EE2-08C – Numerical Analysis of Differential Equations using MATLAB – Coursework 2019

EEE Y2 Group 7

Ammar Chaudhry

Kanav Agarwal

Linghzi Qiao

Raj Jain

Sanjana Ganguly

Sanjith Nambiar

Snehil Kumar

Yinzi Zhang

Word Count: 9783 words

Date: 18th March, 2019

Table of Contents

1. **RL Circuit**……..……..1

    a. **1.1:** Introduction to the RL Circuit……....1
    b. **1.2:** RL Circuit Features……....2
    c. **1.3:** The second-order Runge Kutta Methods: (Heun, Midpoint and Ralston with MATLAB codes……....3

    d. **1.4:** RK2_script.m……....10
    e. **1.5:** Heun Method Plots……....11
    f. **1.6:** Midpoint Method Plots……....26
    g. **1.7:** Ralston Method Plots……41
    h. **1.8:** Explanation of the plots with respect to Heun, Midpoint and Ralston methods..…….... .56

    i. **1.9:** Error Script.. ……....61

2. RLC Circuit……..……..77

    a. **2.1:** Introduction to the RLC Circuit……....77
    b. **2.2:** RLC Circuit Features……....78
    c. **2.3:** The 4th order Runge-Kutta 3/8 Algorithm method ……..…….. 79
    d. **2.4:** The 4th order Runge-Kutta 3/8 Algorithm method implemented on MATLAB as RK4second.m……....81

    e. **2.5:** RLC Series Circuit Modeling…… ……82

3. **Finite Differences for PDE**……..…….98
    a. **3.1:** Introduction to Finite Differences for PDE……..……..…….98
    b. **3.2:** Solving the given heat equation by implementing the finite difference method using Central Algorithm Theorem……..99

    c. **3.3:** Using Taylor series to derive the finite difference approximation and obtaining the error included……126

4. Works Cited……..……..…….128

1 RL Circuit (Exercises 1 and 2)

1.1: Introduction to the RL Circuit (Exercise 1)

In this exercise, a high-pass filter takes an input signal V_{in} and lets only the high-frequency components pass. Usually, high-pass filters are made with capacitors instead of inductors since the latter can be more easily manufactured and are usually, physically smaller.

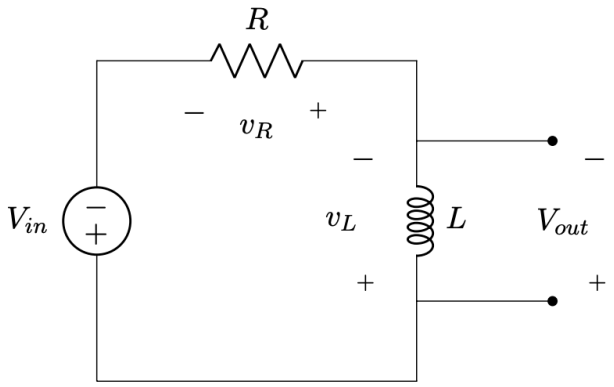

Fig1: *Given RL Circuit with parameters R = 0.5 Ω, L = 1.5 mH*

For the circuit in Fig1, the following equations can be written:

$$v_L(t) + v_R(t) = V_{in}(t)$$
$$L\frac{d}{dt}\, i_L(t) + Ri_L(t) = V_{in}(t)$$

In the aforementioned equations, the state is $i_L(t)$ and $V_{in}(t)$ is the input voltage. Finally, the output voltage V_{out}, is given by the equation below:

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

Furthermore, it is mentioned that at $t = 0$, $i_L(0) = 0\ A$. In this section, a DC motor with inertia 250 μNm/s^2 will be modelled the maximum torque gain should be 50 mN m A^{-1}.

1.2: RL Circuit Features

The transfer function of the circuit in Fig1 is given by the following:

$$T(s) = \frac{Ls}{R + Ls} = \frac{s}{\frac{R}{L} + s}$$

From the equation above, it can be deduced that the corner frequency is equal to $\frac{R}{L} = \frac{0.5}{0.0015} = 333.3\ Hz$. When the input is a DC voltage, the inductor acts as a short circuit, so the output voltage is 0.

The step-response of the RL circuit is given by the following:

$$i(t) = \frac{V_{in}}{R} - \frac{V_{in}}{R}\mathrm{e}^{-\left(\frac{R}{L}\right)*t}$$

Therefore, the voltage across the inductor can be written as follows:

$$v_L = L\frac{di}{dt} = L\frac{d}{dt}\left(\frac{V_{in}}{R} - \frac{V_{in}}{R}\mathrm{e}^{-\left(\frac{R}{L}\right)*t}\right)$$

$$v_L = L\frac{V_{in}}{R}\left(\frac{R}{L}e^{-\left(\frac{R}{L}\right)t}\right) = V_{in}e^{-\left(\frac{R}{L}\right)t}$$

This equation will be useful in later stages to explain plots of output voltage across the inductor against the time.

1.3: The Second Order Runge-Kutta Methods: Heun, Midpoint and Ralston with MATLAB codes

This analysis will focus on second order Runge-Kutta methods. The second order method is written as:

$$\frac{dy}{dx} = f(x, y), y(0) = y_0$$

The next step value of y is given as follows:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h$$

where;

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h)$$

From the Taylor Series expansion for a function of two variables the definitive equation for all possible second order methods can be defined:

$$y_{i+1} = y_i + (a_1 + a_2)hf(t_i, y_i) + h^2 \left(bp \frac{\partial f}{\partial t} + bq \frac{\partial f}{\partial y} f(t_i, y_i) \right) + O(h^3)$$

where;

a_1, a_2: weights

$$p = q = constants$$

$$f(t_i, y_i) = \frac{dy}{dt}$$

This must be equivalent to the Taylor series expansion of y(x + h):

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2} \left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f(t_i, y_i) \right) + \cdots$$

It is observed that this sets a condition for the infinite set of combinations of coefficients of all second order Runge-Kutta methods:

$$a_1 + a_2 = 1$$
$$a_2 p = \frac{1}{2}$$
$$a_2 q = \frac{1}{2}$$

There can be infinitely many sets of solutions for weights a_1, a_2. There are several second order Runge-Kutta methods but in this exercise, the following algorithms will be considered:

1.3.1: Heun's Method

The Heun formula gives a more accurate approximation than the Euler rule and gives an explicit formula for computing y_{n+1}. The basic idea is to correct some error of the original Euler's method. The syntax of this method is similar to that of the trapezoidal rule, but the y value of the function in terms of y_{n+1} consists of the sum of the y value and the product of h and the function in terms of x_n and y_n which is given below:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h$$

where $a_2 = \frac{1}{2}$ is chosen, giving;

$$a_1 = \frac{1}{2}$$
$$p_1 = 1$$
$$q_{11} = 1$$

resulting in;

$$y_{i+1} = y_i + \left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right)h$$

where;

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + h, y_i + k_1 h)$$

```
function [t,Vout] = Heun(func, i0, tf, h, R, L)
    %R resistor value
    %L inductor value
    %tf stop time
    %i0 inital current value
    %func function handle
    ti = 0;                                          %initial time
    N = round((tf-ti)/h);                            %N = number of
iterations
    t(1)=ti;                                         %set t(1) as
initial time value, ti
    i = zeros(1,N);                                  %initialise i array
to zeros
    Vout = zeros(1,N);                               %initialise Vout
array to zeros
    i(1) = i0;                                       %set i(1) as
initial current value i0
    Vout(1) = L*func(t(1),i(1));                     %initial value of
output voltage calculated->Vout(1)=L*di(1)/dt(1);

    for j=1:N-1                                      %loop for N
iterations
        t(j+1) = t(j) + h;                           %Updating the next
value of time with t(j) + h, where t(j) is the previous time value
and h is the step size
        k1 = func(t(j),i(j));                        %gradient at t(j)
        ip = i(j) + h*k1;                            %calculated y-
predictor, ie, predicted value of current at t(j)+h
        k2 = func(t(j)+h,ip);                        %second gradient at
t(j)+h
        kave = 0.5*(k1+k2);                          % average gradient
over [t(j),t(j)+h]
        i(j+1) = i(j) + h*kave;                      %next value of i
calculated from previous values of t,i
        Vout(j+1) = L*func(t(j+1),i(j+1));           %Next value of
output voltage calculated -> Vout(j+1) = L*(di(j+1)/dt(j+1))
    end
end
```

Fig2: *MATLAB code Heun.m implementing the Heun method discussed in this section*

1.3.2: Midpoint Method

Instead of taking approximations of the slopes provided in the function, this method attempts to calculate more accurate approximations by calculating slopes halfway through the line segment. The syntax of this method is given as follows:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h$$

where $a_2 = 1$ is chosen, giving;

$$a_1 = 0$$

$$p_1 = \frac{1}{2}$$

$$q_{11} = \frac{1}{2}$$

resulting in;

$$y_{i+1} = y_i + k_2 h$$

where;

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h)$$

```
function [t,Vout] = Midpoint(func,i0,tf,h,R,L)
    %R resistor value
    %L inductor value
    %tf stop time
    %i0 inital current value
    %func function handle
    ti = 0;                                          %initial time
    N = round((tf-ti)/h);                            %N = number of
iterations
    t(1)=ti;                                         %set t(1) as
initial time value, ti
    i = zeros(1,N);                                  %initialise i
array with zeros
    i(1) = i0;                                       %set i(1) as
initial current value, i0
    Vout(1) = L*func(t(1),i(1));                     %initial value
of output voltage calculated->Vout(1)=L*di(1)/dt(1)

    for j=1:N-1                                      % loop for N
iterations
        t(j+1) = t(j) + h;                           %Updating the
next value of time with t(j) + h, where t(j) is the previous time
value and h is the step size
        k1 = func(t(j),i(j));                        %gradient at
t(j)
        k2 = func(t(j)+0.5*h,i(j)+h*0.5*k1);         %second gradient
at t(j) + 0.5*h, ie, at the midpoint
        i(j+1) = i(j) + h*k2;                        %next value of i
calculated from previous values of t,i
        Vout(j+1) = L*func(t(j+1),i(j+1));           %Next value of
output voltage calculated-> Vout(j+1) = L*di(j+1)/dt(j+1)
    end
end
```

Fig3: *MATLAB code Midpoint.m implementing the Midpoint method discussed in this section*

1.3.3: Ralston's Method

Ralston's method evaluates the slopes similar to other two methods, but with a different set of coefficients from a set of infinite solutions that satisfy the overall second order Runge-Kutta method.

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h$$

where $a_2 = \frac{2}{3}$ is chosen, giving;

$$a_1 = \frac{1}{3}$$

$$p_1 = \frac{3}{4}$$

$$q_{11} = \frac{3}{4}$$

resulting in;

$$y_{i+1} = y_i + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h$$

where;

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h)$$

```
function [t,Vout] = Ralston(func,i0,tf,h,R,L)
    %R resistor value
    %L inductor value
    %tf stop time
    %i0 inital current value
    %func function handle
    ti = 0;                                                %initial time
    N = round((tf-ti)/h);                                  %N = number of
iterations
    t(1)=ti;                                               %set t(1) as
initial time value, ti
    i = zeros(1,N);                                        %initialise i
array to zeros
    i(1) = i0;                                             %set i(1) as
initial current value, i0
    Vout(1) = L*func(t(1),i(1));                           %initial value
of output voltage calculated->Vout(1)=L*di(1)/dt(1);

    for j=1:N-1                                            % loop for N
iterations
        t(j+1) = t(j) + h;                                 %Updating the
next value of time with t(j) + h, where t(j) is the previous time
value and h is the step size
        k1 = func(t(j),i(j));                              %gradient at
t(j)
        k2 = func(t(j) + 2*h/3, i(j) + 2*h*k1/3);          %second
gradient at t(j) + 2*h/3
        i(j+1) = i(j) + h*(k1 + 3*k2)/4;                   %next value of
i calculated from previous values of t,i
        Vout(j+1) = L*func(t(j+1),i(j+1));                 %Next value of
output voltage calculated-> Vout(j+1) = L*di(j+1)/dt(j+1)
    end
end
```

Fig4: *MATLAB code Ralston.m implementing the Ralston method discussed in this section*

1.4: RK2_script.m

```
i0=0;                                                          %initial
current value
R=0.5;                                                         %resistor
value = 0.5ohm
L=0.0015;                                                      %Inductor
value = 1.5mH
tf =  5 * 10^-5;                                               %stop
time
h  = 0.000000001;                                              %step
size

Vin = @(t) 3.5*exp(-(t.^2)/(150e-12));                         %input
voltage

func=@(t,i) (1/L)*(Vin(t)-R*i);                                %function
handle->i'=(1/L)*(Vin(t)-R*i)

[Heun_t,Heun_Vout] = Heun(func,i0,tf,h,R,L);                    %calling
Heun function

figure(1)
plot(Heun_t,Heun_Vout);                                        %plotting
output voltage vs time for Heun method
hold on
title('Vout against time for Heun method');
xlabel('Time / s');
ylabel('Vout / V');
legend('Vout', 'Vin');


[Midpoint_t,Midpoint_Vout] = Midpoint(func,i0,tf,h,R,L);        %calling
Midpoint function

figure(2)
plot(Midpoint_t,Midpoint_Vout);                                %plotting
output voltage vs time for Midpoint method
hold on
title('Vout against time for Midpoint method');
xlabel('Time / s');
ylabel('Vout / V');
legend('Vout', 'Vin');
```

Fig5: *MATLAB code RK2_Script.m implementing the RK2_Script.m discussed in this section (1)*

```
[Ralston_t,Ralston_Vout] = Ralston(func,i0,tf,h,R,L);                    %calling
Ralston function

figure(3)
plot(Ralston_t,Ralston_Vout);                                            %plotting
output voltage vs time for Ralston method
hold on
title('Vout against time for Ralston method');
xlabel('Time / s');
ylabel('Vout / V');
legend('Vout', 'Vin');
```

Fig6: *MATLAB code RK2_Script.m implementing the RK2_Script.m discussed in this section (2, cont..)*

1.5: Heun Method Plots

1.5.1: Step signal with amplitude $V_{in} = 3.5\ V$

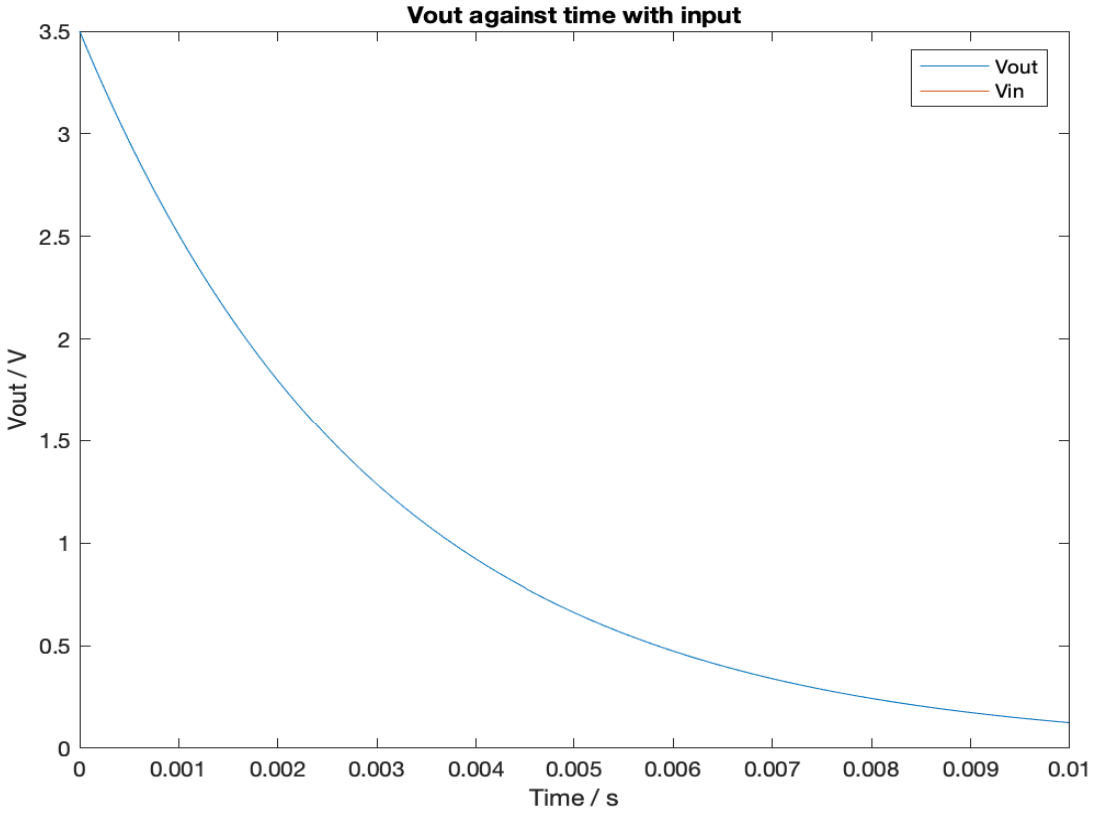

Fig7: *$V_{out} = V_L(t)$ across the inductor when the input is a step signal input voltage $V_{in} = 3.5$*

The following parameters were changed in the MATLAB code to implement the graph in Fig7:

Vin = @(t) 3.5
ti = 0
i0 = 0
tf = 0.01
h = 0.00001

1.5.2: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$

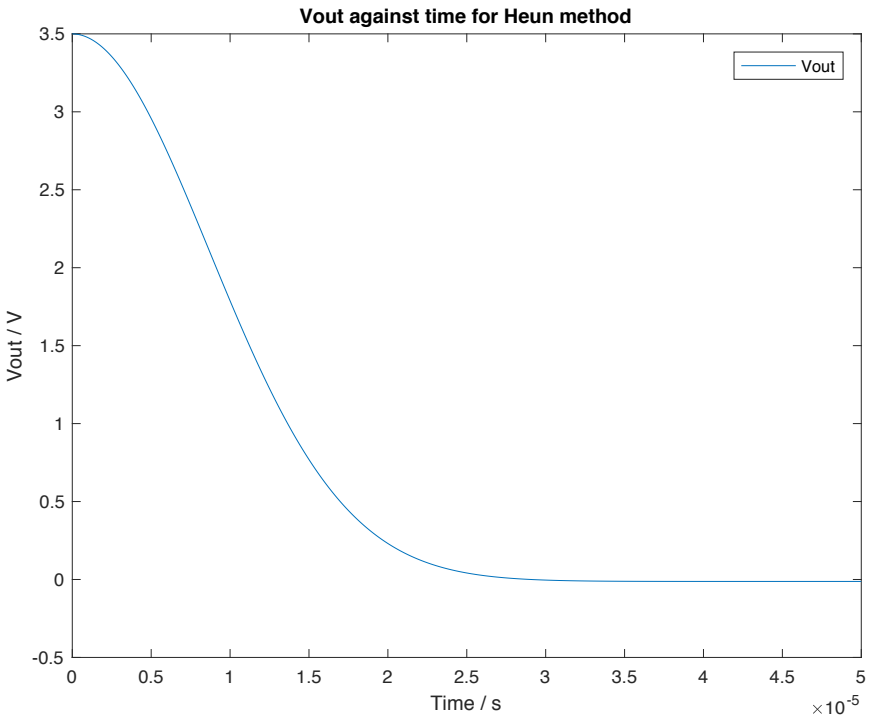

Fig8: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$*

The following parameters were changed in the MATLAB code to implement the graph in Fig8:

Vin = @(t) 3.5*exp(-(t.^2)/(150e-12))
ti = 0
i0 = 0
tf = 5 * 10^-5
h = 0.000000001

1.5.3: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ \mu s$

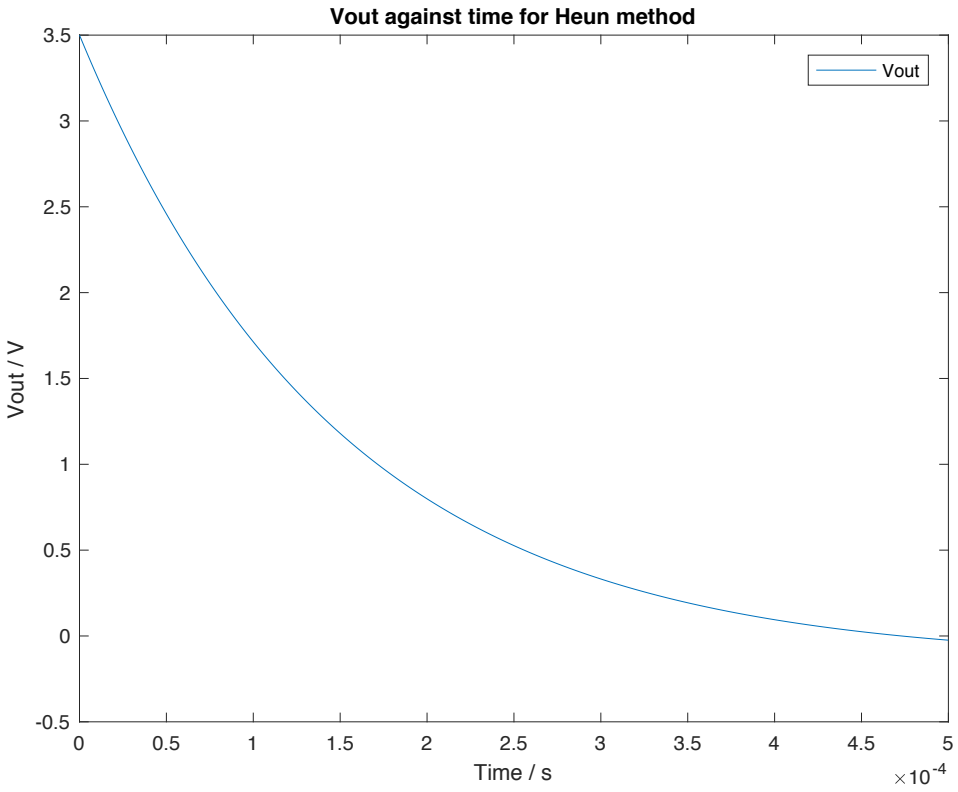

Fig9: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ (\mu s)^2$*

The following parameters were changed in the MATLAB code to implement the graph in Fig9:

Vin = @(t) 3.5*exp(-(t)/(150e-6))
ti = 0
i0 = 0
tf = 5 * 10^-4
h = 0.0000001

1.5.4: Sine wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

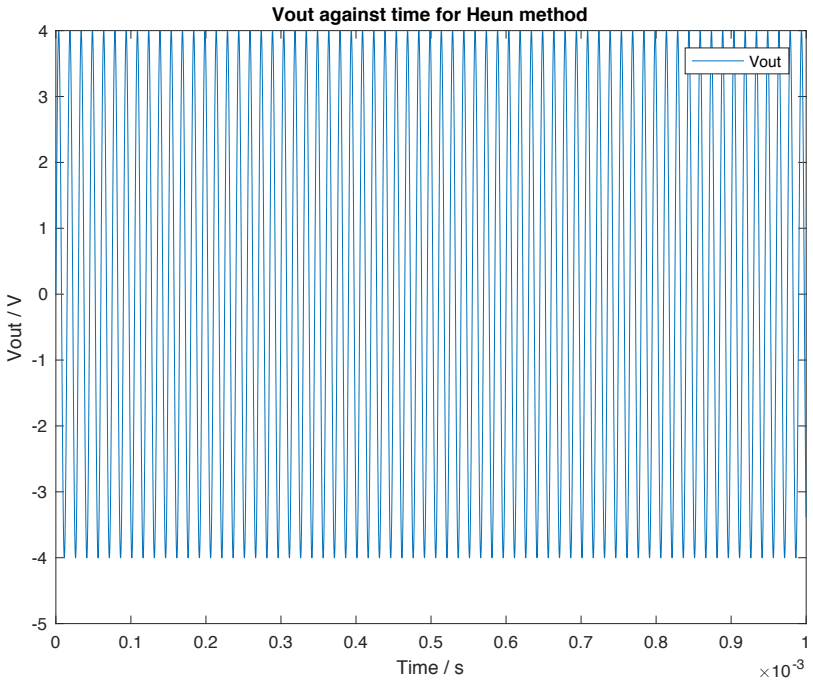

Fig10: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 15\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig10:

Vin = @(t) 4*sin((2*pi/(15e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

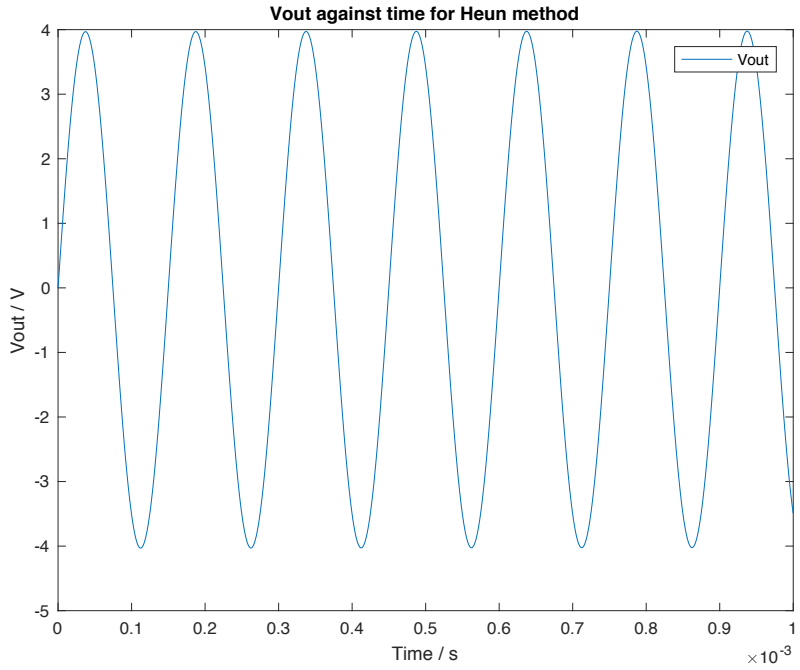

Fig11: $V_{out} = V_L(t)$ *across the inductor when the input is sine wave with amplitude 4 V and time period* $T = 150\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig11:

Vin = @(t) 4*sin((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

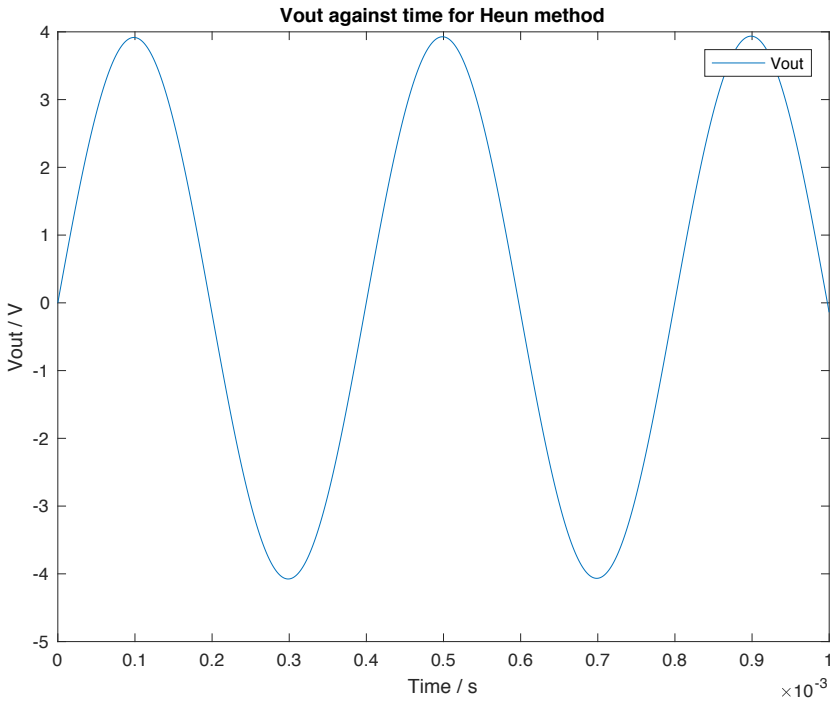

Fig12: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 400\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig12:

Vin = @(t) 4*sin((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

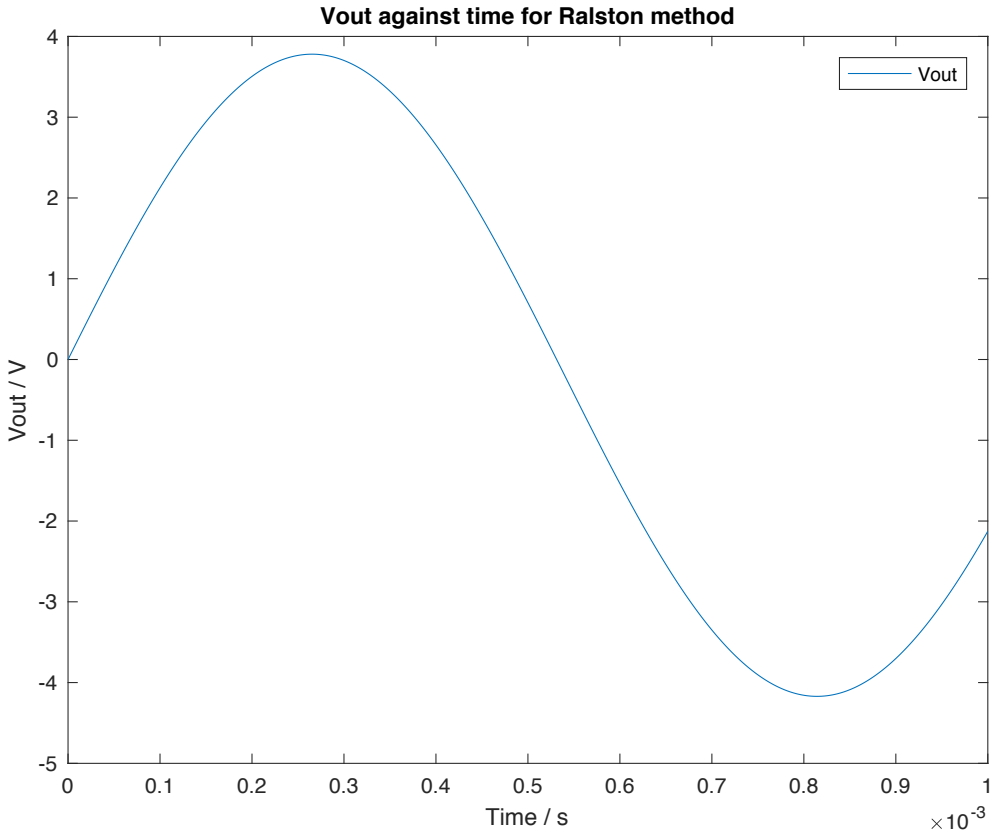

Fig13: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 1100\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig13:

Vin = @(t) 4*sin((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

1.5.5: Square wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

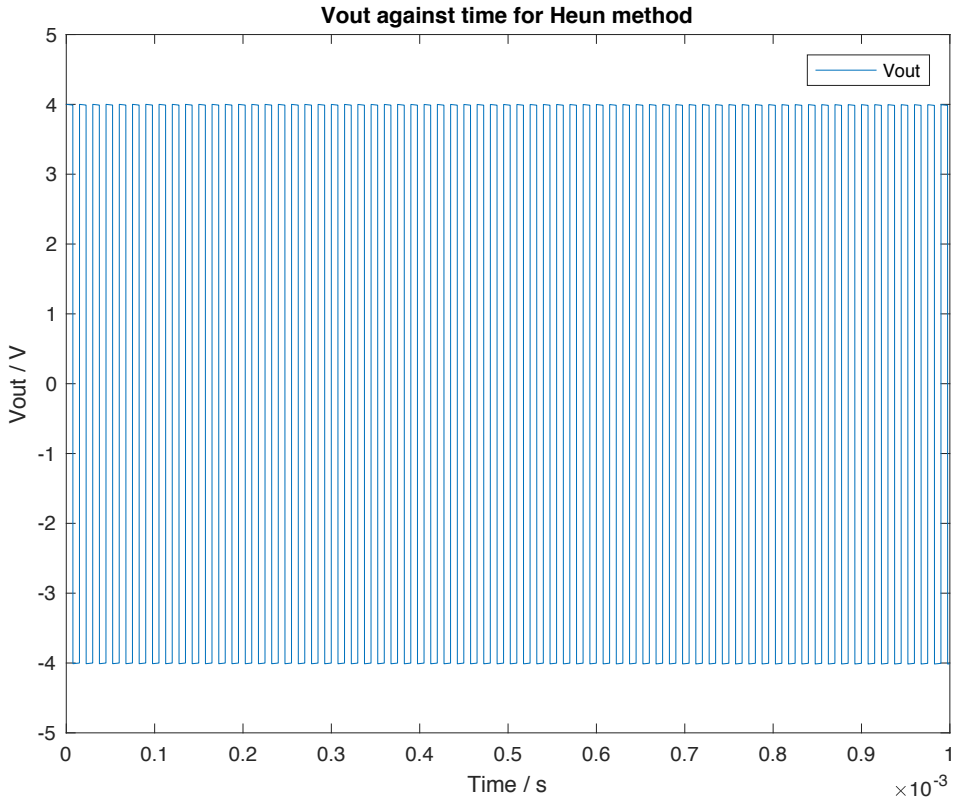

Fig14: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 15\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig14:

Vin = @(t) 4*square((2*pi/(15e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

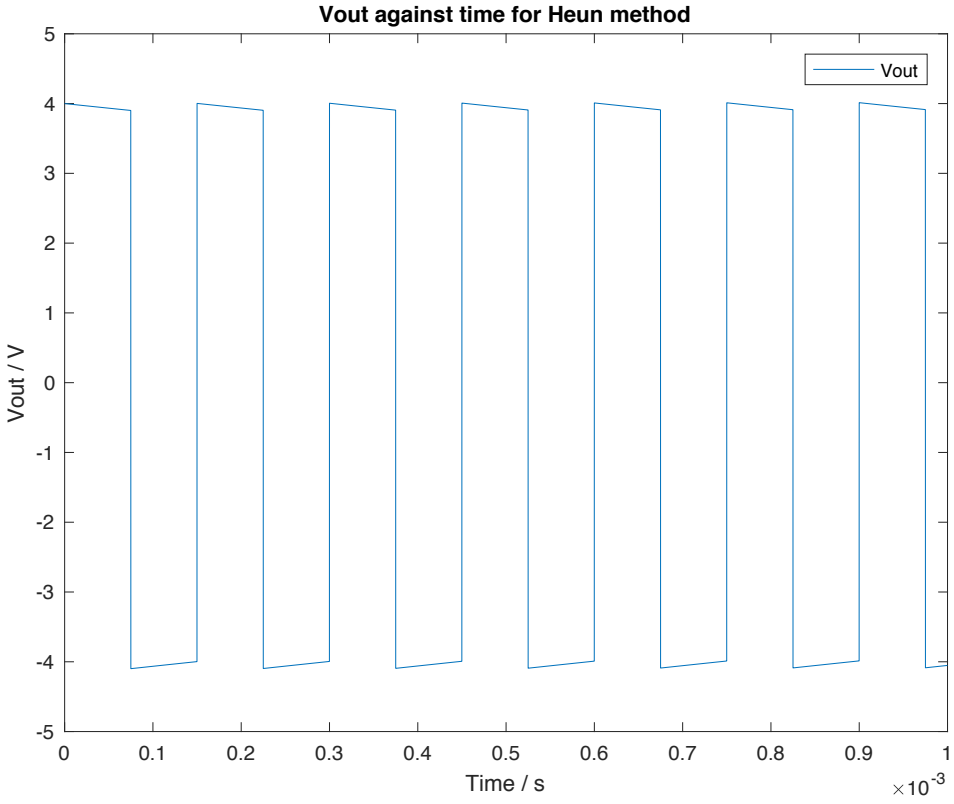

Fig15: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 150\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig15:

Vin = @(t) 4*square((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

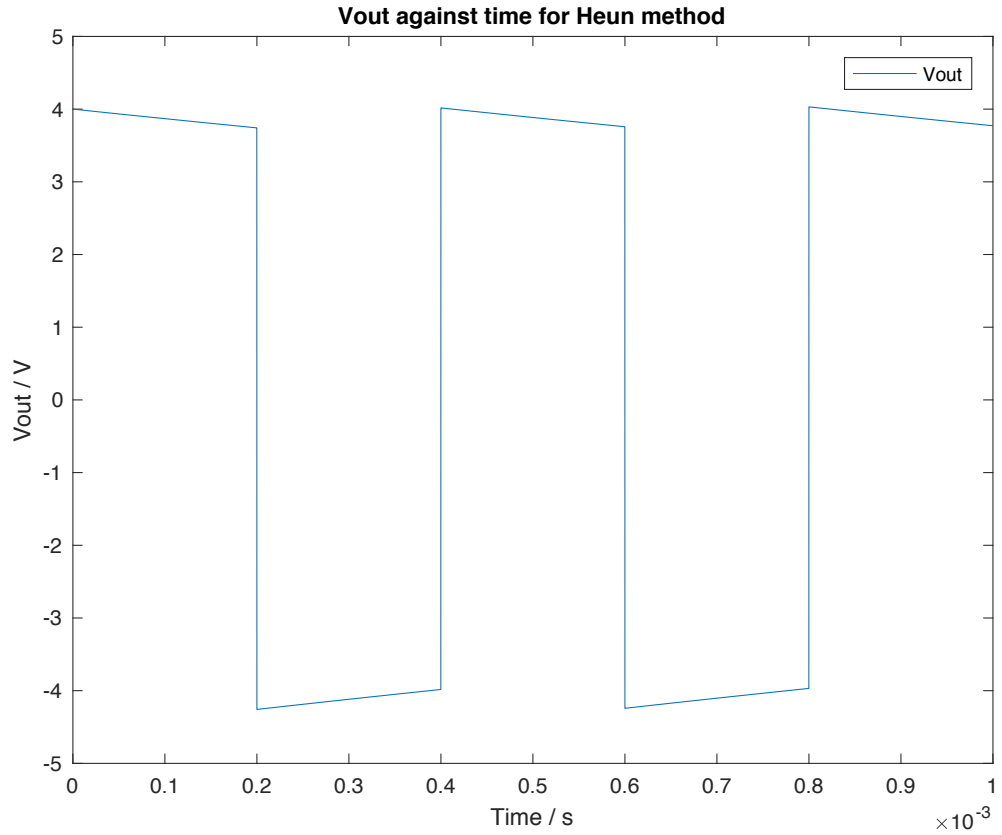

Fig16: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 400\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig16:

Vin = @(t) 4*square((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

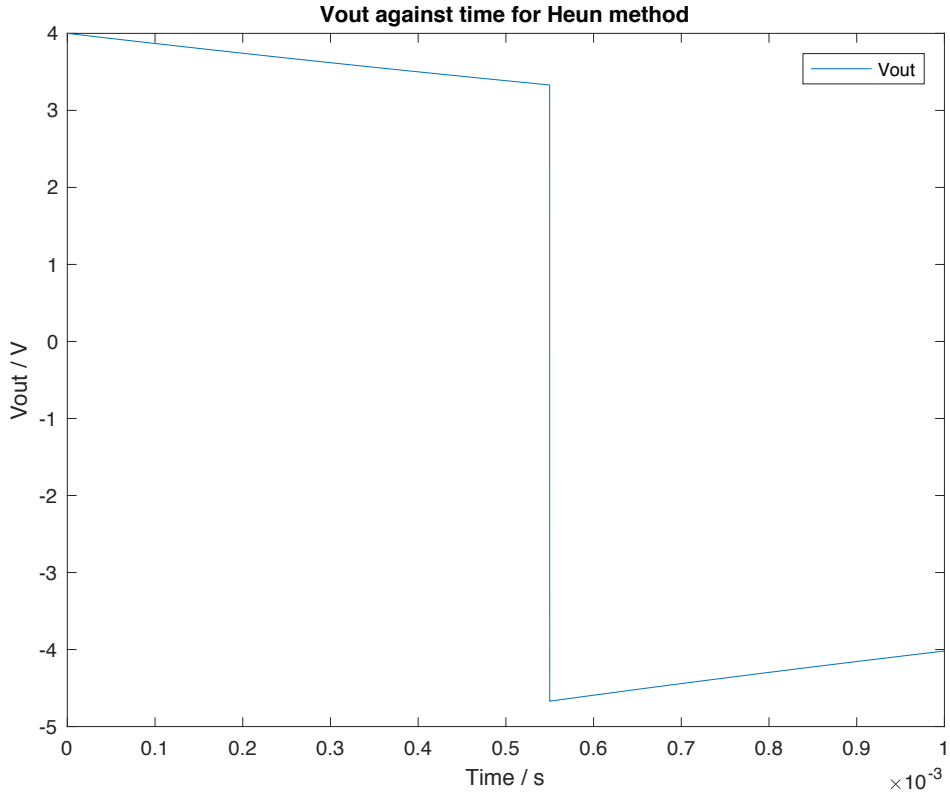

Fig17: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 1100\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig17:

Vin = @(t) 4*square((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

1.5.6: Sawtooth wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

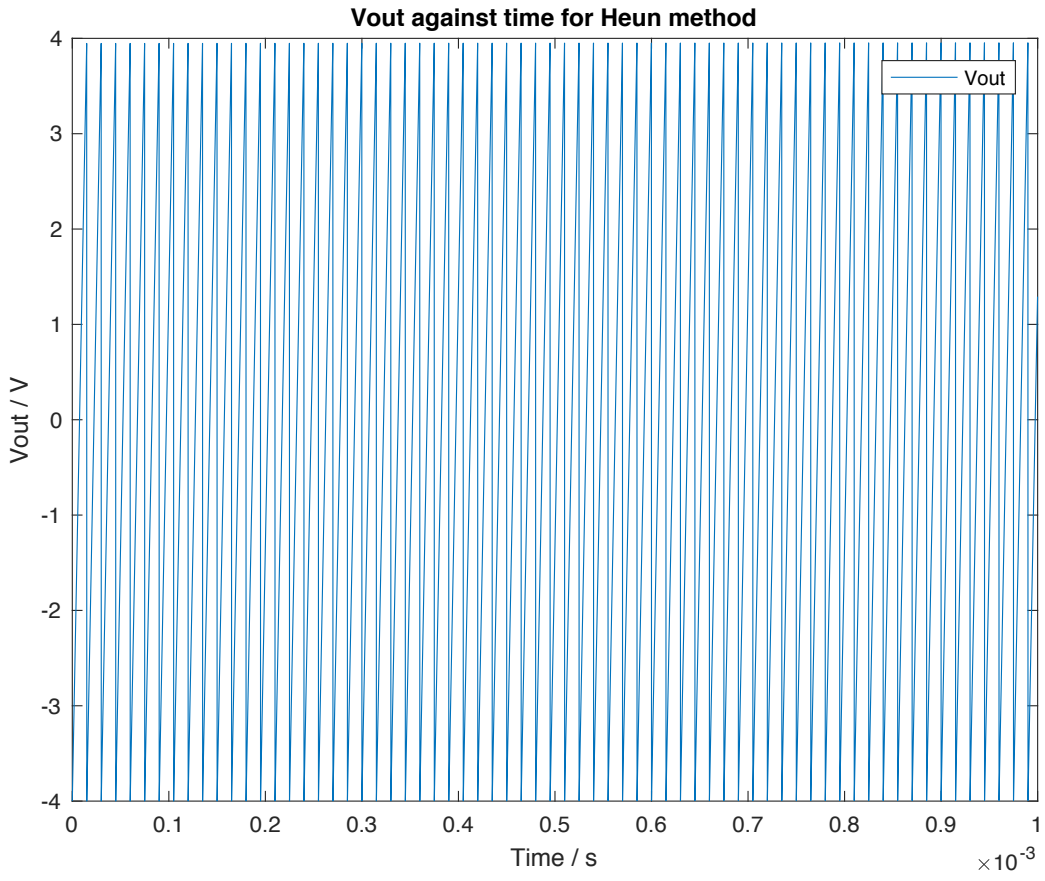

Fig18: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period* $T = 15\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig18:

Vin = @(t) 4*sawtooth((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

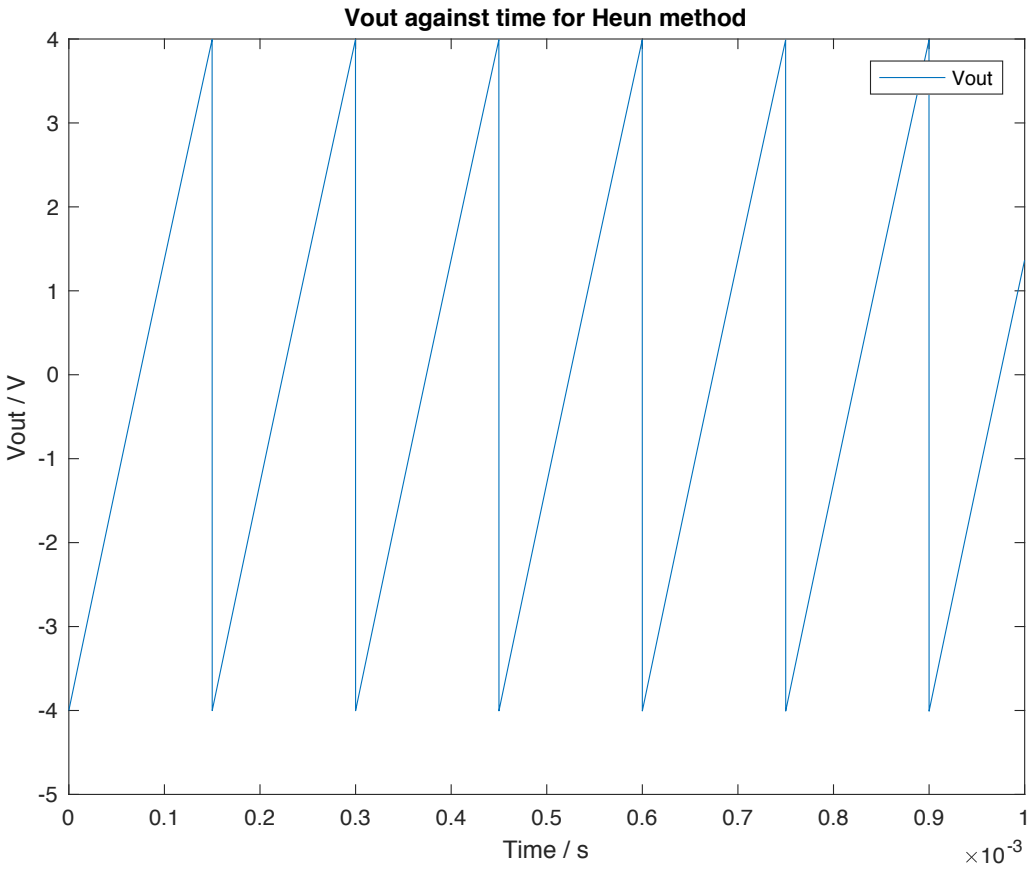

Fig19: $V_{out} = V_L(t)$ *across the inductor when the input is sawtooth wave with amplitude 4 V and time period* $T = 150$ μs

The following parameters were changed in the MATLAB code to implement the graph in Fig19:

Vin = @(t) 4*sawtooth((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

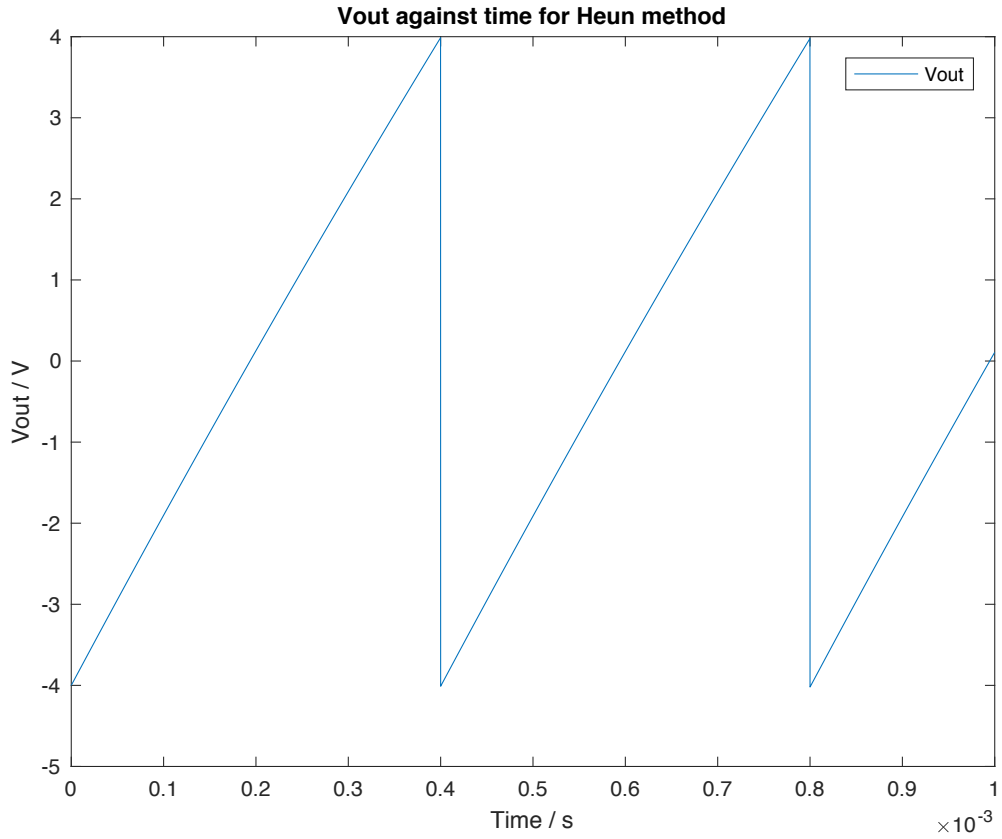

Fig20: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 400\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig20:

Vin = @(t) 4*sawtooth((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

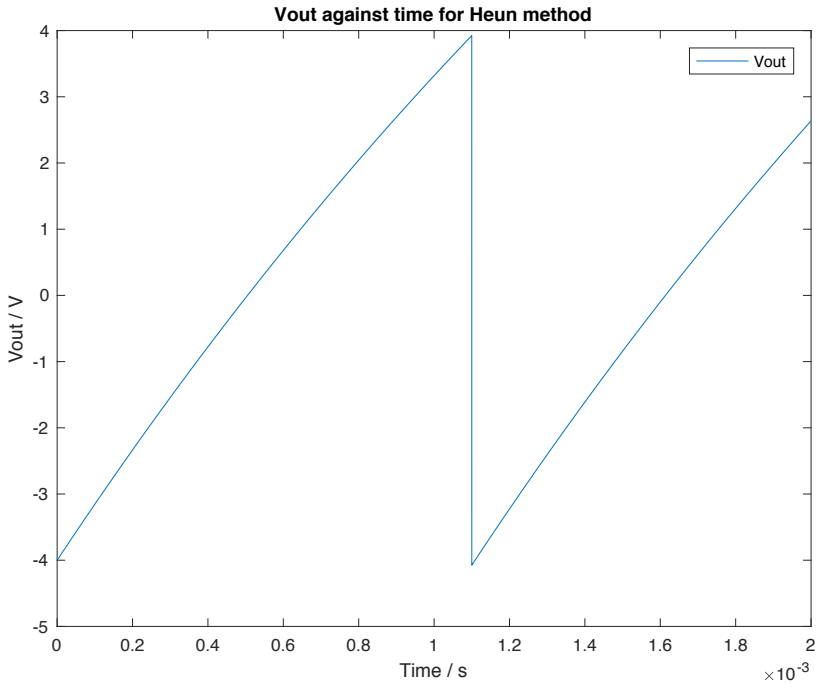

Fig21: $V_{out} = V_L(t)$ *across the inductor when the input is sawtooth wave with amplitude 4 V and time period* $T = 1100\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig21:

Vin = @(t) 4*sawtooth((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.002
h = 0.0000001

1.6: Midpoint Method Plots

1.6.1: Step signal with amplitude $V_{in} = 3.5\ V$

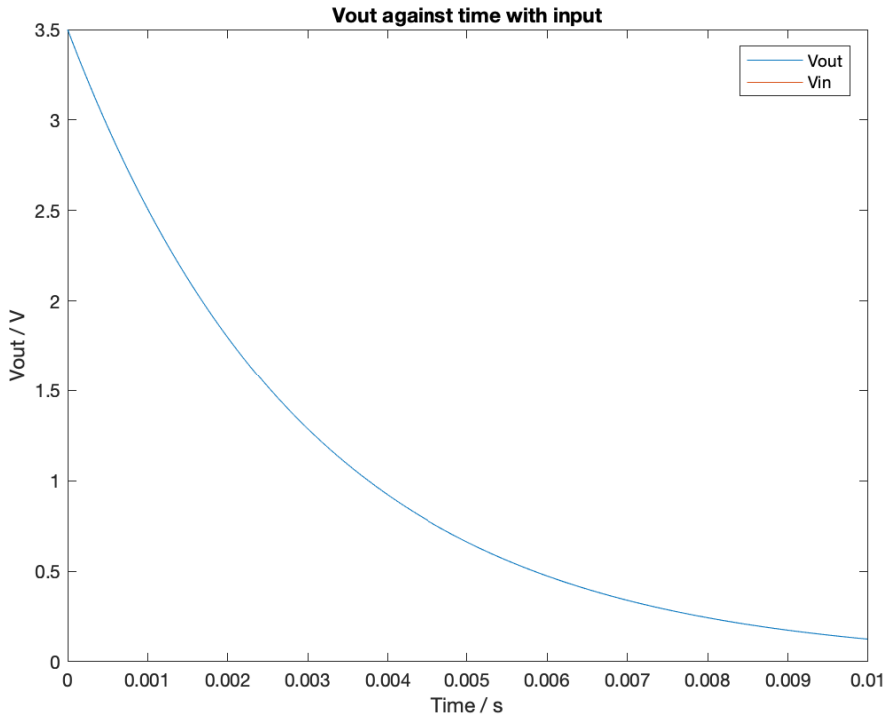

Fig22: *$V_{out} = V_L(t)$ across the inductor when the input is a step signal input voltage $V_{in} = 3.5$*

The following parameters were changed in the MATLAB code to implement the graph in Fig22:

Vin = @(t) 3.5
ti = 0
i0 = 0
tf = 0.01
h = 0.00001

1.6.2: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$

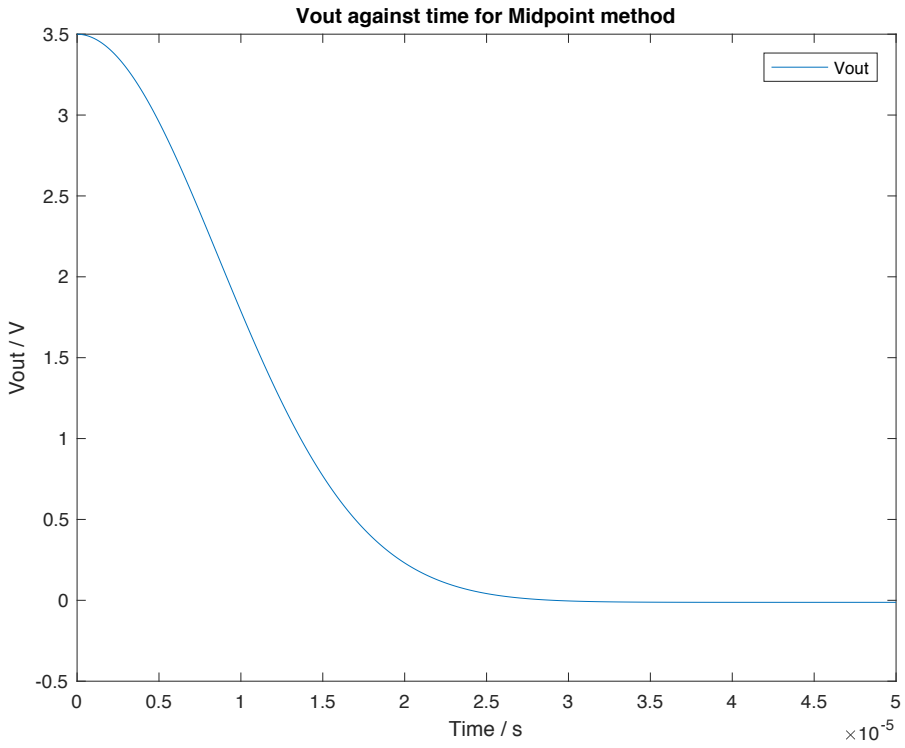

Fig23: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$*

The following parameters were changed in the MATLAB code to implement the graph in Fig23:

Vin = @(t) 3.5*exp(-(t.^2)/(150e-12))
ti = 0
i0 = 0
tf = 5 * 10^-5
h = 0.000000001

1.6.3: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ \mu s$

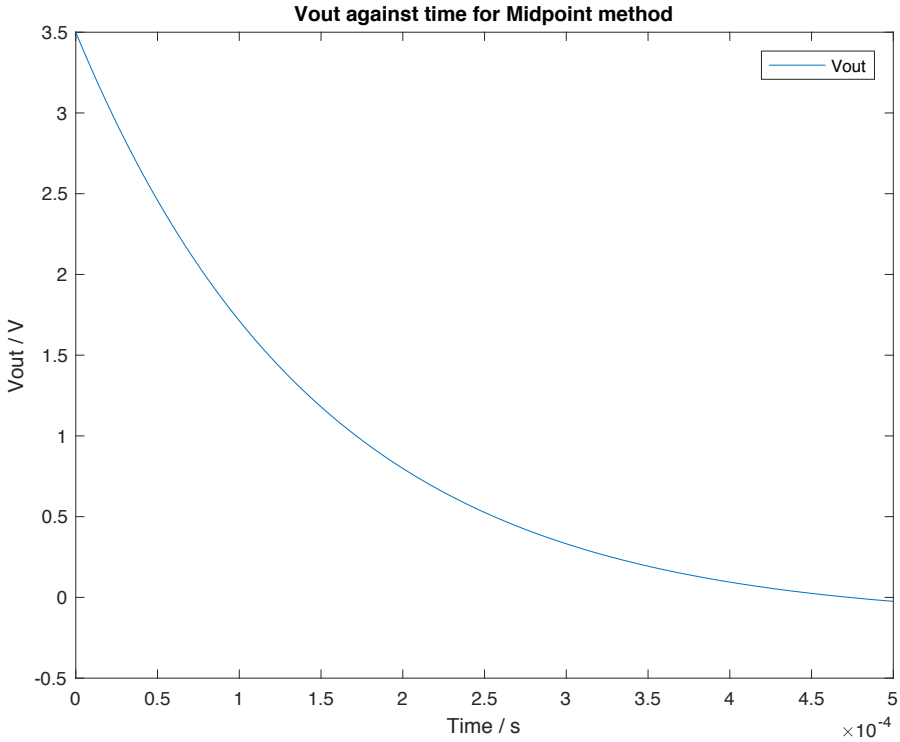

Fig24: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ (\mu s)^2$*

The following parameters were changed in the MATLAB code to implement the graph in Fig24:

Vin = @(t) 3.5*exp(-(t)/(150e-6))
ti = 0
i0 = 0
tf = 5 * 10^-4
h = 0.0000001

1.6.4: Sine wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

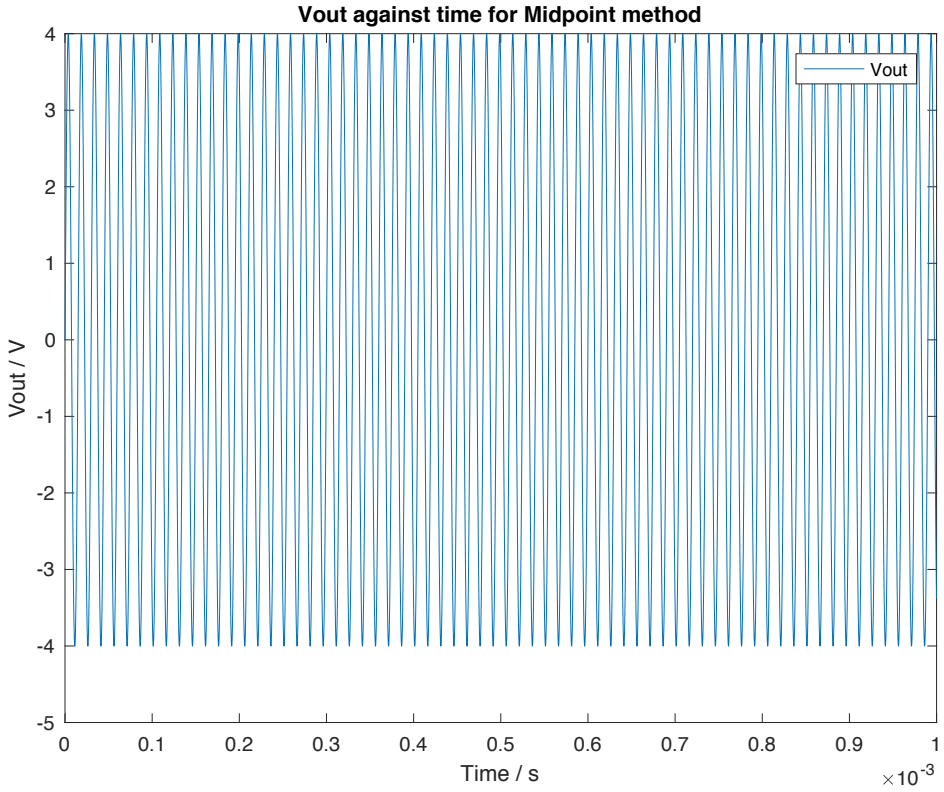

Fig25: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 15\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig25:

Vin = @(t) 4*sin((2*pi/(15e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

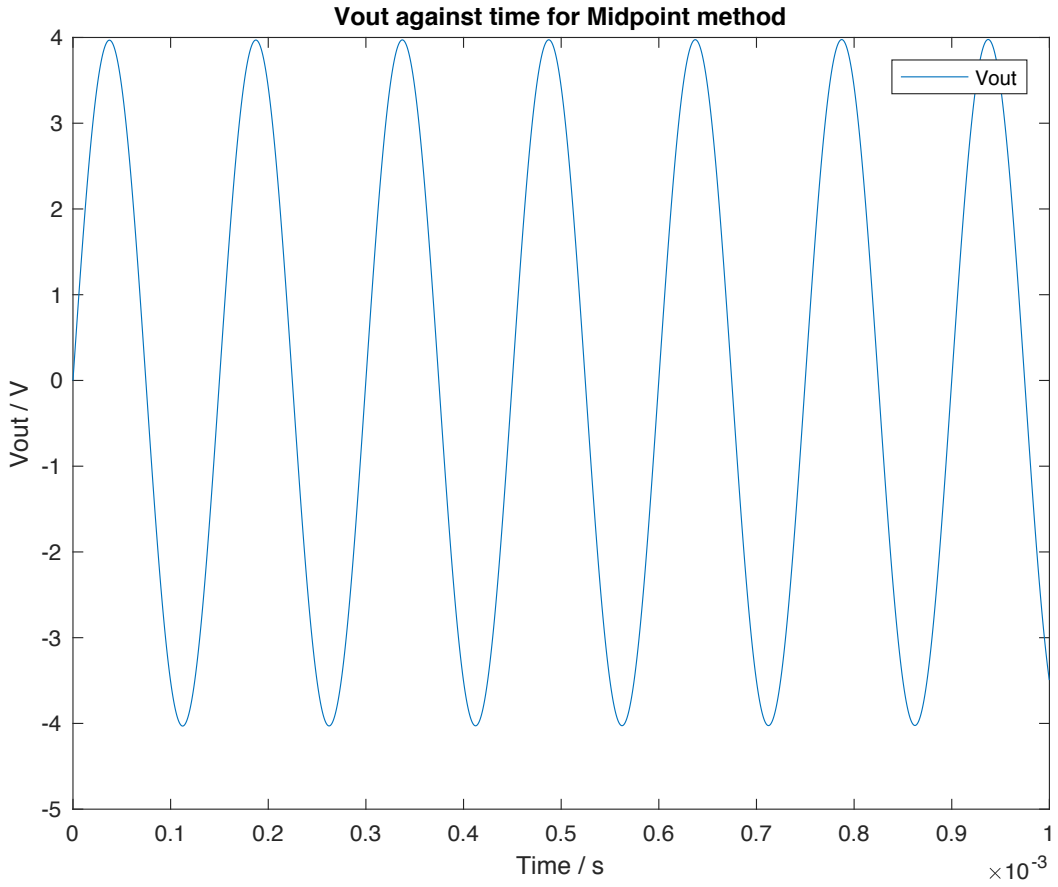

Fig26: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 150$ μs*

The following parameters were changed in the MATLAB code to implement the graph in Fig26:

Vin = @(t) 4*sin((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

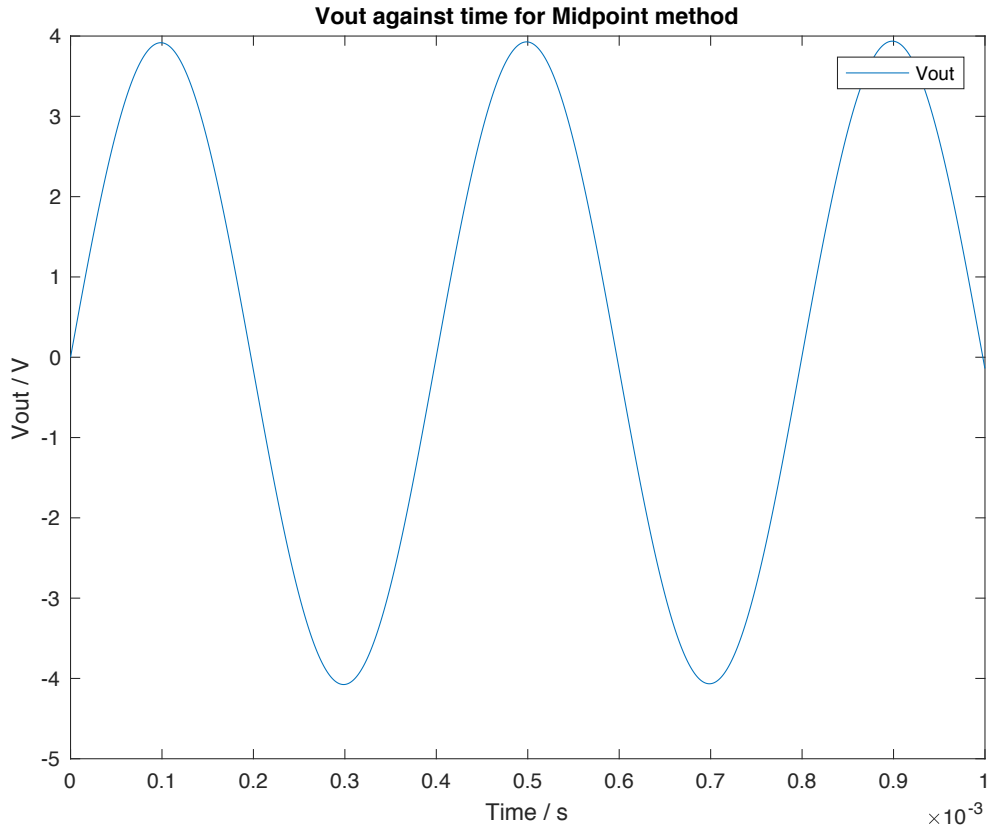

Fig27: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 400\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig27:

Vin = @(t) 4*sin((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

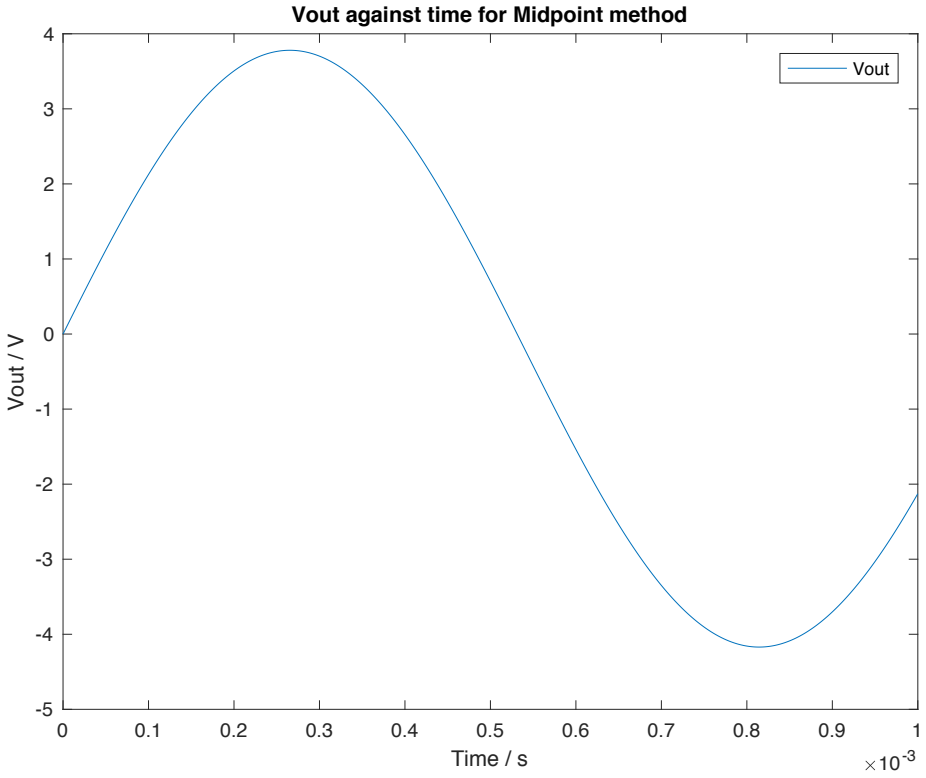

Fig28: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 1100\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig28:

Vin = @(t) 4*sin((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

1.6.5: Square wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

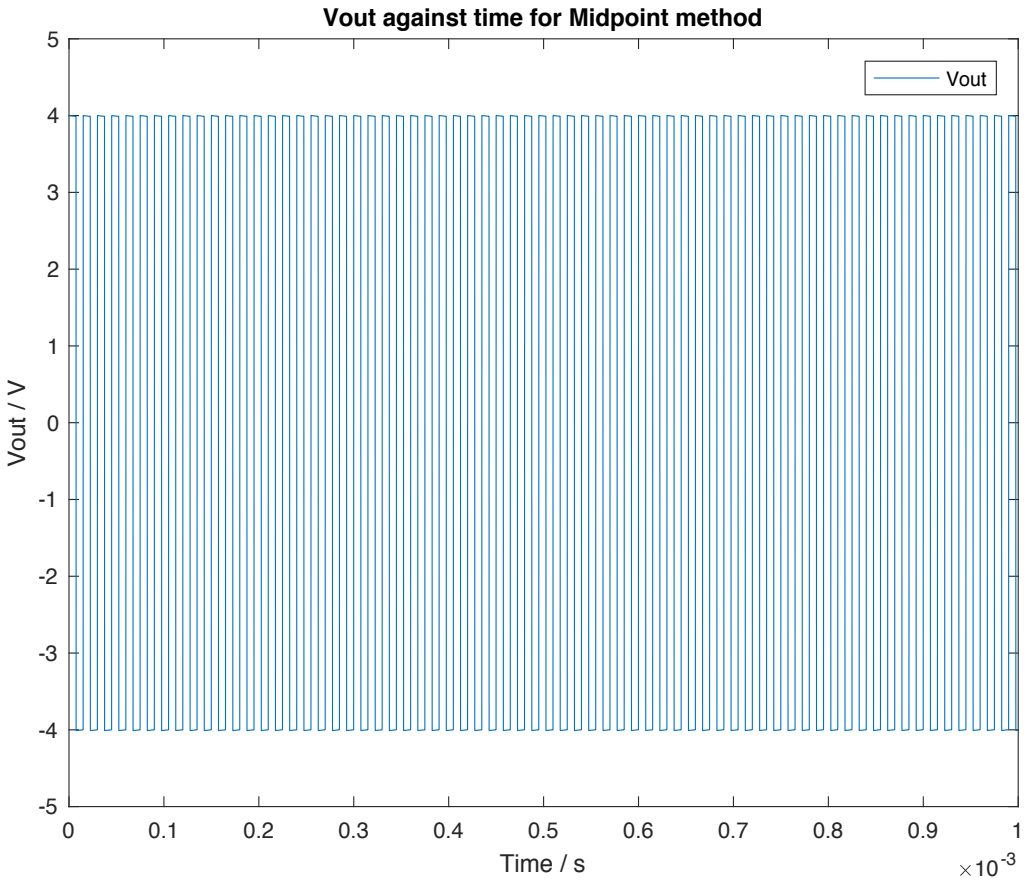

Fig29: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 15\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig29:

Vin = @(t) 4*square((2*pi/(15e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

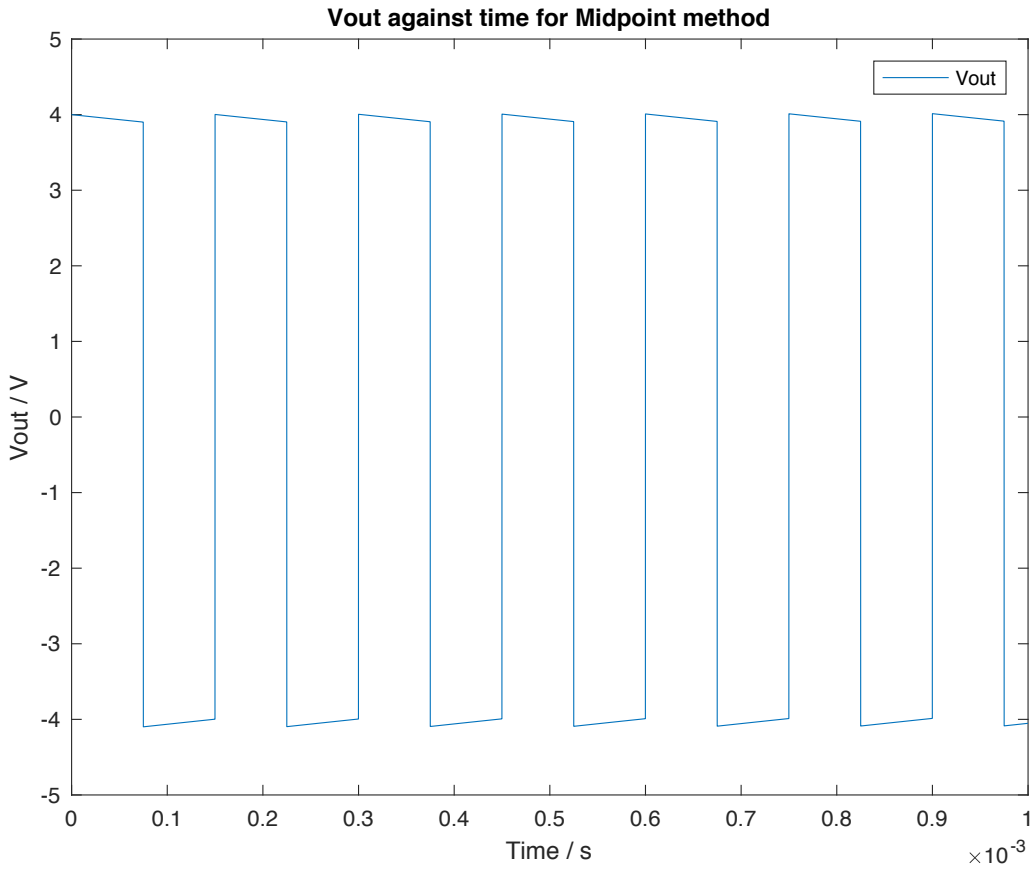

Fig30: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 150\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig30:

Vin = @(t) 4*square((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

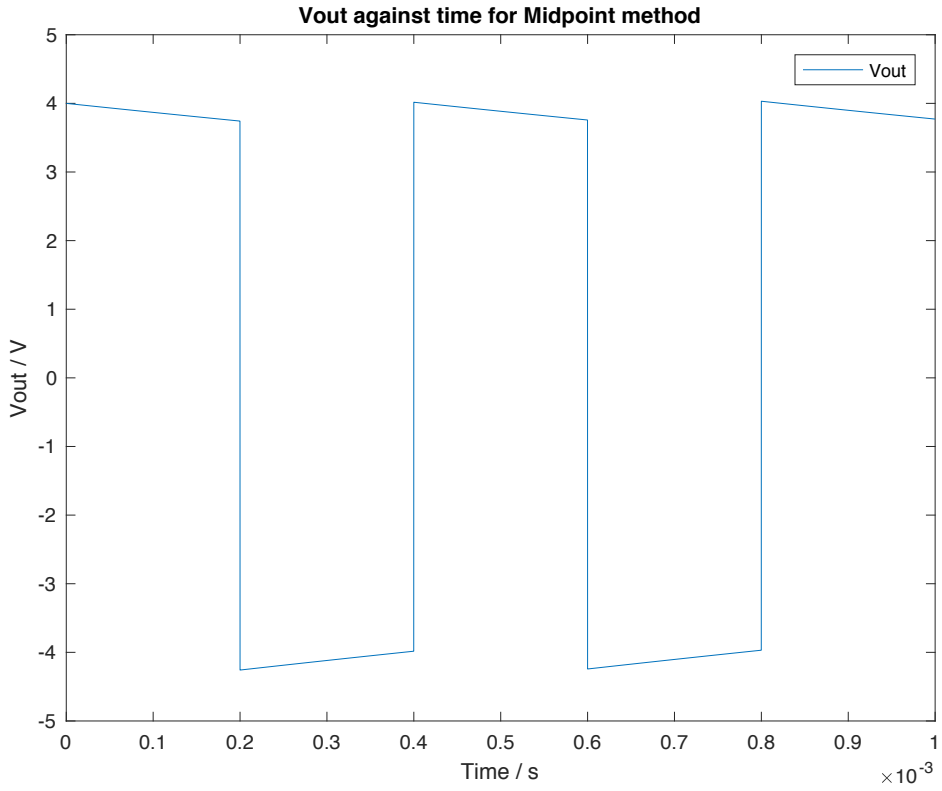

Fig31: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 400\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig31:

Vin = @(t) 4*square((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

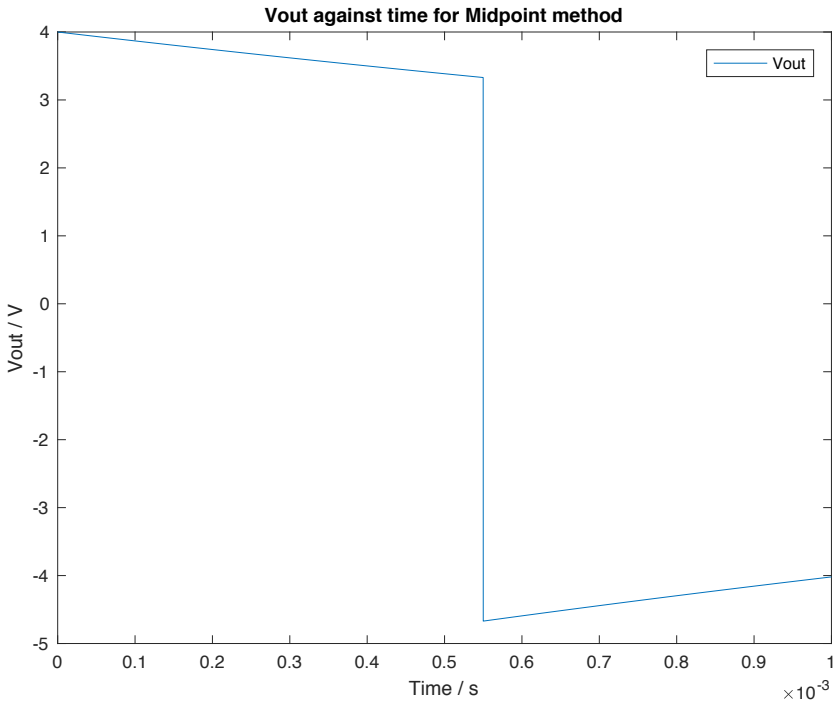

Fig32: $V_{out} = V_L(t)$ *across the inductor when the input is square wave with amplitude 4 V and time period* $T = 1100\ \mu s$

The following parameters were changed in the MATLAB code to implement the graph in Fig32:

Vin = @(t) 4*square((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

1.6.6: Sawtooth wave input with Amplitude 4 $V$and different periods

$T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

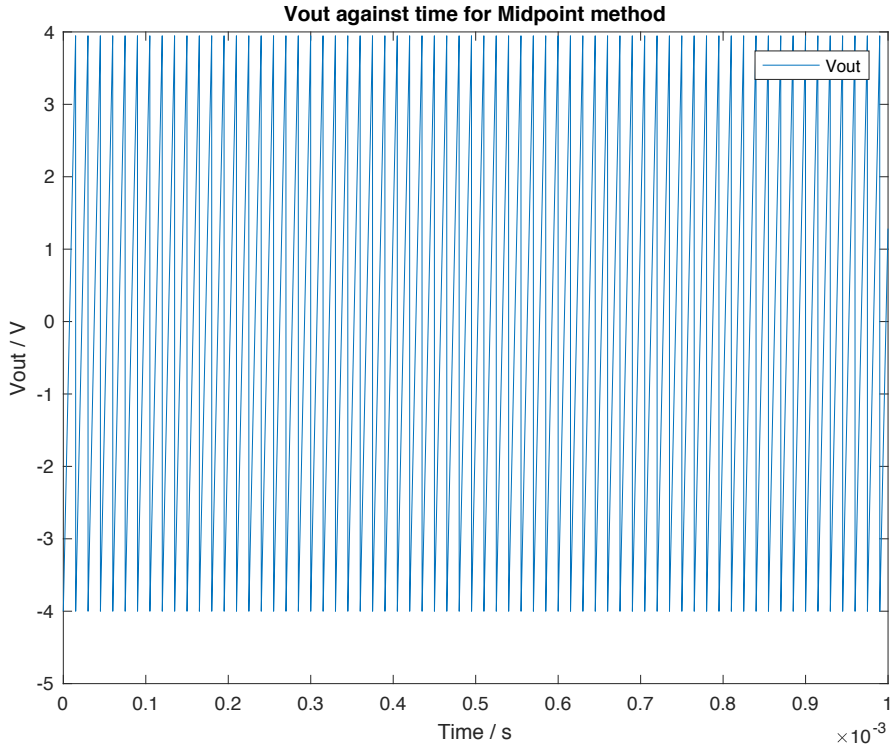

Fig33: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 15\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig33:

Vin = @(t) 4*sawtooth((2*pi/(15e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

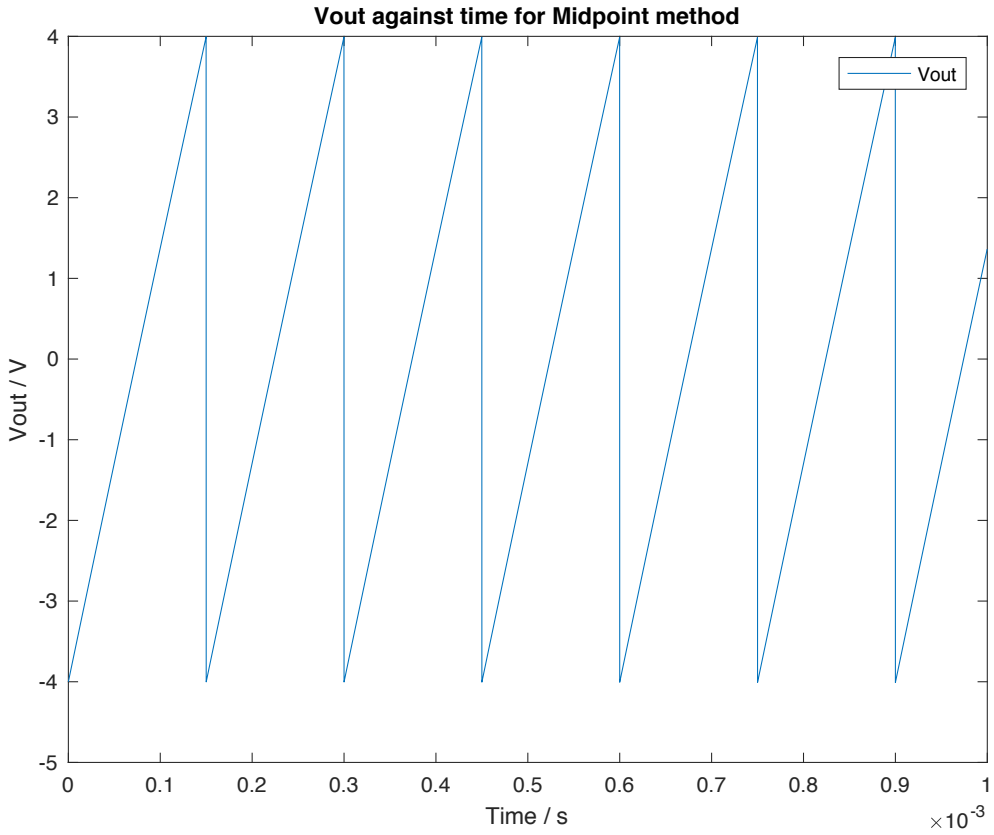

Fig34: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 150\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig34:

Vin = @(t) 4*sawtooth((2*pi/(150e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

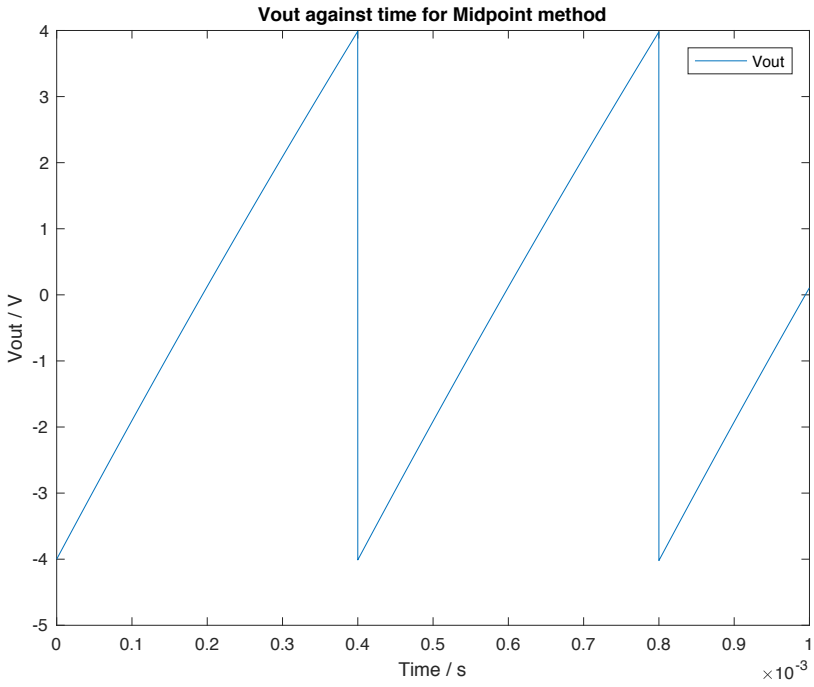

Fig35: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 400$ μs*

The following parameters were changed in the MATLAB code to implement the graph in Fig35:

Vin = @(t) 4*sawtooth((2*pi/(400e-6))*t)
ti = 0
i0 = 0
tf = 0.001
h = 0.0000001

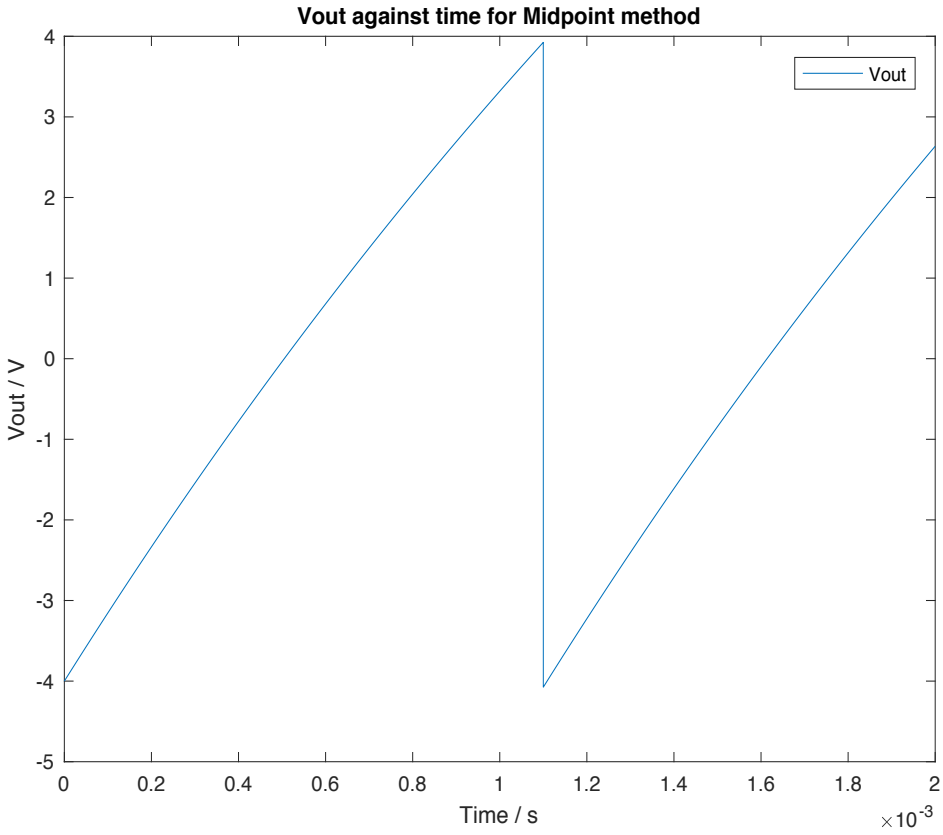

Fig36: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 1100\ \mu s$*

The following parameters were changed in the MATLAB code to implement the graph in Fig36:

Vin = @(t) 4*sawtooth((2*pi/(1100e-6))*t)
ti = 0
i0 = 0
tf = 0.002
h = 0.0000001

1.7: Ralston Method Plots

1.7.1: Step signal with amplitude $V_{in} = 3.5\ V$

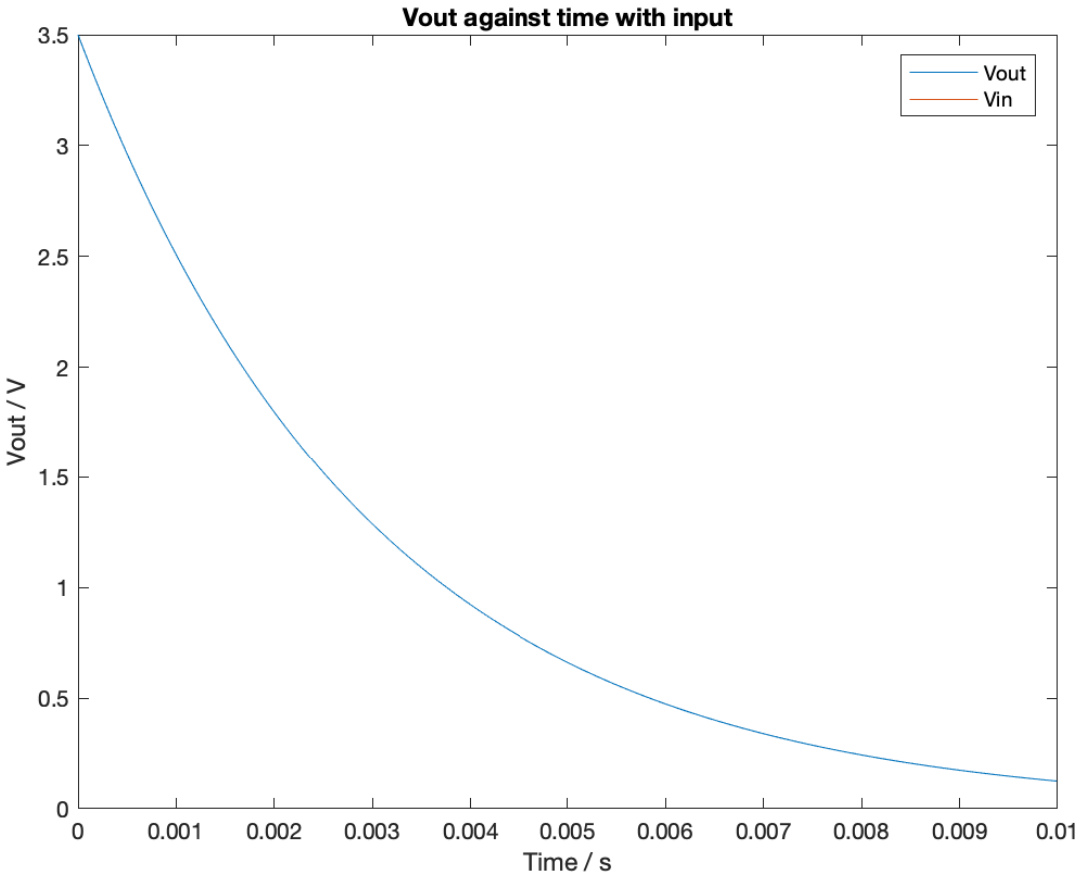

Fig37: *$V_{out} = V_L(t)$ across the inductor when the input is a step signal input voltage $V_{in} = 3.5$*

1.7.2: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$

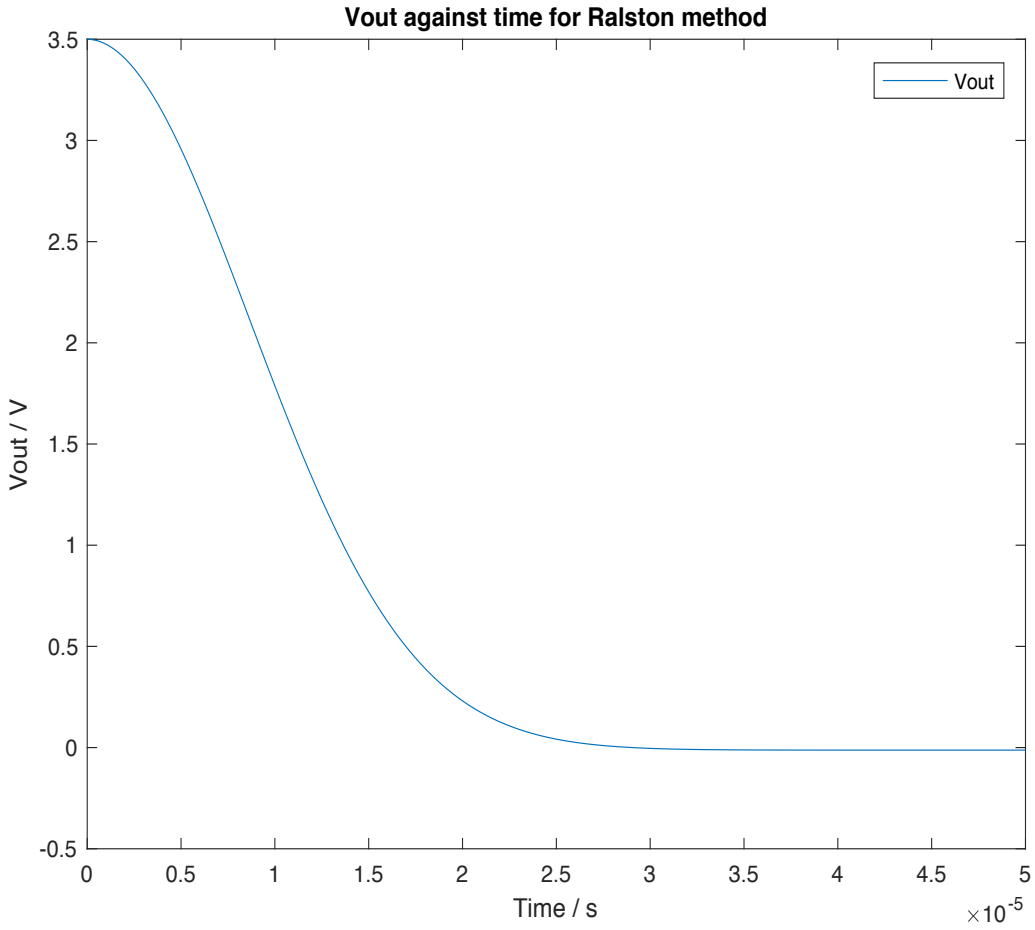

Fig38: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t^2}{\tau})}$ where $\tau = 150\ (\mu s)^2$*

1.7.3: Impulsive signal with decay $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ \mu s$

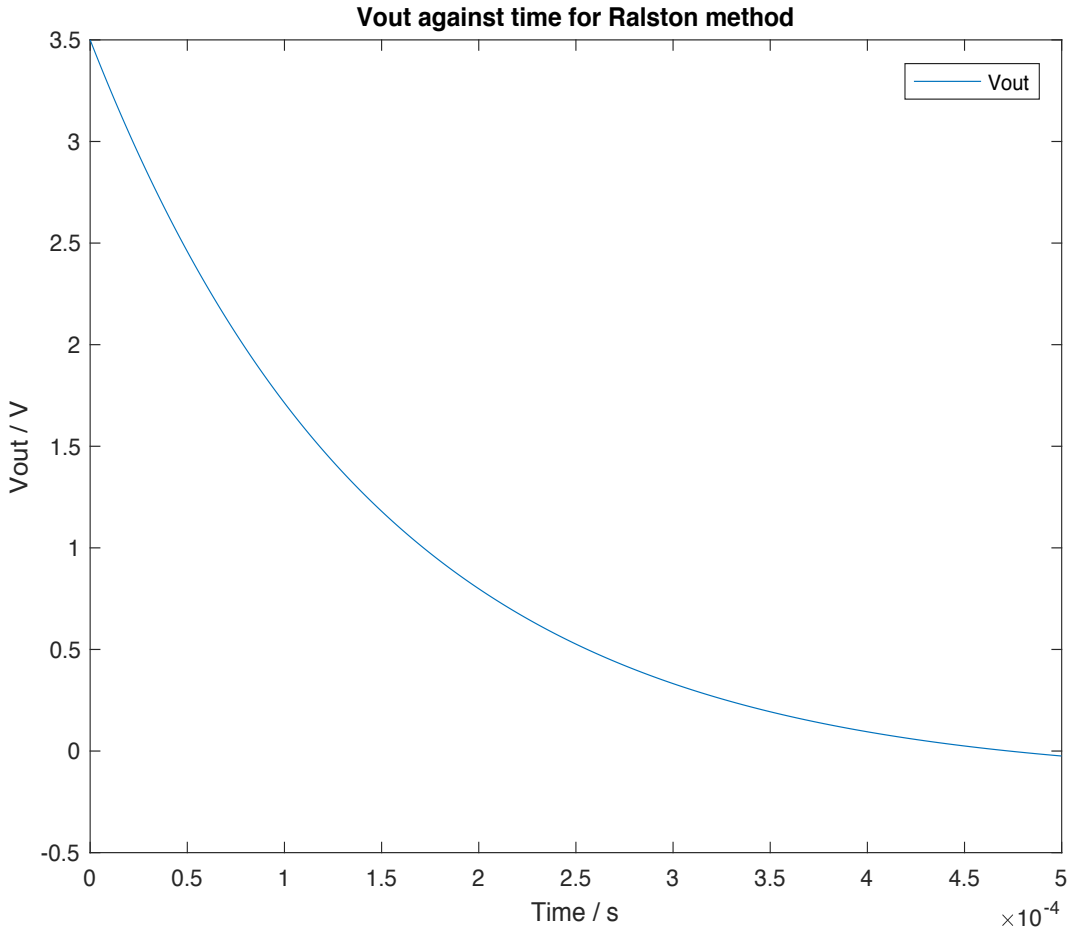

Fig39: *$V_{out} = V_L(t)$ across the inductor when the input is $V_{in} = 3.5e^{(-\frac{t}{\tau})}$ where $\tau = 150\ \mu s$*

1.7.4: Sine wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

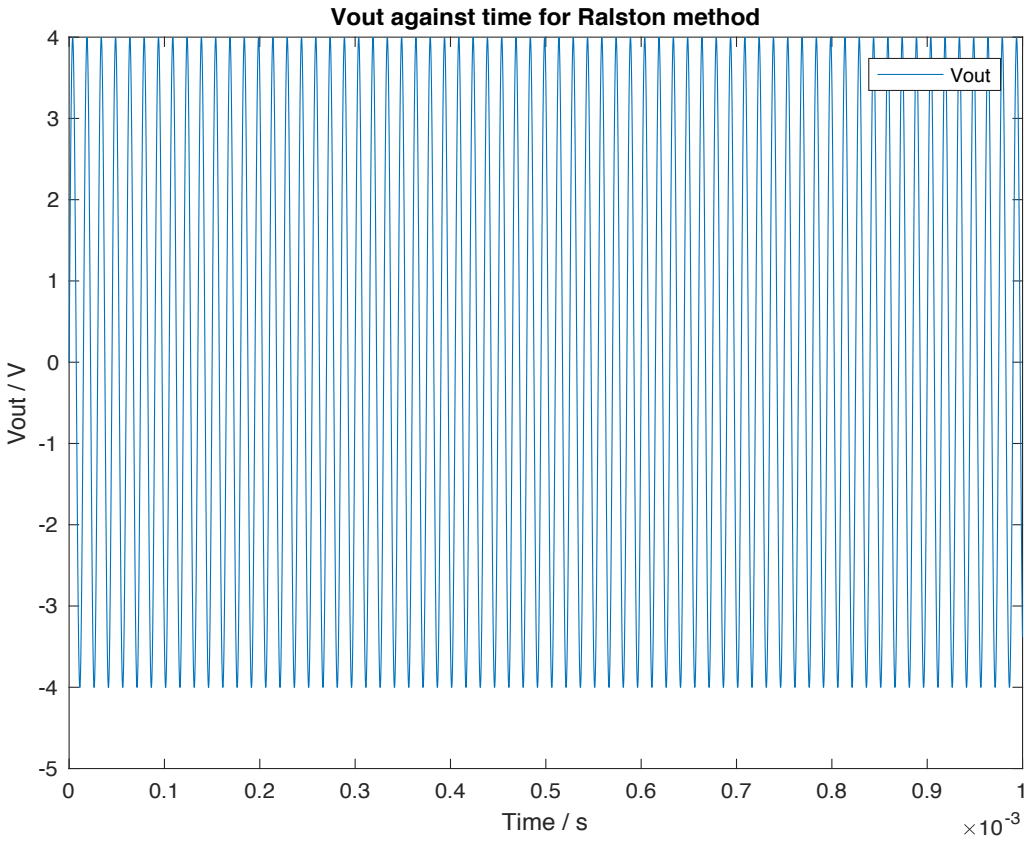

Fig40: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 15\ \mu s$*

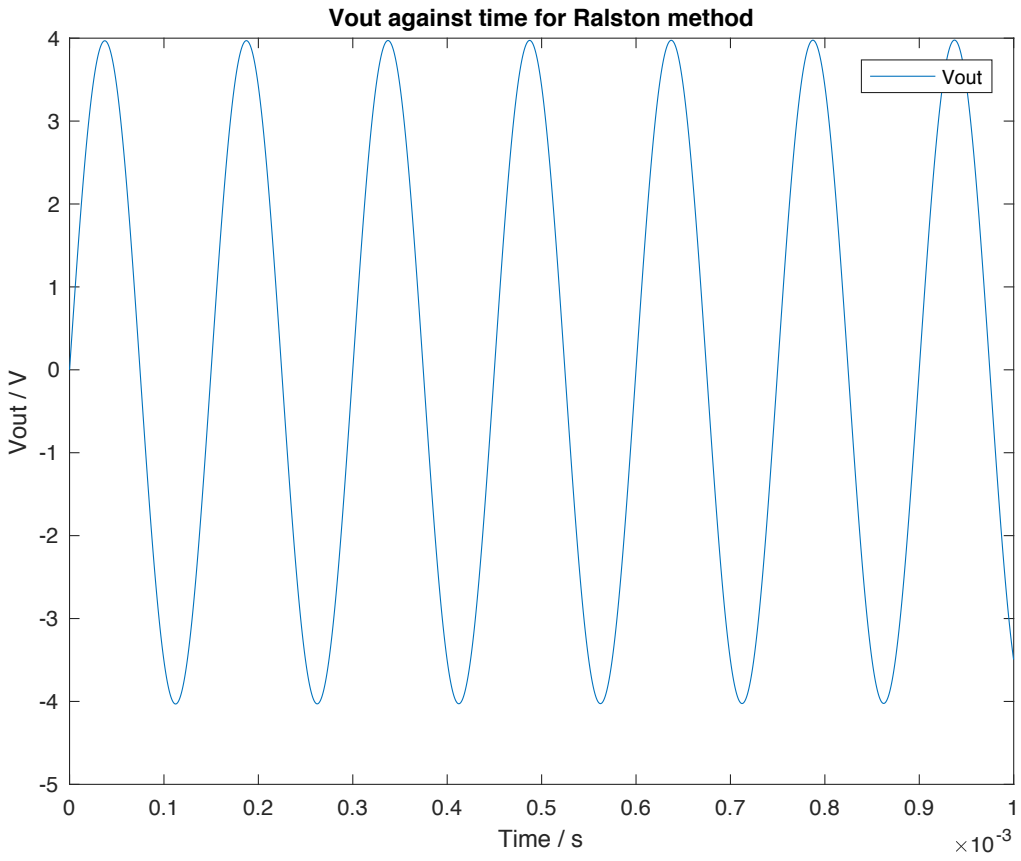

Fig41: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 150\ \mu s$*

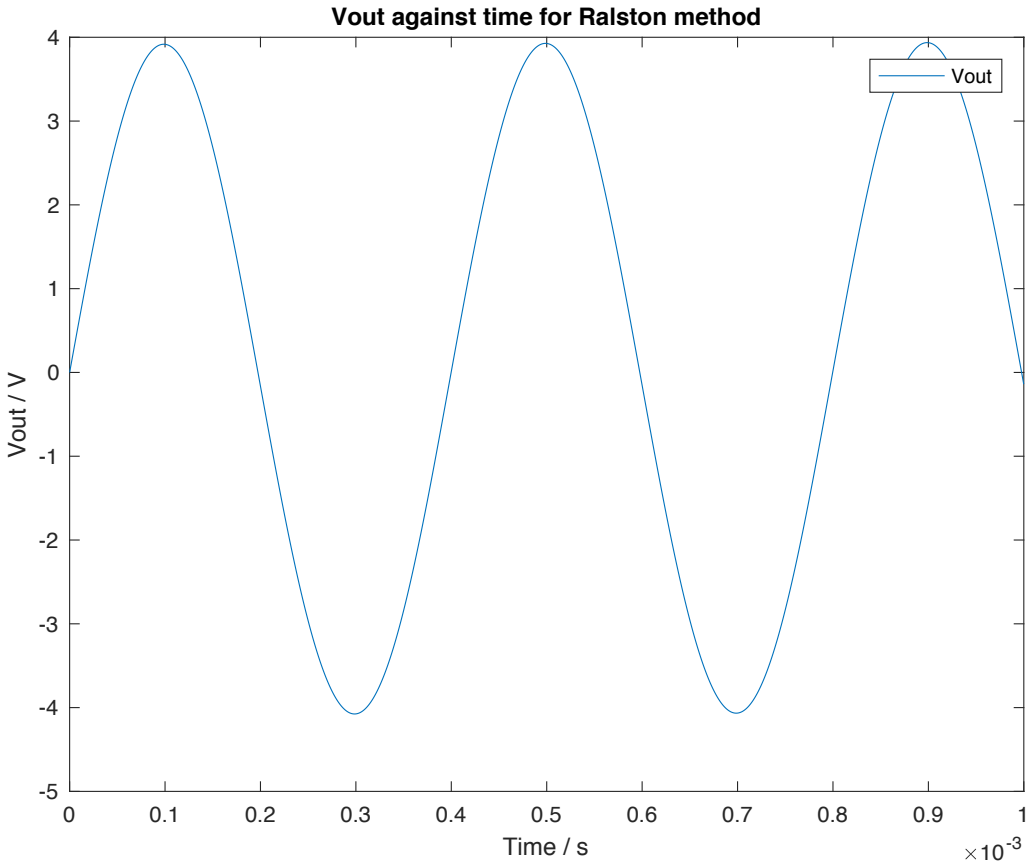

Fig42: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 400\ \mu s$*

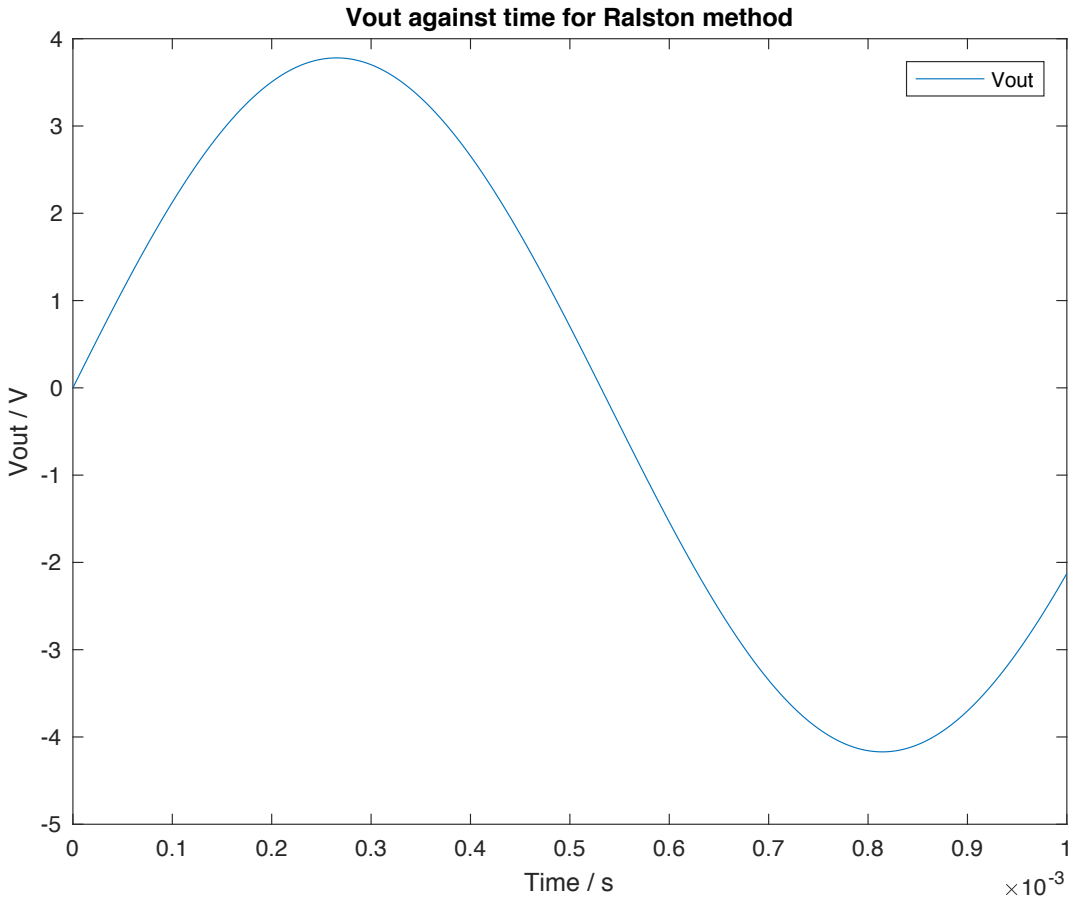

Fig43: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 1100\ \mu s$*

1.7.5: Square wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

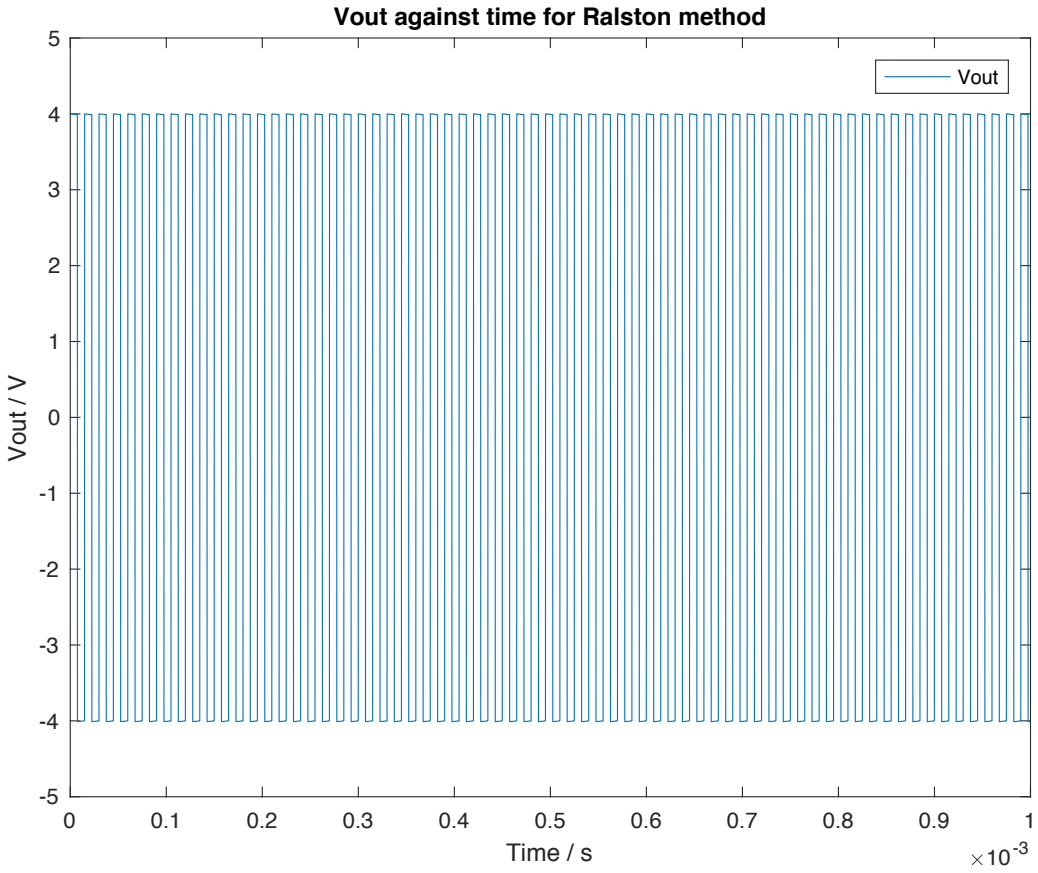

Fig44: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 15\ \mu s$*

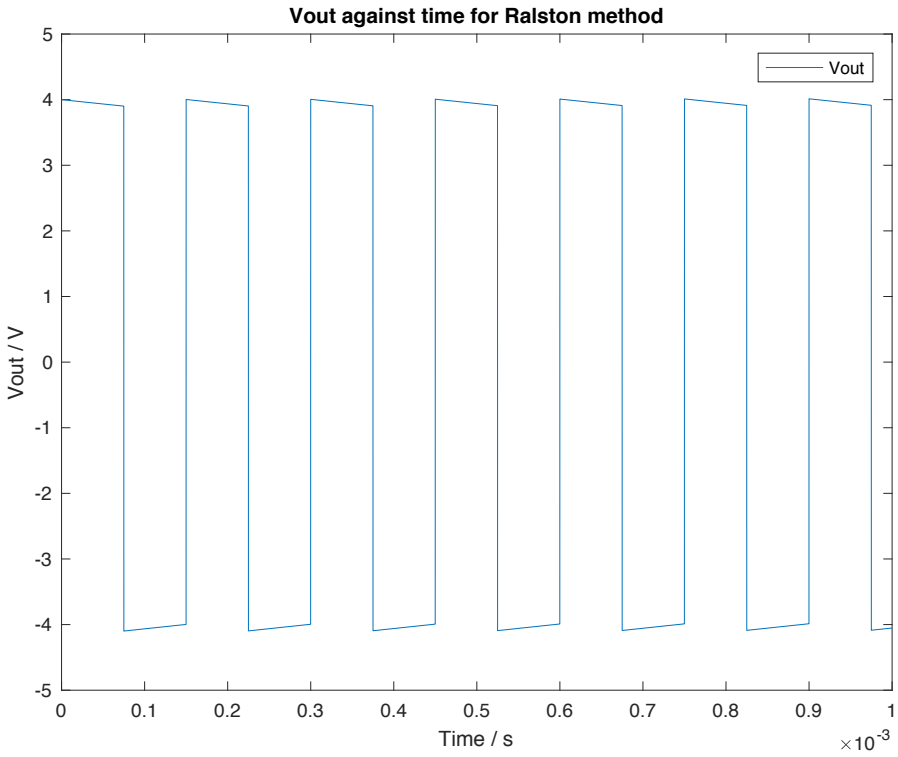

Fig45: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 150\ \mu s$*

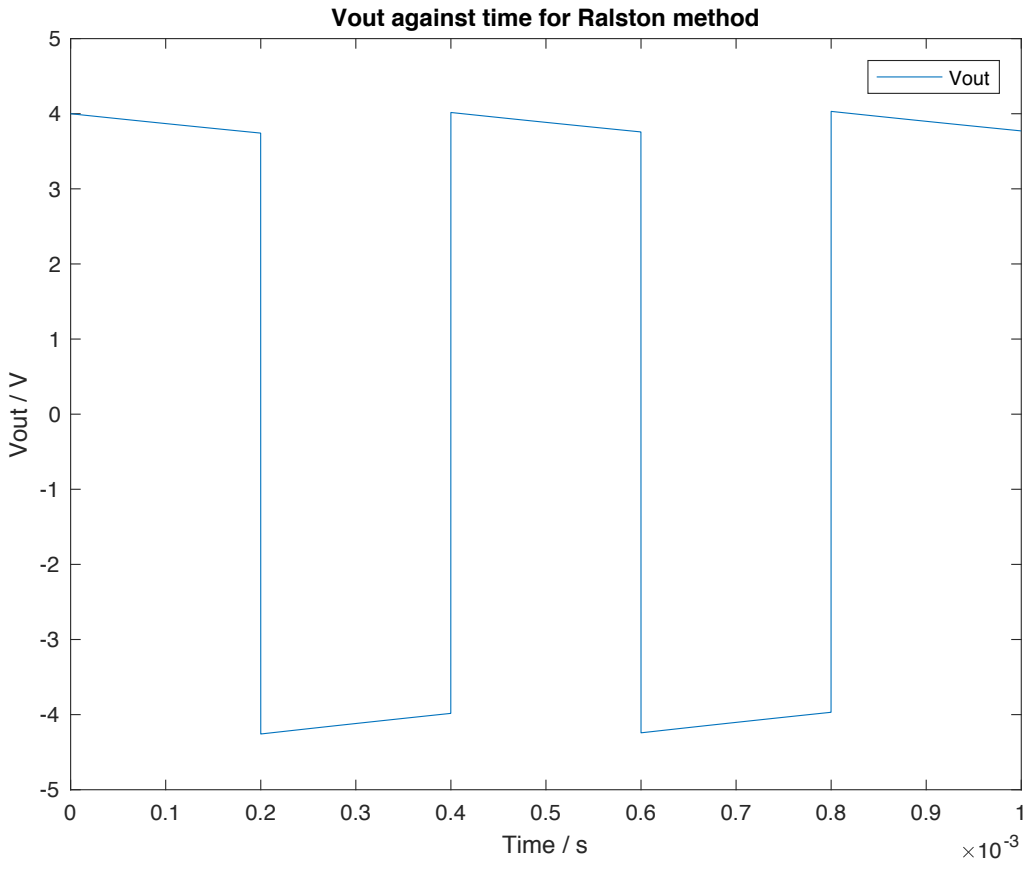

Fig46: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 400\ \mu s$*

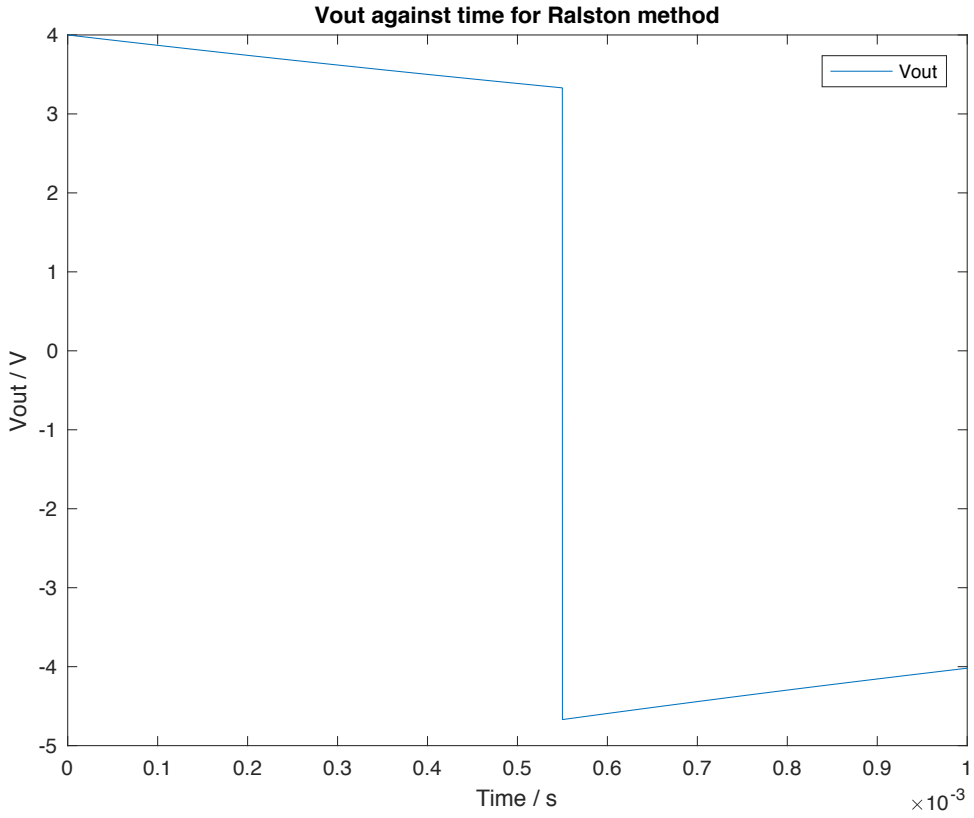

Fig47: *$V_{out} = V_L(t)$ across the inductor when the input is square wave with amplitude 4 V and time period $T = 1100\ \mu s$*

1.7.6: Sawtooth wave input with Amplitude 4 V and different periods

$T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

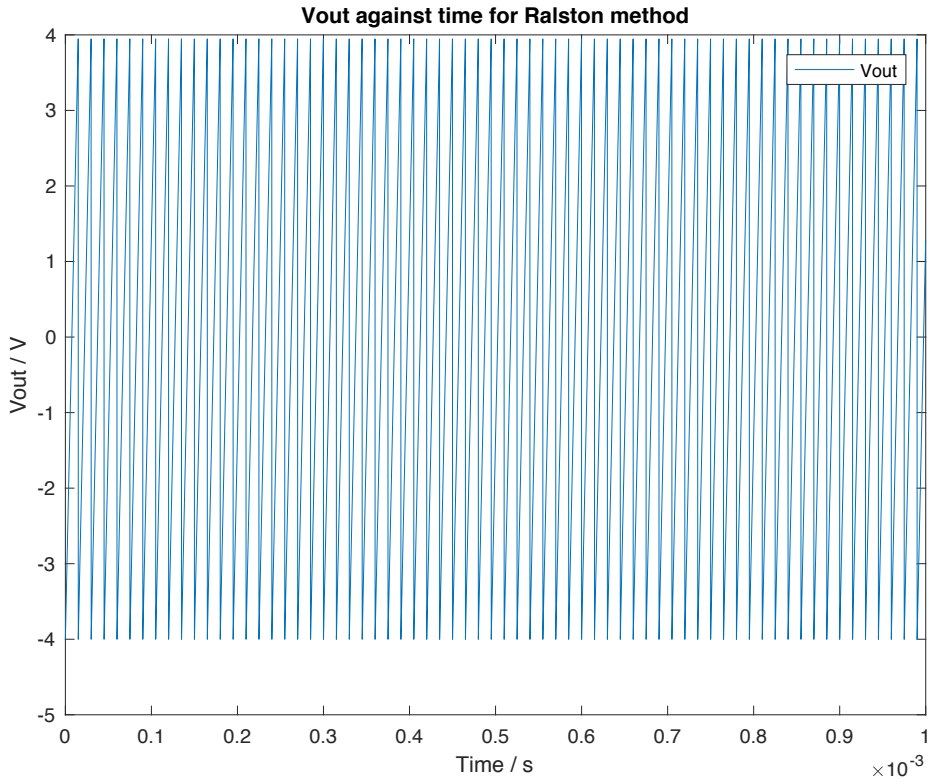

Fig48: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 15\ \mu s$*

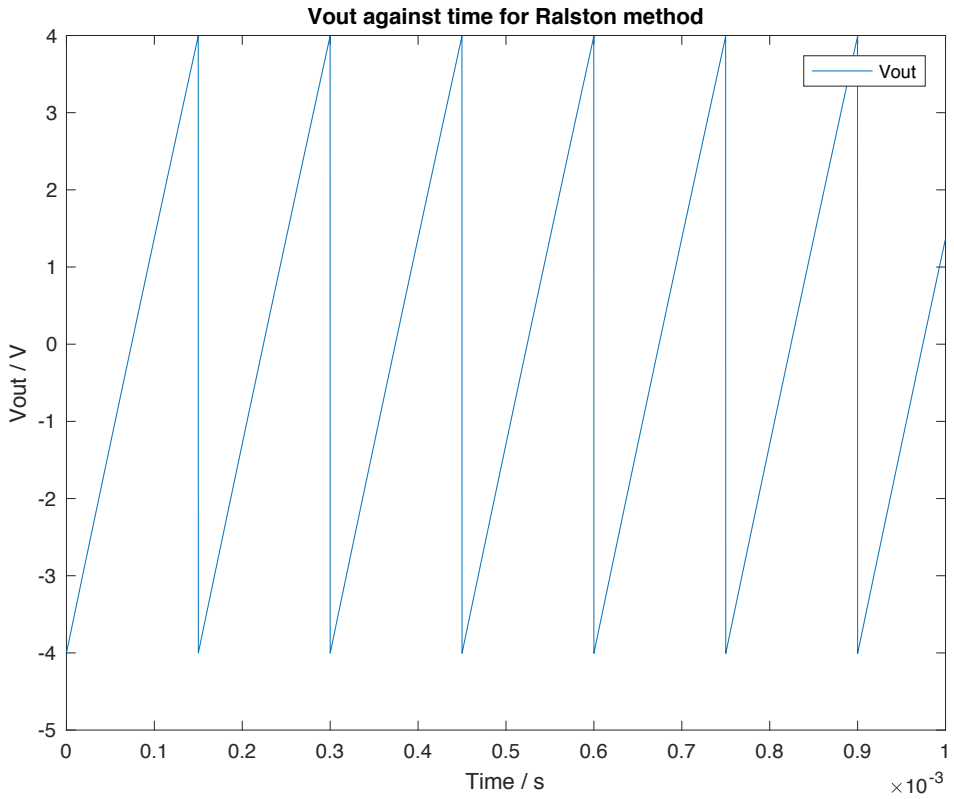

Fig49: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 150\ \mu s$*

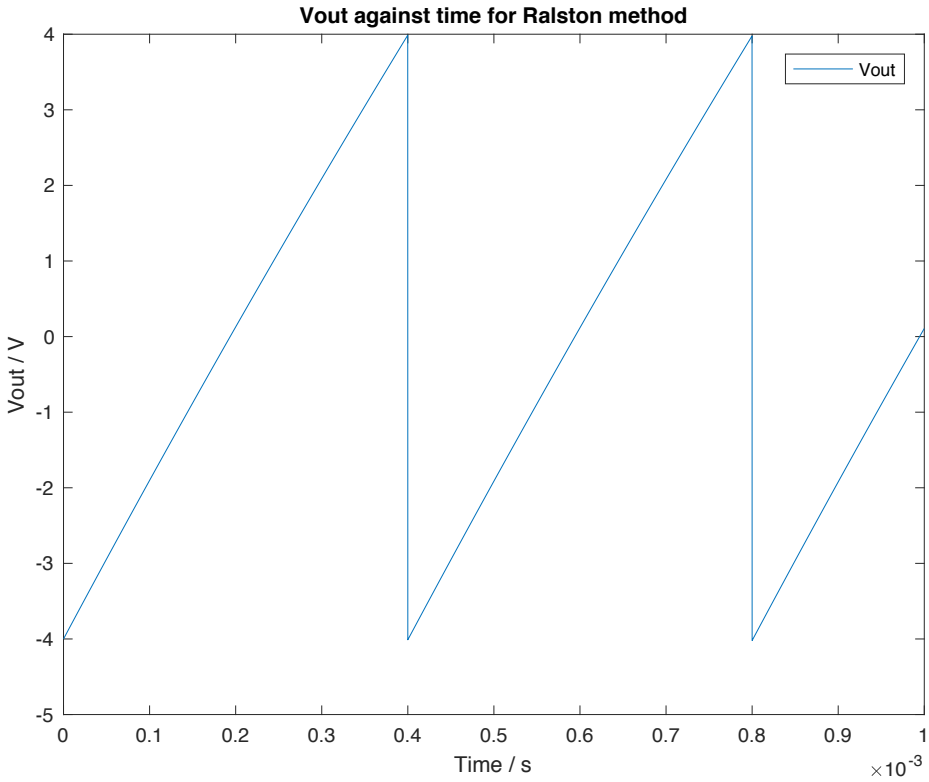

Fig50: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 400\ \mu s$*

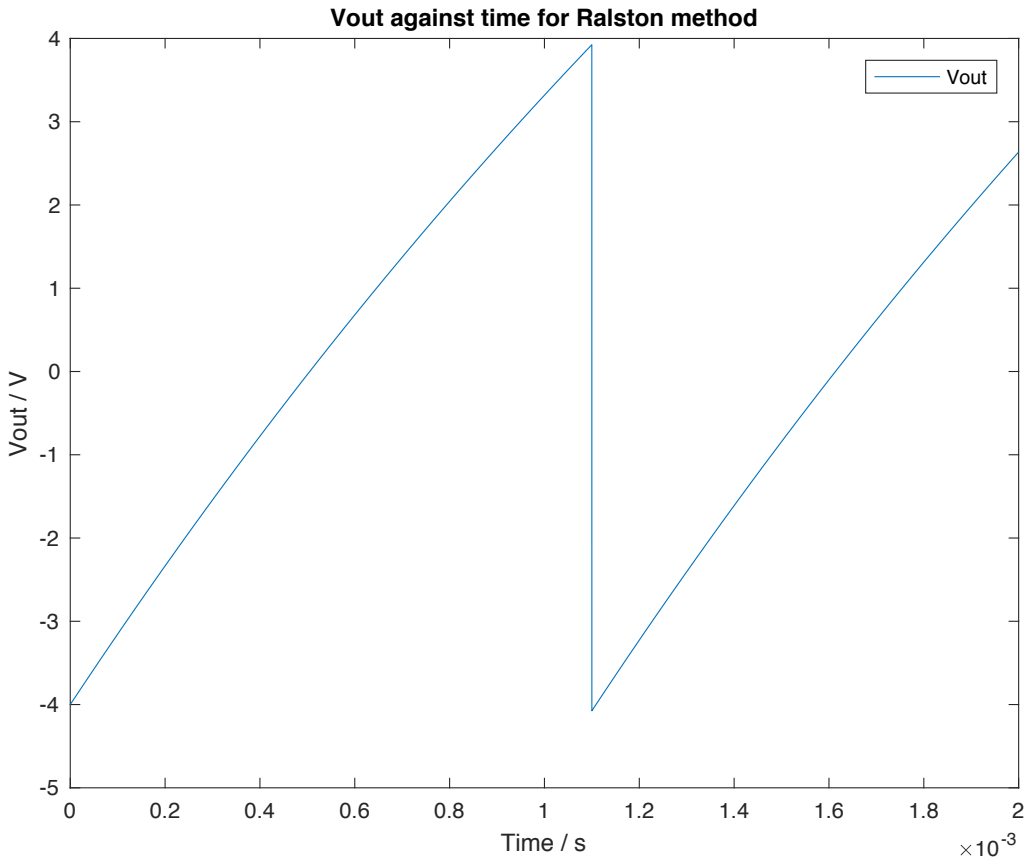

Fig51: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period* $T = 1100\ \mu s$

1.8: Explanation of the Plots with respect to Heun, Midpoint and Ralston methods

1.8.1: When the input is a step signal with amplitude $V_{in} = 3.5\ V$

For a step signal input, the sudden change in input voltage is observed across the output. This is because the current through an inductor doesn't change instantaneously. At the instant when the input signal is applied, there is no current flowing across the inductor. Therefore, the whole input voltage is observed across the output.

The step response as discussed in section 1.2 is given by the following equation:

$$v_L = L\frac{V_{in}}{R}\left(\frac{R}{L}e^{-\left(\frac{R}{L}\right)t}\right) = V_{in}e^{-\left(\frac{R}{L}\right)t}$$

Meanwhile, the transfer function is given by the following formula:

$$T(s) = \frac{Ls}{R + Ls} = \frac{s}{\frac{R}{L} + s}$$

A step input at t = 0 represents a set of high frequency components in the frequency domain. At t = 0, $V_{out} = V_{in}$, that is the transfer function must be equal to 1. This is only possible when frequency is very high, i.e. infinity. The RL high pass filter circuit allows the input voltage to be observed on the output. For t > 0, a decay of voltage occurs at the output as V_{out} is directly proportional to $e^{-\left(\frac{R}{L}\right)t}$, observed in the step response function.

1.8.2: When the inputs are an impulsive signal with decay $V_{in} = 3.5e^{\left(-\frac{t^2}{\tau}\right)}$ where $\tau = 150\ (\mu s)^2$ and an impulsive signal with decay $V_{in} = 3.5e^{\left(-\frac{t}{\tau}\right)}$ where $\tau = 150\ \mu s$

For both exponential functions in input, the rate of change of the input for the exponential decreases as time increases. Therefore, a high voltage appears at the output i.e., the high rate of change of voltage is passed through the high pass filter circuit. However, as the rate of change decreases, the voltage is no longer passed through ($V_{in} = 0$) and the output decreases to 0. Also, as the value of τ decreases, the deviation between the input and output voltage plots increases.

1.8.3: When the input is a sine wave with amplitude 4 V and different periods $T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

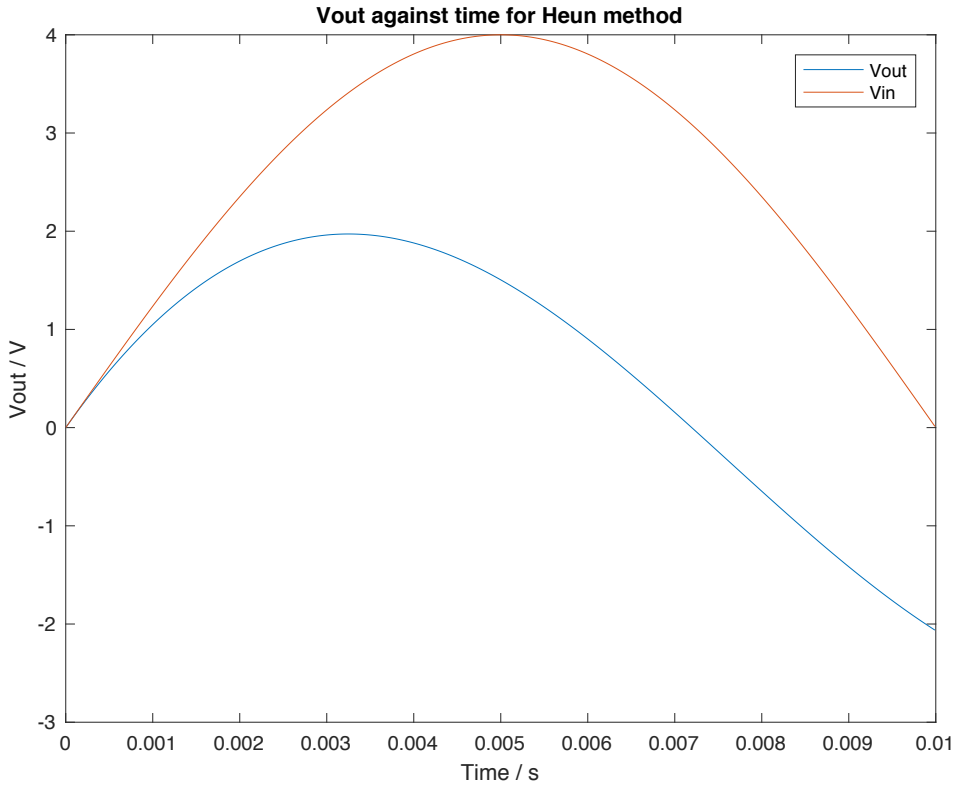

Fig52: *$V_{out} = V_L(t)$ across the inductor when the input is sine wave with amplitude 4 V and time period $T = 20\ \mu s$*

For the sine wave function, all the tested periods provide frequencies well above the cut off corner frequency. Therefore, it is expected that all signals pass through to the output and indeed this is what is observed as the input signal is directly noticed at the output. In Fig52 is a graph

which shows V_{in} (Orange) and V_{in} (Blue) against time at time period 20 ms. The frequency exceeds the cut off frequency of 333.3Hz, therefore the output voltage does not replicate input voltage.

1.8.4: When the input is a square wave with amplitude 4 V and different periods $T = 15\ \mu s,\ T = 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

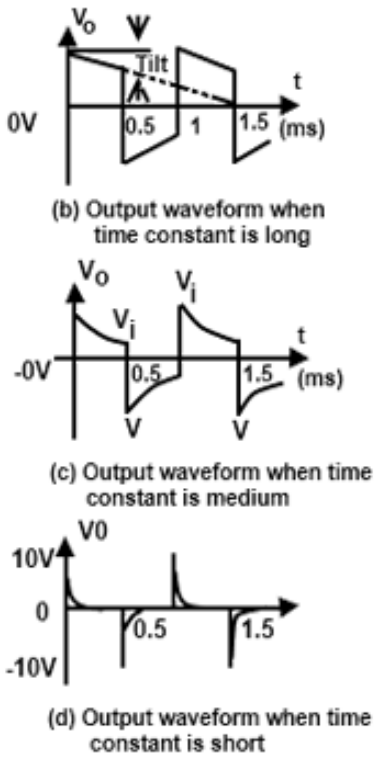

Fig53: *Output waveforms for different values of time constants and the tilting phenomenon*

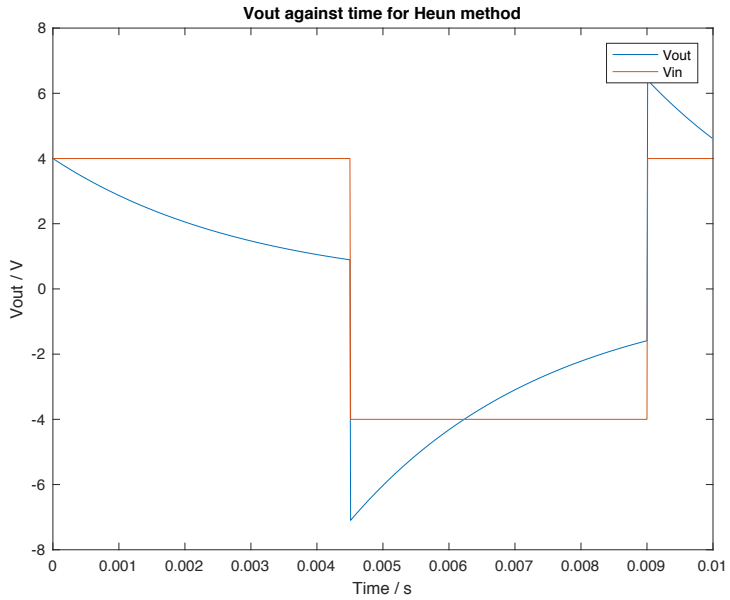

Fig54: *Output waveform for an input square wave and the observation of the tilting phenomenon in the output wave*

For the squared wave, slightly different results were observed despite the frequencies being the same as the sine wave inputs. This is because the square wave in a way models multiple step functions and contains DC levels which cannot pass through and exponentially decay as seen in the step signal input case.

For this reason, the beginning parts of the output start to decay at the points where the square wave is flat. It is observed that the amount of decay increases as the frequency decreases (or time period increases), which is expected and observed in the graph below. Additionally, as the time constant (L/R) increases, there's a greater tilt as shown in Fig53.

1.8.5: When the input is a square wave with amplitude 4 V and different periods $T = 15\ \mu s,\ T = \ 150\ \mu s,\ T = 400\ \mu s,\ T = 1100\ \mu s$

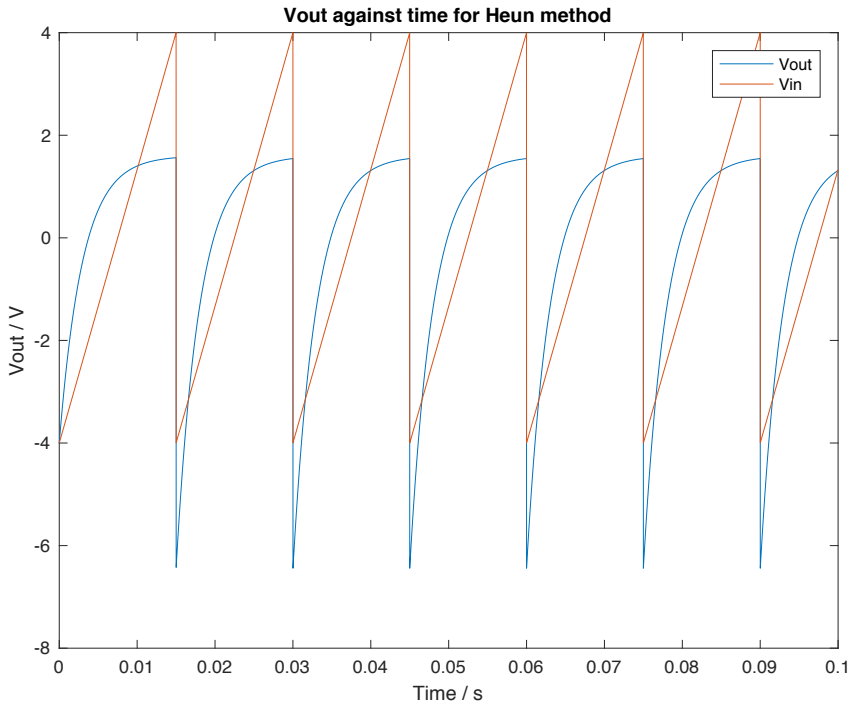

Fig55: *$V_{out} = V_L(t)$ across the inductor when the input is sawtooth wave with amplitude 4 V and time period $T = 15\ ms$*

The sawtooth waveform results are similar to the sine wave results as constant high rates of change (steep slopes) are observed. Frequencies (or time periods) used to plot the graphs in this exercise exceeds the cut off frequency therefore, the input waveform is exactly observed at the output.

When a sawtooth signal at a frequency higher than the cutoff frequency, the output increases exponentially in the steep slope region as observed in Fig55.

1.9: Error Script (Exercise 2)

1.9.1: Introduction to Error Analysis

For this exercise, a MATLAB file called "error_script.m" was created in order to provide a complete error analysis of a given input signal, using all methods provided in the previous exercise.

The input signal was a cosine wave with the following parameters –

- Amplitude V_{in} = 6 V
- Time Period T = 150 μs

The final objectives with a time varying input $V_{in}(t) = 6\cos\left(\frac{2\pi t}{150*10^{-6}}\right)$ were to:

- Plot the output voltage obtained through numerical analysis, output voltage obtained through calculation by hand, and the error function obtained by subtracting the two aforementioned plots for each of the three methods which are Heun, Midpoint and Ralston.
- Vary the time step h for a suitable number and range of values, and obtain a log-log plot to show the order of the error

Based on these tasks, the code has been split into two parts, one to fulfil each objective. In the first part, the circuit component values are initialized as well as circuit conditions. It's worth noting that the value of step size, 'h', is limited by the Nyquist Sampling Rate to a value of $75 * 10^{-6}$ seconds. Since the value of time period is 150μs and the Nyquist sampling rate is twice the frequency, using the relation $f = \frac{1}{T}$, we obtain a maximum value of step size which is half of the time period ($h = 75\mu s$).

After the circuit encoding, the input signal parameters were encoded and values of output voltage across the inductor was calculated by the three methods used in the previous exercise which are Heun, Midpoint and Ralston.

1.9.2: Exact solution of the ODE

The integration-factor method was used to solve the ODE. The following equations were taken into consideration:

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

$$L\frac{di}{dt} + iR = Vcos(\omega t)$$

$$\frac{di}{dt} + \frac{R}{L}i = \frac{V}{L}\cos(\omega t)$$

The integrating factor is as follows:

$$e^{\int \frac{R}{L} dt} = e^{\frac{R}{L}t}$$

Therefore, the equation can be written as follows:

$$ie^{\frac{R}{L}t} = \int \frac{V}{L}\cos(\omega t)\, e^{\frac{R}{L}t}$$

Using integration by parts, the following is the result:

$$i(t) = \frac{V}{R^2 + (L\omega)^2}\,(L\omega\sin(\omega t) + R\cos(\omega t))$$

Since $V_{out} = V_L(t)$, the exact value is calculated by differentiating $i(t)$ once and multiplying it by the inductance L which consequently gives the exact solution:

$$V_{out} = L\frac{di}{dt} = \frac{VL\omega}{R^2 + (L\omega)^2}\,(L\omega\cos(\omega t) - R\sin(\omega t))$$

The above exact solution is plotted along with every numerical method plot of V_{out} against Heun's method time. Since the values of time in arrays of all three methods are the same, it does not matter which time is used for calculation and plotting. Finally, the error for each numerical method is calculated by subtracting the respective solution obtained numerically from the exact solution. Since the error's magnitude is of order 10^{-4}, three separate subplots are used for every method, first one being the numerical plot for the method, second one being the exact plot and third one being the error plot.

1.9.3: Error Analysis implemented on MATLAB as error_script.m

```
% Exercise-2 : Error Analysis

% This script uses the methods implemented in the previous exercise,
% (Heun, Midpoint and Ralston) and calculates the error function by
finding
% the exact solution of the ODE. The error function is finally used to
% carry the required error analysis.

% The following circuit components/conditions were initialised:
clc;
clear;
R = 0.5;                % Value of resistor in ohms
L = 1.5 * 10^-3;        % Value of inductor in Henries
t_initial = 0;          % Initial time
i_initial = 0;          % Current at initial time
t_final = 0.0005;       % Final time
h = 0.000005;           % Time Step, maximum value calculated as per
                        % Nyquist Sampling Rate, since T = 150*10^-6
s,
                        % maximum value of h is 75*10^-6 s.
N = round ((t_final - t_initial/h));

%The given input signal parameters are:

Vin_Amp = 6;                    % Amplitude of signal
T = 150 * 10^-6;                % Time period of cosine wave
f = 1/T;                        % Frequency of wave
w = 2*pi*f                      % Angular velocity
Vin=@(t) Vin_Amp*cos(w*t); % Input Signal
func=@(t,i) (1/L)*(Vin(t)-R*i); % To calculate (di/dt) at any instant
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis*

```
% Next step is to calculate output voltage, Vout, through the three
% proposed methods (Heun, Midpoint and Ralston):


[Heun_t, Heun_Vout]         = Heun(func, i_initial, t_final, h, R,
L);
[Midpoint_t, Midpoint_Vout] = Midpoint (func, i_initial, t_final, h,
R, L);
[Ralston_t, Ralston_Vout]   = Ralston (func, i_initial, t_final, h,
R, L);

% The next step is to calculate the exact solution of the ODE. This
Ode is
% solved using integration factor method. Refer to the report for the
% solution of the ODE.

exact = zeros(1,N);

% Time t will be the same for all three methods because h is equal.
exact = ((Vin_Amp*L*w)/((R^2)+(L*L*w*w)))*((L*w*cos(w*Heun_t)) -
(R*sin(w*Heun_t)));


% Calculating the errors:
Heun_error      = exact - Heun_Vout;
Midpoint_error  = exact - Midpoint_Vout;
Ralston_error   = exact - Ralston_Vout;

% Plotting the three solutions (Exact solution, Numerical solution
and Error):

% Heun

figure(1) ;

subplot (3 ,1 ,1) ;
plot(Heun_t, Heun_Vout, 'r');
title ([ 'Numerical solution by Heuns method, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

```
subplot (3 ,1 ,2) ;
plot(Heun_t, exact, 'b');
title ([ 'Exact solution by Heuns method, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');

subplot (3 ,1 ,3) ;
plot(Heun_t, Heun_error, 'g');
title ([ 'Error in Heuns method, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Error [V]');

% Midpoint

figure(2) ;

subplot (3 ,1 ,1) ;
plot(Midpoint_t, Midpoint_Vout, 'r');
title ([ 'Numerical solution by Midpoint, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');

subplot (3 ,1 ,2) ;
plot(Midpoint_t, exact, 'b');
title ([ 'Exact solution by Midpoint, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');

subplot (3 ,1 ,3) ;
plot(Midpoint_t, Midpoint_error, 'g');
title ([ 'Error in Midpoint, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Error [V]');

% Ralston

figure(3) ;

subplot (3 ,1 ,1) ;
plot(Ralston_t, Ralston_Vout, 'r');
title ([ 'Numerical solution by Ralston, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

```
subplot (3 ,1 ,2) ;
plot(Ralston_t, exact, 'b');
title ([ 'Exact solution by Ralston, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Vout [V]');

subplot (3 ,1 ,3) ;
plot(Ralston_t, Ralston_error, 'g');
title ([ 'Error in Ralston, h = ' num2str(h)]);
xlabel('Time [s]');
ylabel('Error [V]');


%%%%%%%

% Finally, we carry out the error analysis in this section. We keep on
% modifying the value of 'h' and the value of maximum error obtained
for
% every method is recorded, as are the values of 'h'. An additional
plot is
% used to show the order dependency of O(h^2).

K = 20;             % To plot for 20 different values of h

% Initialising new matrices for analysis:

Heun_err             = zeros(1,K);               % Arrays to store max
error
Midpoint_err         = zeros(1,K);
Ralston_err          = zeros(1,K);
h_matrix             = logspace (-5, -4.1, K);   % Region where order 2
                                                 % properties are
observed.
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

```
line_dependency     = zeros(1,K);                % Reference curve
for order
                                                 % 2 properties.
% Now we initialise the loop to calculate values for different
values of
% 'h', keeping in mind the Nyquist Sampling Rate and effect on
% computing speed.

for j = 1:K
    N = round ((t_final - t_initial)/h_matrix(j));

    % Solving for the new value of 'h':

    [Heun_t, Heun_Vout]         = Heun(func, i_initial, t_final,
h_matrix(j), R, L);
    [Midpoint_t, Midpoint_Vout] = Midpoint(func, i_initial, t_final,
h_matrix(j), R, L);
    [Ralston_t, Ralston_Vout]   = Ralston(func, i_initial, t_final,
h_matrix(j), R, L);


% Now we initialise the loop to calculate values for different
values of
% 'h', keeping in mind the Nyquist Sampling Rate and effect on
% computing speed.

for j = 1:K
    N = round ((t_final - t_initial)/h_matrix(j));

    % Solving for the new value of 'h':

    [Heun_t, Heun_Vout]         = Heun(func, i_initial, t_final,
h_matrix(j), R, L);
    [Midpoint_t, Midpoint_Vout] = Midpoint(func, i_initial, t_final,
h_matrix(j), R, L);
    [Ralston_t, Ralston_Vout]   = Ralston(func, i_initial, t_final,
h_matrix(j), R, L);
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

```
  % Solving for the new value of 'exact':


    exact = ((Vin_Amp*L*w)/((R^2)+(L^2*w^2)))*((L*w*cos(w*Heun_t))
- (R*sin(w*Heun_t)));

    % Solving for the new values of 'error':

    Heun_error      = abs(exact - Heun_Vout);
    Midpoint_error  = abs(exact - Midpoint_Vout);
    Ralston_error   = abs(exact - Ralston_Vout);

    % Solving for maximum errors for every method for every
corresponding
    % value of 'h':

    Heun_err(j)          = max(Heun_error);
    Midpoint_err(j)      = max(Midpoint_error);
    Ralston_err(j)       = max(Ralston_error);
    line_dependency(j)   = h_matrix(j)^2;

end

figure(4);

% Heun

subplot(3, 1, 1);
loglog (h_matrix , Heun_err , '*r') ;
title('Max Error vs Step Size (Heuns Method)');
xlabel('Step Size (s)');
ylabel ('Error in Vout (V)') ;
hold on;
logx = log(h_matrix);
logy = log(Midpoint_err);
Const = polyfit(logx, logy, 1)
hold on
plot(h_matrix, exp(polyval(Const, logx)));
legend('Numerical error','Exact error');
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

```
% Midpoint

subplot(3, 1, 2);
loglog (h_matrix , Midpoint_err , '*r') ;
title('Max Error vs Step Size (Midpoint Method)');
xlabel('Step Size (s)');
ylabel ('Error in Vout (V)') ;
hold on;
logx = log(h_matrix);
logy = log(Midpoint_err);
Const = polyfit(logx, logy, 1)
hold on
plot(h_matrix, exp(polyval(Const, logx)));
legend('Numerical error','Exact error');

% Ralston

subplot(3, 1, 3);
loglog (h_matrix , Ralston_err , '*r') ;
title('Max Error vs Step Size (Ralston Method)');
xlabel('Step Size (s)');
ylabel ('Error in Vout (V)') ;
hold on;
logx = log(h_matrix);
logy = log(Ralston_err);
Const = polyfit(logx, logy, 1)
hold on
plot(h_matrix, exp(polyval(Const, logx)));
legend('Numerical error','Exact error');

figure(5)

% Proof of second order

loglog (h_matrix , Heun_err);
hold on
loglog (h_matrix , Midpoint_err);
loglog (h_matrix , Ralston_err);
loglog(h_matrix, line_dependency);
legend( 'Heuns method', 'Midpoint method', 'Ralstons method',
'y=x^2');
title('Max Error vs Step Size');
xlabel('Step Size (s)');
ylabel ('Error in Vout (V)') ;
```

Fig56: *MATLAB code implementing error_script.m for the purposes of error analysis (cont..)*

1.9.4: Comparing the numerical solution with the exact solution and obtaining the error as a function of t

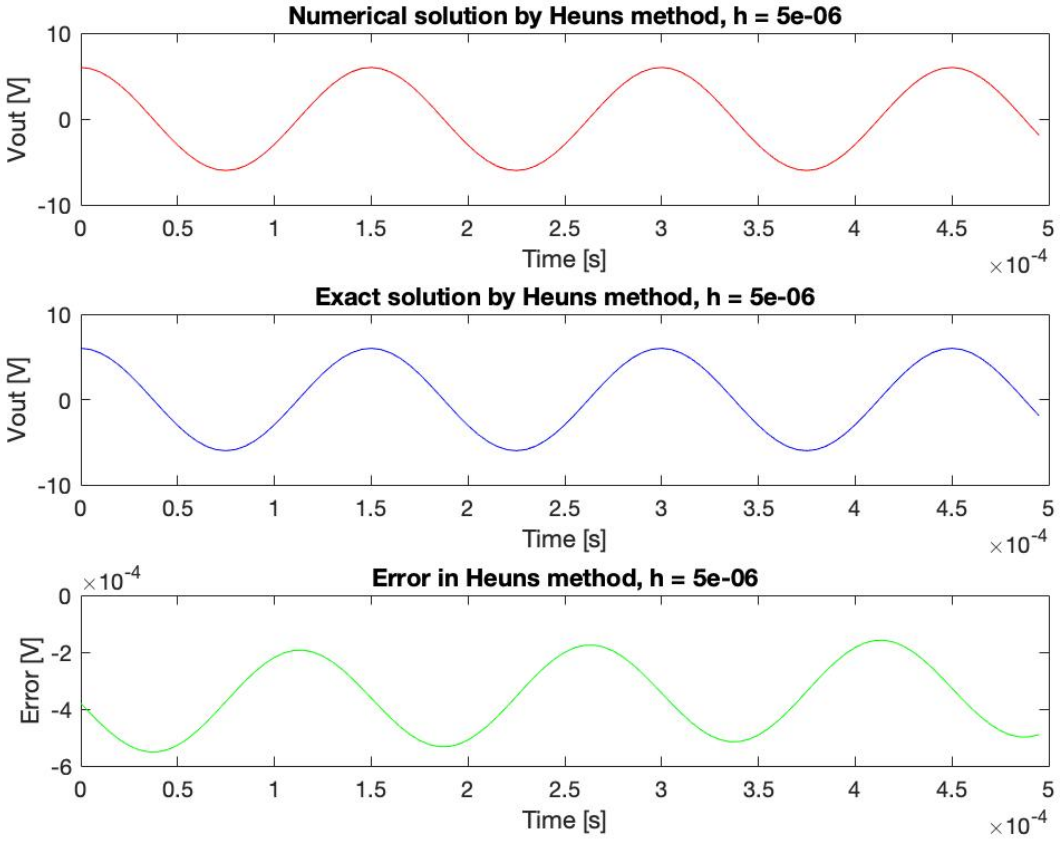

Fig57: *Plots for numerical solution by Heun's method and the exact solution. The error plot in Heun's method is found by subtracting the numerical solution for the Heun's method from the exact solution.*

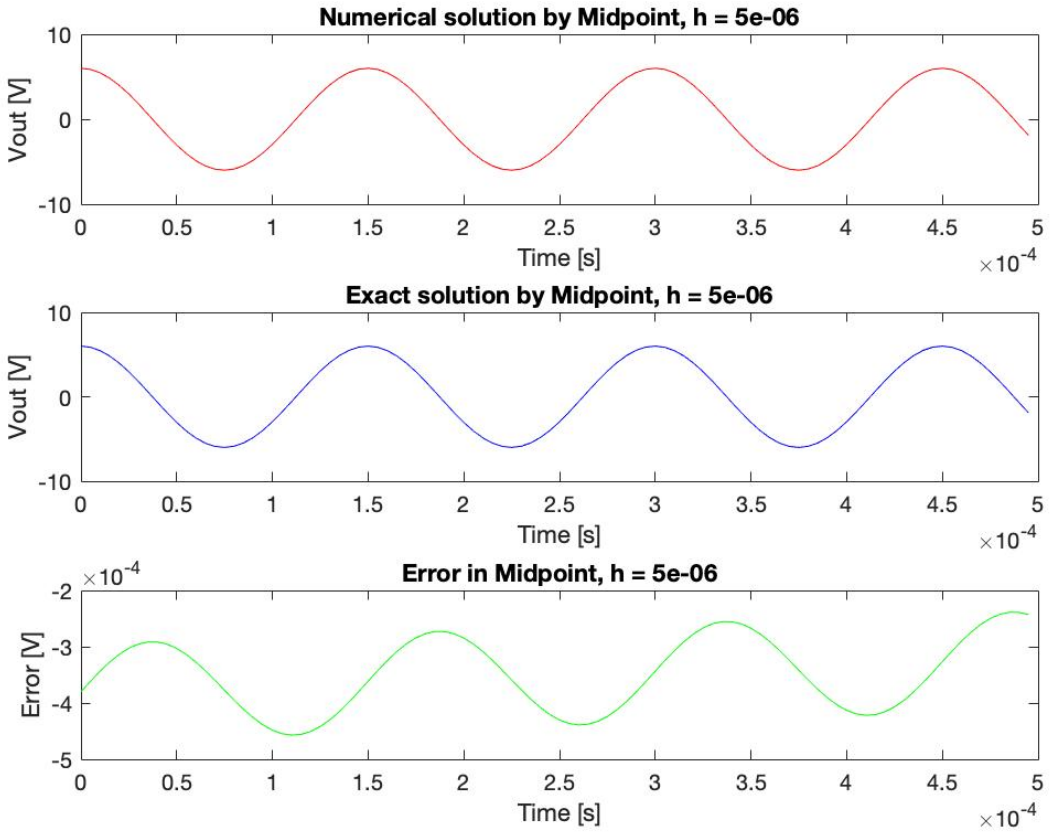

Fig58: *Plots for numerical solution by Midpoint method and the exact solution. The error plot in Midpoint method is found by subtracting the numerical solution for the Midpoint method from the exact solution.*

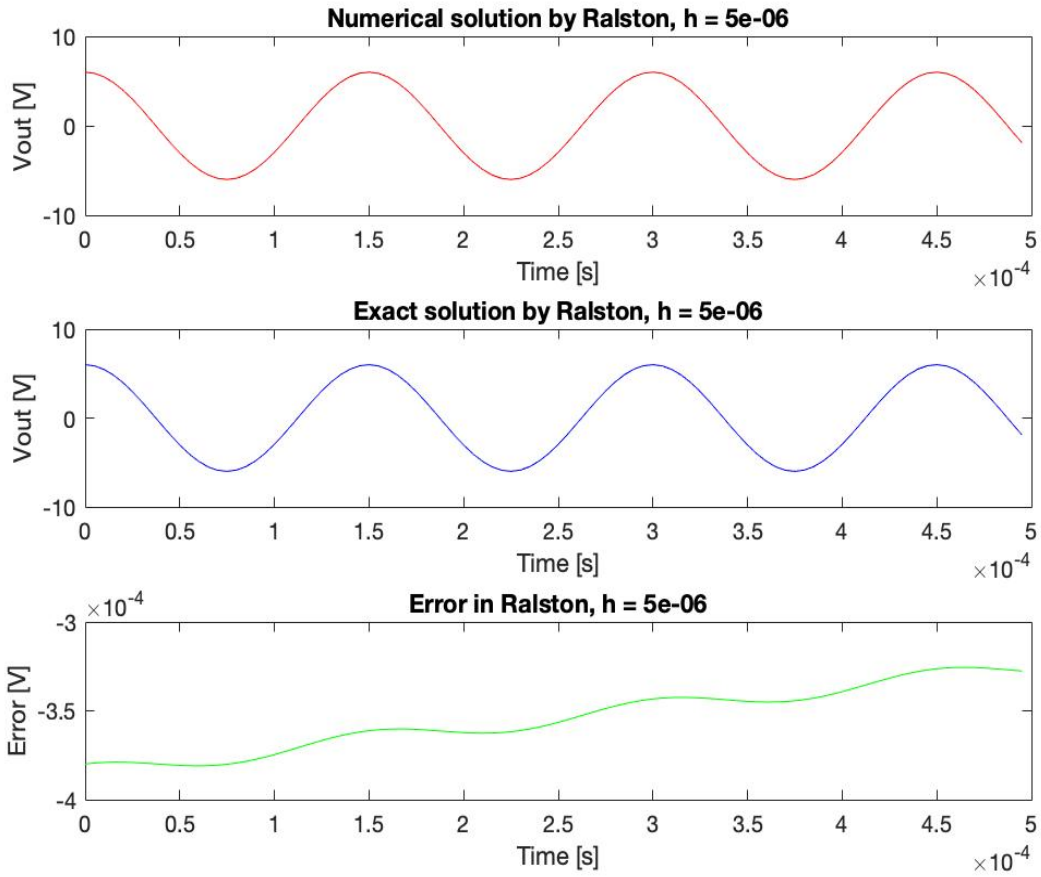

Fig59: *Plots for numerical solution by Ralston method and the exact solution. The error plot in Ralston method is found by subtracting the numerical solution for the Ralston method from the exact solution.*

1.9.5: Varying the time step h for a suitable number and range of values and the Log-Log plot to show the order of the error

In the second part of the script, the objective is to highlight the variation in the error due to a change in the value of step size 'h'. A new array was made, called h_matrix, which contains values of Step Size for which maximum error for each value and method were calculated. There are 20 values of step size, spaced out between 10^{-5} and $10^{-4.1}$ using the logspace function.

To store the maximum error of each method, a loop was used which would run for 20 iterations (equal to values of step size). At the beginning of every loop, the numerical solution for every method, exact solution as per the new step size and error for every method was calculated in the same way as in the first part of the code.

Since error obtained can be both negative or positive, the absolute value of the error was taken, and max function was used to store it in an array of the respective method (Heun, Midpoint or Ralston). Finally, these maximum error values are plotted as subplots (Log-Log plots), along with a best fit plot of the points. The red stars show the maximum error plot while the blue line shows the best fit line.

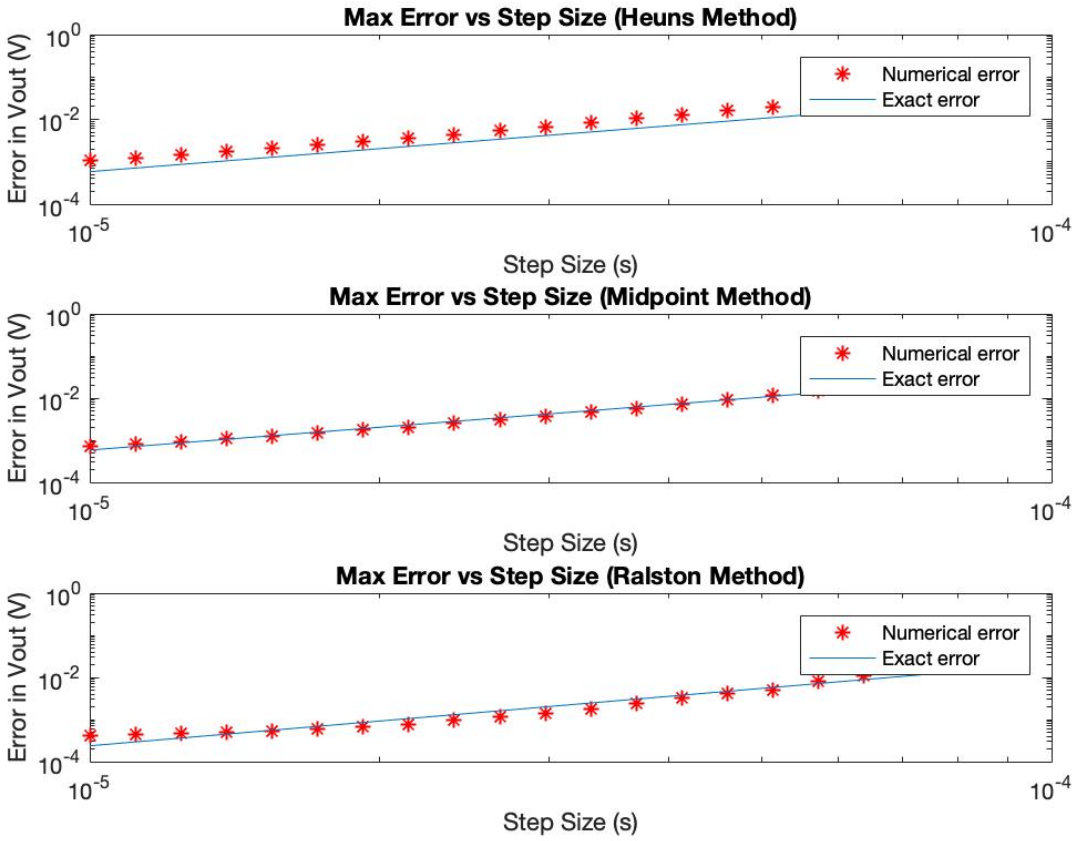

Fig60: *Log-Log plots for the error (found by subtracting numerical solution from the exact solution) as the step size 'h' is varied. Error vs the step-size 'h' Log-Log plots are shown for the Heun, Midpoint and the Ralston methods*

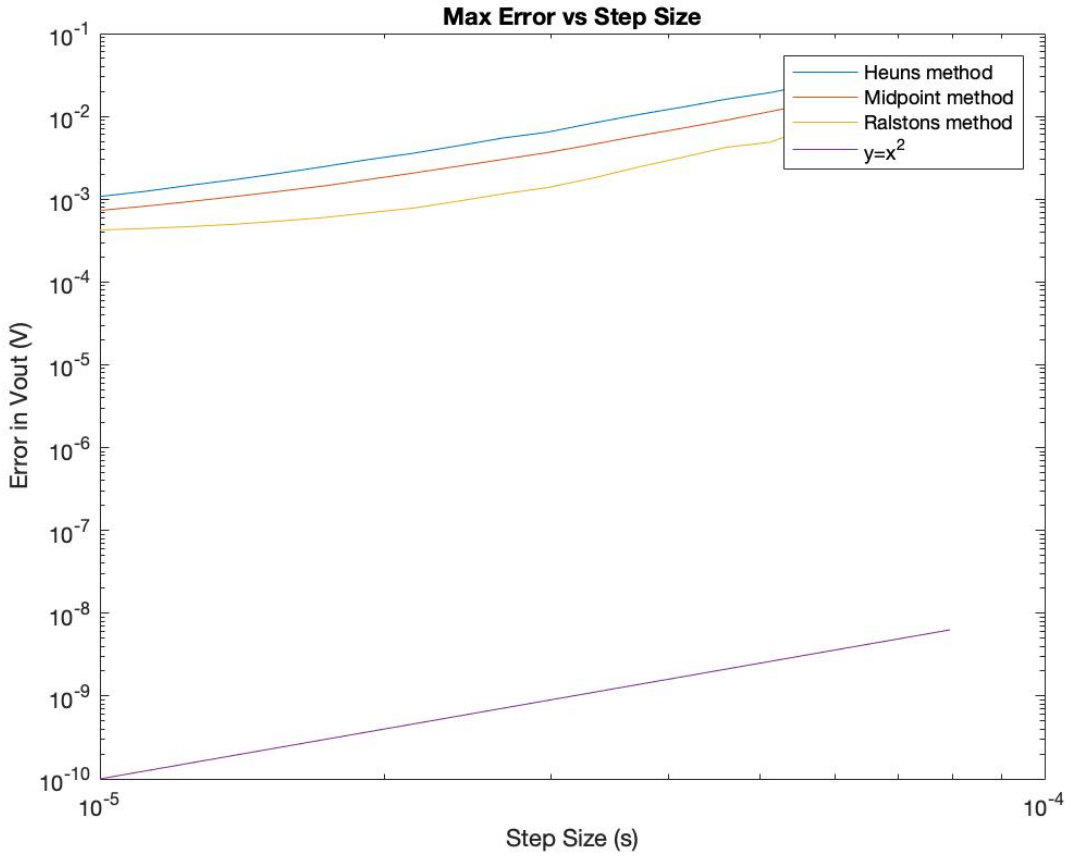

Fig61: *Error vs the step-size 'h' Log-Log plots are shown in the same graph for the Heun, Midpoint and the Ralston methods. The Log-Log plot for* $y = x^2$ *is also shown*

1.9.6: Comparison of the error between the three methods (Heun, Midpoint, Ralston)

The graph in Fig61 shows all the three loglog error plots together with a second order Log-Log plot for the equation $y = x^2$. This graph is very important for the final conclusion that is drawn from this experiment. It can be observed that the scale of error is the largest when Heun's method is used and the least when we use Ralston's algorithm. Also, as the step size 'h' is increased, it can be seen that the error increases since the number of samples decreases.

Some more results that follow are:

- Error in Heun's Method varies from -6×10^{-4} to -2×10^{-4} (approximately)
- Error in Midpoint Method varies from -4.7×10^{-4} to -2.2×10^{-4} (approximately)
- Error in Ralston's Method varies from -3.8×10^{-4} to -3.2×10^{-4} (approximately)

The truncation errors for the second order Range-Kutta Methods are $O(h^3)$ because, in the expansion of Taylor Series, truncation is observed after the term containing h^3. Also, the truncation error after a single iteration is proportional to h^3. So, on moving from $t_{initial}$ to t_{final}, one might encounter errors of different signs which might cancel out other errors out, which is why the worst possible case of assuming all errors having the same sign and summing them is considered.

From the Taylor series truncation, we know that the global error for this case is proportional to Nh^3. Since N is inversely proportional h by the relation $N = (t_{final} - t_{initial})/h$, the global error is proportional to h^2.

This is also observed in Fig61 where the maximum error for each case is plotted against various values of h and so is the plot for a quadratic relation. Since the gradient is same for all the plots, we can say that the error is of order 2.

2 RLC Circuit (Exercise 3)

2.1: Introduction to the RLC circuit (Exercise 3)

In the given exercise, an RLC series circuit is given which is normally used to model a harmonic oscillator which is a device used to resonate to a sinusoidal input signal. Applications of the RLC circuit include tuning of analogue radio receivers and usually, they are used as band-pass filters. Only signals at resonance are passed in when an inductor and a capacitor are connected in series since all the other frequencies are eliminated.

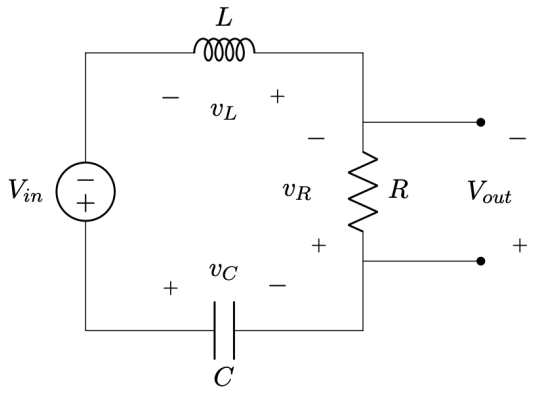

Fig62: *Given RLC Circuit with parameters R = 250 Ω, C = 3 μF, L = 0.650 H*

For the given RLC circuit, the following equations can be written:

$$v_L(t) + v_R(t) + v_C(t) = V_{in}(t)$$

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

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

Fig63: *RLC Circuit equations wherein $V_{in}(t)$ stands for the input voltage, $v_L(t)$ is the voltage across the inductor, $v_R(t)$ stands for the voltage across the resistor, $v_C(t)$ stands for the voltage across the capacitor.*

Additional parameters of the circuit are given. $q_C(t)$ indicates the state of the circuit and the output of the voltage is measured across the resistor which is indicated by $v_R(t)$ and $R\frac{d}{dt}q_C(t)$. It is also assumed that at time $t = 0$, the capacitor is pre-charged with $q_C(0) = 500nC$ and no initial current flows through the inductor; i.e. $i_L(0) = \frac{d}{dt}q_C(0) = 0\ A$.

2.2: RLC Circuit Features

Resonance:

For an RLC series circuit, the resonant frequency occurs at a point such that the magnitude of the impedance of the capacitor is the same as that of the inductor. But there, is a phase shift of 180°. Moreover, the maximum amplitude occurs at the resonant frequency. The latter can be confirmed by plotting the graphs on MATLAB:

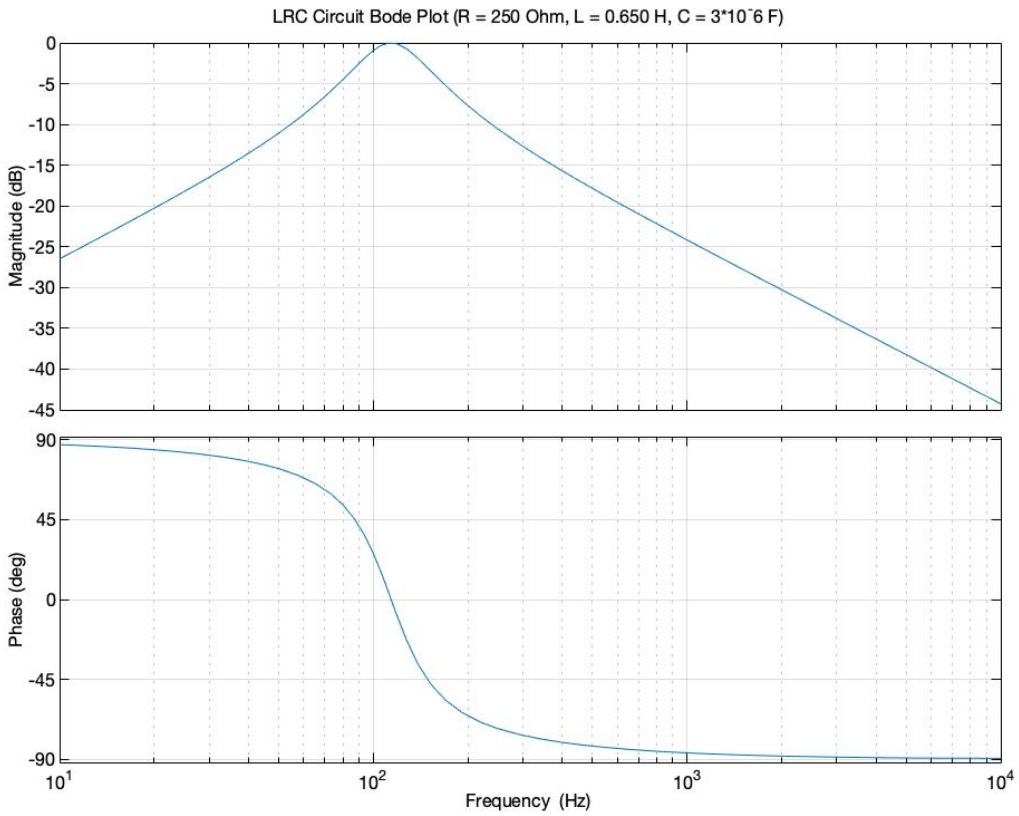

Fig64: *Bode Plot of the RLC series circuit with R = 250 Ω, C = 3 μF, L = 0.650 H*

In the above graph, the resonant frequency is given by around 113 Hz. This can be confirmed with the formula which is (Step Response of RLC Circuits):

$$f = \frac{1}{2\pi\sqrt{LC}} = 113.97\ Hz$$

The theoretical phase shift due to the filter is 0° at the resonant frequency (Step Response of RLC Circuits).

2.3: The 4th order Runge-Kutta 3/8 Algorithm method

The problem $\frac{dy}{dx} = f(x, y)$ and $y(x_0) = y_0$, can be evaluated using the 4th order Runge-Kutta 3/8 algorithm which provides an approximated solution for first order differential equations. The integrand $f(x, y)$ is evaluated four times per step which is given by the following formula (*Runge-Kutta 3/8 Method*):

$$y_{i+1} = y_i + \frac{1}{8}[k_1 + 3k_2 + 3k_3 + k\_4]$$

where;

$$k_1 = hf(x_i, y_i)$$

$$k_2 = hf\left(x_i + \frac{h}{3},\ y_i\ + \frac{k_1}{3}\right)$$

$$k_3 = hf\left(x_i + \frac{2h}{3}, y_i - \frac{k_1}{3} + k_2\right)$$

$$k_4 = hf(x_i + h, y_i + k_1 - k_2 + k_3)$$

The next value of x_i which is x_{i+1} can be calculated by the following formula:

$$x_{i+1} = x_i + h$$

The given RLC circuit in Fig1 has the following equation:

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

The third equation is a second order differential equation wherein capacitive charge is expressed as a function of the applied input voltage. Furthermore, the following initial conditions are given as defined in section 2.1:

$$V_{out} = v_R(t) = R\frac{d}{dt}q_C(t)$$

$$At\ t = 0;\ i_L(0) = \frac{d}{dt}q_C(0) = 0\ A$$

$$q_c(0) = 500\ nC$$

The second order differential equation can be written as a system of second order differential equations to apply the Runge-Kutta 3/8 algorithm. Therefore, it is done as follows:

$$q_C(t) = x$$

$$\frac{d}{dt}q_C(t) = \frac{dx}{dt} = y$$

The equations are modified as follows:

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

$$L\frac{dy}{dt} + \frac{Rdx}{dt} + \frac{x}{C} = V_{in}(t)$$

$$L\frac{dy}{dt} + Ry + \frac{x}{C} = V_{in}(t)$$

The above equation can be rearranged as follows:

$$\frac{dy}{dt} = \frac{1}{LC}(CV_{in}(t) - x - RCy)$$

Consequently, the above equation can be applied for the Runge-Kutta 3/8 algorithm and it was coded as follows:

2.4: The 4th order Runge-Kutta 3/8 Algorithm implemented as RK4second on MATLAB

```
function [Yi, Xi] = RK4second(R,C,L,h,X,Y,t,vin) %definining the
function required

%generates Yi and Xi, given the Y and X
vin1 = vin(t);
coup=@(t,X,Y) Y;
pre_coup=@(t,X,Y) (vin1 - (R*Y) - X/C)/L; %circuit equation derived
above expressing dy/dt


% setting the method of calculating Yi and Xi

K_1 = feval(coup, t, X, Y);
L_1 = feval(pre_coup, t, X, Y);

K_2 = feval(coup, t+h/3, X+h*K_1/3, Y+h*L_1/3);
L_2 = feval(pre_coup, t+h/3, X+h*K_1/3, Y+h*L_1/3);

K_3 = feval(coup, t+2*h/3, X+h*(-K_1/3 + K_2), Y+h*(-L_1/3 + L_2));

L_3 = feval(pre_coup, t+2*h/3, X+h*(-K_1/3 + K_2), Y+h*(-L_1/3 +
L_2));

K_4 = feval(coup, t+h, X+h*(K_1 - K_2 + K_3), Y+h*(L_1 - L_2 + L_3));

L_4 = feval(pre_coup, t+h, X+h*(K_1 - K_2 + K_3), Y+h*(L_1 - L_2 +
L_3));

X_new = (K_1 + 3*K_2 + 3*K_3 + K_4)/8;
Y_new = (L_1 + 3*L_2 + 3*L_3 + L_4)/8;

Yi = Y + h*Y_new; %Yi formula
Xi = X + h*X_new; %Xi formula

end
```

Fig65: *RK4second.m function implementing the 4th order Runge-Kutta 3/8 algorithm*

2.5: RLC Series Circuit Modeling

The given RLC series circuit is modelled using RLC_script.m file which has been written for different input values and forms of the input voltage. In this section of the report, a MATLAB code will be presented for the different cases and the results will be analyzed in the context of the circuit. There are four cases to consider in this experiment:

2.5.1: Step signal with amplitude $V_{in} = 5V$

```
R = 250; %resistance in Ohms
L = 0.650; %inductance in henries
C = 3*10^-6; %capacitance in farads
qi = 500*10^(-9); %initial value of the charge
i0 = 0; %inital value of the current
ti = 0; %t = 0; (initial time)

figure;

vin = @(t) 5; %step input signal
h = 0.00001; %step size
tfinal = 0.05;
N = round((tfinal-ti)/h); %to set the upper bound of the loop
Ya = zeros(1,N); %array for current with zeros
Xb = zeros(1,N); %array for charge (state of the equation) with zeros
Tc = zeros(1,N); %array for time with zeros

Xb(1) = qi; %first element of the array equal to inital charge
Ya(1) = i0; %first element of the array equal to inital current
Tc(1) = ti; %first element of the array equal to initial time

for i=1:N-1;
    [Ya(i+1), Xb(i+1)] = RK4second(R,C,L,h,Xb(i), Ya(i), Tc(i),vin);
    %calling the 3/8 runge kutta 4th order method
    Tc(i+1)=Tc(i)+h; %increment the time
end

subplot(3,3,1)
plot(Tc,R*Ya); %Graph between the output voltage vs time
title('Vout with step signal input with 5 V amplitude');
ylabel ('Vout/V');
xlabel ('t/s');
```

Fig66: *RLC_script.m script implementing the step signal input voltage V_{in} = 5 V with an amplitude of 5*

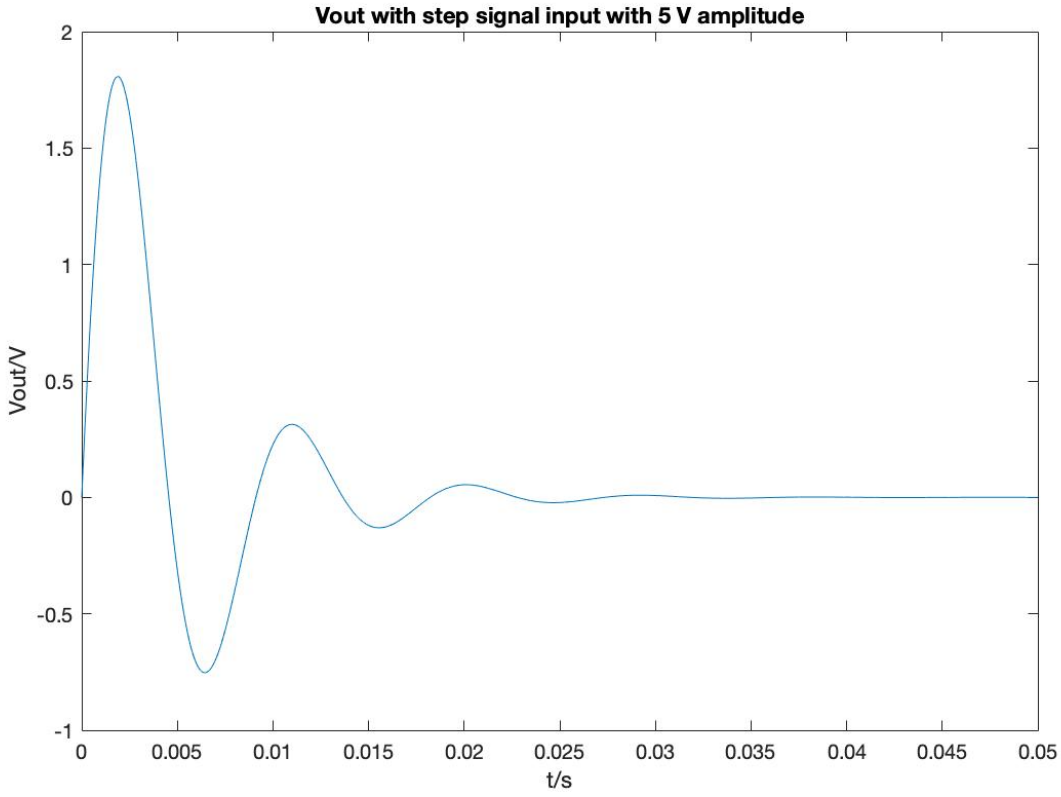

Fig67: *V_{out} when the input is a step signal input voltage with amplitude V_{in} = 5 V*

The output of a circuit to a step-input consists of a steady state response and a transient. The initial transient response is expected due to a sudden change in the voltage at t = 0 which is followed by a steady state response. The steady state output in Fig67 is seen to be 0 V. This is because for low frequencies, the capacitor becomes an open circuit and consequently, no current flows through producing a V_{out} = 0 V (Brookes).

This result can also be analyzed through a different lens. When the step-input is 5 V, the capacitor starts to charge slowly and when, it's fully charged, the current stops flowing. Therefore, the capacitor starts to discharge, and the energy is transferred to the inductor. But simultaneously, the current across the inductor can't change instantly and therefore, the voltage across the resistor decreases so that the capacitor can discharge. The energy sloshes between the capacitor and the inductor and this continues, until a steady-state value of 0 V is reached. Therefore, the output voltage finally decreases to 0 (Brookes).

The oscillations in Fig67 can be explained in the context of damping in RLC series circuits. In general, there are three types of damping namely: overdamped, critically damped and underdamped. In any RLC series circuit, KVL around the loop gives a second order differential equation which can be divided into the following parts (Step Response of RLC Circuits):

$$x(t) = x_n(t) + x_p(t)$$

where $x_n(t)$ gives the complementary solution and $x_p(t)$ gives the particular solution. Looking at the complementary solution, it can be expressed as follows:

$$s^2 + 2\zeta w_n s + w_n^2 = 0$$

In the above equation, ζ is the damping ratio and w_n is the undamped resonant frequency. The roots of the characteristic equation are given by the following (Step Response of RLC Circuits):

$$s_{1,2} = -\zeta w_n \pm w_n\sqrt{\zeta^2 - 1}$$

The RLC series circuit given has the following characteristic equation:

$$s^2 + \frac{R}{L}s + \frac{1}{LC} = 0$$

For the given RLC series circuit parameters, the circuit is underdamped with $\zeta < 1$ and consequently the equation; $s^2 + 2\zeta w_n s + w_n^2 = 0$ producing two complex conjugate roots which are as follows:

$$s = -\sigma \pm jw_d = -\zeta w_n \pm jw_n\sqrt{\zeta^2 - 1}$$

As a result, the complementary solution can be expressed as follows:

$$x_n(t) = e^{\sigma t}(A_1 cosw_d t + A_2 sinw_d t) = e^{-\sigma t}(Acosw_d t + \theta)$$

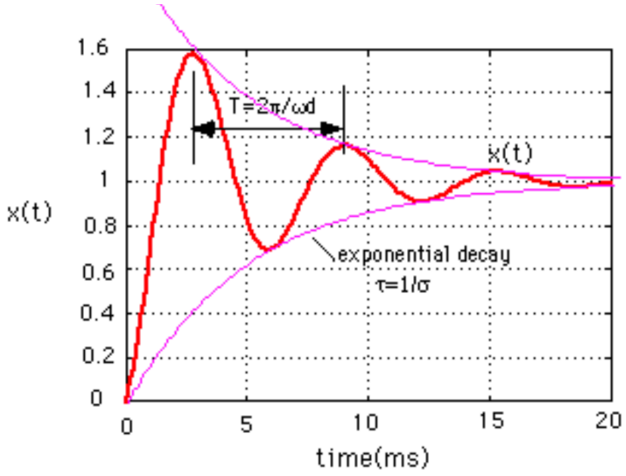

Fig68: *Underdamped response to a step function when* $\zeta < 1$ (Step Response of RLC Circuits)

The transfer function of the given RLC circuit is as follows:

$$\frac{V_{out}\,(s)}{5} = \frac{\frac{R}{L}}{s^2 + \frac{R}{L}s + \frac{1}{LC}}$$

Completing the square of the above equation, the following results are obtained (Brookes):

$$\frac{V_{out}\,(s)}{5} = \frac{\frac{R}{L}}{(s+a)^2 + w_d^2}$$

$$\frac{V_{out}(s)}{5} = \frac{R}{Lw_d}\frac{w_d}{(s+a)^2 + w_d^2}$$

$$\frac{V_{out}(t)}{5} = \frac{R}{Lw_d}e^{-at}\sin\,(w_d t)$$

where;

$$w_d = \sqrt{w_0^2 - a^2} = w_0\sqrt{1-\zeta}$$

Theoretically, the fact that damping ratio is less than 1 of the given RLC series circuit is shown by the following formula (Step Response of RLC Circuits):

$$\zeta = \frac{R}{2}\sqrt{\frac{C}{L}} = 0.269 < 1$$

Therefore, it can be confirmed that the theoretical result of the damping factor being less than 1 implies that the complementary solution $x_n(t)$ would result in oscillations to a step input function as seen in Fig68. The oscillations were also seen in the actual graph wherein a step signal of 5 V was applied to the RLC series circuit in Fig1 and V_{out} was measured across the resistor displayed an initial transient response with oscillations before reaching a steady state value of 0 V.

2.5.2: Impulsive signal with decay $V_{in} = 5e^{(-\frac{t^2}{\tau})}$ where $\tau = 3\ (ms)^2$

```
vin = @(t) 5 * exp(-t^2/(3*10^-6));

for i=1:N-1;
    [Ya(i+1), Xb(i+1)] = RK4second(R,C,L,h,Xb(i), Ya(i), Tc(i),vin);
    Tc(i+1)=Tc(i)+h;
end

subplot(3,3,2);
plot(Tc,R*Ya);
title('Vout with a 5 V input impulsive signal with decay');
ylabel ('Vout/V');
xlabel ('t/s');
```

Fig69: *RLC_script.m script implementing the impulsive signal with decay* $V_{in} = 5e^{(-\frac{t^2}{\tau})}$ *where* $\tau = 3\ (ms)^2$

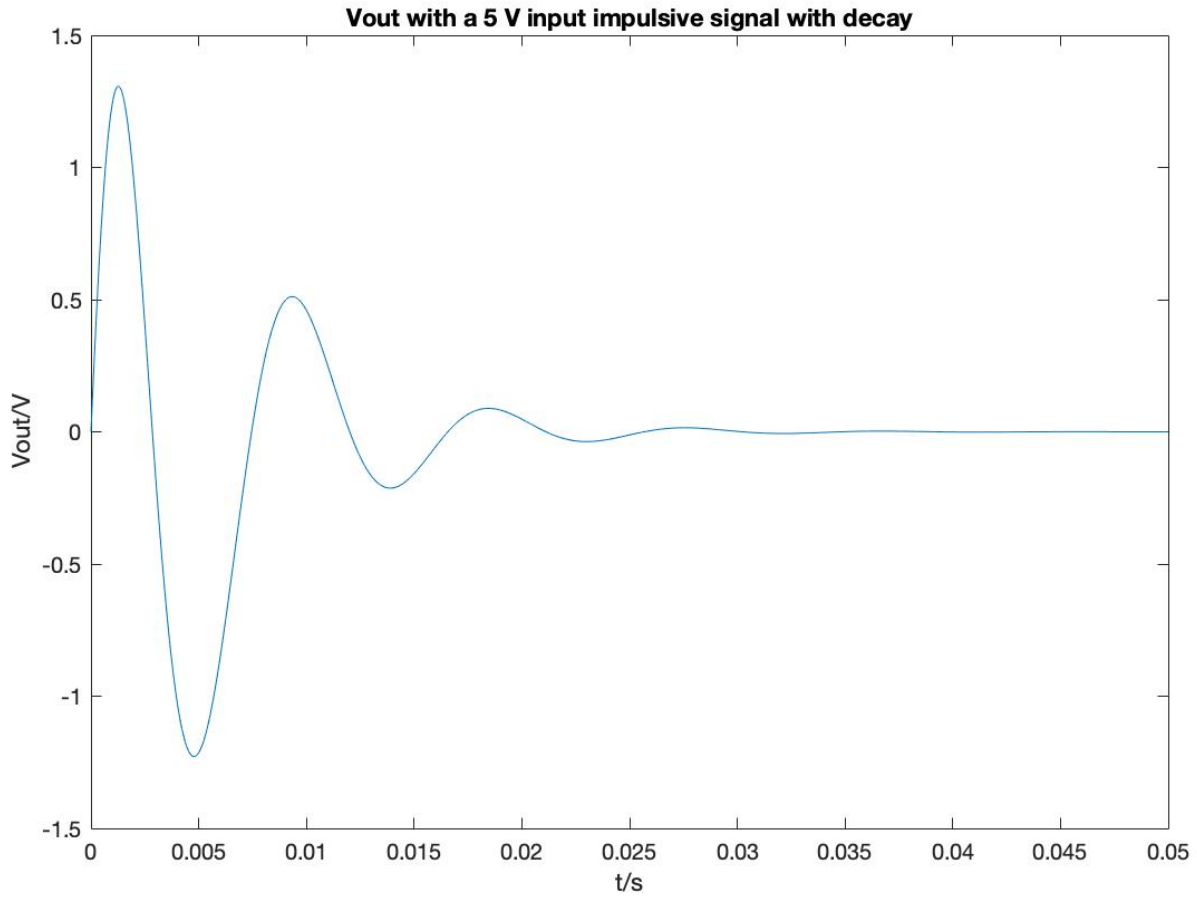

Fig70: *V_{out} when the input is an impulsive signal with decay $V_{in} = 5e^{(-\frac{t^2}{\tau})}$ where $\tau = 3\ (ms)^2$*

It is expected that the graph in Fig70 is similar to the graph of the output voltage to a step signal as shown in Fig67. For the purposes of this explanation, it is safe to assume that the input voltage is an impulse input with an amplitude of 5 V. Neither the capacitor nor the resistor can absorb the impulse voltage since for the latter, the circuit current would have to become infinite and the inductor current would not allow that. Therefore, the entire impulse voltage is applied across the inductor. There's a sudden change in the current from 0 to 5/L A due to 5 Wb-T flux linkage applied across the inductor (Kumar).

While the inductor current $i_L(t)$ at t = 0^+ becomes 5/L A, the capacitor voltage $v_C(t)$ at t = 0^+ remains at 0 V. Since this is an impulse input of 5 V at t = 0, the input source becomes 0 V for $t \geq 0^+$and therefore, it becomes a short circuit. However, the energy stored in the inductor is still $\frac{1}{2}L\,J$ from the impulse input while the capacitor energy is 0 J. So, the process of energy being exchanged between the capacitor continues until V_{out} settles down to 0 V of steady-state response (Kumar).

2.5.3: Square wave input with Amplitude $V_{in} = 5\ V$ and different frequencies $f = 5\ Hz, f = 100\ Hz, f = 500\ Hz$

```
%square wave with amplitude Vin = 5V for different frequencies:
%f = 5 Hz, 100 Hz, 500 Hz

%f=5 Hz

vin = @(t) 5*square(10*pi*t);
h = 0.0001;
tfinal = 0.5;
N = round((tfinal-ti)/h);
Ya = zeros(1,N);
Xb = zeros(1,N);
Tc = zeros(1,N);

Xb(1) = qi;
Ya(1) = i0;
Tc(1) = ti;

for i=1:N-1
    [Ya(i+1), Xb(i+1)] = RK4second(R,C,L,h,Xb(i), Ya(i), Tc(i),vin);
    Tc(i+1)=Tc(i)+h;
end

subplot(3,3,3);
plot(Tc,R*Ya);
title('Vout with square wave input with f = 5 Hz and amplitude 5
V');
ylabel ('Vout/V');
xlabel ('t/s');

Vin = vin(Tc);
hold on;
plot(Tc, Vin);
```

Fig71: *RLC_script.m script implementing the input signal which is a square wave with amplitude 5 V and f = 5 Hz*

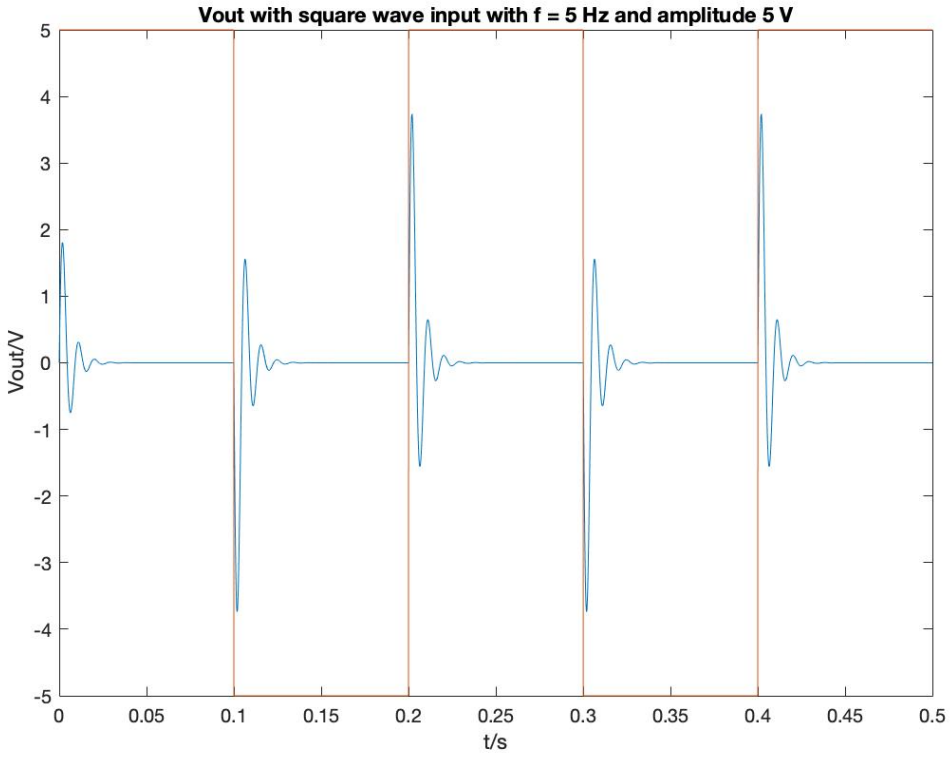

Fig72: *V_{out} when the input is a square wave with amplitude $V_{in} = 5\ V$ with a frequency of 5 Hz*

```
%f=100 Hz

vin = @(t) 5*square(200*pi*t);
h = 0.00001;
tfinal = 0.05;
N = round((tfinal-ti)/h);
Ya = zeros(1,N);
Xb = zeros(1,N);
Tc = zeros(1,N);

Xb(1) = qi;
Ya(1) = i0;
Tc(1) = ti;

for i=1:N-1
    [Ya(i+1), Xb(i+1)] = RK4second(R,C,L,h,Xb(i), Ya(i),
Tc(i),vin);
    Tc(i+1)=Tc(i)+h;
end

subplot(3,3,4);
plot(Tc,R*Ya);
title('Vout with square wave input with f = 100 Hz and amplitude
5 V');
ylabel ('Vout/V');
xlabel ('t/s');

Vin = vin(Tc);
hold on;
plot(Tc, Vin);
```

Fig73: *RLC_script.m script implementing the input signal which is a square wave with amplitude 5 V and f = 100 Hz*

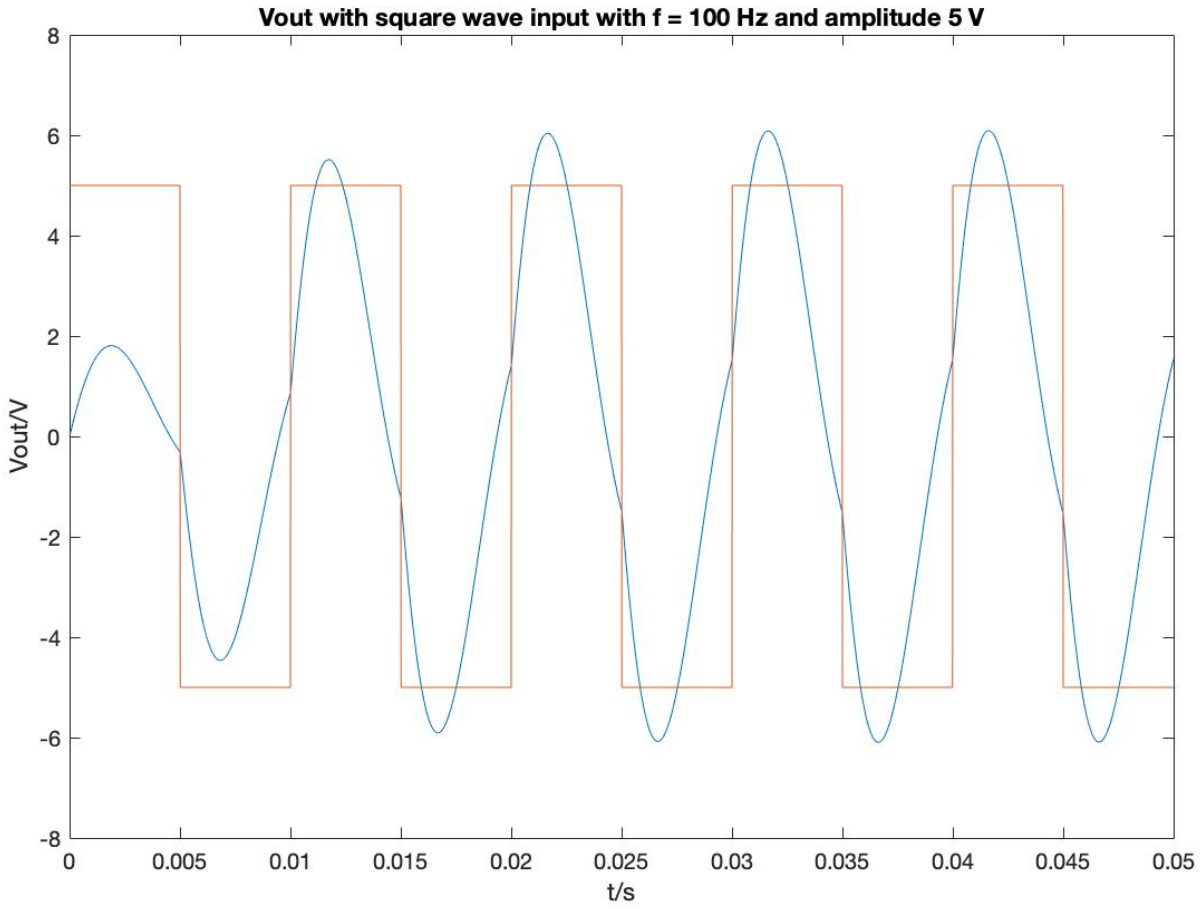

Fig74: *V_{out} when the input is a square wave with amplitude $V_{in} = 5\,V$ with a frequency of 100 Hz*

```
%f=500 Hz

vin = @(t) 5*square(1000*pi*t);
h = 0.00001;
tfinal = 0.01;
N = round((tfinal-ti)/h);
Ya = zeros(1,N);
Xb = zeros(1,N);
Tc = zeros(1,N);

Xb(1) = qi;
Ya(1) = i0;
Tc(1) = ti;

for i=1:N-1
    [Ya(i+1), Xb(i+1)] = RK4second(R,C,L,h, Xb(i), Ya(i),
Tc(i),vin);
    Tc(i+1)=Tc(i)+h;
end

subplot(3,3,5);
plot(Tc,R*Ya);
title('Vout with square wave input with f = 500 Hz and amplitude 5
V');
ylabel ('Vout/V');
xlabel ('t/s');

Vin = vin(Tc);
hold on;
plot(Tc, Vin);
```

Fig75: *RLC_script.m script implementing the input signal which is a square wave with amplitude 5 V and f = 500 Hz*

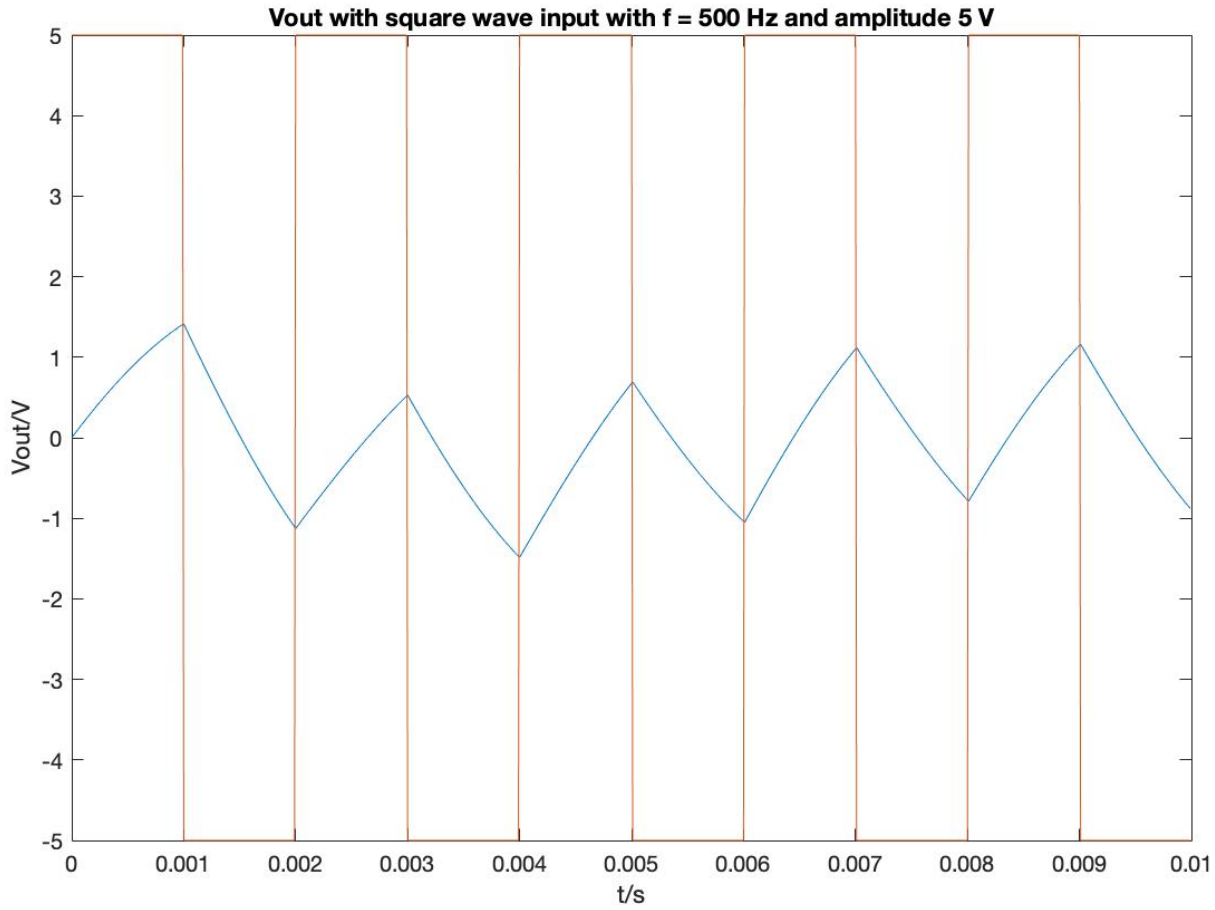

Fig76: *V_{out} when the input is a square wave with amplitude $V_{in} = 5\,V$ with a frequency of 500 Hz*

From Fig72, Fig74 and Fig76, it can be observed that the output voltage oscillates between positive and negative transients. Specifically, in the output voltage graph observed in Fig72 is similar to a step input signal voltage due to the fact that frequency of the square wave is 5 Hz.

2.5.4 Sine wave input with amplitude $V_{in} = 5\ V$ and different frequencies $f = 5\ Hz, f = 100\ Hz, f = 500\ Hz$

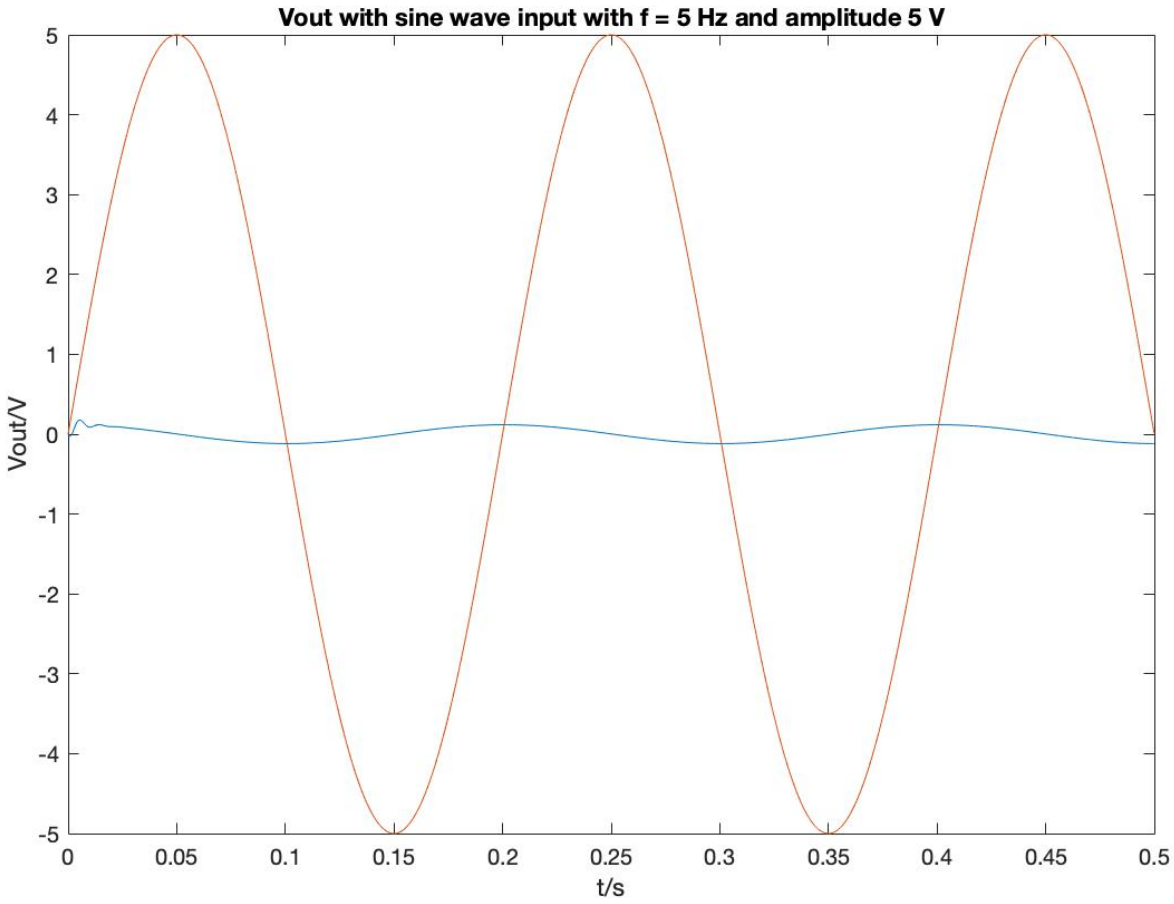

Fig77a: *V_{out} when the input is a sine wave with amplitude $V_{in} = 5\ V$ with a frequency of 5 Hz*

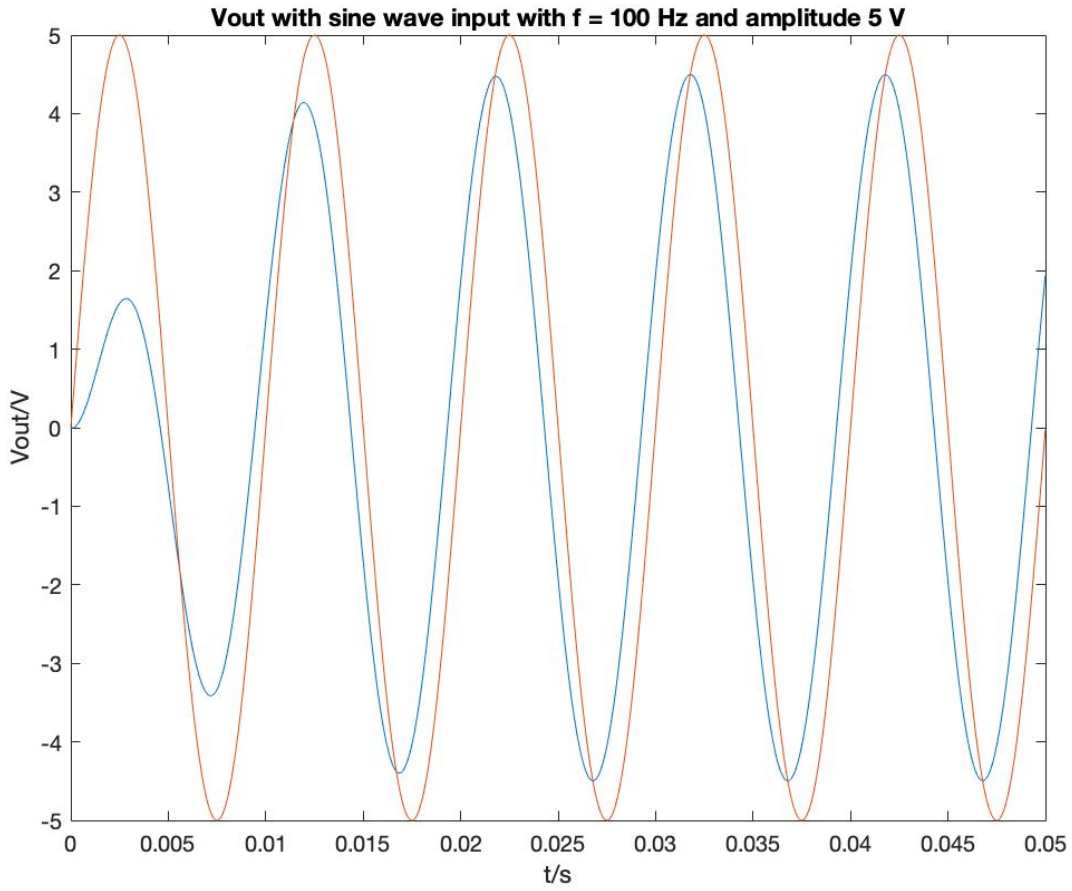

Fig77b: *V_{out} when the input is a sine wave with amplitude $V_{in} = 5\,V$ with a frequency of 100 Hz*

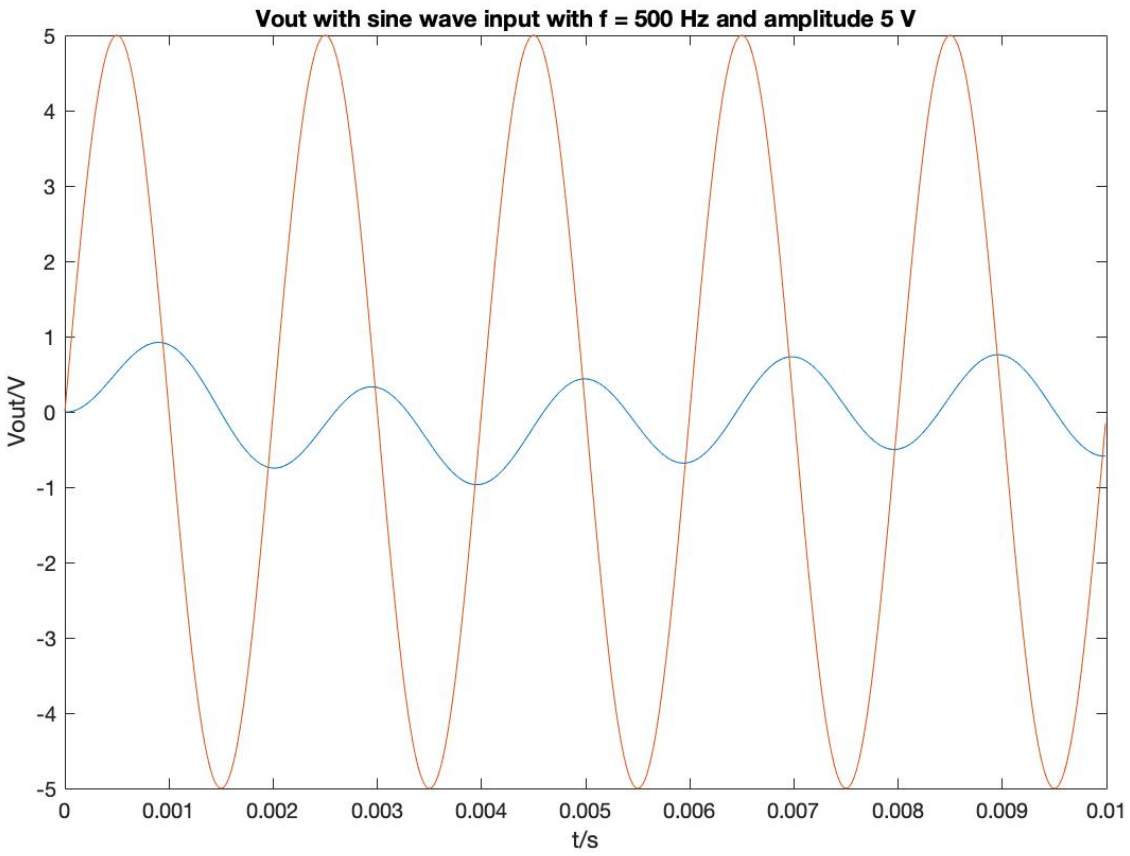

Fig77c: *V_{out} when the input is a sine wave with amplitude $V_{in} = 5\,V$ with a frequency of 500 Hz*

For the case when the input is a sine wave, it can be seen that for f = 5 Hz, the output of graph is out of phase by 90°. This is also shown by the bode plot of the circuit in Fig64 wherein for lower frequencies, there's a phase shift indicated at the output by 90°. On the other hand, at f = 100 Hz, the output sine wave is almost in phase as that of the input sine wave. This is due to the fact that the input frequency is closer to resonant frequency (natural frequency of the circuit) which is approximately, 113 Hz, as predicted in section 2.2 and confirmed along with the bode diagram of the given RLC series circuit. At the resonant frequency, there's a phase shift of 0° between the output and the input of the circuit as shown by the bode plot in Fig64.

Finally, at higher frequencies of f = 500 Hz, the output sine wave is also out of phase of the input sine wave. This time, it is out of phase by approximately -90° which is once again, in agreement with the bode plot in Fig64. For higher frequencies, the phase shift between the output and the input is -90°. All the three aforementioned cases of the output wave being out of phase by

90°, being in phase and being out of phase by -90° also apply to square wave inputs for f = 5 Hz, f = 100 Hz and f = 500 Hz.

3 Finite Differences for PDE (Exercises 4, 5 and 6)

3.1: Introduction

Given 1-D heat equation: $\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}$, $0 < x < 1$, $t > 0$ along with zero boundary conditions $y(0,t) = y(1,t) = 0$, and initial condition $y(x,0) = y_0(x)$, can be solved numerically using the finite difference method. The given equation can physically translate to the heat distribution across a thin, insulated metal rod, of length 1 m, with boundary conditions that refer to the ends of the rod being at temperature of 0 K. This section of the report will contain the following:

- Solving the given 1-D heat equation along with finite_script.m to implement the finite difference method.
- Testing the conditions which are zero-boundary, constant and time varying, for the three basic cases which are a tent function and a sinusoidal function given by:
    - $y_0(x) = \sin(2\pi x)$
    - $y_0(x) = |\sin(2\pi x)|$
    - $y_0(x) = f(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2-2x, & x \in [0.5,1] \end{cases}$
    - $y_0(x) = \cos\frac{\pi x}{2}$
    - $y_0(x) = (x-0.5)^2$
    - $y_0(x) = e^{x+1} - 5$

Points such as how to allocate boundary conditions, implementing the central algorithm (using MATLAB), choosing the values of h, k and plotting U_j^m for sensible choice of values of m will be considered.

3.2: Solving the given heat equation by implementing the finite difference method using the central algorithm theorem (Exercise 4 and 5)

```
N=50; h=1/N; c=1; k=1/10000; v=k/(h^2); B=1-2*v;
u=zeros(N-1,N+1); % defining a 2D array of zeros
for j=0:h:1 % initialize the array, u, with the desired input %function
%To plot one, comment out other two u(1,c) lines in the loop
  u(1,c)=(sin(2*pi*j)); % initializing u for the t=1 case
% u(1,c)=abs(sin(2*pi*j));
% u(1,c)=cos((pi*j)/2); % to test for non-zero boundaries
% u(1,c)=(j-0.5)^2; %additional testing case with polynomial term
% u(1,c)=exp(j+1)-5; %additional testing case with exponential term
  c=c+1; % c increments as j increments from 0 to 1 in intervals of %0.02
end
% u=triangularPulse(0,1,x); %for tent function, comment out the above % for
loop

for m=1:20 %m controls loop iterations, ie number of %lines plotted
    for j=2:N %runs N-1 times
% Implementing %central algorithm using the 2D array:
        u(m+1,j)=v*u(m,j-1)+B*u(m,j)+v*u(m,j+1);
% TESTING FOR DIFFERENT BOUNDARY CONDITIONS: please comment out the % other 2
cases
% Zero Boundary Conditions:
             u(m+1,1)=0;
             u(m+1,N+1)=0;
% Constant non-zero boundary conditions:
%          u(m+1,1)=0.5;
%          u(m+1,N+1)=0.5;
% Time Varying Non-Zero boundary conditions:
          u(m+1,1)=cos((pi*m*k)/4);
          u(m+1,N+1)=cos((pi*m*k)/4);
    end
end
x=[0:h:1];
 for i=1:m
    hold on
plot(x,u(i,:))
title('Variation of heat distribution across a metal rod with respect to
time');
ylabel ('Temperature/K');
xlabel ('Length of the rod/m');
 end
```

Fig76: *finite_script.m, a MATLAB script, implementing the finite difference method using the Central Algorithm Theorem which is* $y_j^{m+1} = v \times y_{j-1}^m + (1-2v) \times y_j^m + v \times y_{j+1}^m$ *to solve the1-D heat equation* $\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}, 0 < x < 1,\ t > 0$

3.2.1: Calculating the value of N, number of segments (or h) where $h = \frac{1}{N}$

The value of 'h' (Δx) (distance between two segments) chosen is 0.02 and it depends on the value of N (number of segments) which is chosen as 50. The relationship between the value of h and N is defined as $h = \frac{1}{N}$. The suitable value of 'h' was found by conducting trial and error methods at extreme values of N and they are explored below:

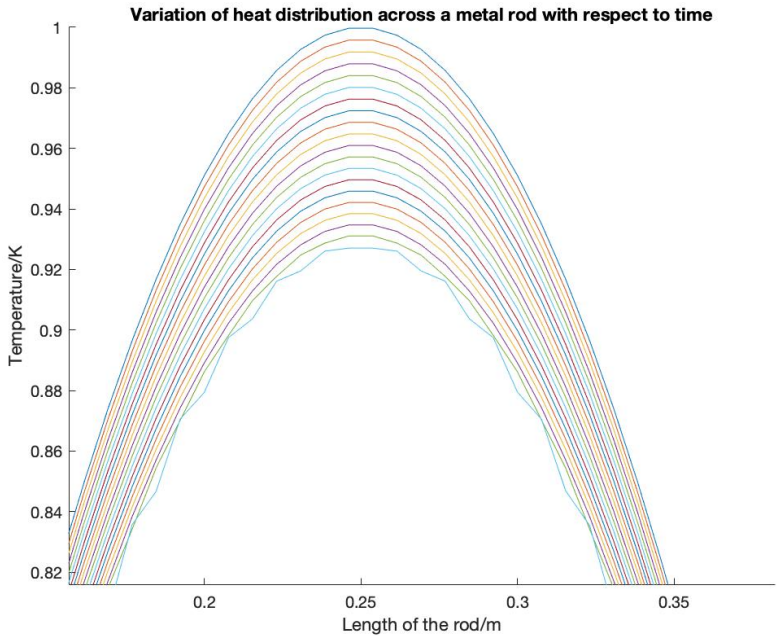

Fig77: *Plot of sine wave with zero boundary conditions, for N = 130*

For N = 130, the discretization of the lines can be seen and specifically, the bottom blue line fails to retain its shape. This is undesirable for the purposes of this experiment.

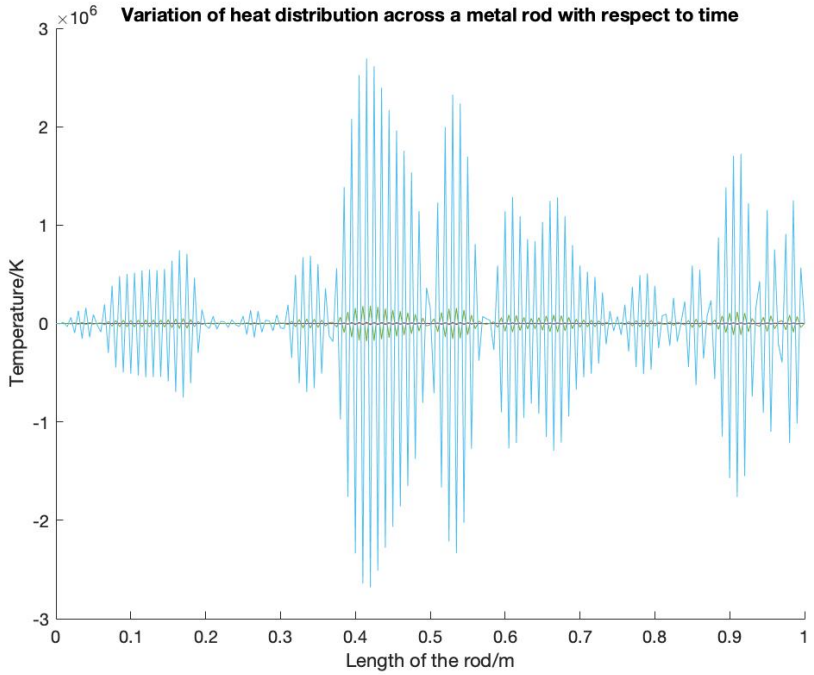

Fig78: *Plot of sine wave with zero boundary conditions, for N = 200.*

In Fig78, the value of N is increased to 200 and it can be seen that the scale of y-axis becomes insignificant because of the small values of y. From this it can be concluded that, for higher values of N (or consequently, lower values of h), the plot fails to retain its shape for the zero-boundary condition sine wave and hence, lower values of N need to be considered (or higher values of h).

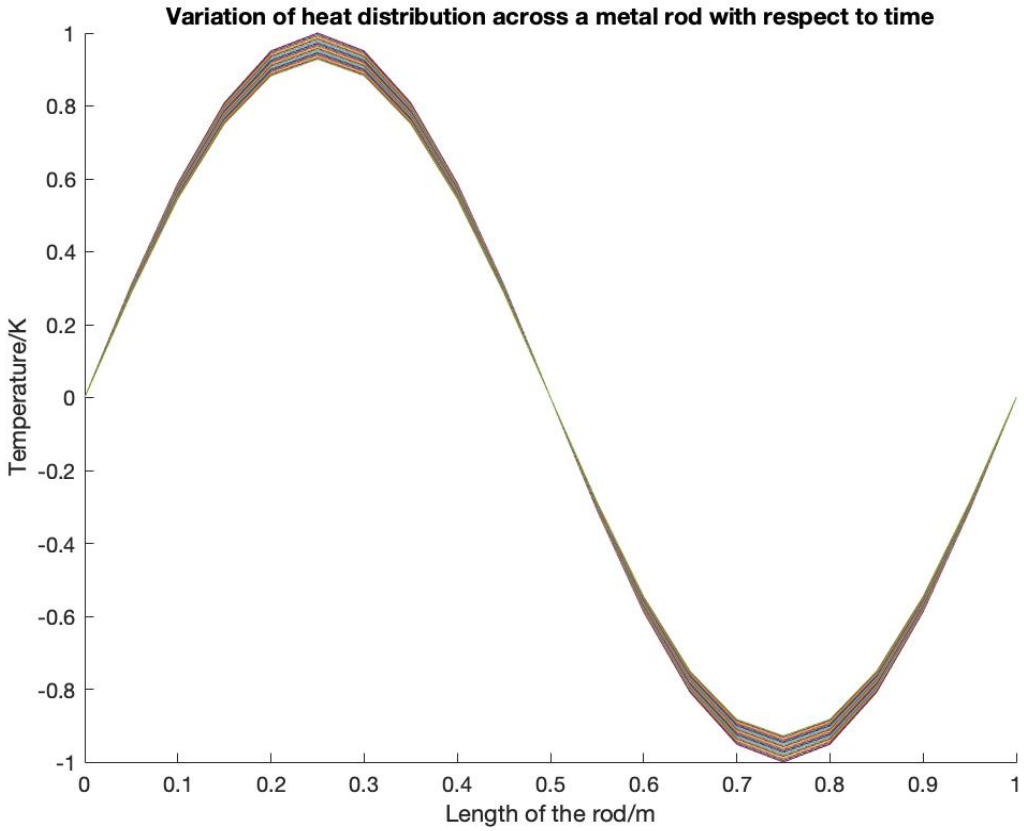

Fig79: *Plot of sine wave with zero boundary conditions, for N = 20.*

In Fig79, the value of N is lowered to 20 and the resultant zero-boundary condition sine wave is plotted. It can be observed that for N = 20, the graph isn't smooth and consequently, the straight-line approximations can be seen.

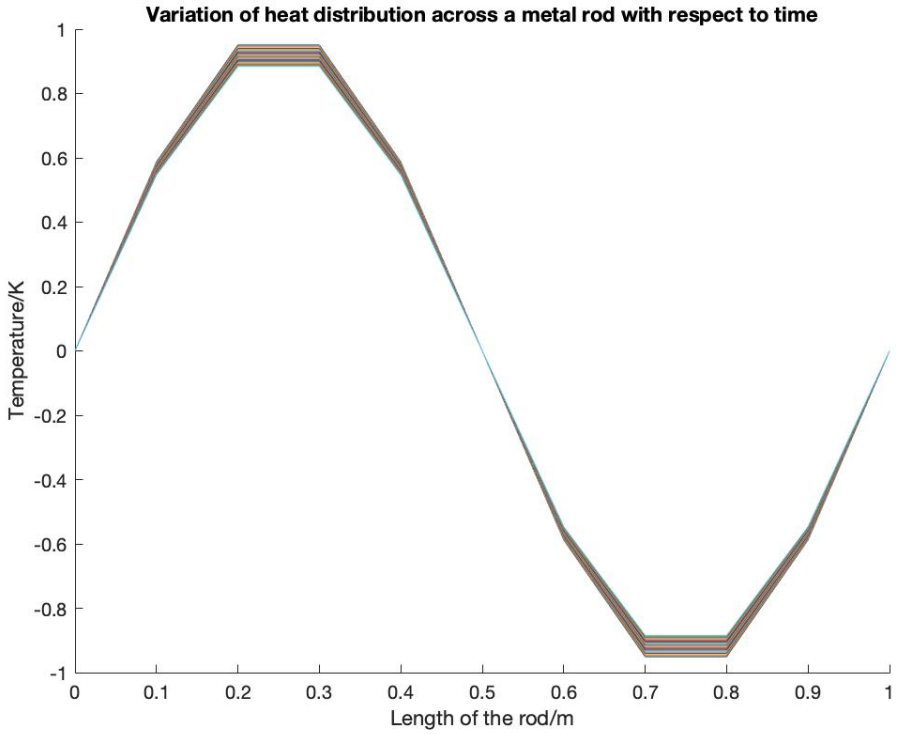

Fig80: *Plot of sine wave with zero boundary conditions, for N = 10.*

By decreasing the value of N further to 10, it is observed that the graph becomes less smooth along with an increase in straight-line observations. In conclusion, at extreme values of N, the plot of the zero-boundary condition sine wave doesn't behave as expected. Therefore, through extensive trial and error methods, it was found that at N = 50, the graph becomes smooth.

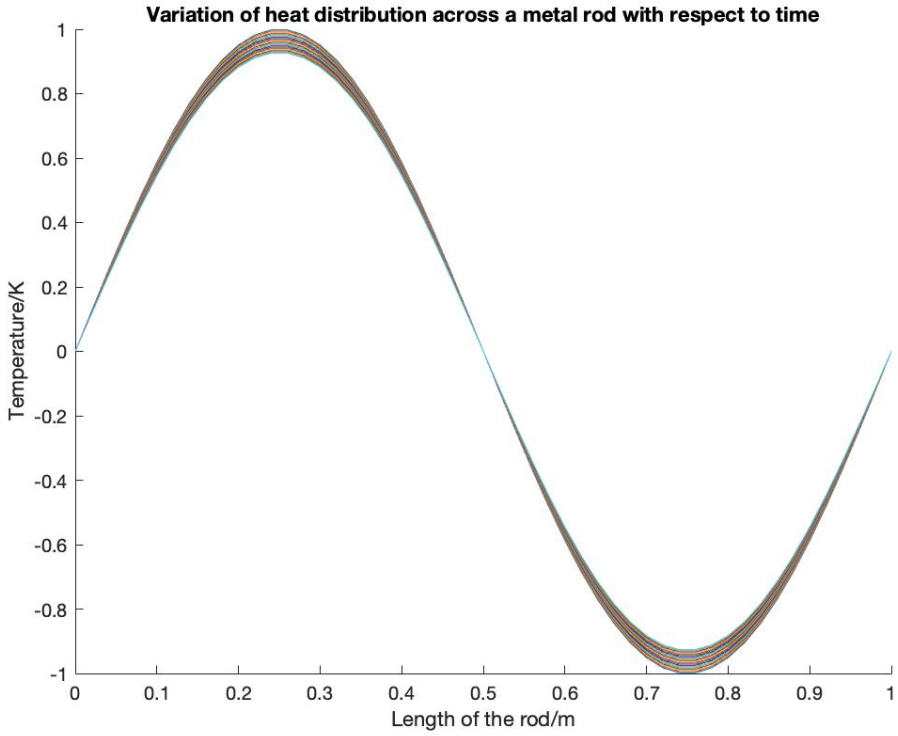

Fig81: *Plot of sine wave with zero boundary conditions, for N = 50 or h = 0.02.*

3.2.2: Calculating the value of k

The total number of points including boundaries is given by $N + 1 = 50 + 1 = 51$. It is given that $v = \frac{k}{h^2}$ where v is a key parameter used to discretize the 1D heat equation. It combines the coefficient of the $\frac{\partial^2 y}{\partial x^2}$ term (known as the diffusion coefficient), that has a value of 1 in this case, and the parameters: Δx and Δt into a single, dimensionless parameter. k points to arbitrary position in time (Nucinkis).

If the value of k decreases (or v), the general shape of the plot is indistinguishable. Since, by error analysis, $v \leq \frac{1}{2}$, it can be deduced that $k{:}\,h^2 = 1{:}\,2$ which implies that $k{:}\,h = 1{:}\,\sqrt{2}$ (Nucinkis). For $h = 0.02,$ as was derived in section 3.2.1, $k = \frac{0.02}{\sqrt{2}} = 0.0014$. Hence, the required result is, $k < 0.0014$. While experimenting, it was found that for $k < \frac{1}{4800}$, the general shape begins to get appear, however is still distorted.

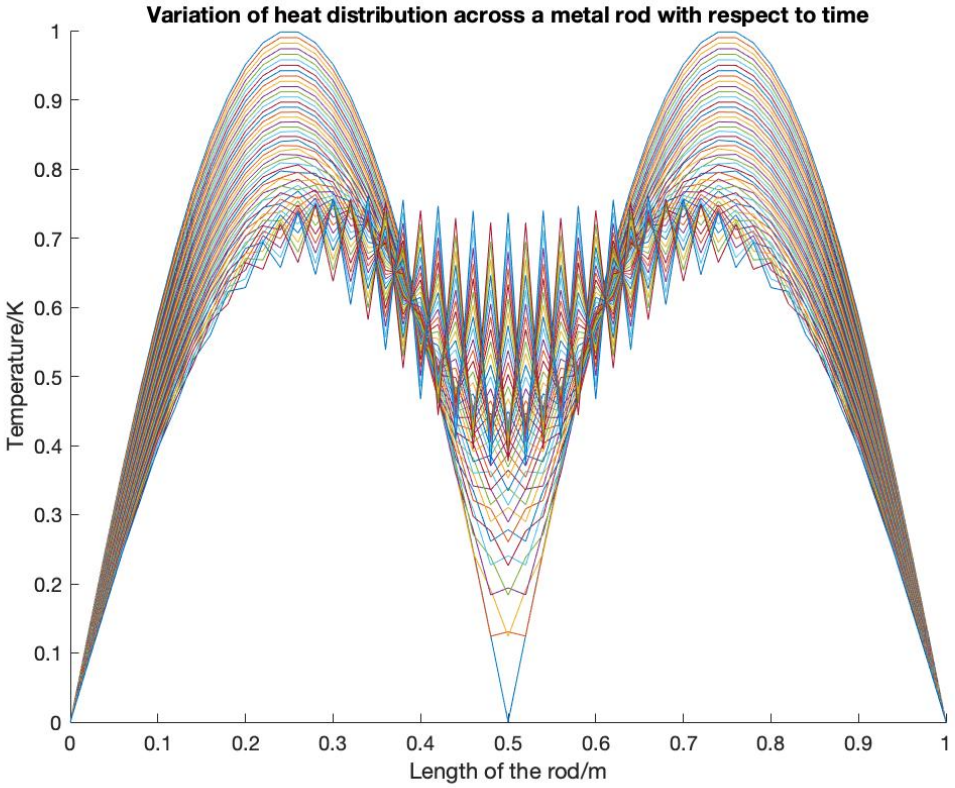

Fig82: *Distortion observed for* $k = \frac{1}{4800}$ *when plotting the absolute sine wave with zero boundary conditions*

When the value of k is reduced even further to $k = \frac{1}{1000000}$, it is observed that the spacing between iterations becomes unnoticeable.

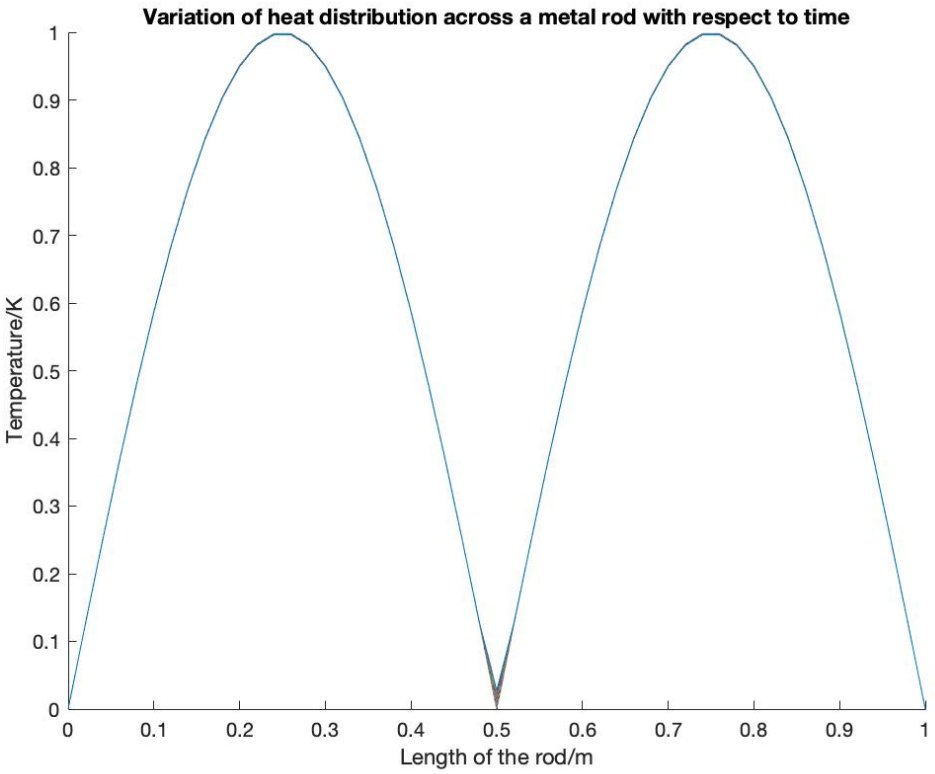

Fig83: *Unnoticeable space between iterations for* $k = \frac{1}{1000000}$ *when plotting the absolute sine wave with zero boundary conditions*

As in section 3.2.1, through extensive trial and error methods, it was found that the plots become desirable for $k = \frac{1}{10000}$.

3.2.3: Plots for $y_0(x) = \sin(2\pi x)$ for varying boundary conditions

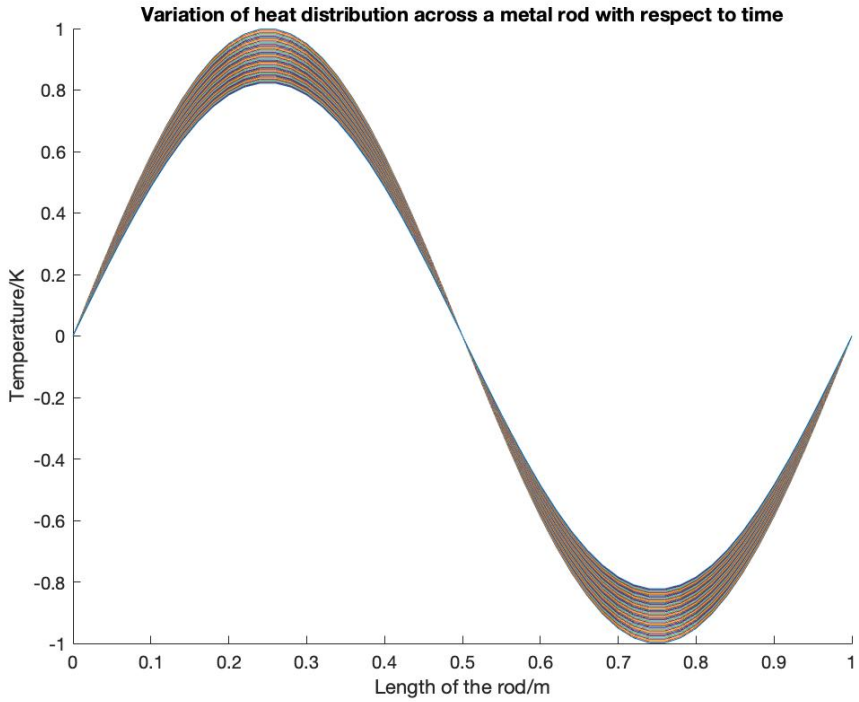

Fig84: *Plot for* $y_0(x) = sin(2\pi x)$ *for zero-boundary conditions*

Fig84 shows an accurate sine wave plot for m = 50 where m is the number of iterations. It can be observed that the zero points are x = 0, x = 0.5, and x = 1 (since the original x-axis values have been multiplied by 2π). The points are approached by a clean linear approximation of lines. Energy can be calculated using the formula given below, where T is the time period:

$$E = \int_{-T/2}^{T/2} [y(x)]^2 dx$$

Using this, the net energy of a sine wave is zero. As m, or time, increases, the energy remains at zero as there is no net change and consequently, the sine wave distribution tends to a horizontal line y = 0 since the net temperature is 0 K.

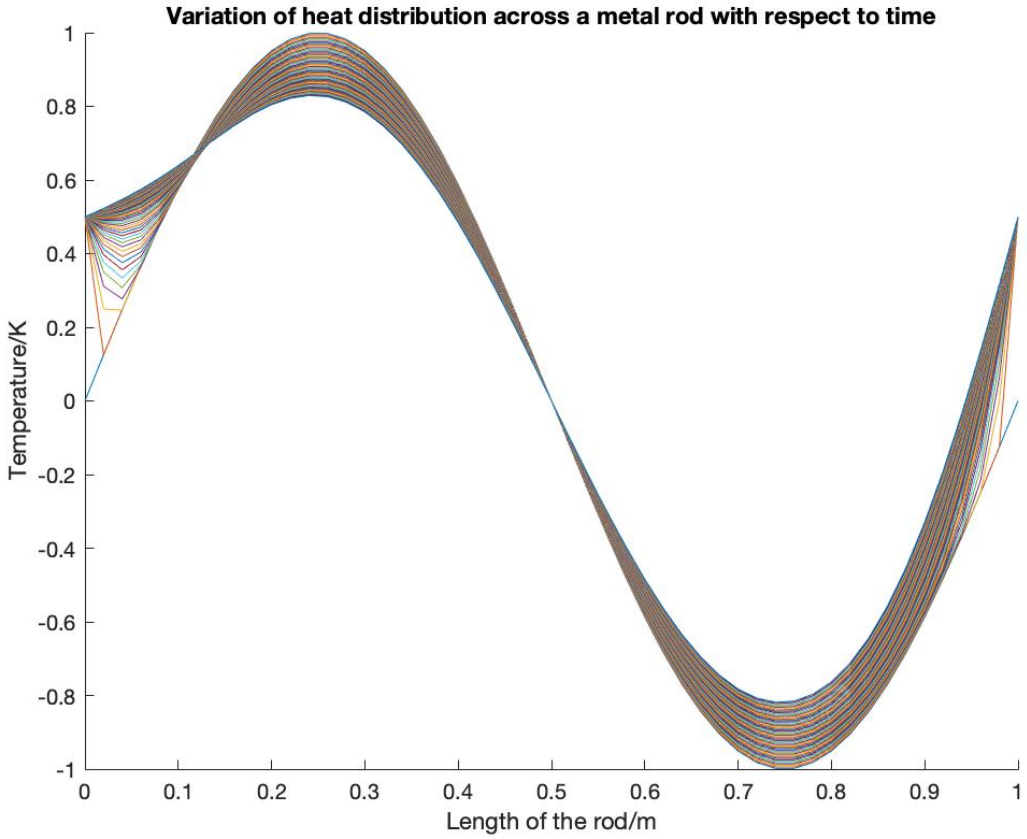

Fig85: *Plot for* $y_0(x) = sin(2\pi x)$ *for constant (0.5) non-zero-boundary conditions*

From Fig85, it can be observed that, the results are as expected. Given the net energy of the sine curve is zero, the boundary conditions can be seen as supplying a temperature to the system. The net temperature is no longer 0 K, due to the positive component being added by the boundary condition (Hancock). The maximum net temperature is the sum of the energy of the function, and the temperature being supplied from the boundaries. Since there is no set method to calculate the exact net temperature, it can be concluded, that it is within the range: $0\ K < T \leq 0.5\ K$, from the plots in Fig19.

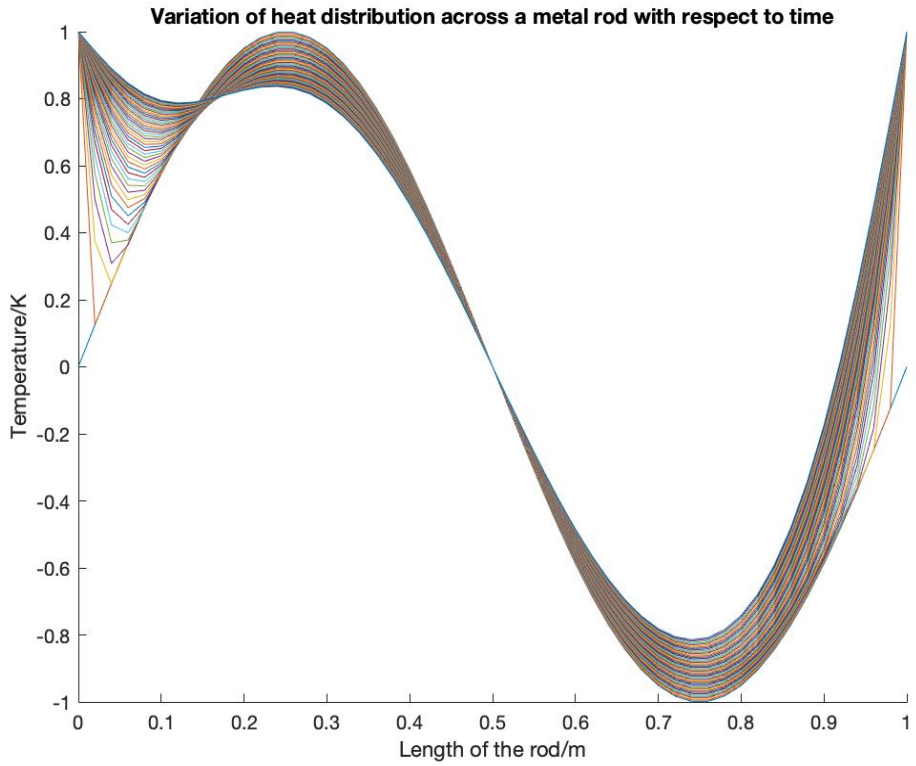

Fig86: *Plot for* $y_0(x) = sin(2\pi x)$ *for time varying boundary conditions for m = 20*

From Fig86, it can be observed that as m increases, the gradient at which the approximation curve deviates from the zero conditions (0, 0), (1, 0) to the points (0, 1), (1, 1) and consequently, increases. The point of inflexion (0.5, 0), is approached with a high degree of accuracy. All time varying boundary plots shown will have boundaries tending to the line y = 1, because the boundaries have been fixed by a cos((pi * m * k) / 4) term in the code. The magnitude of this cos term is 1, hence the time-varying boundary peaks at 1.

The net temperature in this case can be said to be a function of time, as the value of m very clearly affects the concentration of lines outside the sine function, converging to 1. Since the energy of a sine wave is zero, the minimum net temperature in this distribution is 0 K. From the plots in Fig20, it can concluded, that the temperature is within the range: $0\ K < T \leq 0.5\ K$.

3.2.3: Plots for $y_0(x) = |\sin(2\pi x)|$ for varying boundary conditions

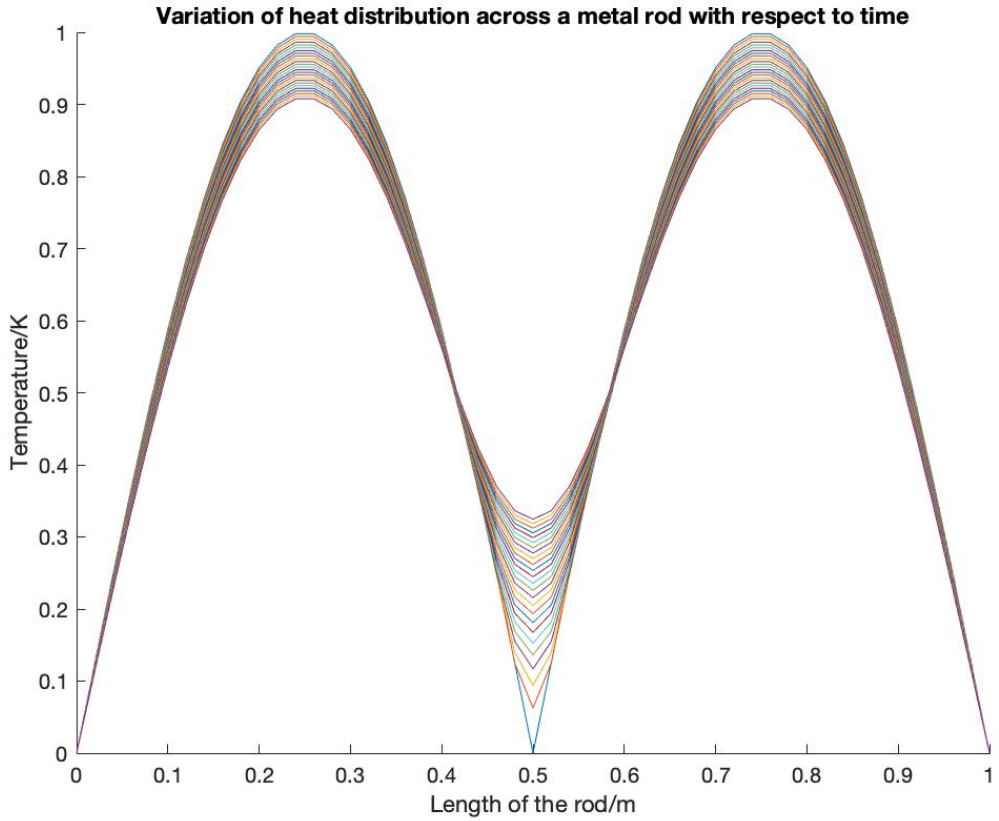

Fig87: *Plot for* $y_0(x) = |sin(2\pi x)|$ *for zero-boundary conditions for m = 25*

The plot in Fig87 also reflects a good level of accuracy as the linear approximation near 0 and 1 is very clear. Between the x = 0.4 and x = 0.6, the curves are clearly tending towards a triangular fit near x = 0.5. Energy can be calculated as follows: $\int_0^{0.5} |\sin x|^2 dx = \frac{1}{\sqrt{2}} = 0.707$ which is also equal to the net temperature.

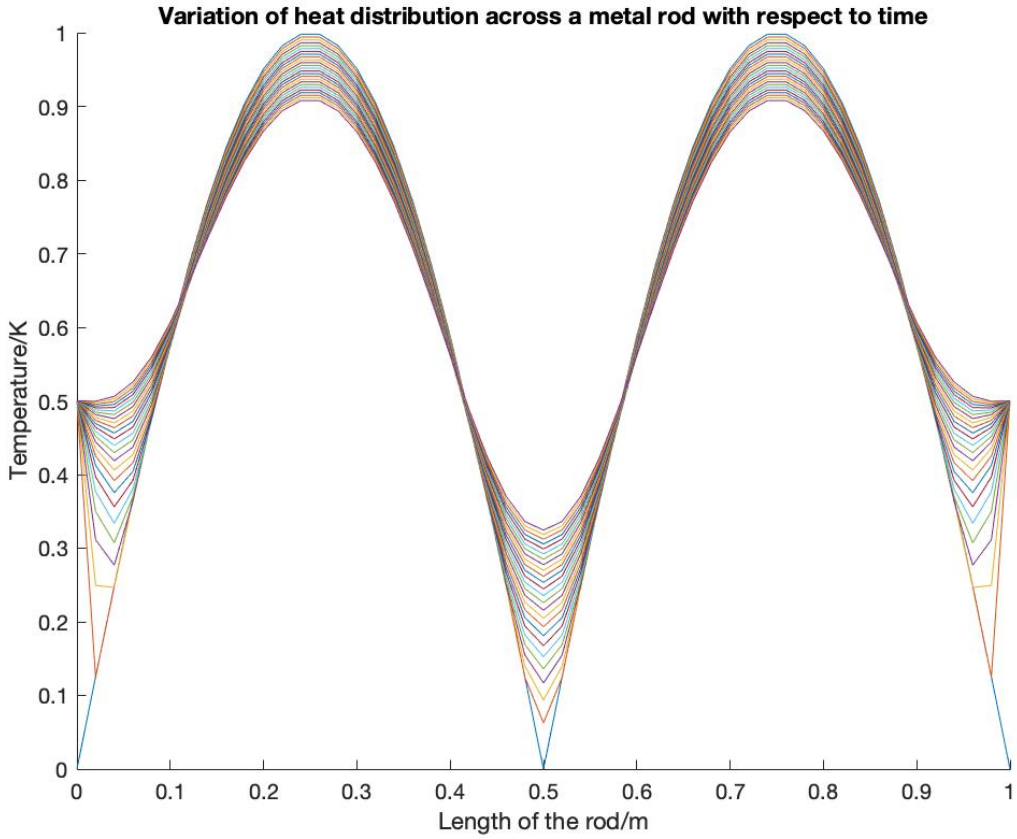

Fig88: *Plot for* $y_0(x) = |sin(2\pi x)|$ *for constant (0.5) non-zero-boundary conditions*

Outer lines converge at (0, 0.5) and (1, 0.5) respectively, in line with our expected results. The net temperature must be greater than $0.7\ K$, as the boundaries also supply 0.5 K. Thus, the net temperature must lie within the range: $0.7\ K < T < 1.2\ K$.

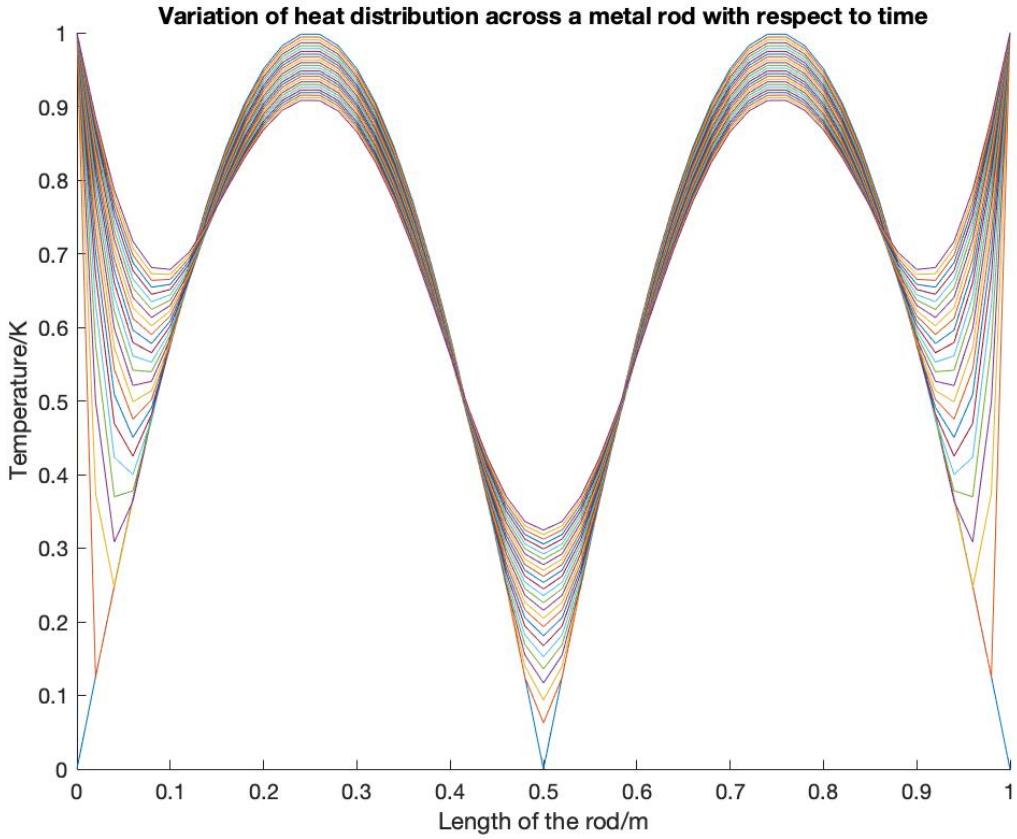

Fig89: *Plot for* $y_0(x) = |sin(2\pi x)|$ *for time varying boundary conditions for m = 25*

As expected, the curves rise upward towards the boundary point 1, occurring at x = 0 and x = 1, the concentration of lines is relatively less, corresponding to the smaller value of m taken. The net temperature of the time-varying case depends on time, and therefore on the value of m.

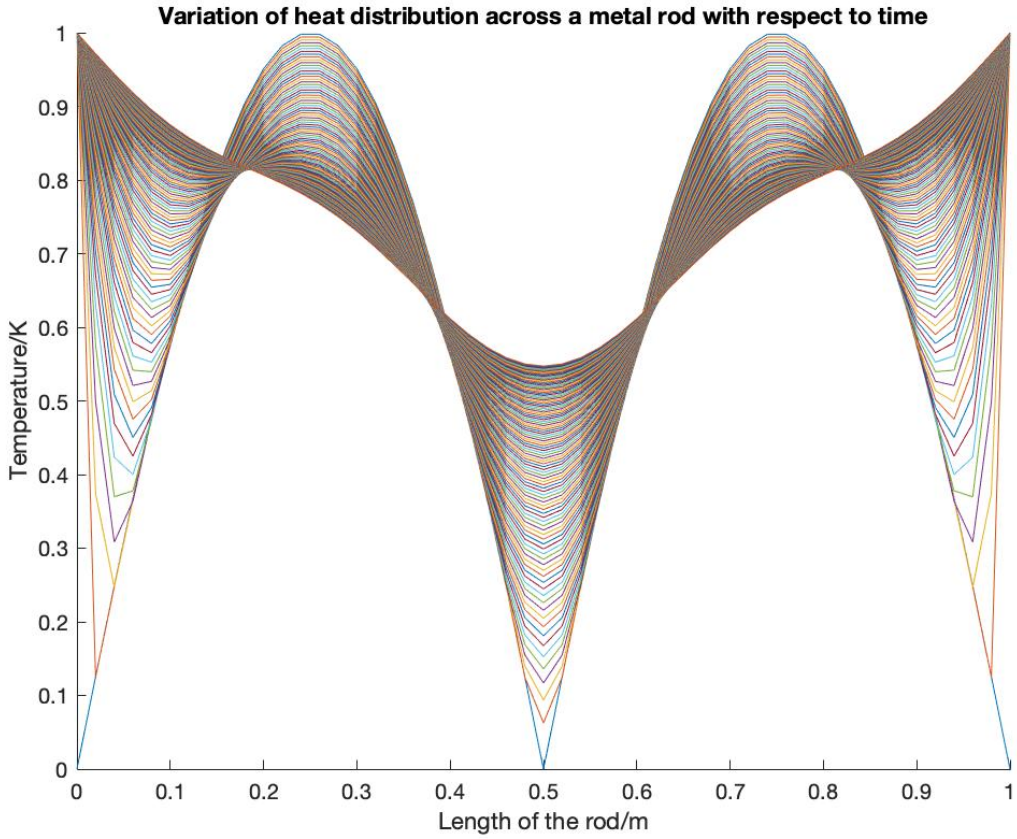

Fig90: *Plot for* $y_0(x) = |sin(2\pi x)|$ *for time varying boundary conditions for m = 100*

For large values of m, the absolute sine wave plotted can be clearly seen "twisting" at about x = 0.4 and x = 0.6. At each "twisting" point, the rate of the gradient change of the distribution reaches a maximum, causing a greater heat diffusion. Hence, a concentrated number of lines are obtained at this point. This phenomenon can be observed better for larger values of m, as illustrated in the plots in Fig90 as compared to Fig89.

3.2.4: Plots for $y_0(x) = f(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2 - 2x, & x \in [0.5,1] \end{cases}$ for varying boundary conditions

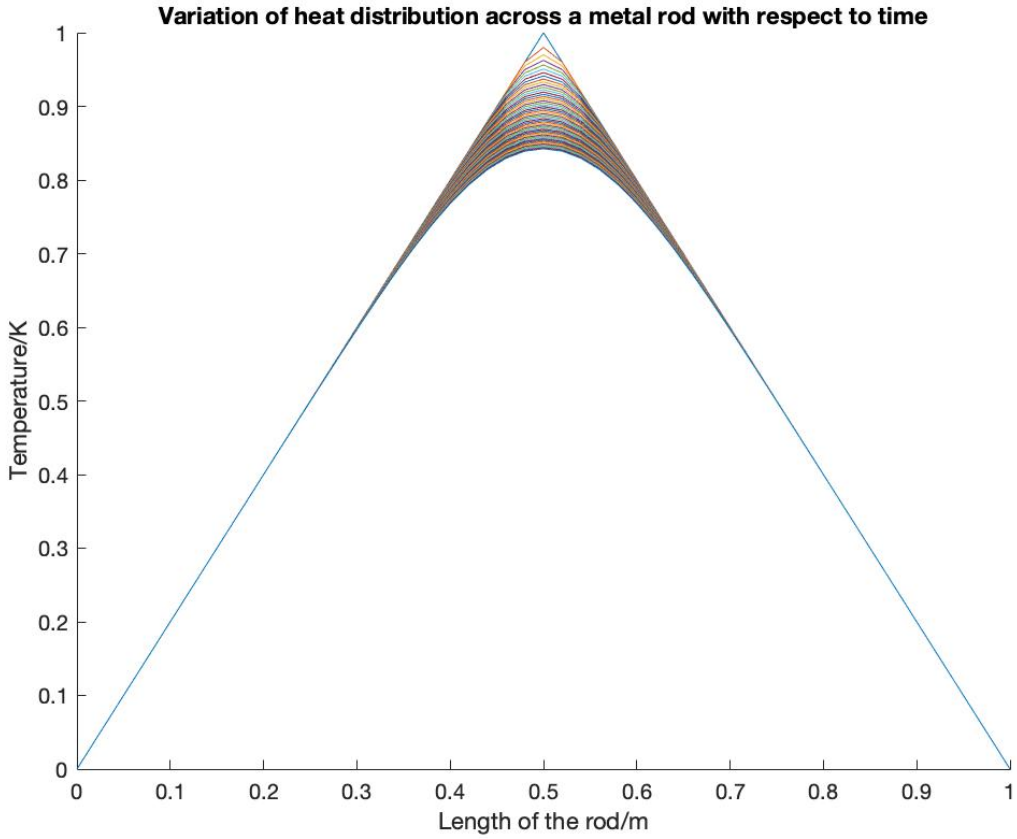

Fig91: *Plot for* $y_0(x) = f(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2 - 2x, & x \in [0.5,1] \end{cases}$ *for zero-boundary conditions for* $m = 50$

The linear approximation is very accurate and therefore, no significant deviation is observed. The energy for the zero-boundary condition can be calculated using $E = \int_0^1 f(x)^2 dx$ which is equivalent to finding the area inside the tent function. Thus, the net temperature can be calculated to be 0.5 K.

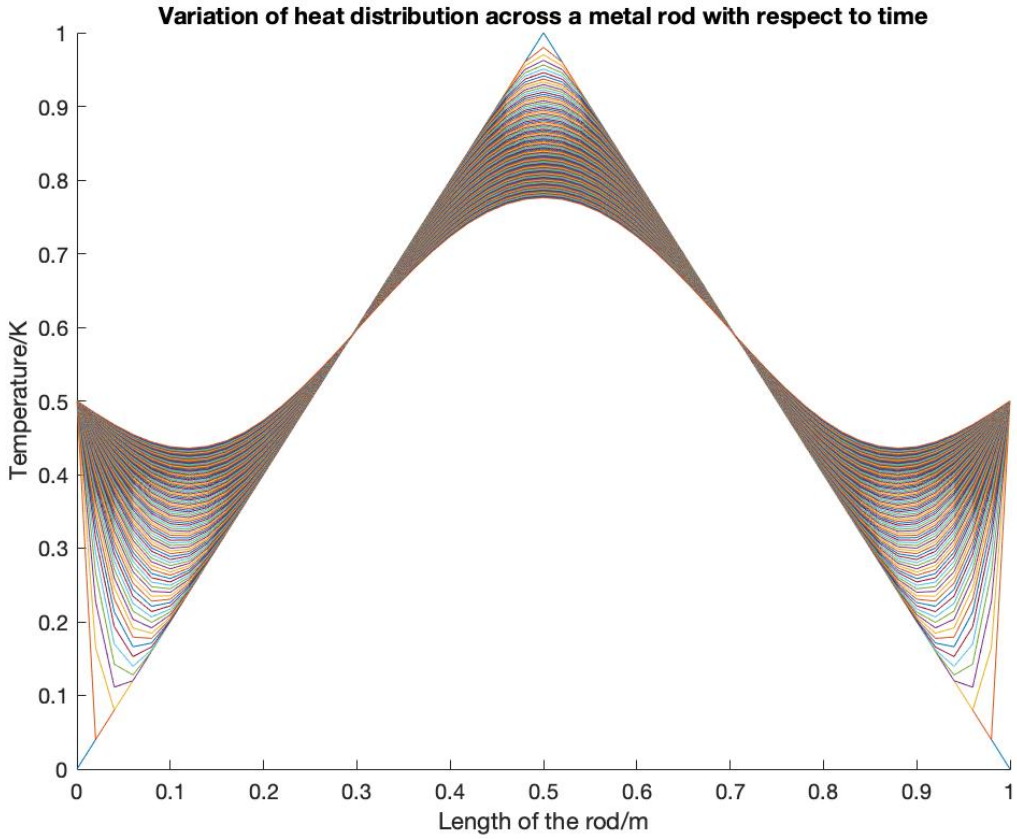

Fig92: *Plot for* $y_0(x) = f(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2 - 2x, & x \in [0.5,1] \end{cases}$ *for constant (0.5) non-zero boundary conditions for m = 100*

All lines outside the tent clearly converge at (0, 0.5) and (1, 0.5), as is expected by fixing a boundary at 0.5. For this boundary condition, the tent function itself has a net temperature of 0.5 K, and the boundary supplies a temperature of 0.5 K. Therefore, it can be hypothesized that the net temperature will fall within $0.5\ K < T < 1\ K$.

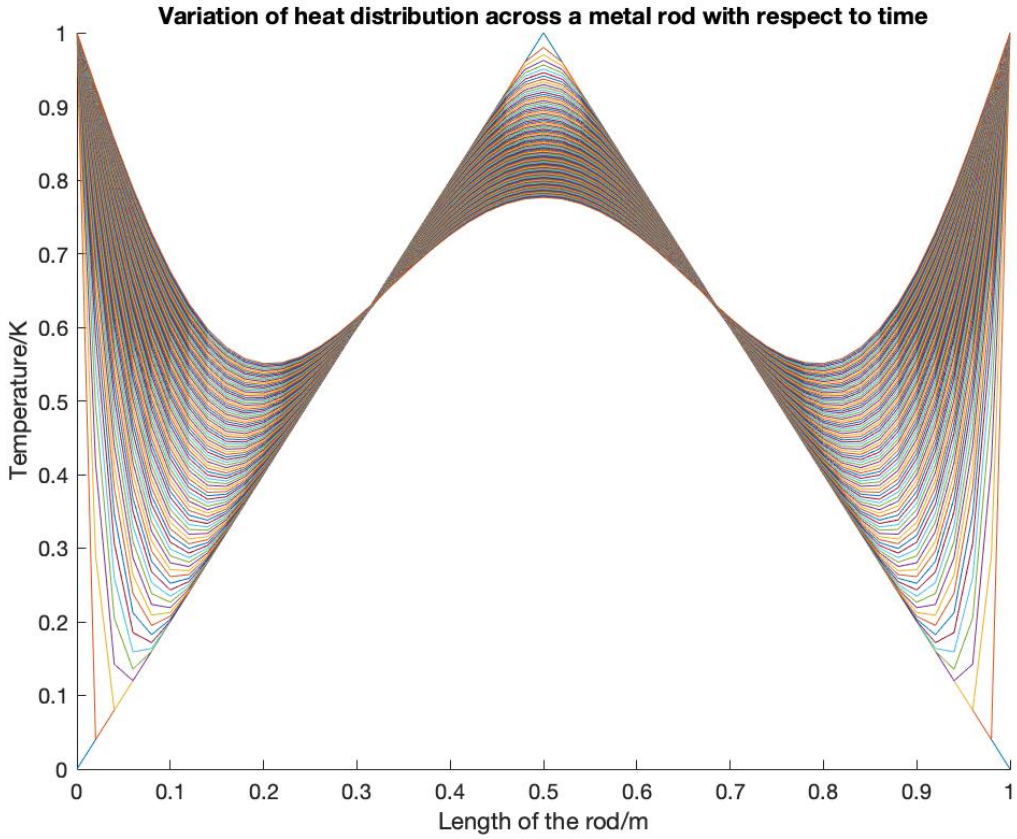

Fig93: *Plot for* $y_0(x) = f(x) = \begin{cases} 2x, & x \in [0,0.5] \\ 2-2x, & x \in [0.5,1] \end{cases}$ *for time varying boundary conditions for m = 100*

Using similar reasoning to the plots above, the net temperature for the time-varying boundary condition can also be discussed. As mentioned before in section 3.2.1, the net temperature in this case is the function of time as the value of m clearly affects the concentration of lines outside the tent function converging to 1.

Thus, for larger values of m, the net temperature across the metal rod increases. Net temperature must be greater than 0.5 K as that is the temperature of the tent function independent of the boundary conditions imposed. Finally, the temperature will fall between $0.5\ K < T < 1.5\ K$.

3.2.5: Plots for $y_0(x) = \cos\frac{\pi x}{2}$ for varying boundary conditions

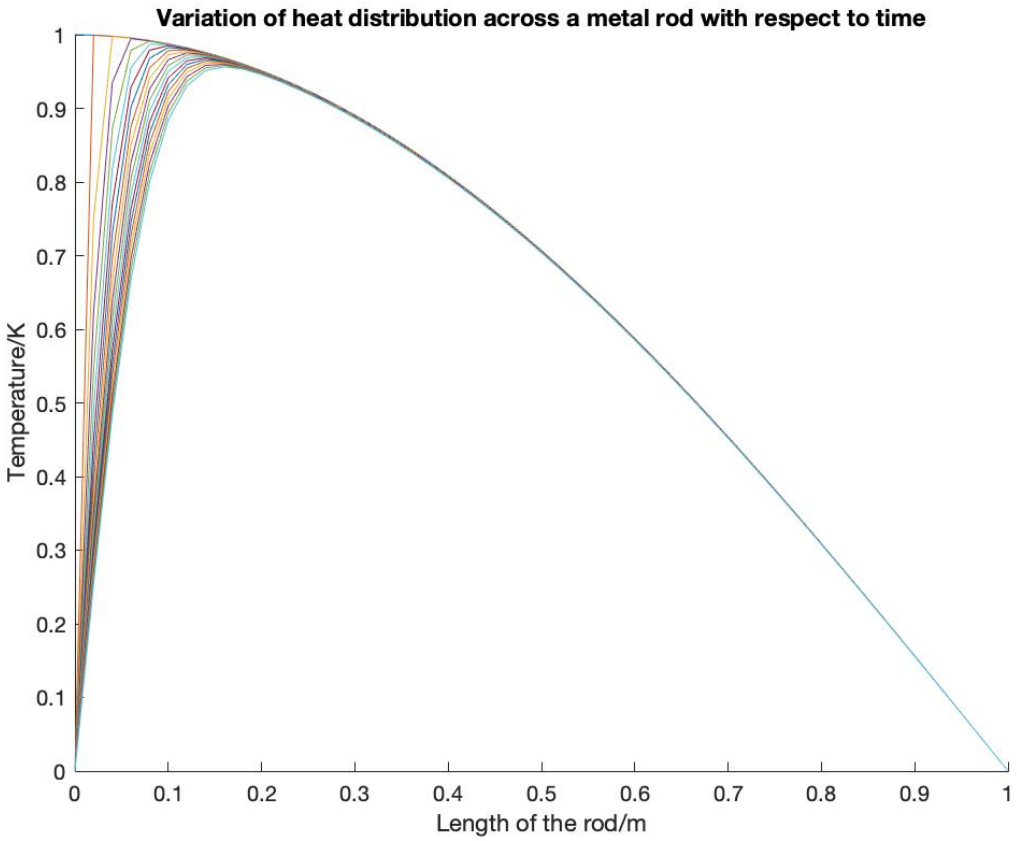

Fig94: *Plot for* $y_0(x) = \cos\frac{\pi x}{2}$ *for zero-boundary conditions for m = 20*

The curve tends to zero as the x value approaches zero from the right-hand side, enforcing the zero-boundary condition.

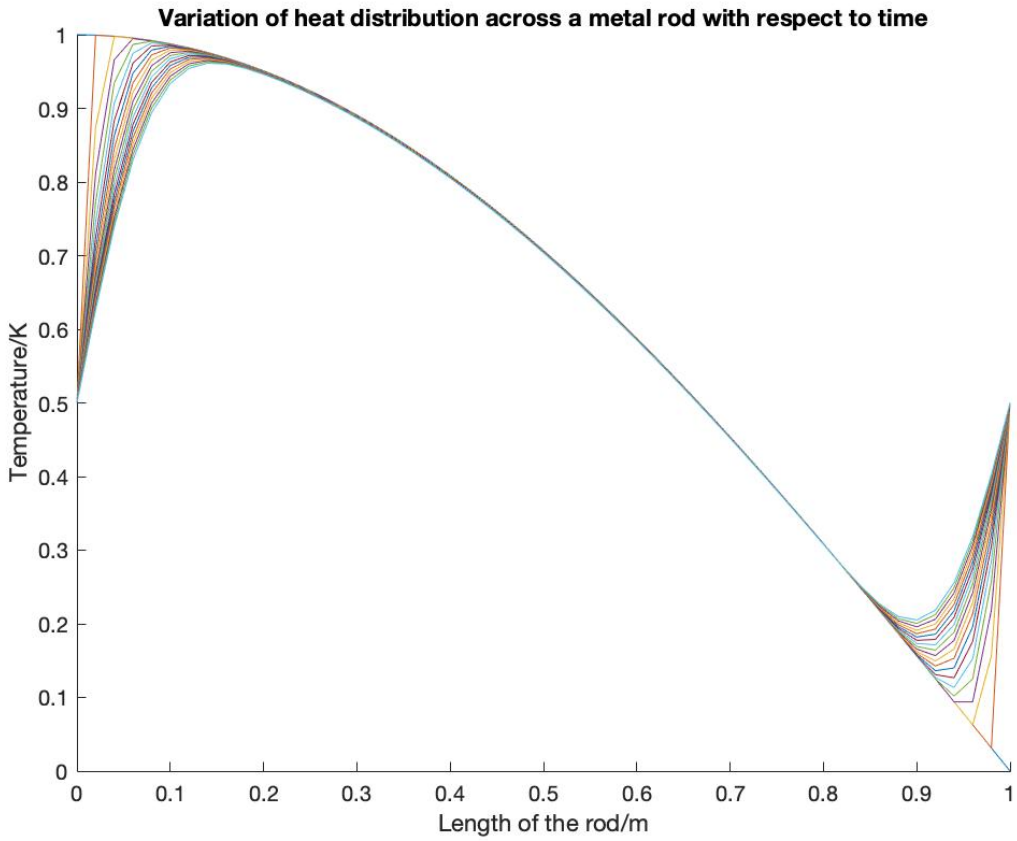

Fig95: *Plot for* $y_0(x) = \cos\frac{\pi x}{2}$ *for constant (0.5) non-zero boundary conditions for m = 20*

The lines are tending towards the points (0, 0.5) and (1, 0.5) from the exact, blue line. Since 0.5 is below the curve at x=0, the curves tend downwards. At x = 1, the curve is below a value of 0.5, thus the lines shoot upwards.

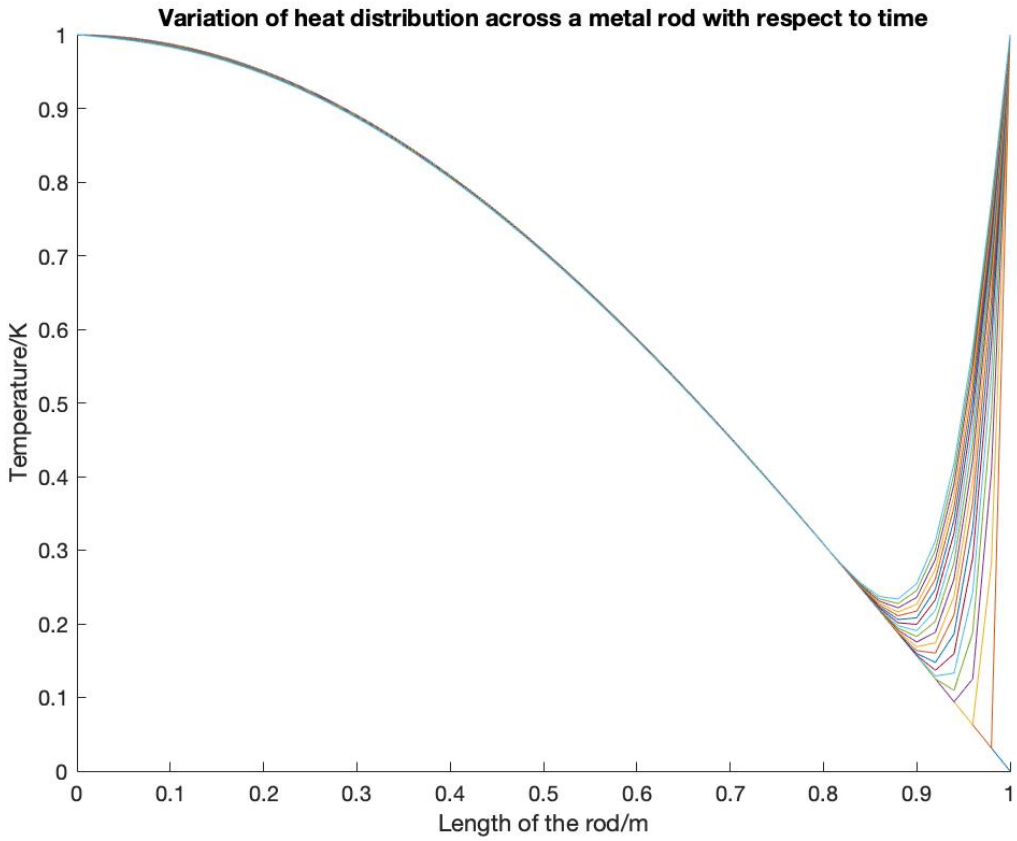

Fig96: *Plot for* $y_0(x) = \cos\frac{\pi x}{2}$ *for times varying boundary conditions for m = 20*

As m increases from 1 to 20, and x approaches 1 from the left-hand side, the approximation curves rise exponentially with different gradients until the point (1,1).

3.2.6: Plots for $y_0(x) = (x - 0.5)^2$ for varying boundary conditions

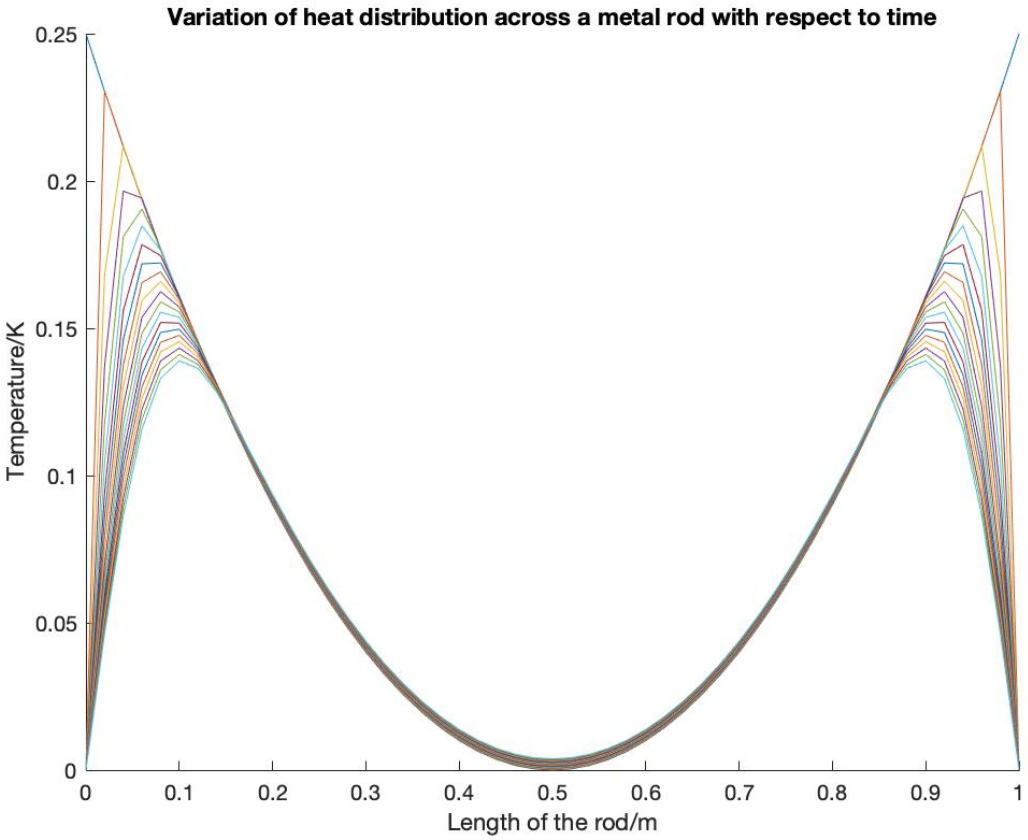

Fig97: *Plot for* $y_0(x) = (x - 0.5)^2$ *for zero-boundary conditions for m = 20*

Zero boundaries at (0, 0) and (1, 0) are clear. Since the curve is above the origin at both ends, approximation lines tend downwards towards the boundaries. Using the following formula - $E = \int_0^1 (x - 0.5)^4 dx$ - the energy of the parabolic function within the domain [0, 1] has a value of 0.0125. Given the zero boundary conditions, the net temperature of this heat distribution is roughly 0.0125 K.

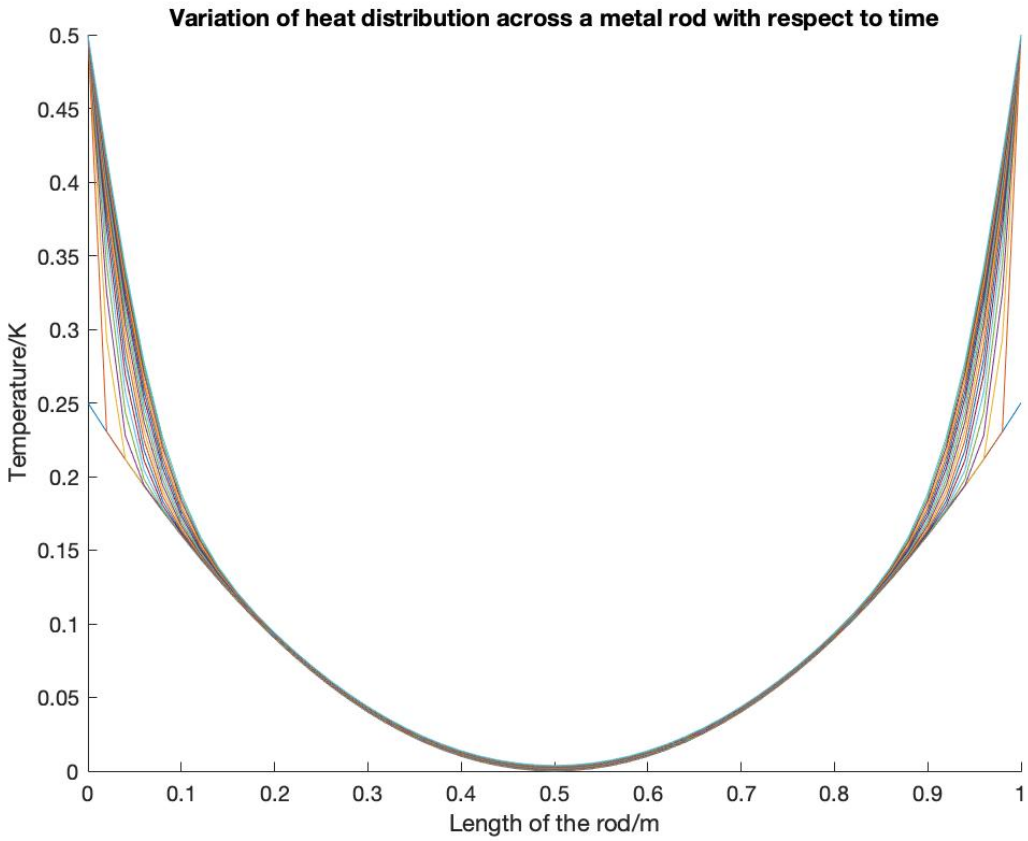

Fig98: *Plot for* $y_0(x) = (x - 0.5)^2$ *for constant (0.5) non-zero boundary conditions for m = 20*

Boundaries at (0, 0.5) and (1, 0.5) for m > 1 can be seen. Using the energy calculated for the zero-boundary condition and extrapolating it for the 0.5 boundary condition case, the net temperature (T) is: $0.0125\ \text{K} < \text{T} < 0.5125\ \text{K}$.

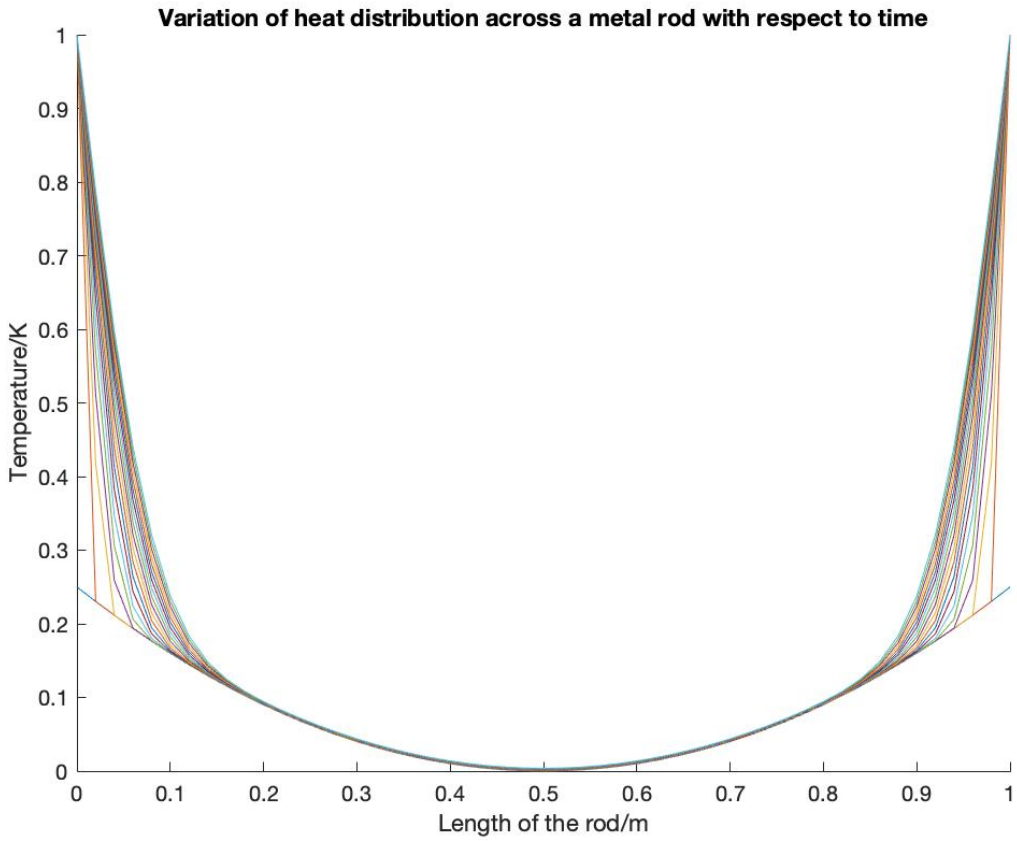

Fig99: *Plot for* $y_0(x) = (x - 0.5)^2$ *for time varying boundary conditions for m = 20*

Boundaries at (0, 1) and (1, 1) for m > 1 are observed. As shown before, the net temperature for this particular heat distribution appears to be a function of time due to the time varying nature of the boundary conditions. It can be roughly stated that temperature is given by the following range $0.0125\ \text{K} < \text{T} < 1.0125\ \text{K}$ for this case.

3.2.6: Plots for $y_0(x) = e^{x+1} - 5$ for varying boundary conditions

Repeating the plots for an exponential function to observe the behavior at boundaries.

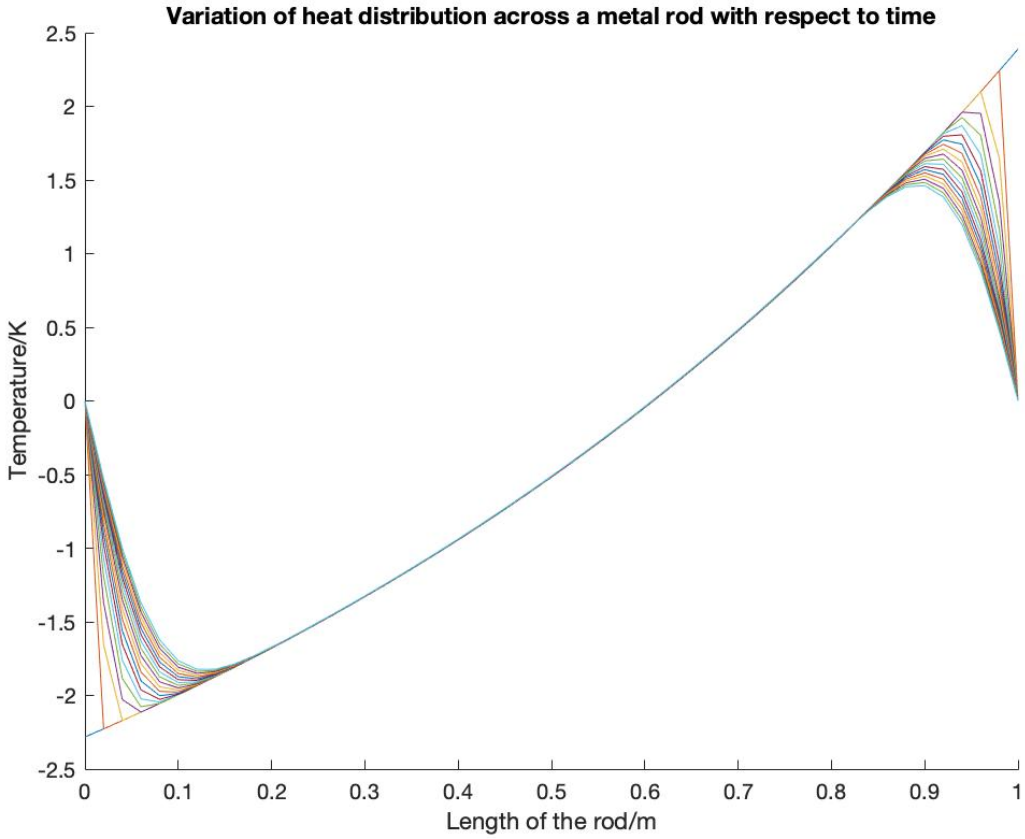

Fig100: *Plot for* $y_0(x) = e^{x+1} - 5$ *for zero-boundary conditions for m = 20*

The curve is below 0.5 at x = 0, and above it at x = 1; thus for $1 < m \leq 20$, the lines tend upwards and downwards respectively, towards the boundary points at (0, 0.5) and (1, 0.5). For m = 1, the exact plot of $y_0(x)$ has been plotted. The energy for this function can be calculated by:

$$E = \int_0^1 (e^{x+1} - 5)^2 dx = 1.897.$$

Thus, the net temperature of this heat distribution should be around 1.9 K.

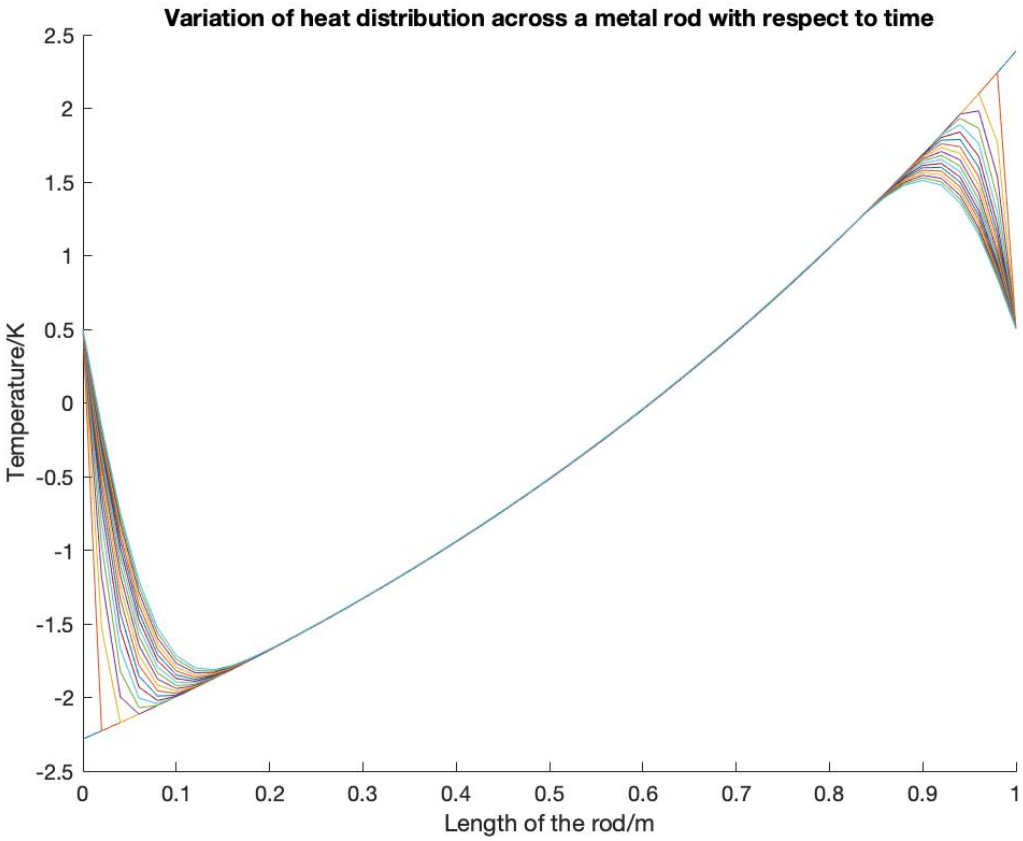

Fig101: *Plot for* $y_0(x) = e^{x+1} - 5$ *for constant (0.5) non-zero boundary conditions for m = 20*

Following the energy calculated for the zero-boundary condition, the net temperature in this heat distribution case should fall roughly between $1.9\ \text{K} < \text{T} < 2.4\ \text{K}$.

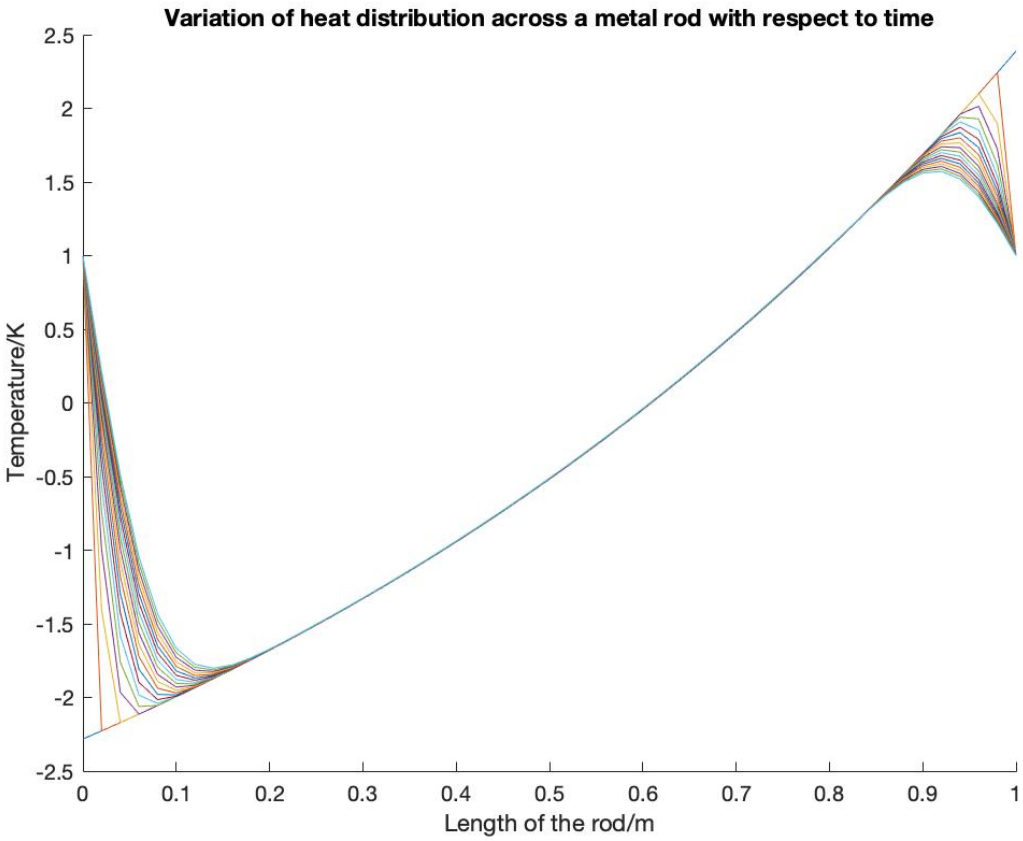

Fig102: *Plot for* $y_0(x) = e^{x+1} - 5$ *for time varying boundary conditions for m = 20*

In accordance with results discussed previously, the net temperature of this heat distribution can be taken to be a function of time which is likely to fall in the range $1.9\ \text{K} < \text{T} < 2.9\ \text{K}$. For larger values of m, the concentration of lines converging at the boundary points increases, thus corresponding to an increase in net temperature across the metal rod.

All zero-boundary condition plots converged at (x, 0) such that x is a point on the x-axis. Usually x = 0 and x = 1, but for sinusoidal plots, x = 0.5 was also a valid solution. All constant non-zero boundary condition plots had lines (corresponding to different values of m) shooting upwards or downwards, depending on the position of the curve relative to the line y = 0.5, to converge at the boundary points occurring at (0, 0.5) and (1, 0.5). The time-varying boundary condition plots can also be explained similarly, however since the code involved a cosine term that set this boundary, the boundary points themselves occurred at value 1 - (0, 1) and (1, 1) - corresponding to the magnitude of the cosine term.

3.3: Using Taylor Series to derive the finite difference approximations and obtaining the order of the error incurred (Exercise 6)

Using the Taylor Series it is known that:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}}{3!}(x-a)^3 + \cdots \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$

Then the Taylor Series expansions for $f(x \pm h)$ can be found:

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2!}f''(x) \pm \frac{h^3}{3!}f^{(3)}(x) + \cdots$$

Then to find an approximation for the first derivative of a given function, $f(x+h) - f(x-h)$ is performed as shown below:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f^{(3)}(x) + \cdots$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f^{(3)}(x) + \cdots$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3!}f^{(3)}(x) + \cdots$$

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{h^2}{3!}f^{(3)} + \cdots$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3!}f^{(3)} + \cdots$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

As it can be seen, this approximation for the derivative, which is known as the central difference approximation as it is centered around *x*, has an error of second order with respect to *h* (i.e. h^2).

To form an approximation for the second derivative, $f(x+h)+f(x-h)$ can be done and rearrange the resulting equation to obtain $f''(x)$. This is shown below:

$$f(x+h)+f(x-h)=2f(x)+h^2f''(x)+\cdots$$

$$f(x+h)+f(x-h)-2f(x)=h^2f''(x)+\cdots$$

$$f''(x)=\frac{f(x+h)+f(x-h)-2f(x)}{h^2}-\frac{2}{4!}h^2f^{(4)}(x)+\cdots$$

$$f''(x)=\frac{f(x+h)+f(x-h)-2f(x)}{h^2}+O(h^2)$$

As it can be seen, the error of this approximation is of second order with respect to *h* (i.e. h^2).

Works Cited

"Step Response of RLC Circuits." *RLC Step Response*, UNIVERSITY of PENNSYLVANIA, www.seas.upenn.edu/~ese206/RLCresponse/rlcreponse.html.

Runge-Kutta 3/8 Method, Mathematics Source Library C & ASM, www.mymathlib.com/diffeq/runge-kutta/runge_kutta_3_8.html.

Brookes, Mike. "Resonance." *E1.1 Analysis of Circuits*, Imperial College London, www.ee.imperial.ac.uk/hp/staff/dmb/courses/ccts1/ccts1.htm.

Kumar, K. S. Suresh. "Electric Circuits and Networks." *O'Reilly | Safari*, Pearson India, learning.oreilly.com/library/view/electric-circuits-and/9789332503328/xhtml/ch11-sub11.7.xhtml.

Hancock, Matthew J. "The 1-D Heat Equation." *Online Courseware MIT*, MIT, ocw.mit.edu/courses/mathematics/18-303-linear-partial-differential-equations-fall-2006/lecture-notes/heateqni.pdf.

Nucinkis, Daniel. "Numerical Analysis of Differential Equations." Numerical Analysis 10C. Numerical Analysis 10C, London, Imperial College London.