see that as time goes on, throughout the body, the heat dissipates, tending towards 0.

In Figure 34, the input is a polynomial with the roots $x = 0$, 0.3, 0.5 and 1, showing a very random distribution of the heat throughout the body. At the start of time, the heat from $0 < x < 0.5$ smooths off a lot quicker than the second half of the graph due to the fact that there is a maximum at the point of $x \approx 0.83$ and this takes a while for the heat to dissipate. As with all the rest, as time travels towards infinity, the heat will tend to 0 unifomally across the body.

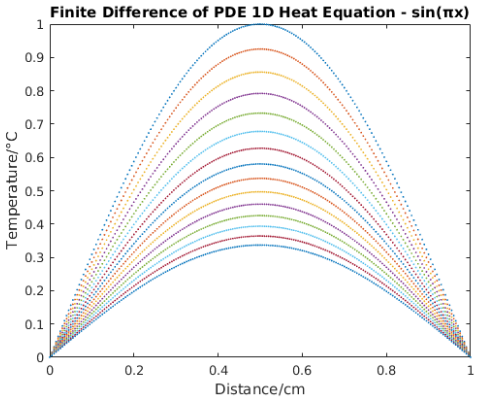

Figure 33: $T_{in}(x) = sin(\pi x)$

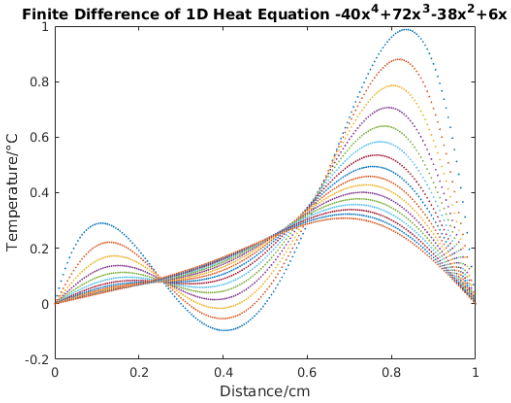

Figure 34: $T_{in}(x) = -40x^4 + 72x^3 - 38x^2 + 6x$

4.5 Bonus Exercises

4.5.1 Boundary Conditions Not Met

In Figure 35, the input was the straight line $T_in(x) = 1 - x$ where the left hand portion of the graph doesn't meet the left hand condition. When the boundary conditions are both 0, it causes the temperature from just after x = 0 to fall away considerably quicker than any other portion of the graph, due to the greater difference between the boundary condition and the points to the side of it, as you can see in Figure 35.

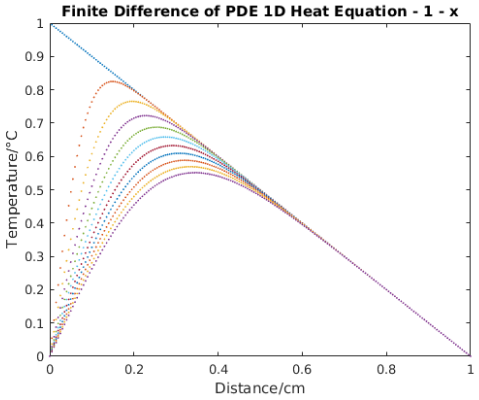

Figure 35: $T_{in}(x) = 1 - x$

4.5.2 Non Zero Boundary Conditions

In Figure 36, the input is a tent function, but rather than having boundary conditions which were equal to 0, for the boundary condition at x = 0, the temperature is equal to 0.75 and at x = 1, the value is equal to 0.25. As you can see, the points to the left of the graph all pull up towards the value of 0.75 and on the other side, all the points are pulled towards 0.25. As time goes to infinity, you would get the heat distributed as a straight line going from 0.75 to 0.25.

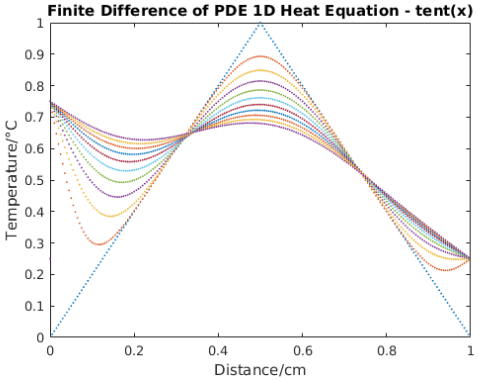

Figure 36:
$T_{in}(x) = tent(x)$ $y_0(t) = 0.75, y_1(t) = 0.25$

4.5.3 Time Varying Boundary Conditions

In Figure 37, the input is a tent function, initially with zero boundary conditions, but as time moves forward, the temperature at the end points increase following the equation $y_0(t) = y_1(t) = 20t$, so the heat in the body will end up heating up from the edges inwards.

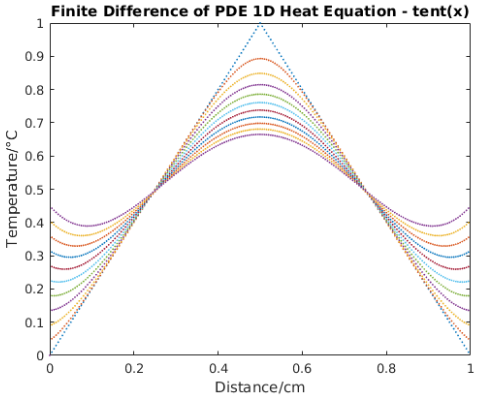

Figure 37: $T_{in}(x) = tent(x)$ $y_0(t) = y_1(t) = 20t$