**RL circuit**

In part one we have a high-pass filter, which takes an input signal, Vin, and let only the high-frequency components pass. Figures 1 shows a simple RL circuit. The purpose of this section is to use three different methods (Heun’s, midpoint and my method) to solve the first order ODE equation and calculate the value of the output signal.

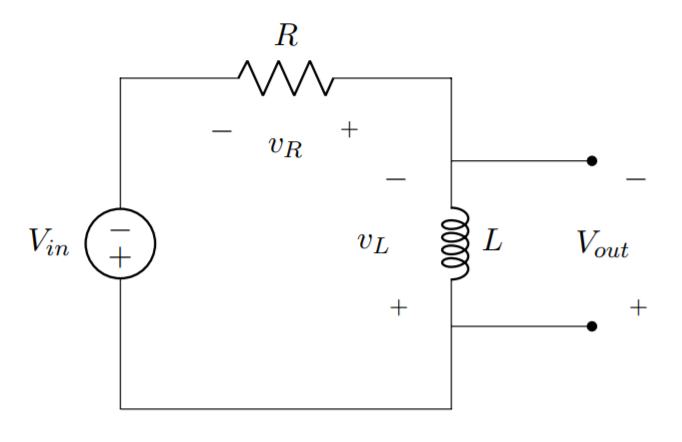


Figure 1. RL circuit

For this circuit, the ODE equation can be written:

where iL(t) is the current and Vin(t) is the input signal.

The output is calculated by:

With corresponding values for the inductance and resistance are:

*R = 0.5Ω*

*L = 1.5mH*

We assume the initial current through the inductor at time *t=0* is *iL(0)=0A*.

Methods:

Euler’s method is a classic Runge-Kutta method, it stands as a base for many methods to follow within this document. It works by dividing a section of a curve into *N* sections, each section with width of *h (in the x direction).* it uses the gradient at the smallest x value of the section, , to approximate the curve to a straight line from to , or .

For each section the straight line therefore has the following equation:

Which can be rewritten as

Where is the derivative of the curve at point .

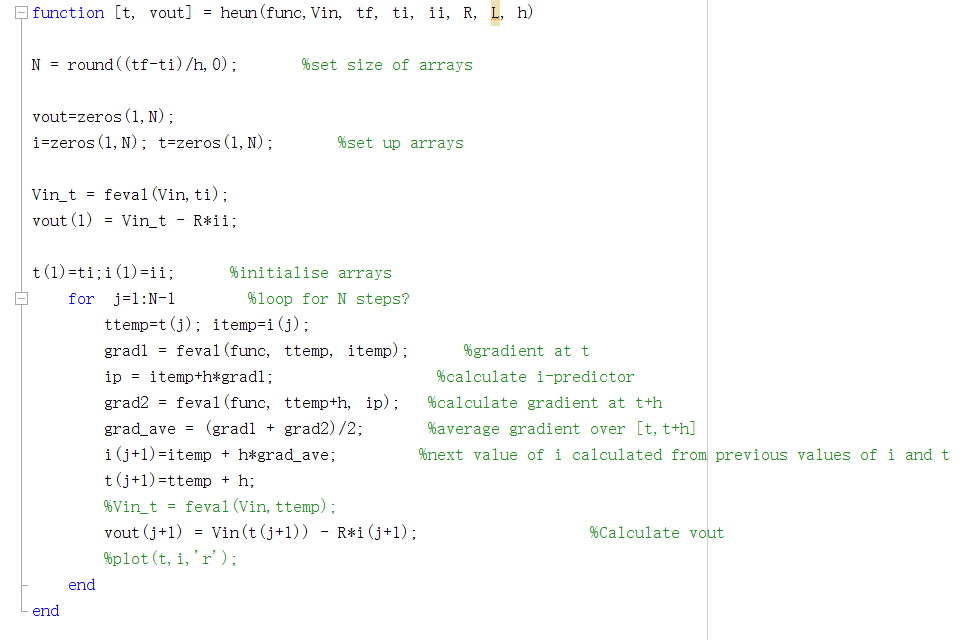
However, since the concavity of the curve cannot be predicted and remains consistent, Euler’s method may give an overestimate or an underestimate at different points. Thus, the following methods are introduced to help solve this problem.

1. *Heun’s Method*

This method is an improvement on Euler’s method. Heun’s method considers the interval spanned by the tangent line segment as a whole. Both the left end-point where the left tangent line is formed, and the right end-point ( are considered. Therefore, the right tangent line is the slope of the tangent line passing through the right end-point. To get the desired end-point, interpret the ‘ideal prediction line’ by calculating the average of the slopes of the left and right tangent lines. This way we can tackle the problem of over/underestimating.

The following equations implement Heun’s method:

Where:



The Heun’s “function” takes a number of arguments as input parameters; the ODE function, the input signal Vin, the initial time ti, the initial current ii, the final time tf, the step size h, the resistance R and the inductance L.

The value of N indicates the size of the arrays, which are initialised with zeros and later will be used to stored values once evaluated by the “function”. Vin\_t is a variable which evaluated at ti and holds the initial value of the Vin. This is used to calculate the value of the first element in the vout array.

The function then enters a loop with the iterator j, starting at 1 and counting up to N-1. At this stage, two temporary variables are created in order to store the current value that has been evaluated.

The Heun’s method is then implemented according to the equations above. The function uses the values at the current time t to predict the values at t+1, therefore when the loop finishes, the current array stores the values from 0s to the final time.

The last step in the for loop is to evaluate the vout array at t+1 by subtracting the production of resistance and the current at t+1 from the input signal value, which is evaluated at t+1.

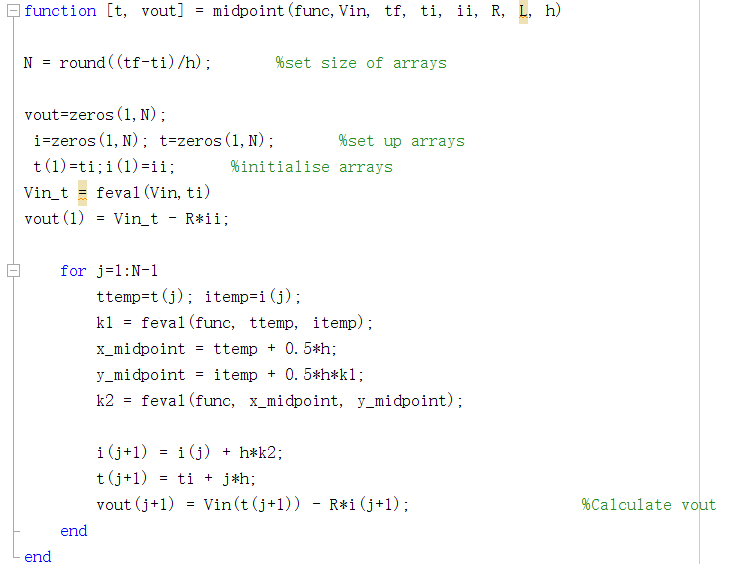
1. *Midpoint Method*

The midpoint method can be thought of as an extension to Euler’s method like other methods used in this document. Just as Euler’s method uses *N* different sections to approximate the target equation from a to b (where a<b), again . However, where Euler’s method uses the gradient of the smallest *x* value () of each section for the approximation, midpoint uses the gradient of the interval at midpoint, thus the constant changes to . Giving the following general iterative formula:

Which we can rewrite as

Where,

The following code shows how this was achieved within MATLAB. Commented out is the code used for Euler’s method to show the similarities between the two methods. The only differences are the updated value of to find the gradient at the midpoint.



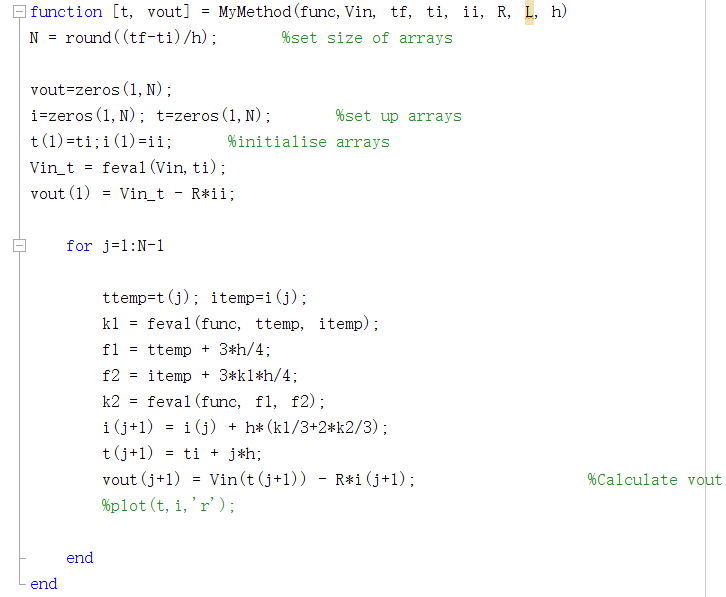
1. *My method*

In this method, a constant of is chosen, which means that it takes the gradient at of the interval.

The following equations implement my method:

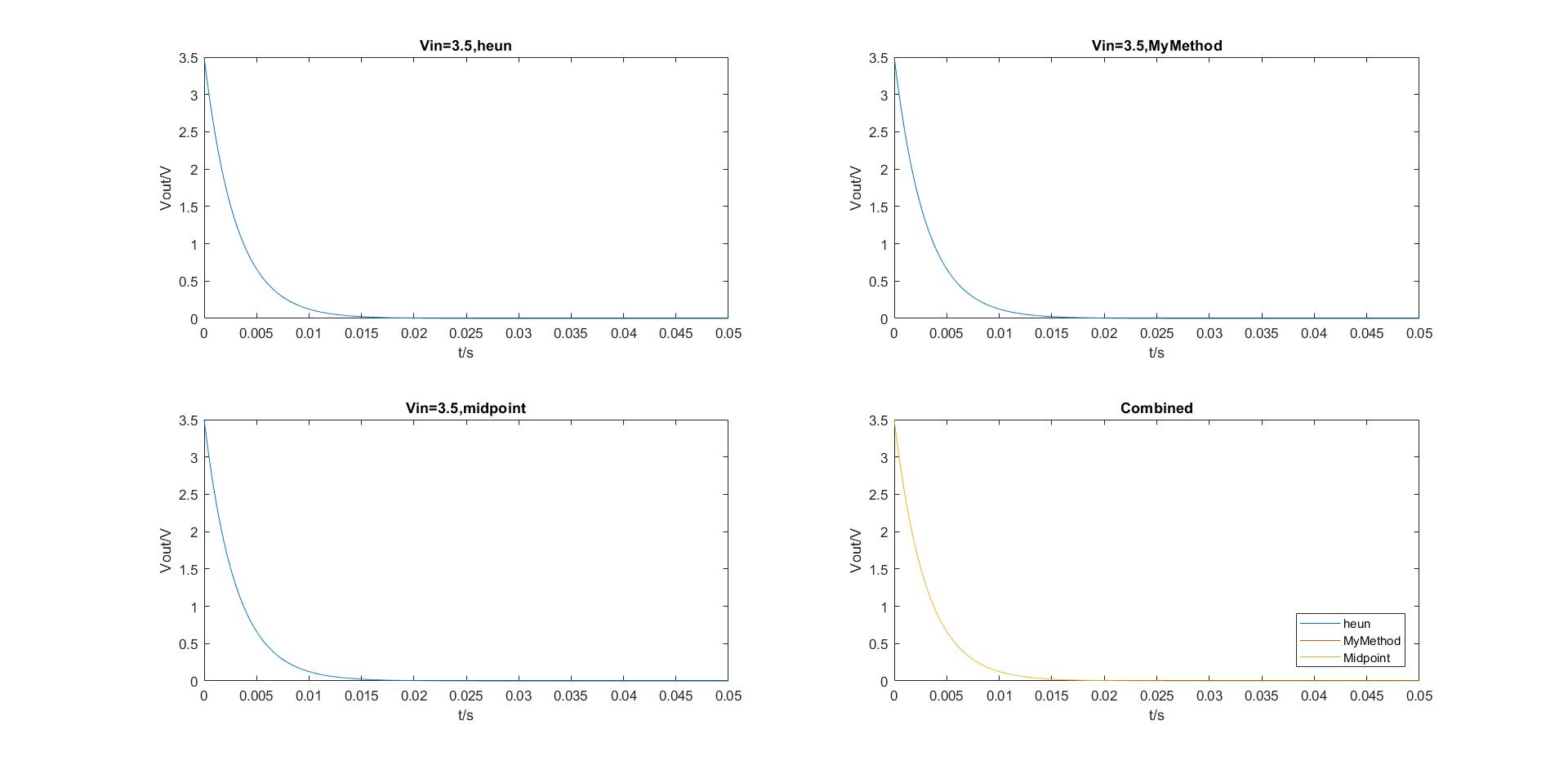
where,

For implementing this method, the algorithm is very similar to the previous one as the three methods are differentiated by its constant.



Inputs:

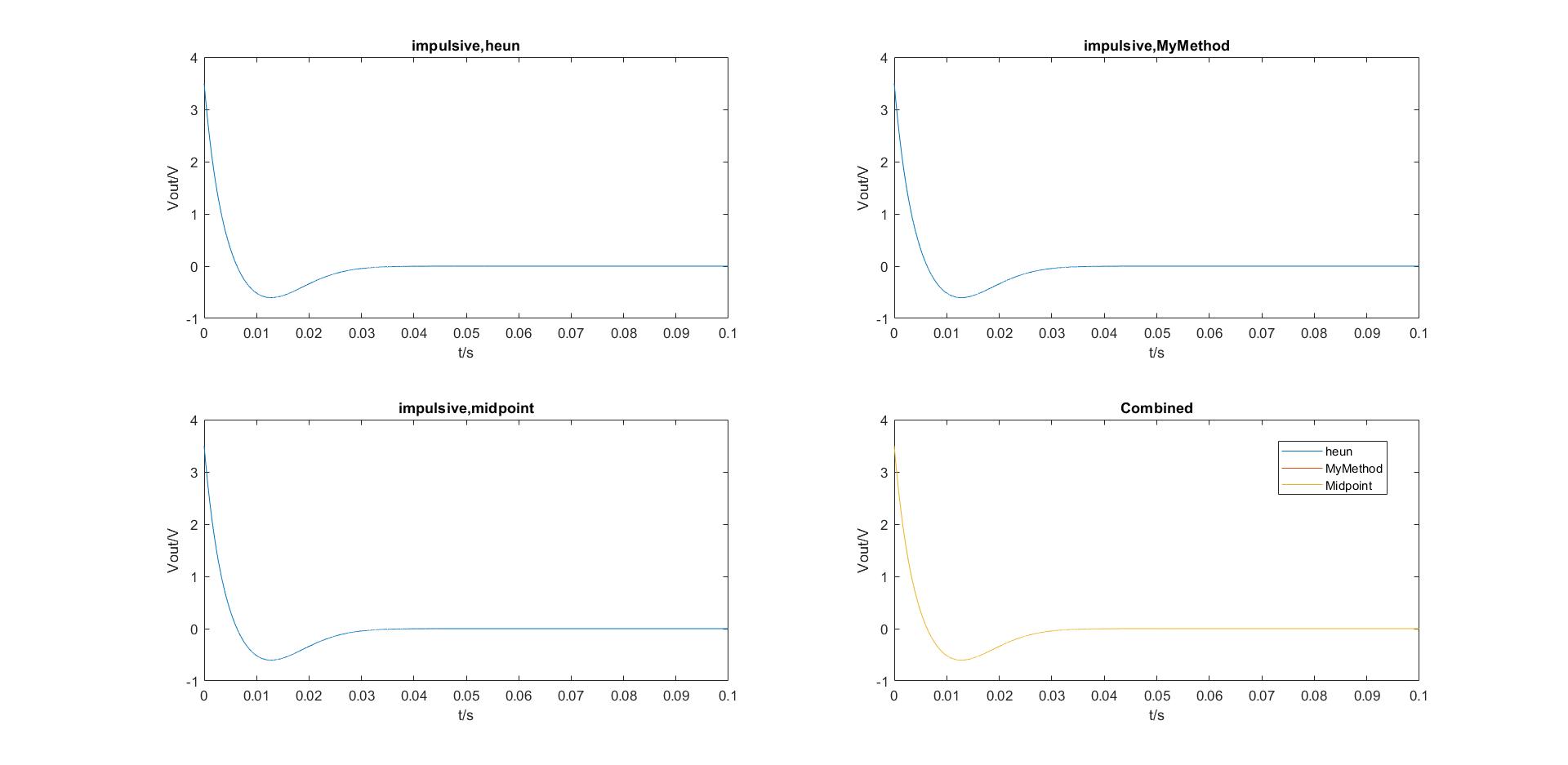
1. Step signal with amplitude Vin = 3.5V

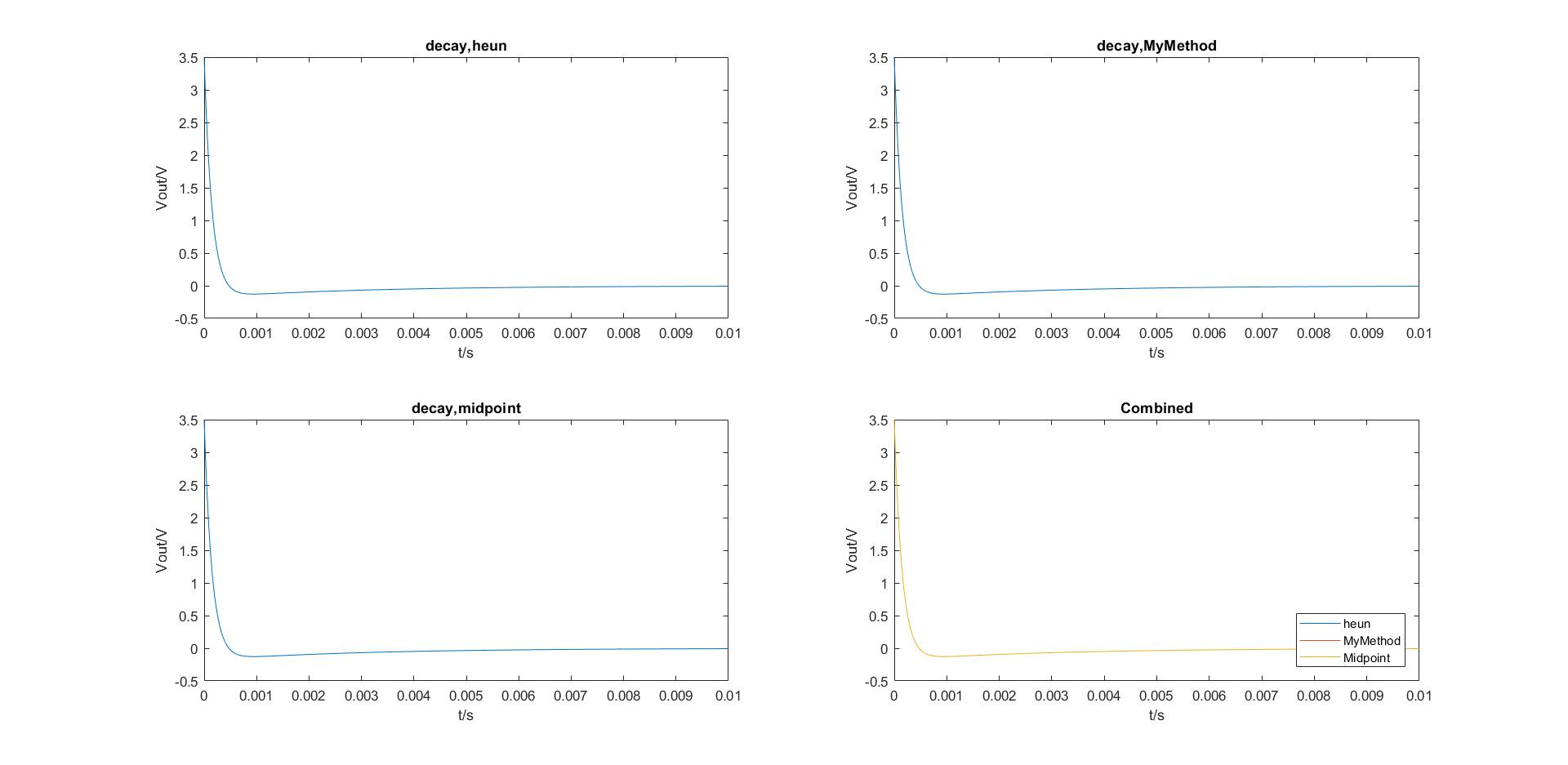
With the step signal as an input, we observed how the output signal decays exponentially. The current flowing through the inductor gradually increases from 0A to 7A, this is because the current through an inductor cannot change instantaneously. However, the voltage across an inductor can. Therefore the initial gradient of output signal is the greatest also where the current is the highest. The following graph shows the behaviour of the output voltage using three different methods to interpret the result. The comparison of all methods is shown in the fourth quadrant.

As the graph illustrates, the voltage does not drop down to zero immediately and the amplitude of the output signal depends on the initial value of input voltage. Furthermore, it can be seen from the graph that all methods obtain similar values and no significant errors are introduced.

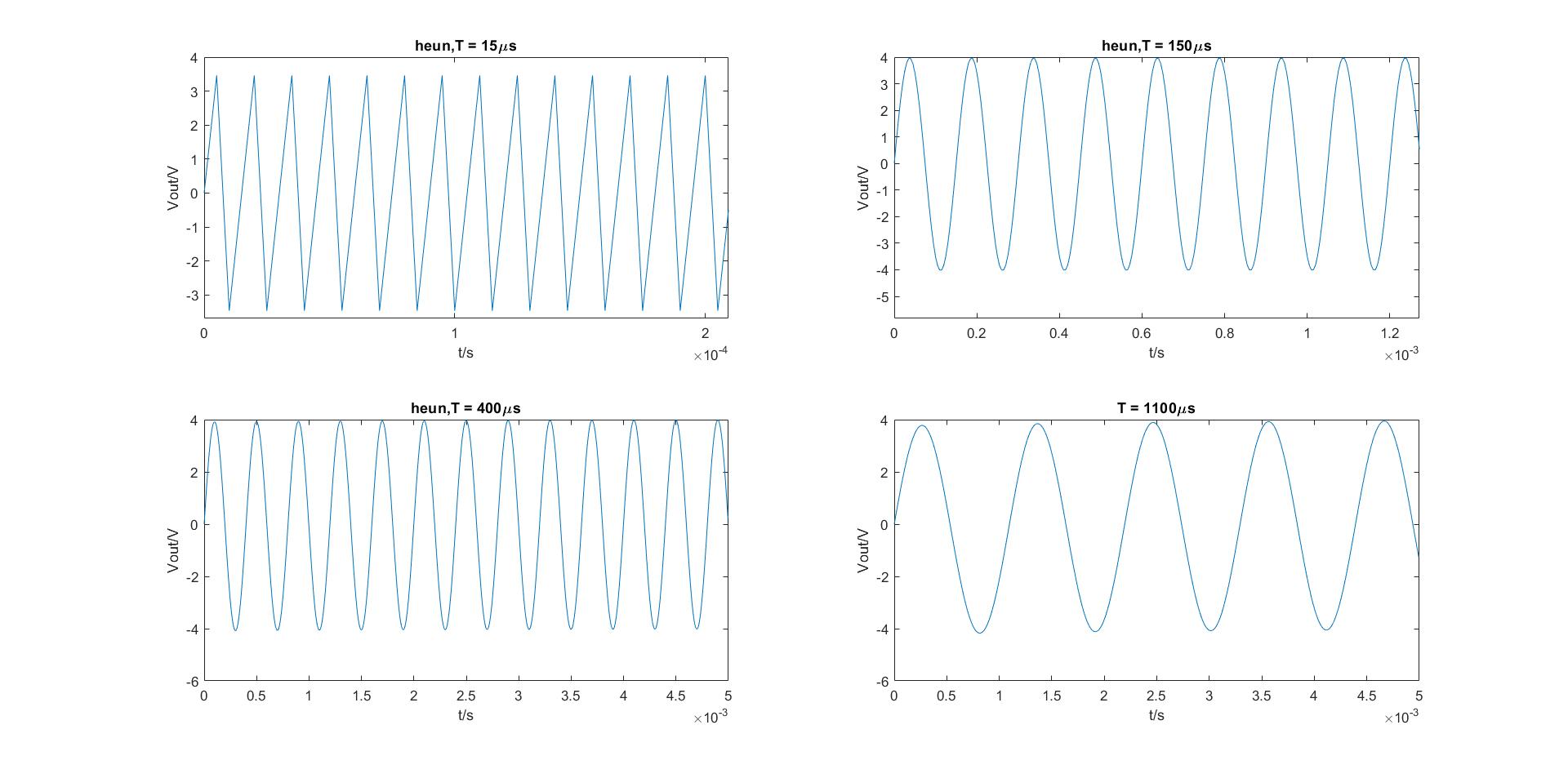
1. Impulsive signal and decay

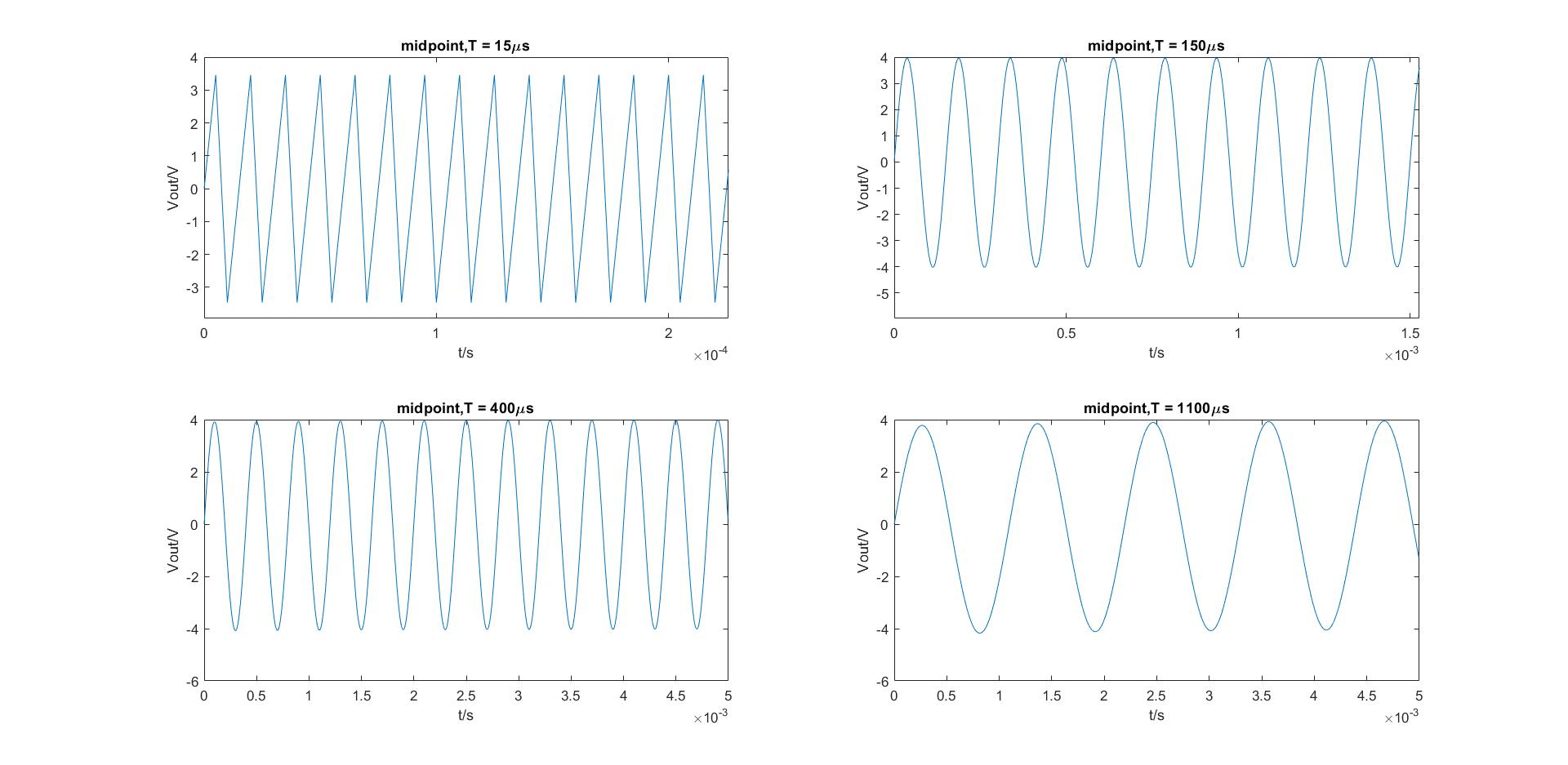
With and τ=150(μs)2 resp.

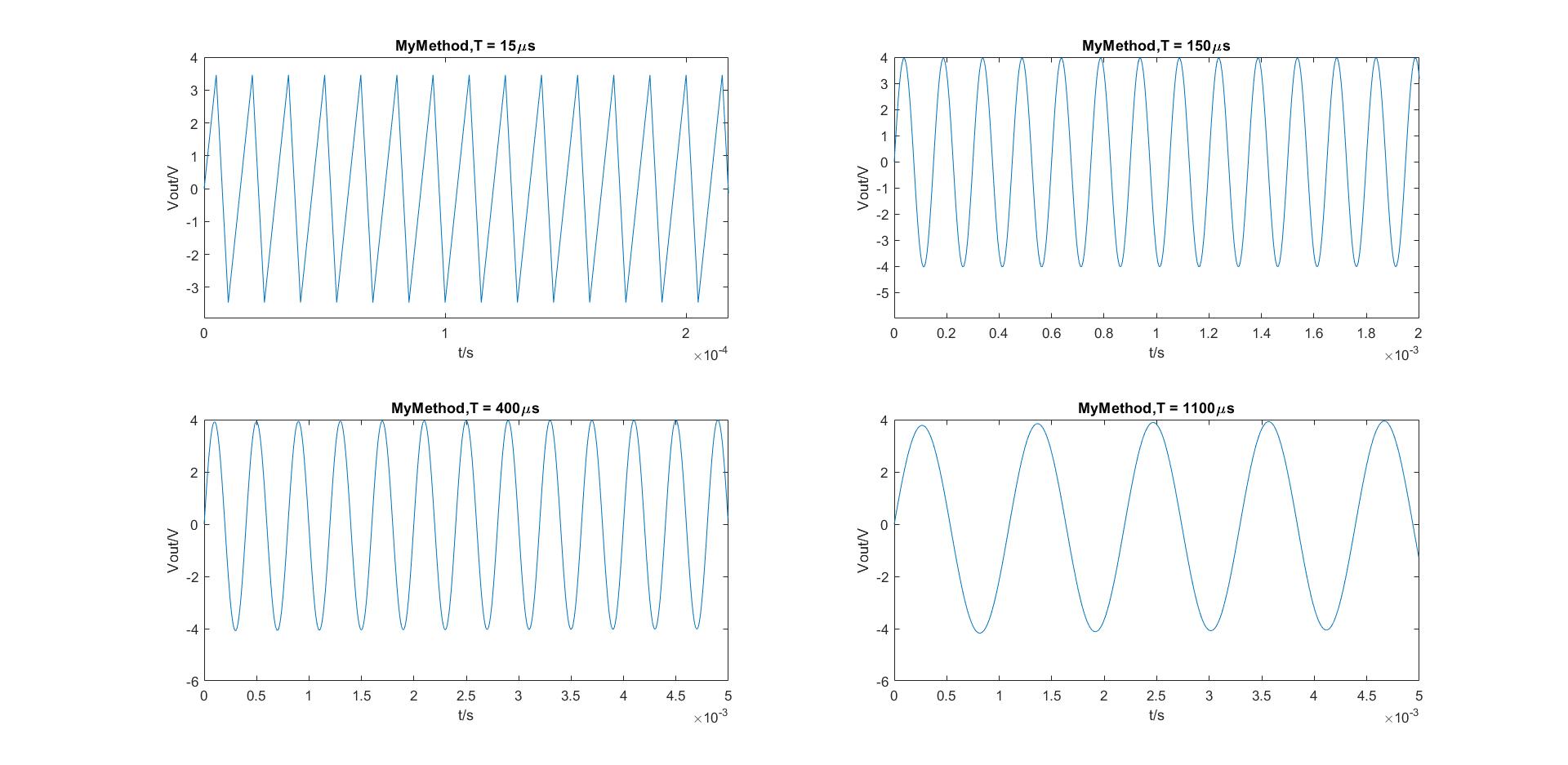
The figure shows the impulsive signal, the output voltage decays exponentially with initial amplitude of 3.5. However, during the time period, 0.00614s to 0.02957s, the output signal forms a local minimum with negative amplitude, then gradually increases back to zero.

Comparing the impulsive to the decay signal, shows the following: the decay curve does not curve down to negative values as much as the impulsive signal. Moreover, the decay signal stays steady at around 0.003s, whereas it takes the impulsive signal longer to decay to zero. This means that the decay signal has larger decay rate than the impulsive signal.

1. Sine waves with amplitude with periods: T = 15 µs, T = 150µs, T = 400µs, T = 1100µs

Heun’s method

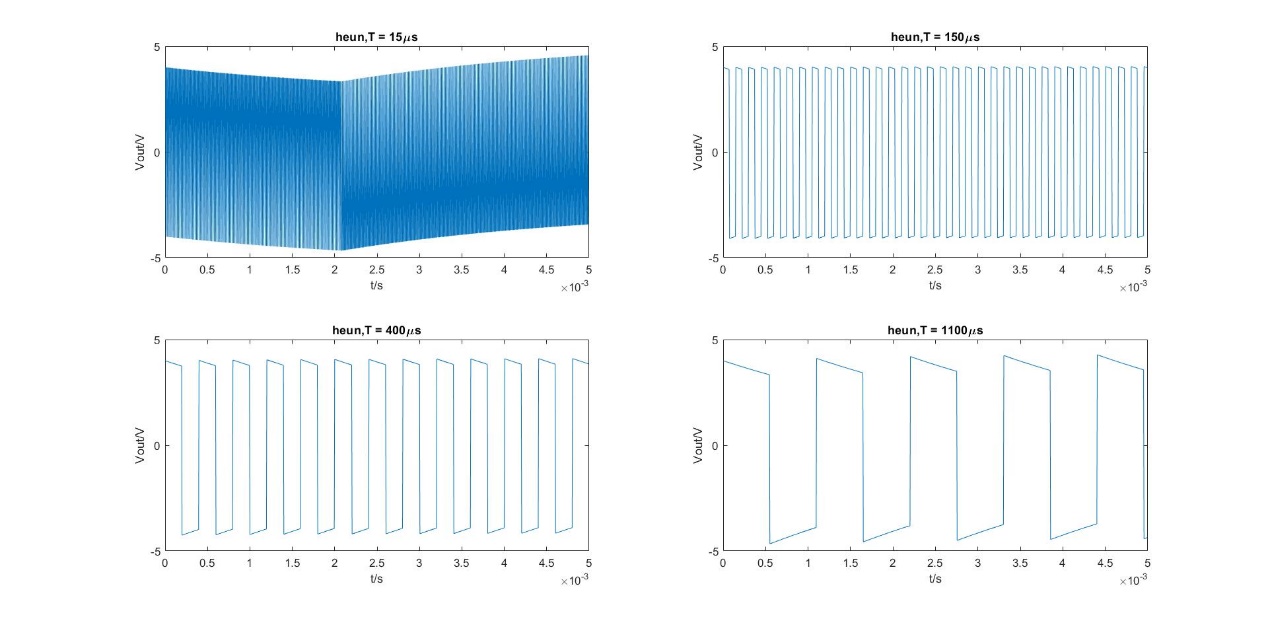
Midpoint method

My method

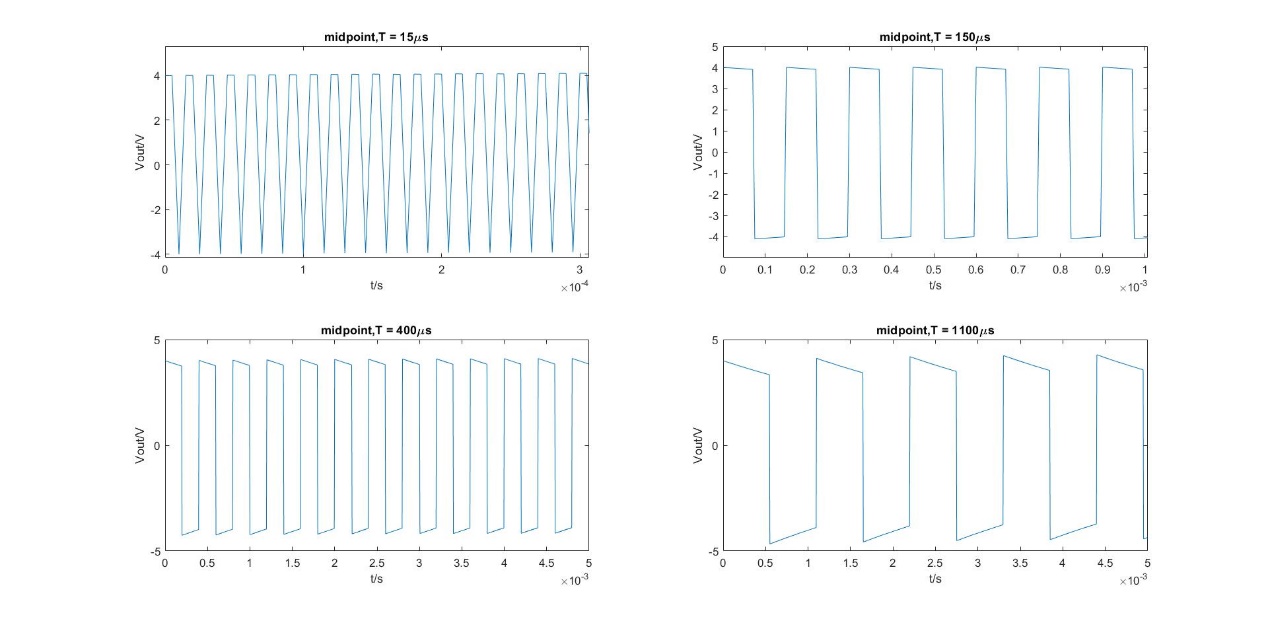
We observed that for all three methods, as the time period decreases to 15µs, the produced output signal is distorted to a triangle wave instead of a sine wave. During the testing, the step size h is fixed to 0.000005 and it is slightly smaller than the time period of 15µs. As the step size h must be small in comparison to the time period. The behaviour when T=15µs is worse than different time periods. Thus, the larger the time period, the better the accuracy with a fixed step size.

3. Square waves with amplitude with periods: T = 15 µs, T = 150µs, T = 400µs, T = 1100µs

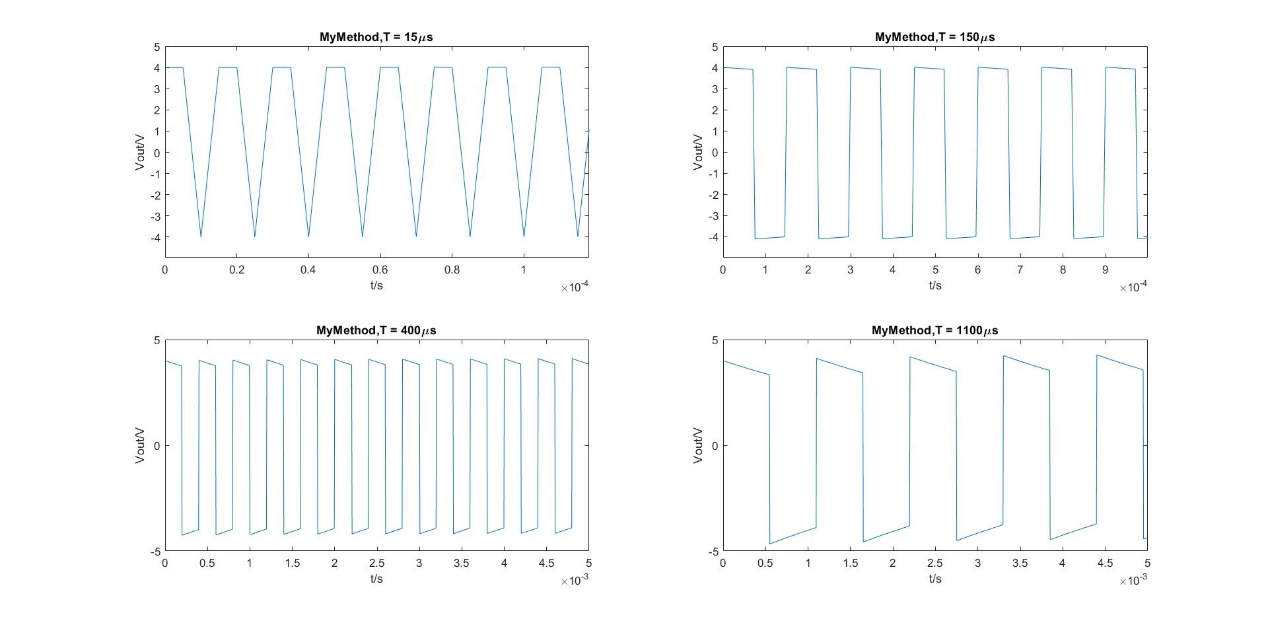
Heun’s method



Midpoint method



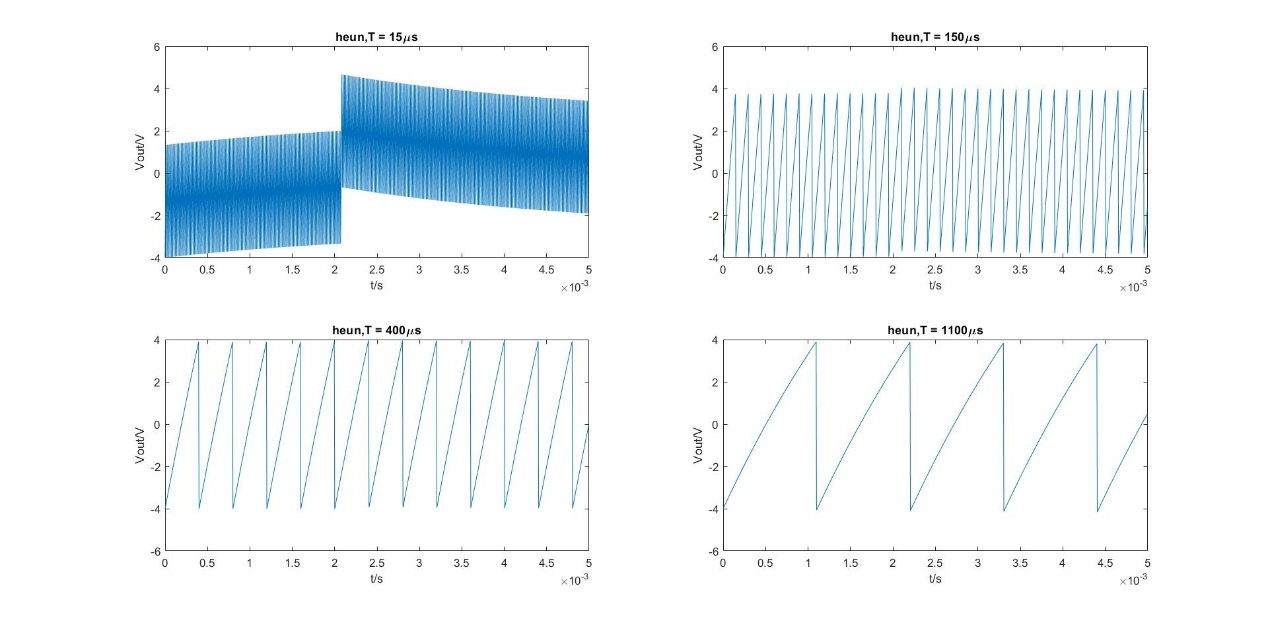
My method



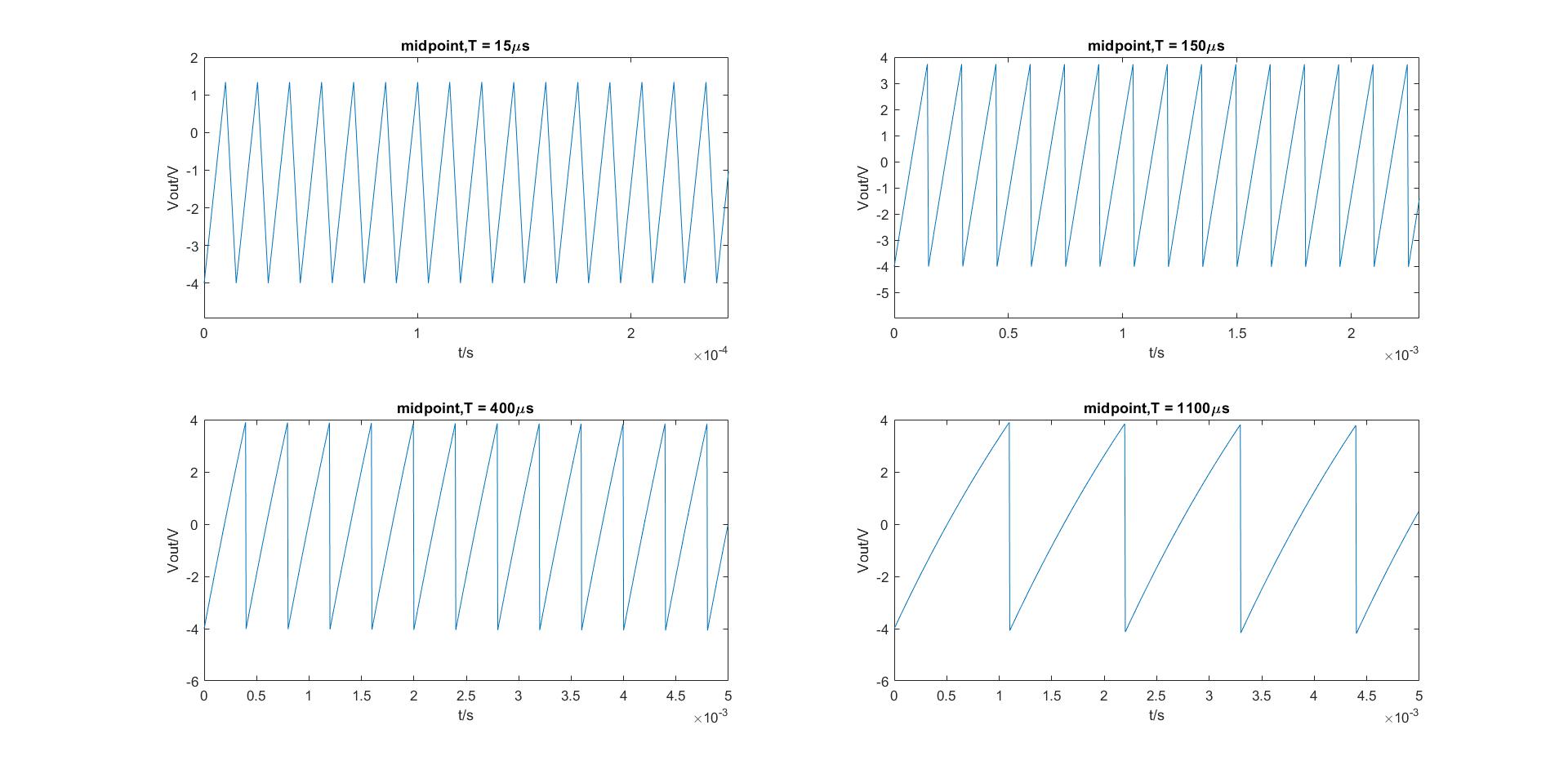
It can be seen from the graph that Heun’s method gives a distorted output wave at T=15 µs. This is because Heun’s method is the least accurate method and whys this is the case will be discussed in detail later. The step size h has been kept consistent for the same reasons as before. The output is distorted when the time period is shorter than 150 µs.

3. Sawtooth waves with amplitude with periods: T = 15 µs, T = 150µs, T = 400µs, T = 1100µs

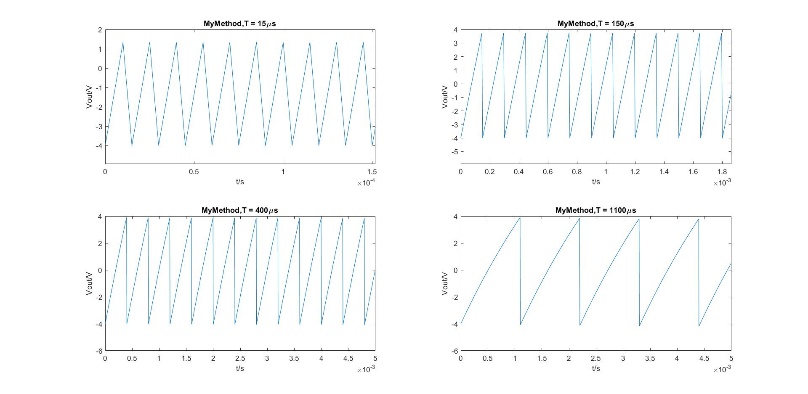
Heun’s method



Midpoint method



My method



All of the methods give a reasonable response at periods 400µs and longer. Any periods at 15µs produce distorted outputs. Since Heun’s method is the least accurate method overall, the produced output signal has a worse distortion at time period of 15µs.

**Open-ended**

To further investigate the behaviour of RL circuit, a number of input parameters have been changed throughout the test.

Firstly, we increase the value of resistance to 500Ω. This results a decreasing in the time constant, since time constant can be calculated using:

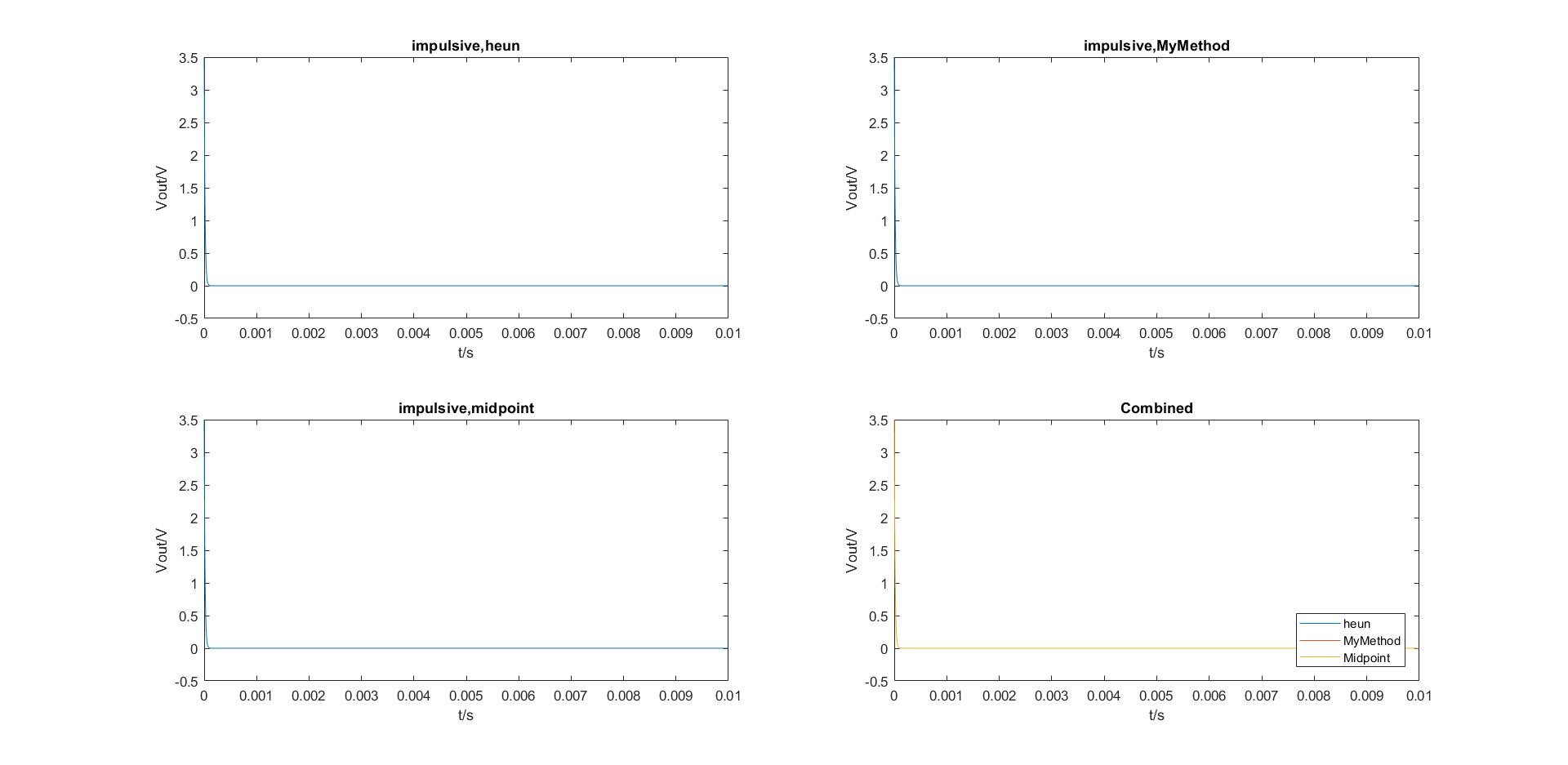


Figure x

The input signal is an impulsive wave and the value of inductance keeps at 1.5mH. As figure x shows, the rate of decay increases significantly and the output reaches 0V within a shorter period of time.

The time constant of the RL circuit represents the time taken by the current to build up to 63.6% of the steady state value. The time required for the current to reach its maximum steady state is equivalent to 5. This is because that for a fixed inductance, when the resistance increases to a very large value, the inductance becomes negligible when compared with the resistance. Thus, the shorter the time constant is, the faster the output will reach its steady state.

Secondly, the effect of increasing the time period has been investigated. We have changed the input signal to a sine wave, with T=0.5s and the amplitude of , which is equivalent to a sine wave with frequency of 2Hz.

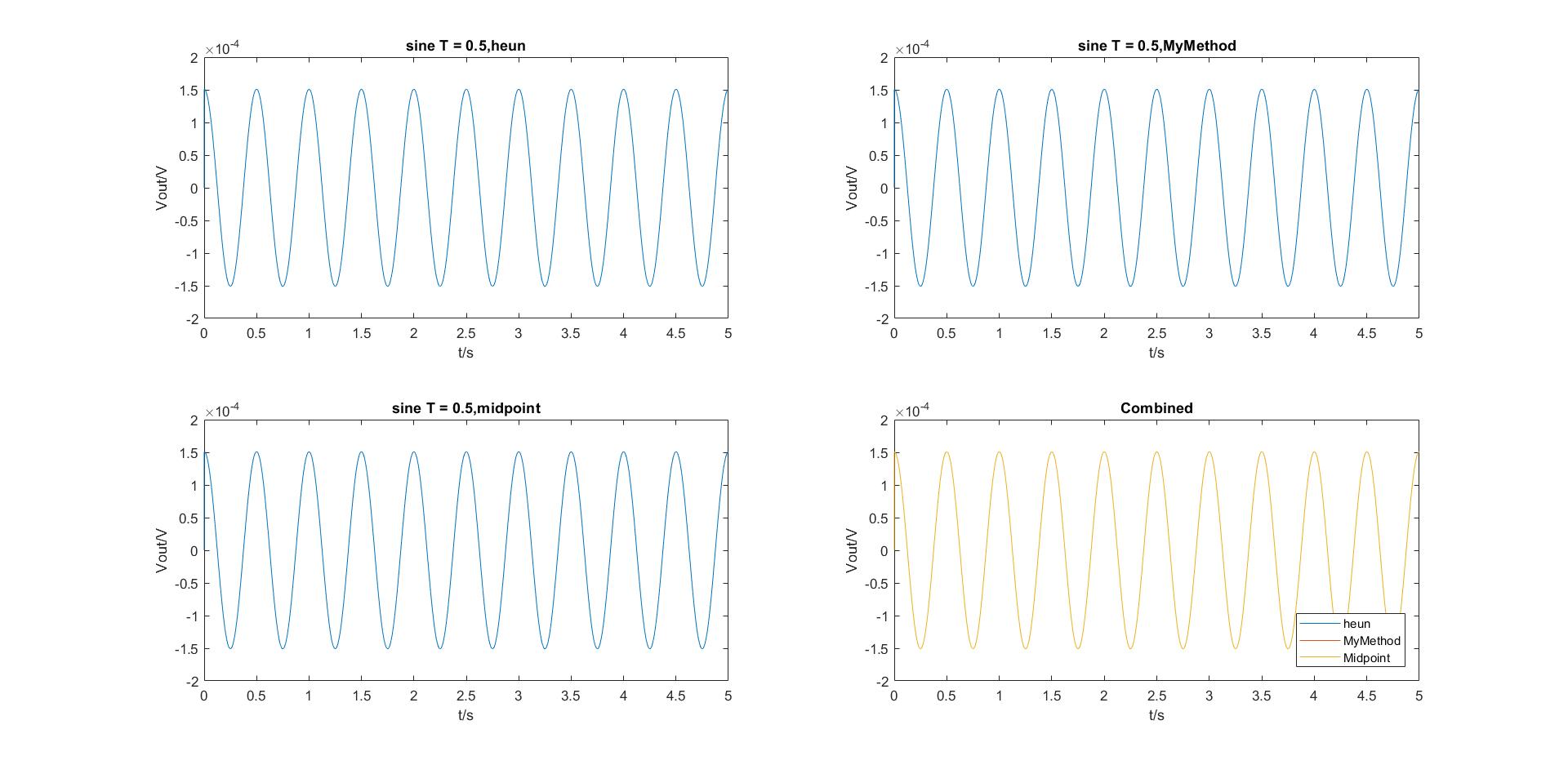


Figure x

It can be seen from the graph that the amplitude of the output signal decreases to nearly half of . This is because that the RL circuit behaves as a high pass filter, it compresses low frequency components and allows high frequency components to pass through.

**Error Analysis**

In this exercise, we aim to carry out an error analysis given that the input signal is a cosine wave of period *T = 150µs* and the amplitude of . In order to examine the error introduced by using numerical methods, an exact solution of the ODE equation needed to be obtained and is used to compare with the numerical solutions.

The first ODE equation is shown as below:

where

This ODE equation can be solved by using integrating factor,

Then multiply both sides by the integrating factor:

Now integrating both sides and simplify:

Integrating:

c

To obtain the value of c, the initial conditions substitute into the equation:

Thus, the exact solution can be evaluated as:

To obtain the exact solution of vout:

The file error\_script.m is shown below to illustrate the method of obtaining the error function and calculate the order of the error.

clear all; close all;

%initialise all the input parameters

tf = 0.005;

ti = 0;

ii = 0;

R = 0.5;

L = 0.0015;

h = 0.000001;

V = 6;

T = 0.00015;

a = 2\*pi/T;

%ln(7/2) = log10 (7/2)/log10 (e);

Vin = @(t) V\*cos(2\*pi\*t/T);

func = @(t,i) (1/L)\*(Vin(t) - R\*i);

[t1,vout1] = heun(func, Vin, tf, ti, ii, R, L,h);

[t2,vout2] = MyMethod(func, Vin, tf, ti, ii, R, L,h);

[t3,vout3] = midpoint(func, Vin, tf, ti, ii, R, L,h);

solution = @(t) (6\*R/(R^2+a^2\*L^2))\*cos(a\*t)+(6\*L\*a/(R^2+a^2\*L^2))\*sin(a\*t)-(6\*R/(R^2+a^2\*L^2))\*exp(-R\*t/L);

exact = @(t) Vin(t) - R\*solution(t); %exact solution of ODE

%-----------------------errors for three methods--------------------

figure(2);

exact\_value = feval(exact,t1);

error\_heun = exact\_value - vout1;

error\_MyMethod = exact\_value - vout2;

error\_Midpoint= exact\_value - vout3;

plot(t1,error\_heun);

hold on;

plot(t2,error\_MyMethod);

hold on;

plot(t3,error\_Midpoint);

hold off;

xlabel('t/s'),ylabel('Error'),title('Error function');

legend('error heun','error MyMethod','error Midpoint','Location','northeast');

%--------------------Ploting order of the error-----------------

i=1;

for j=15:25

clear vout1 vout2 vout3;

h = 2^(-j); %varing value of h

[t1,vout1] = heun(func, Vin, tf, ti, ii, R, L,h);

[t2,vout2] = MyMethod(func, Vin, tf, ti, ii, R, L,h);

[t3,vout3] = midpoint(func, Vin, tf, ti, ii, R, L,h);

exact\_value2 = feval(exact,t1);

error\_order\_heun(i) = max(abs(exact\_value2 - vout1));

error\_order\_MyMethod(i) = max(abs(exact\_value2 - vout2));

error\_order\_Midpoint(i)= max(abs(exact\_value2 - vout3));

h\_temp(i) = h;

i=i+1;

end

figure(3);

loglog(h\_temp,error\_order\_heun);

hold on;

loglog(h\_temp,error\_order\_MyMethod);

hold on;

loglog(h\_temp,error\_order\_Midpoint);

hold off;

xlabel('h'),ylabel('error'),title('Orders of error');

legend('error order heun','error order MyMethod','error order Midpoint','Location','northeast');

The array ‘exact\_value’ stores the values of exact solution evaluated covering the full time period. Therefore, the error function can be calculated by subtracting the output signal from the exact\_value.

The for loop with iterator j is used to change the value of step size h, since h is calculated by The value of j is chosen from 15 to 25, the range of the values cannot be too large since it might overload MATLAB and h should be small enough; since step size should be less than the time period of the cosine wave and the smaller the step size is, the more accurate the approximation.

The error\_order arrays stores the absolute value of the maximum error and the order of the error is showed by a log-log plot.

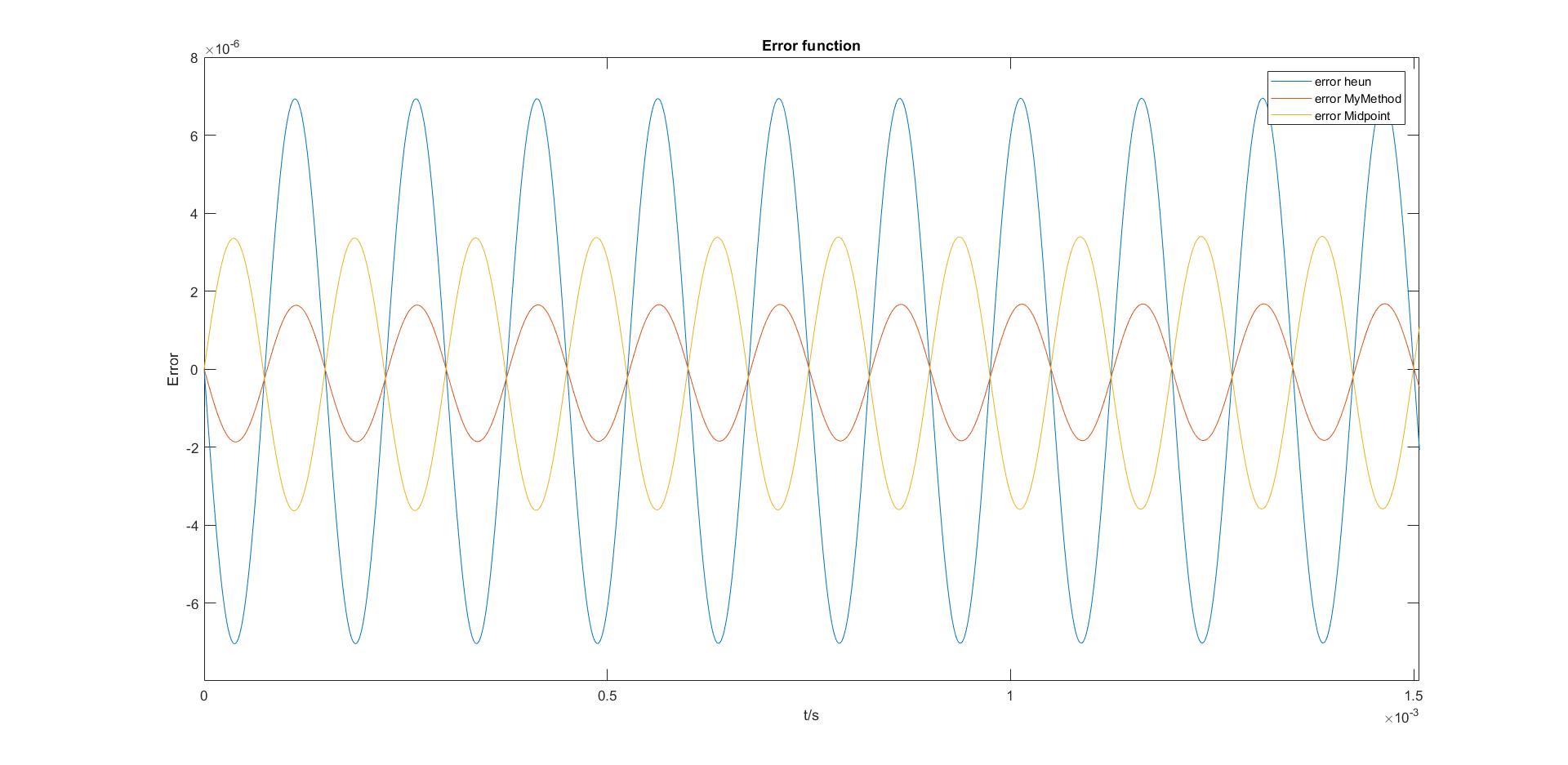


Figure x

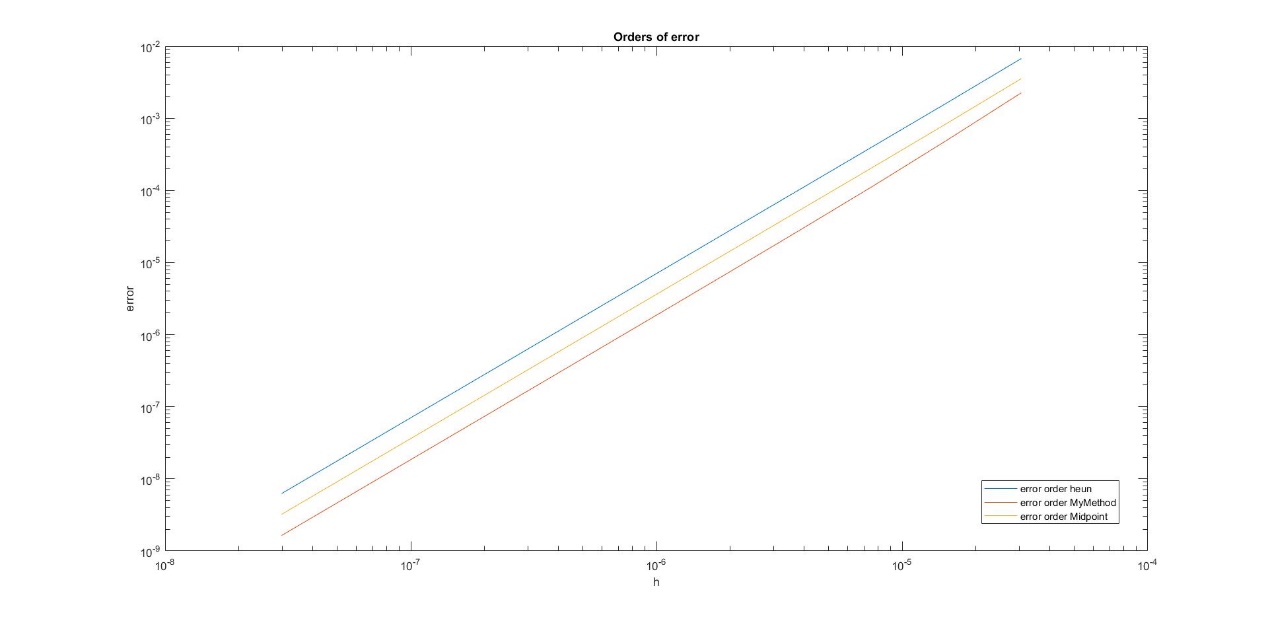


Figure y.

The performance of each method can be seen in figure x. As the graph shows, my method is the most accurate among three methods and Heun’s method is the least accurate, since it has more than three times the error of my method. Moreover, there is a phase shift for the midpoint method.

It can be seen from figure y that the error order function of three methods has similar gradient, but with different value of y-axis intersection. This indicates that there is a consistency within the global truncation error of second-order Runge-Kutta methods as O(h2).