**Numerical Analysis of Diﬀerential Equations using Matlab -2019**

Dr Daniel Nucinkis

**Group 6**

Weiyue Leng wl4817, Kailai Lin kl5017, Zhenming Zhang zz217, Zhiqing Zhong zz8317, Oliver Babington Ellis ob1717, Rudic Grigoryan rg517, Alex Constantinou ac517

**Exercise 1**

In part one we have a high-pass filter, which takes an input signal, Vin, and only lets 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.

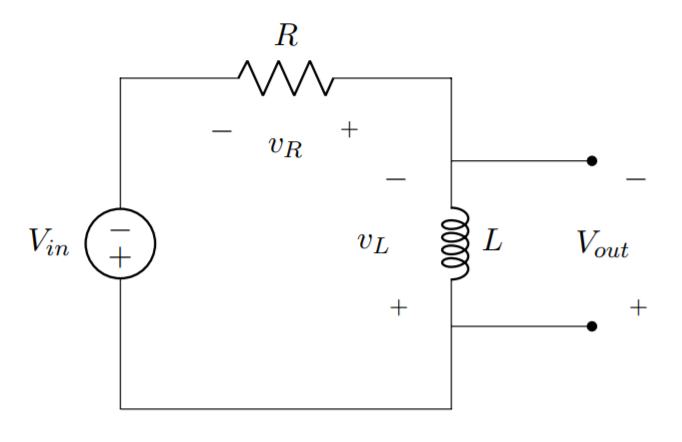


Figure 1. RL circuit

For this circuit, the ODE equation can be written as:

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

The output is calculated by:

The 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 .

Therefore, for each section the straight line has the following equation:

Which can be rewritten as

Where is the derivative of the curve at point .

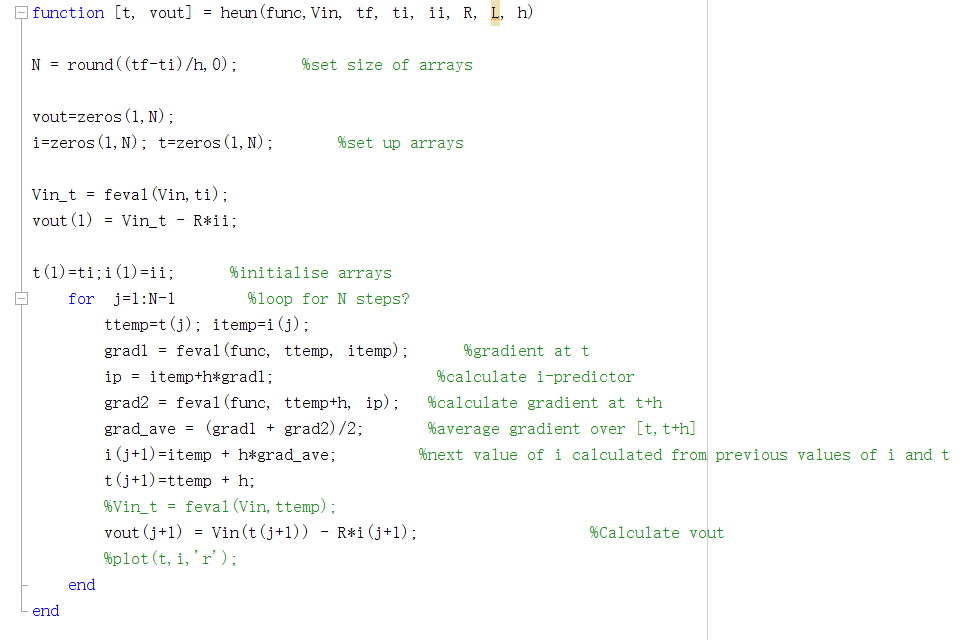
However, since the concavity of the curve cannot be predicted and remains consistent, Euler’s method may give an overestimate or an underestimate at different points. Thus, the following methods are introduced to help solving this problem.

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/under-estimating.

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 will later be used to store values once evaluated by the “function”. Vin\_t is a variable which is 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.

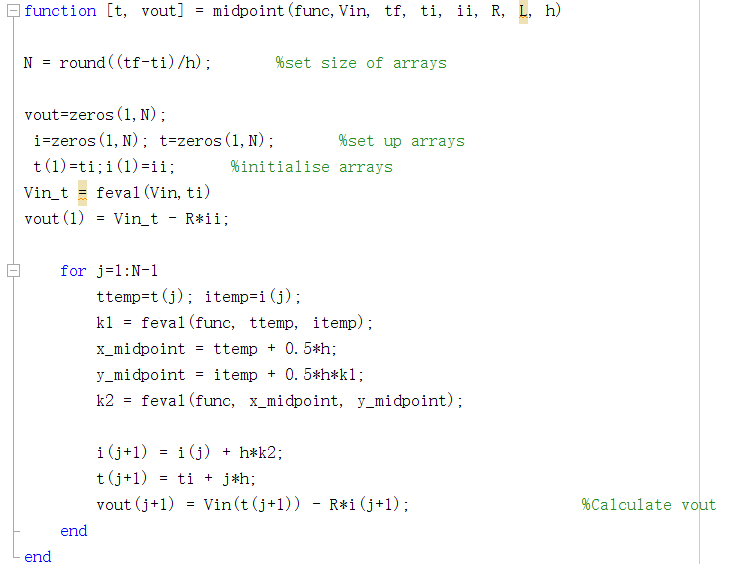
1. *Midpoint Method*

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, at the point 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 code is also used for Euler’s method, thus shows the similarities between the two methods. The only difference is the updated value for to find the gradient at the midpoint.



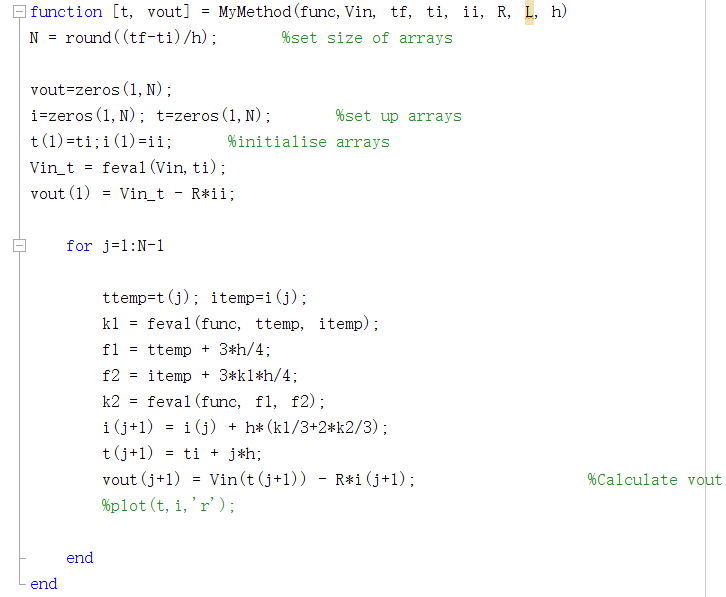
1. *My method*

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 their constants.



Inputs:

1. Step signal with amplitude Vin = 3.5V

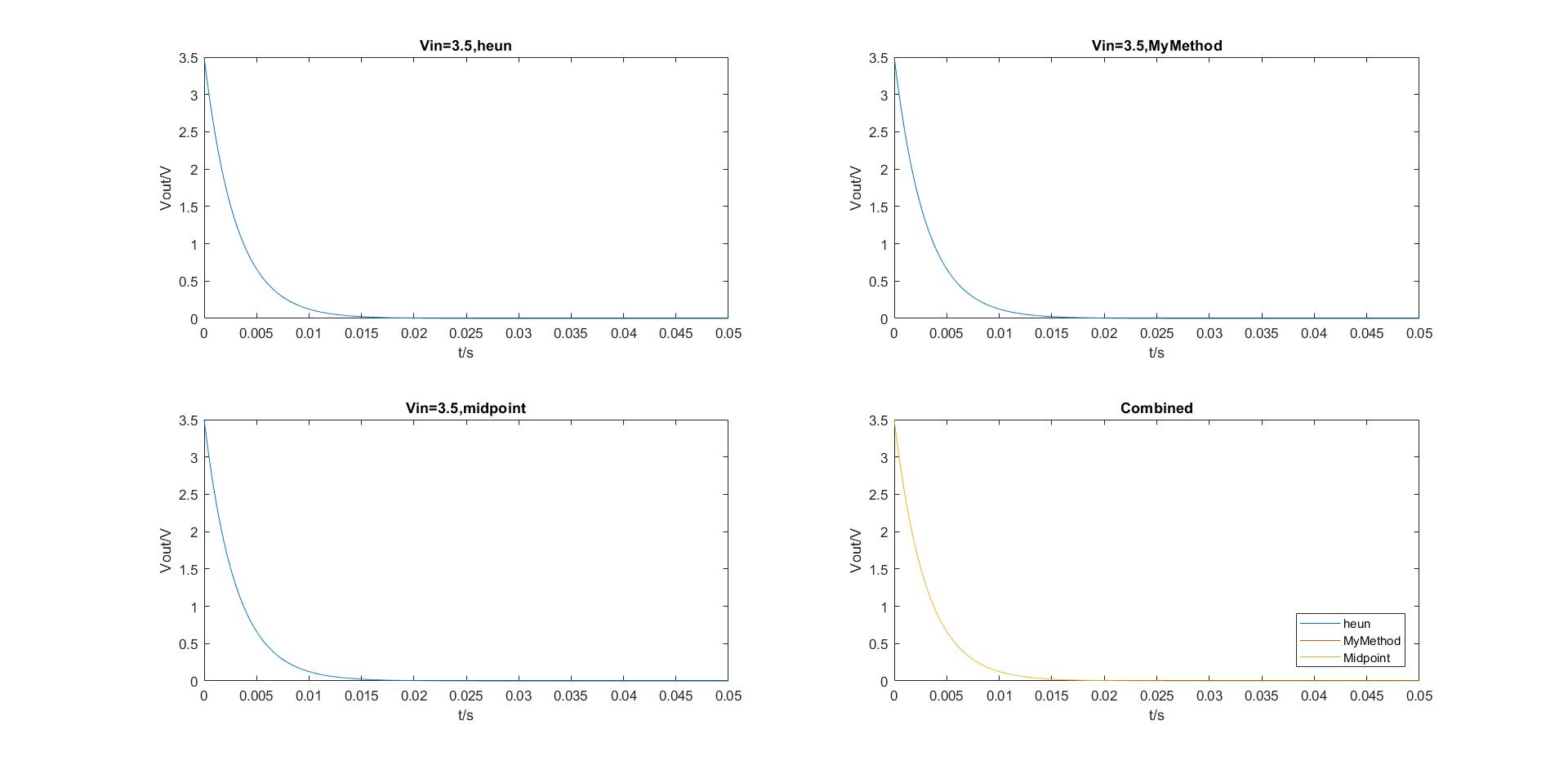
With the step signal as an input, we observed how the output signal decays exponentially. The current flowing through the inductor gradually increases from 0A to 7A, this is because the current through an inductor cannot change instantaneously. However, the voltage across an inductor can. Therefore, 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.

Figure 2. Step Signal with Amplitude Vin = 3.5V

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.

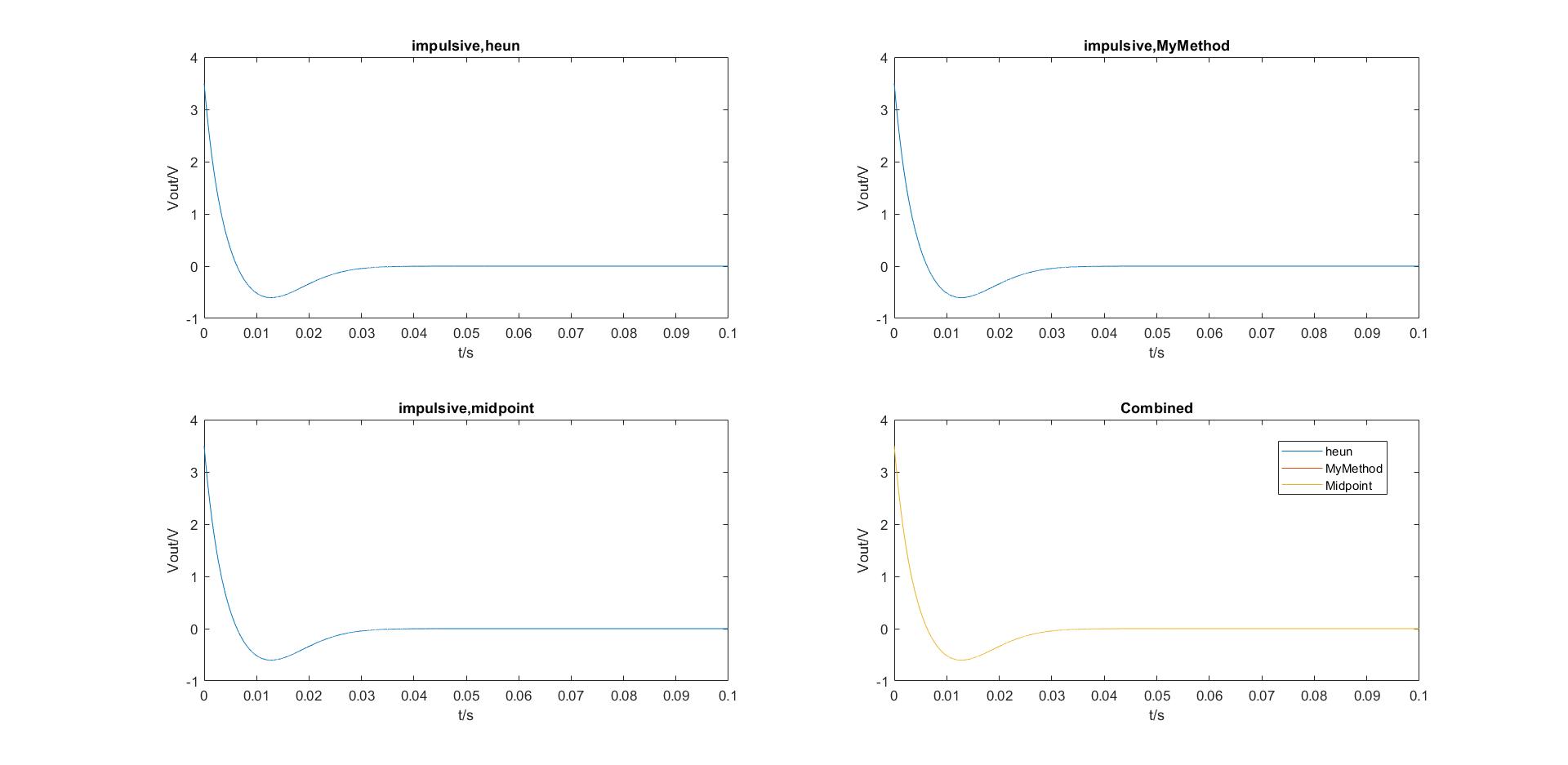
The figure shows the impulsive signal, the output voltage decays exponentially with initial amplitude of 3.5. However, during the time period, 0.00614s to 0.02957s, the output signal forms a local minimum with negative amplitude, then gradually increases back to zero.

Figure 3. Impulsive Signal

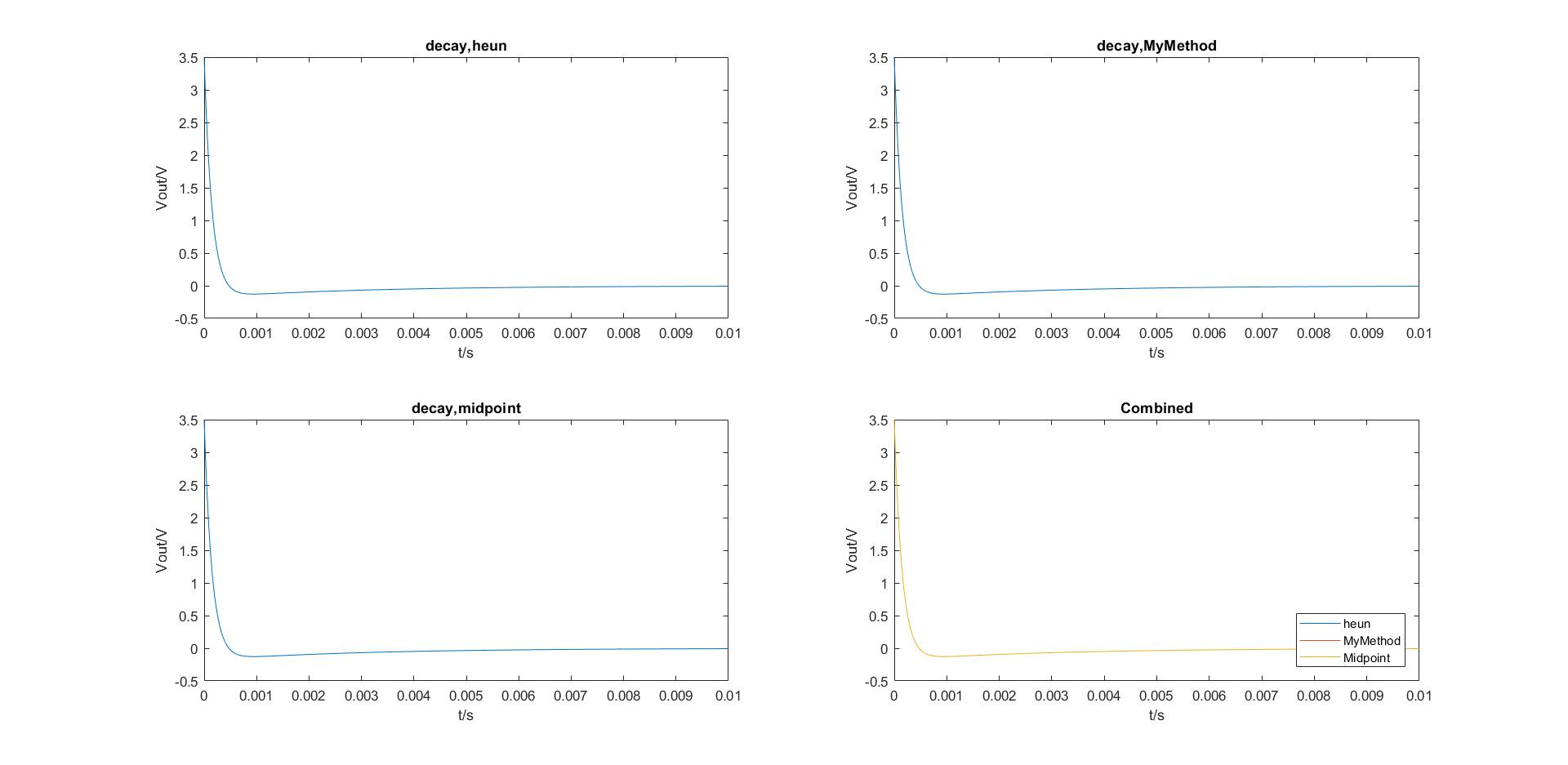
Comparing the impulsive to the decay signal, the following is observed: the decay curve does not curve down to negative values as much as the impulsive signal. Moreover, the decay signal stays steady at around 0.003s, whereas for the impulsive signal it takes longer to decay to zero. This means that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

Figure 4. Decay Signal

1. Sine waves with amplitude

with periods: T = 15 µs, T = 150µs, T = 400µs, T = 1100µs

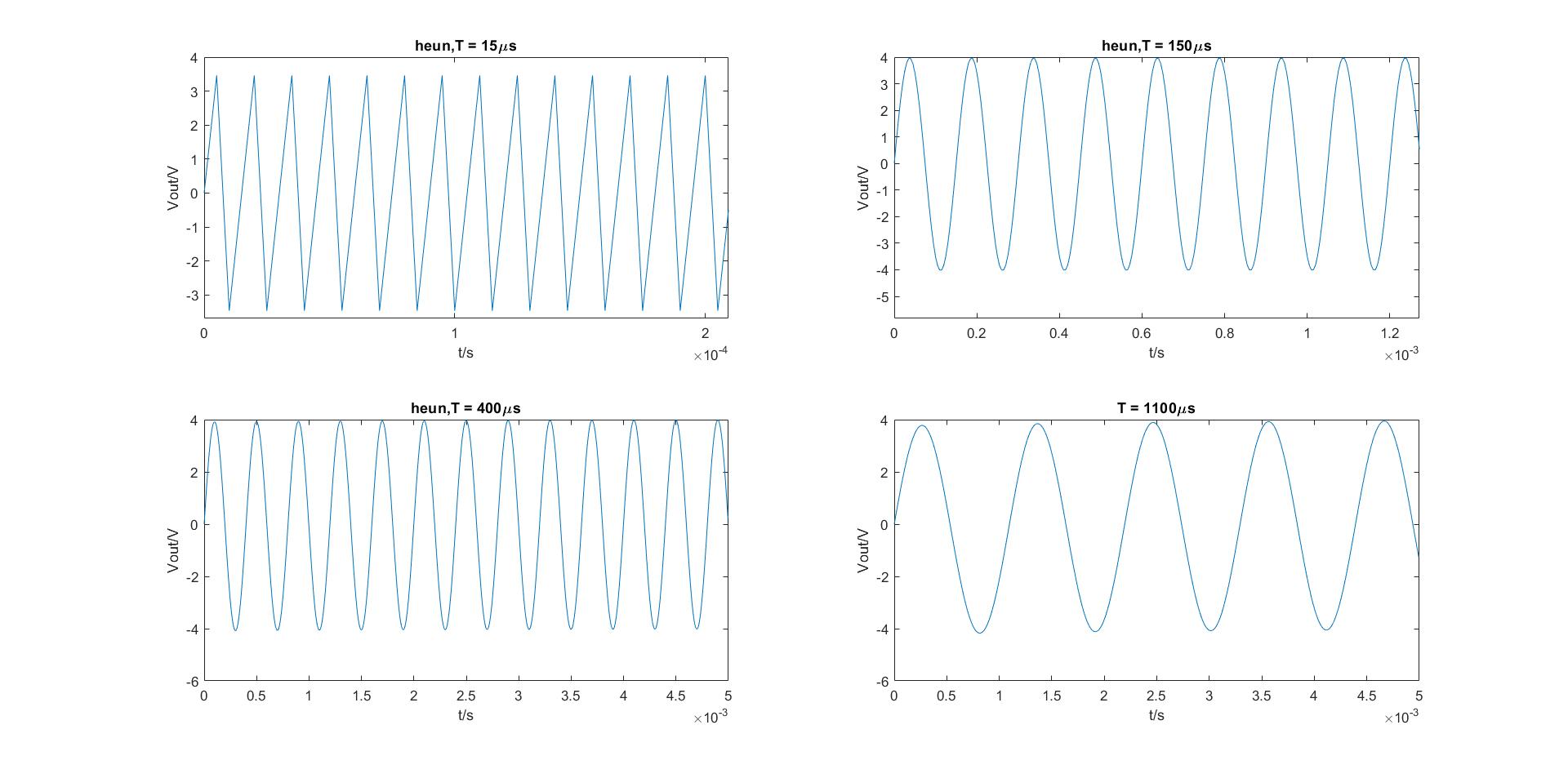
Heun’s method

Figure 5. Sine Waves Generated by Heun’s Method

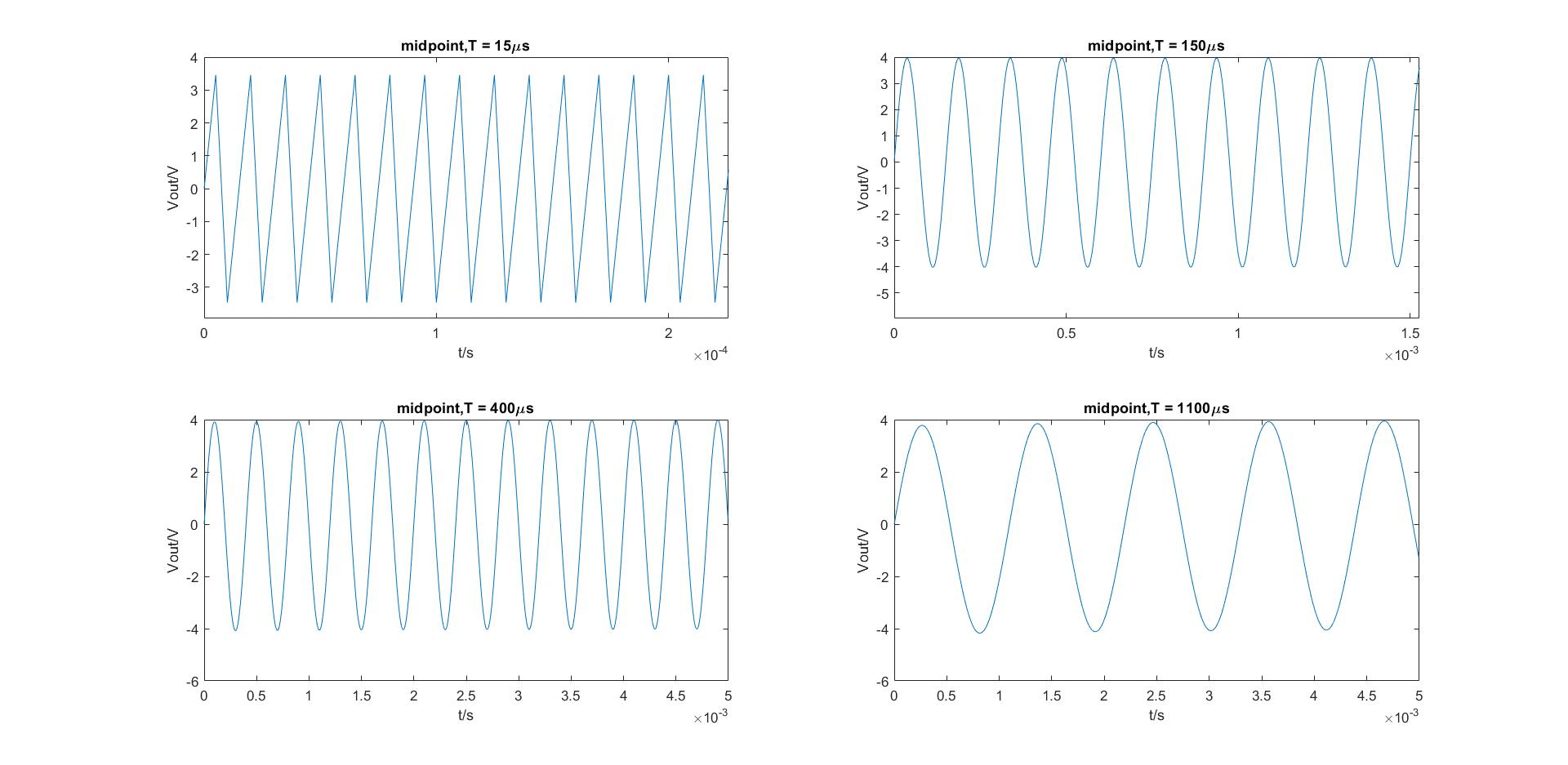
Midpoint method

Figure 6. Sine Waves Generated by Midpoint Method

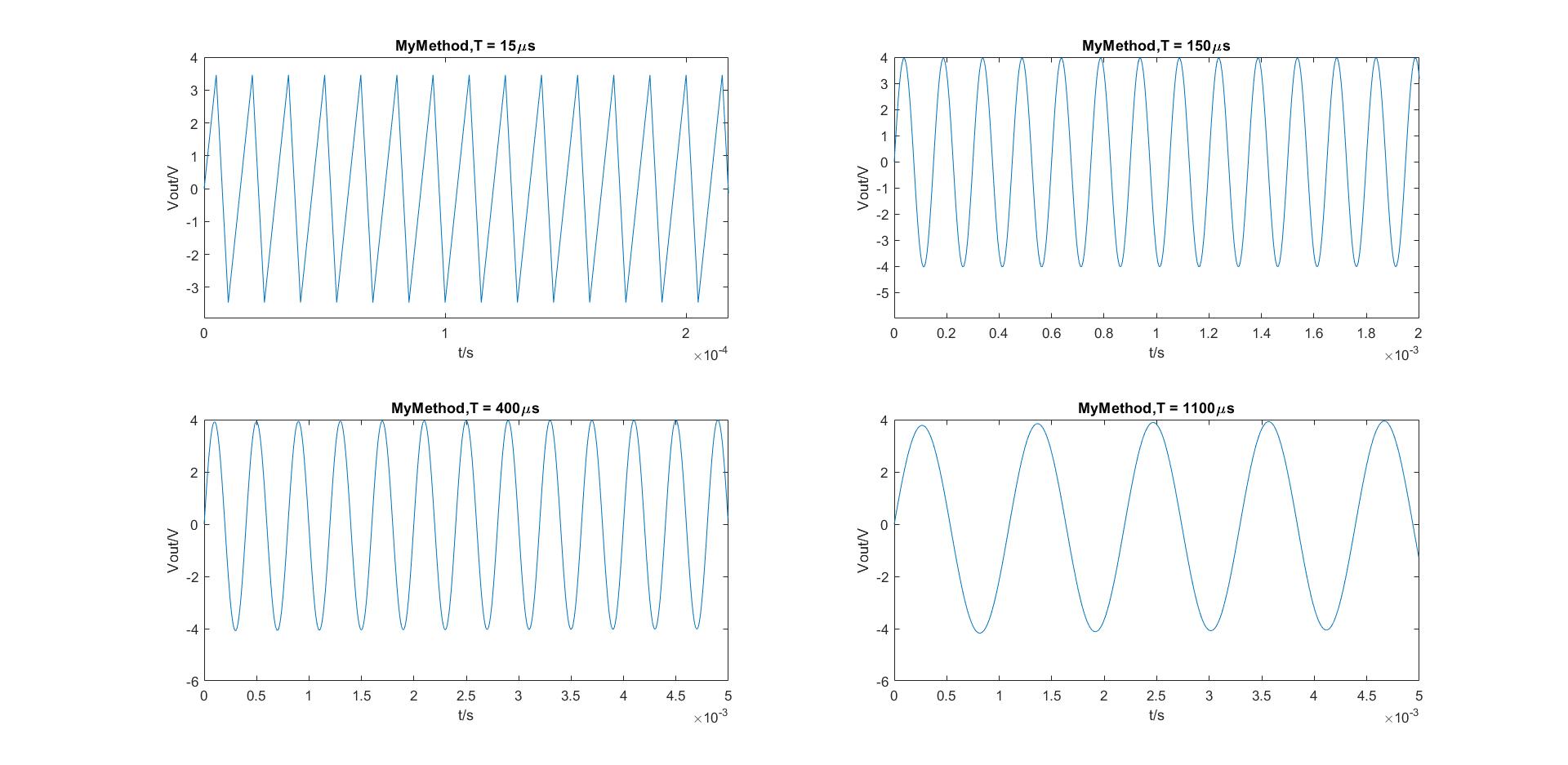
My method

Figure 7. Sine Waves Generated by 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. 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

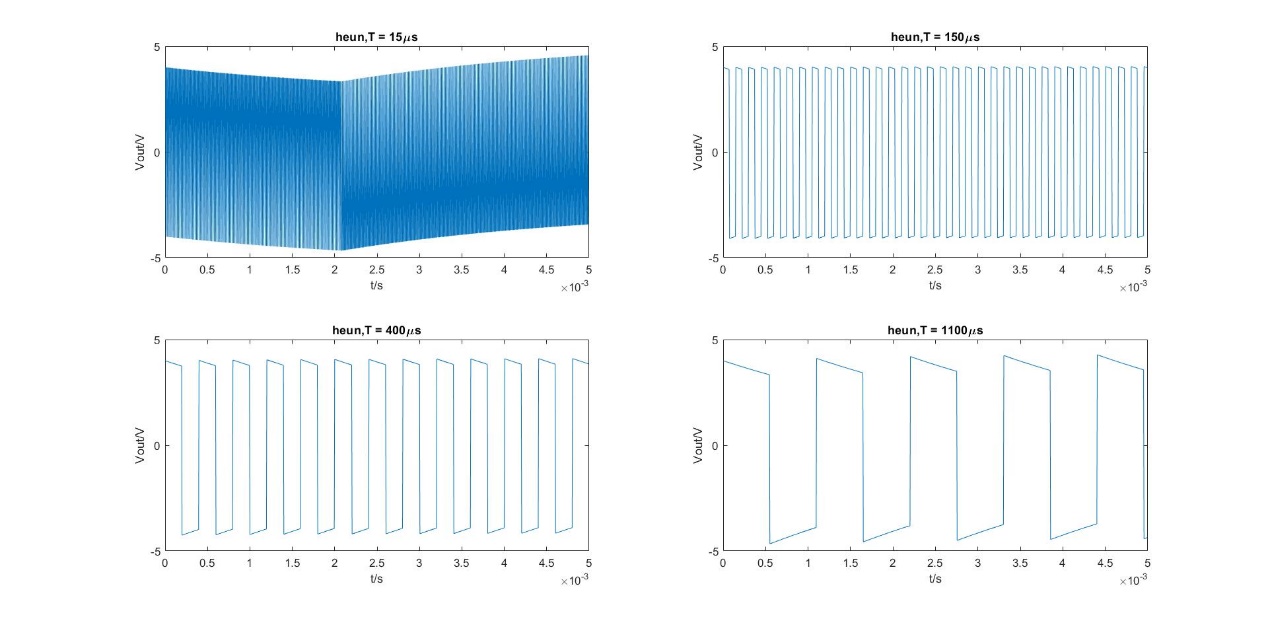
Heun’s method

Figure 8. Square Waves Generated by Heun’s Method

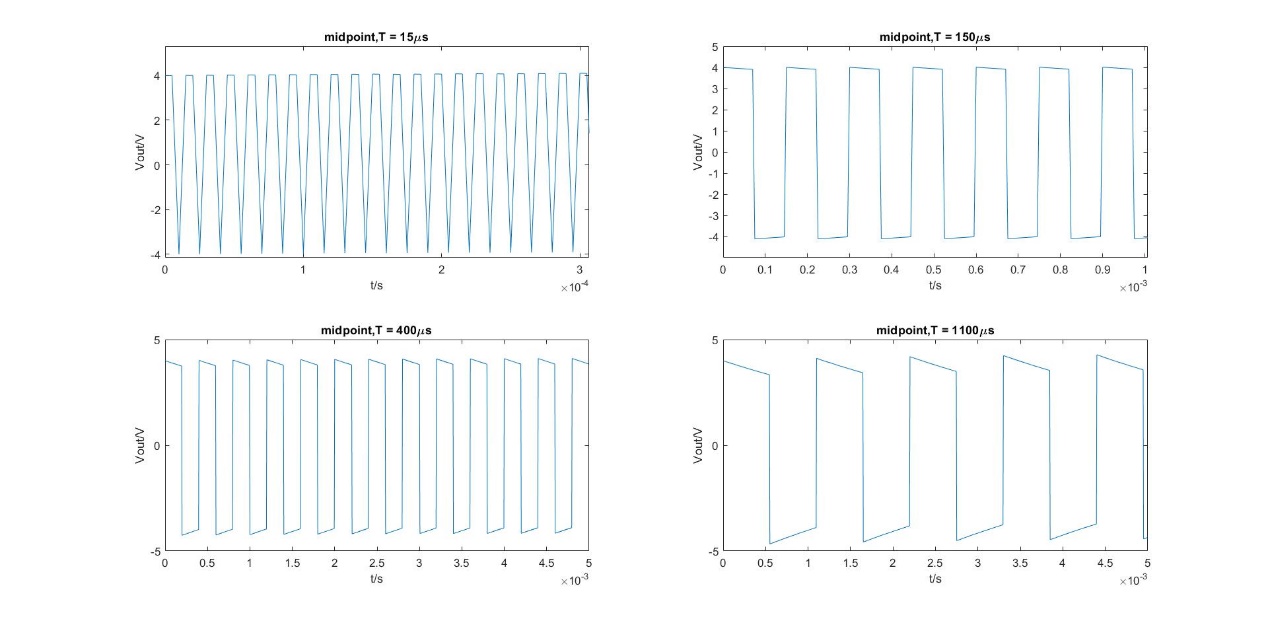
Midpoint method

Figure 9. Square Waves Generated by Midpoint Method

My method

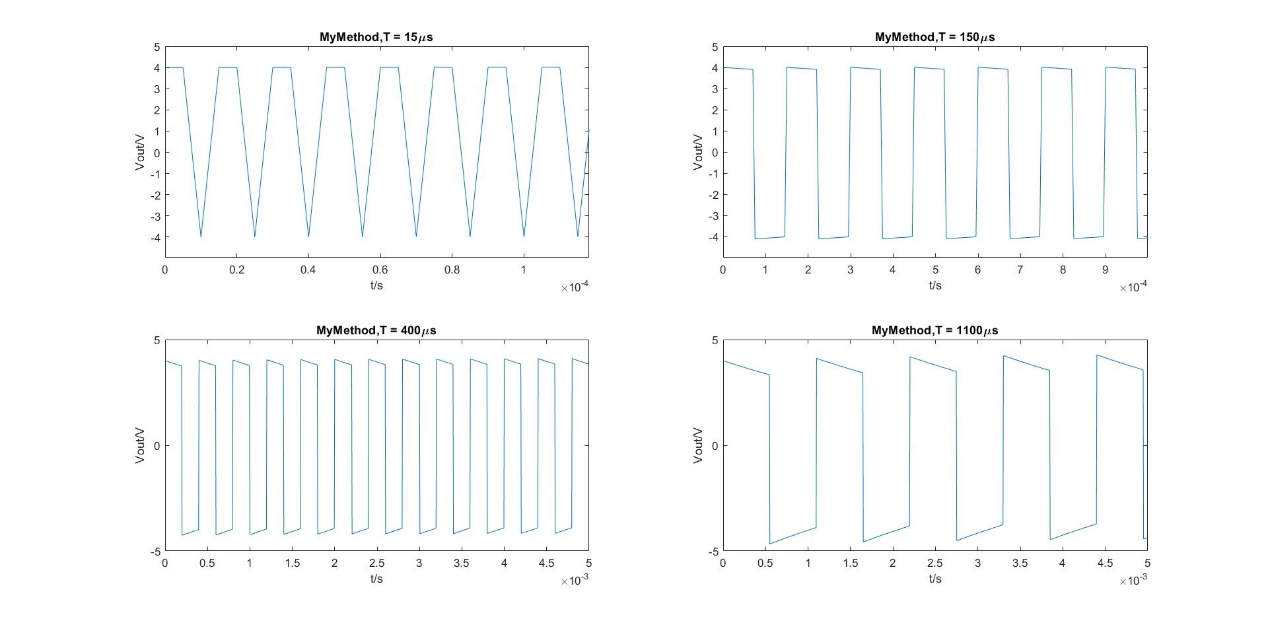


Figure 10. Square Waves Generated by My Method

It can be seen from the graphs 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 the reason for this 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

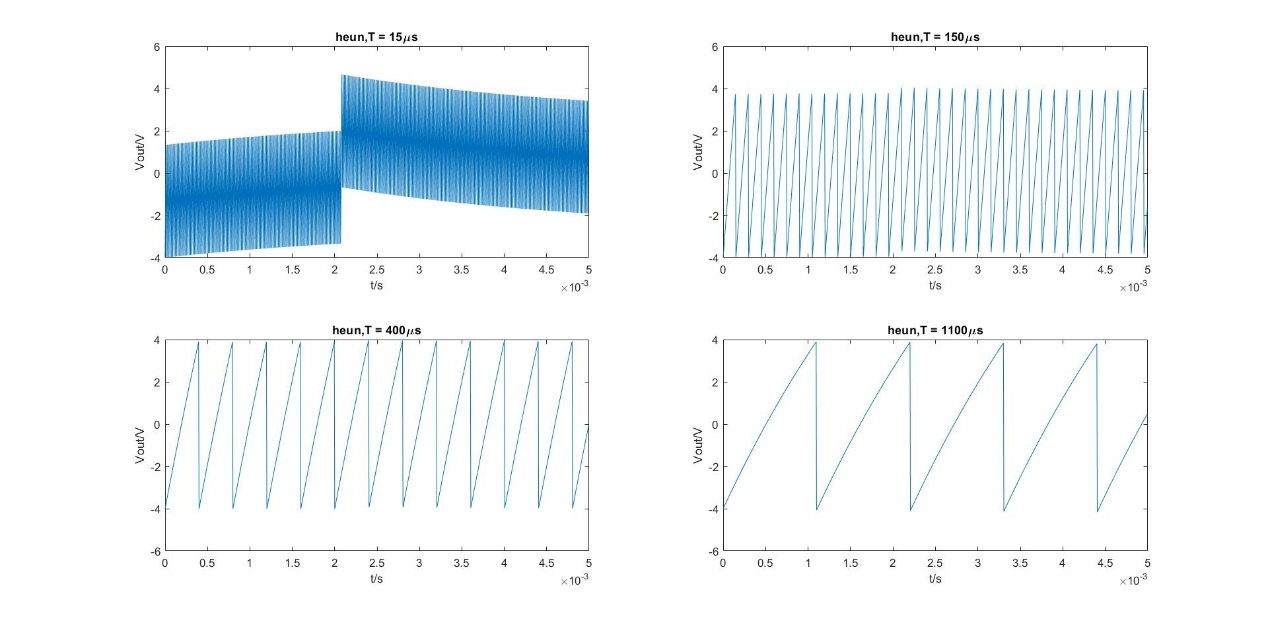


Figure 11. Sawtooth Waves Generated by Heun’s Method

Midpoint method

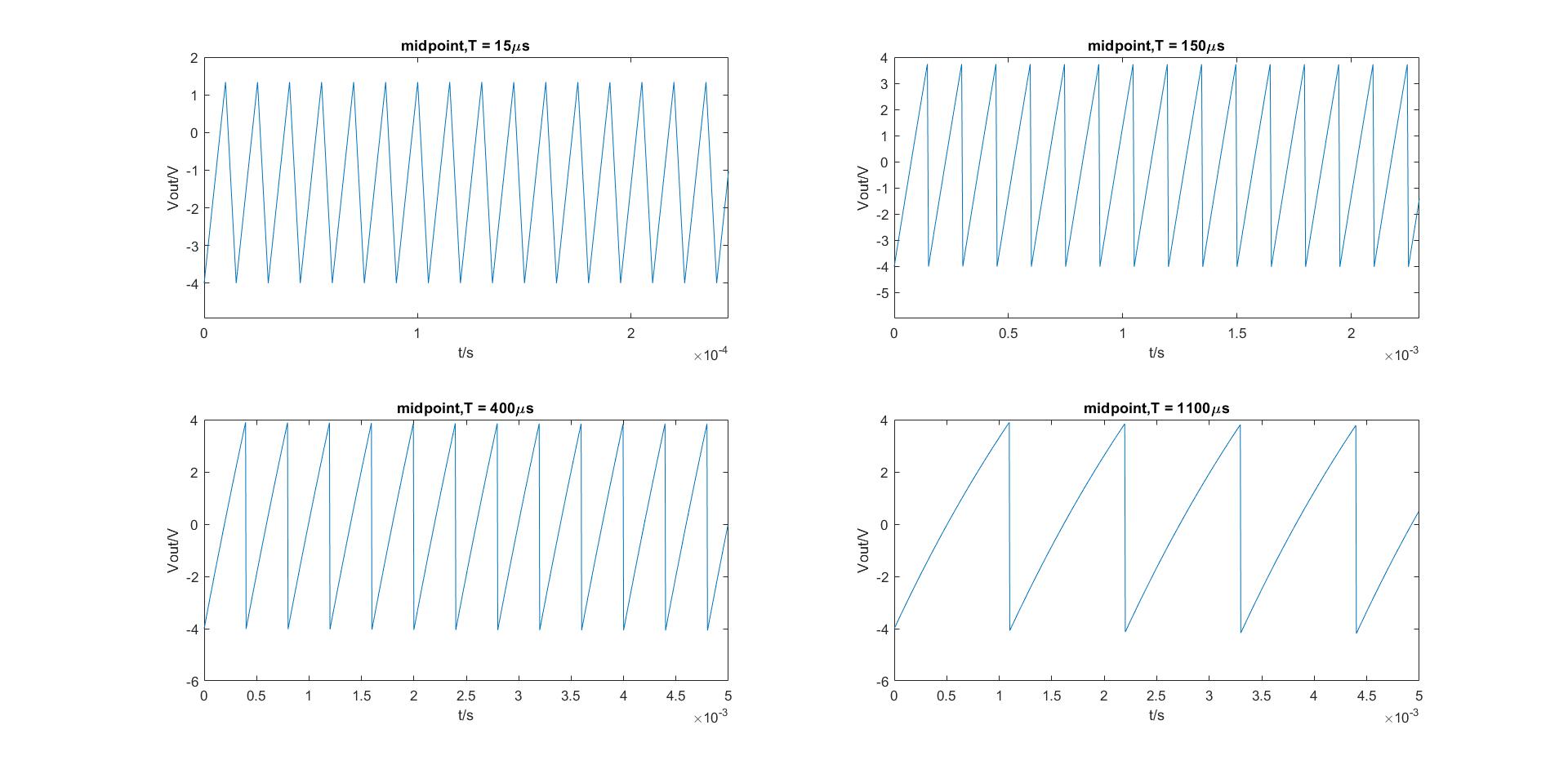


Figure 12. Sawtooth Waves Generated by Midpoint Method

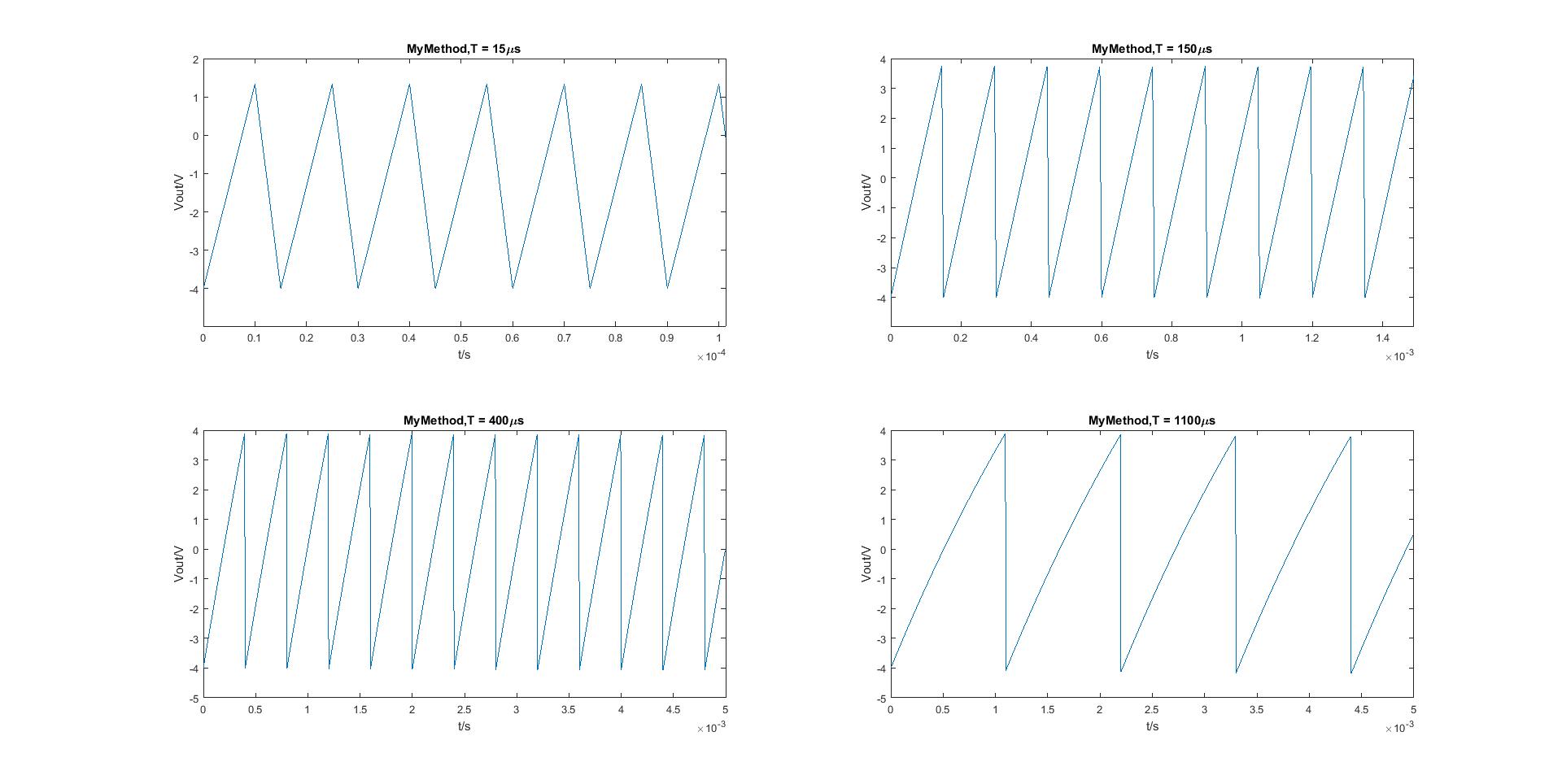
My method

Figure 13. Sawtooth Waves Generated by My Method

All of the methods give a reasonable response at periods 400µs or 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, number of input parameters have been changed throughout the test.

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

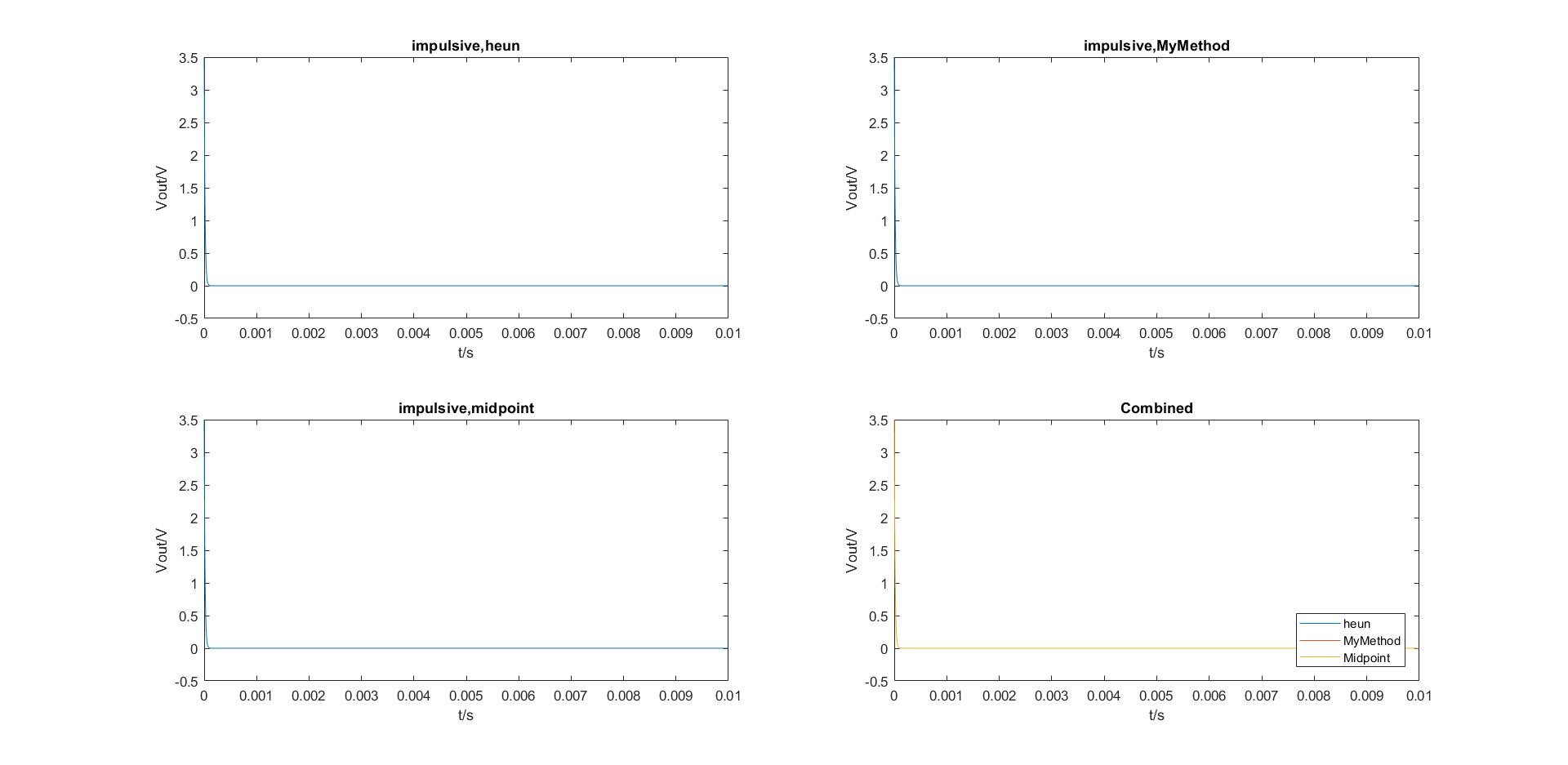


Figure 14. Impulsive Waves with R=500

The input signal is an impulsive wave and the value of inductance stays 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 for a fixed inductance, when the resistance increases to a very large value, the inductance becomes negligible compared to 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.

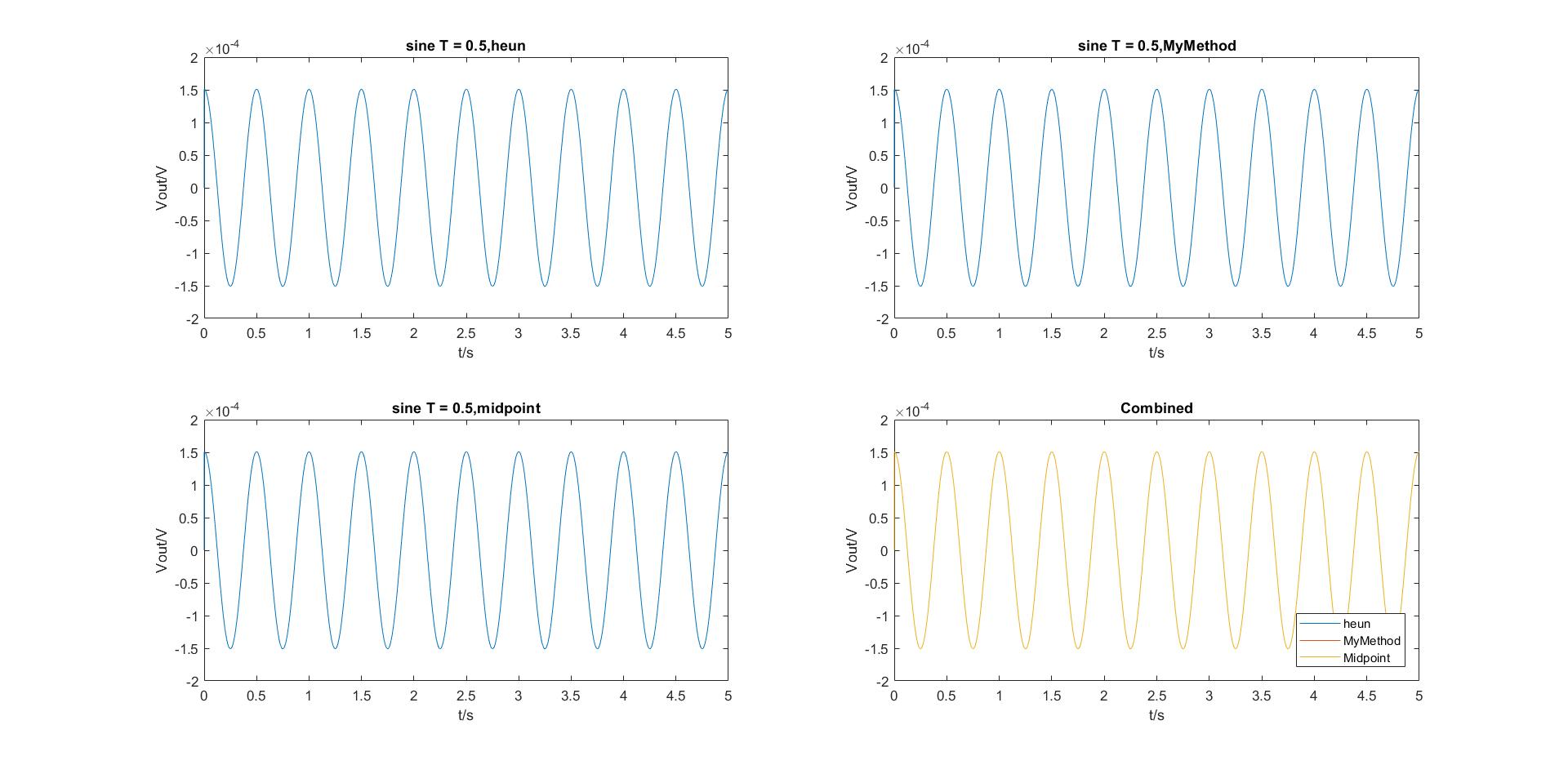


Figure 15. Sine Waves with T = 0.5

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

**Exercise 2**

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 the numerical methods, an exact solution of the ODE equation needs to be obtained and is used to compare to 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 solutions 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, the more accurate is 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.

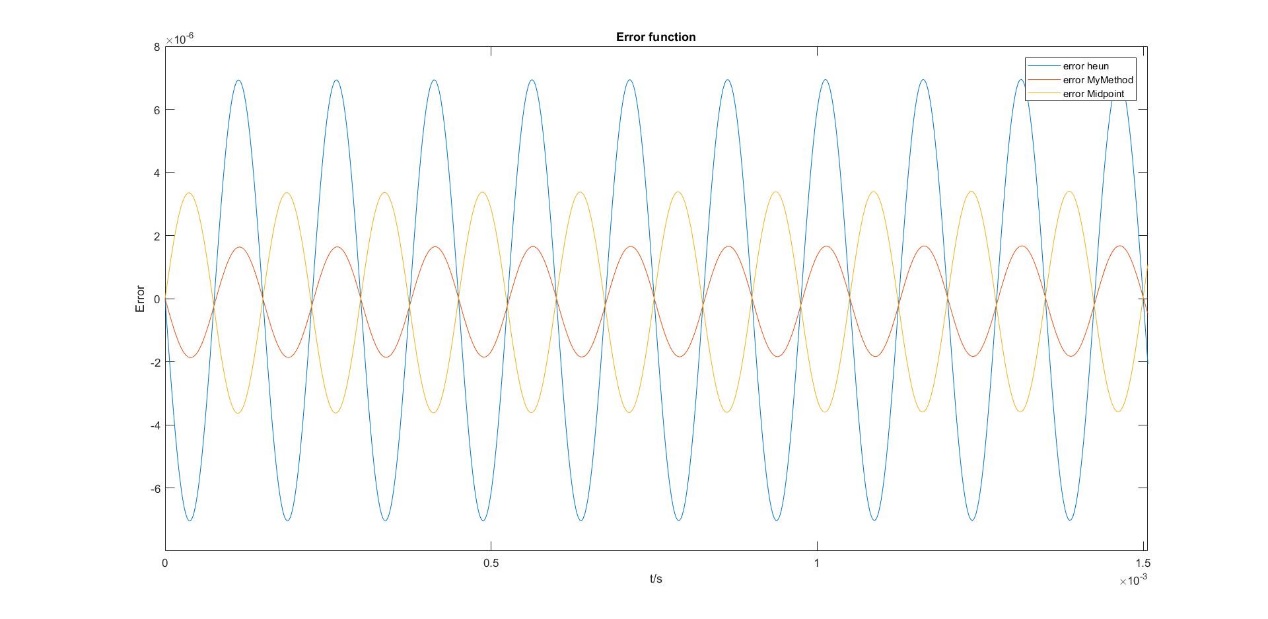


Figure 16. Error Functions

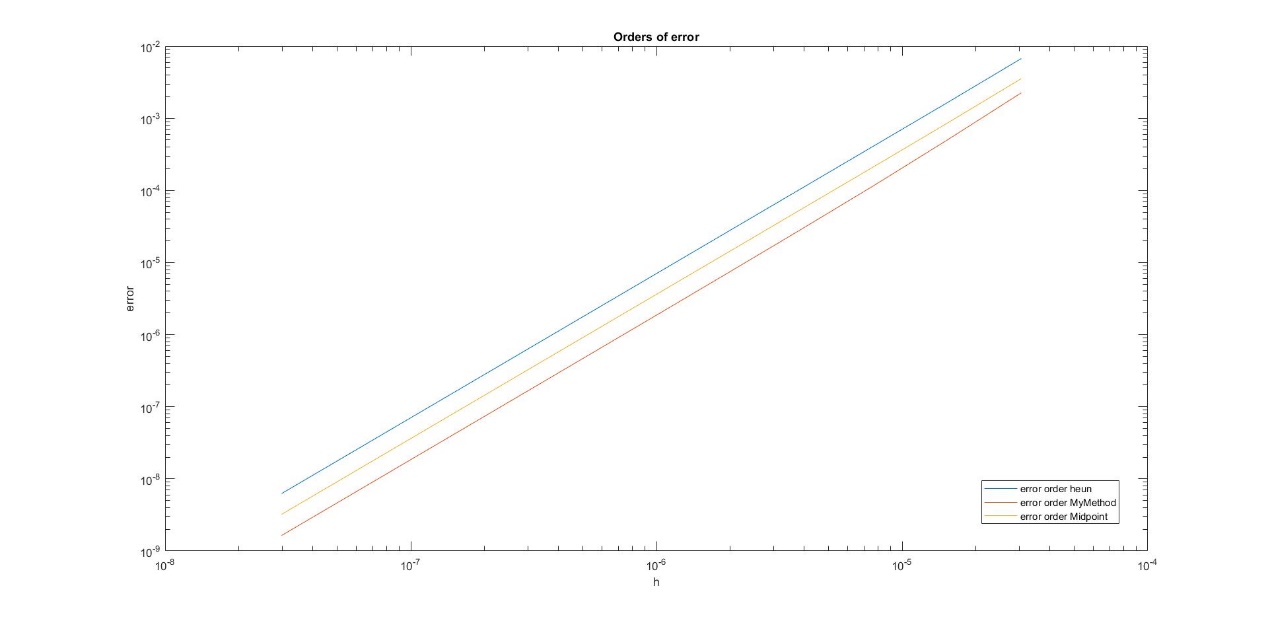


Figure 17. Orders of Error

The performance of each method can be seen in figure 16. 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 17 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).

**Exercise 3**

An RLC circuit consists of resistors, inductors and capacitors. It represents a standard example of a harmonic oscillator, namely a device which is able to resonate to a sinusoidal input signal (e.g. voltage or current). The figure below is an RLC circuit with one resistor, one capacitor and one inductor.

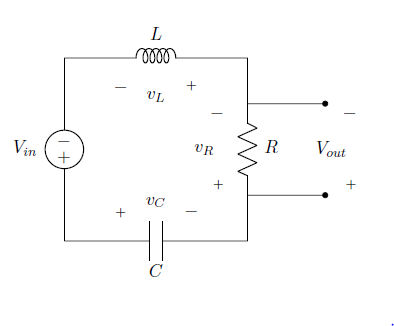
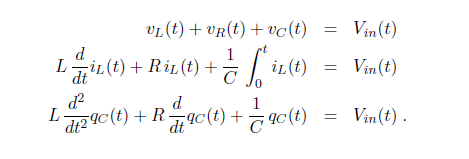


Figure 18. RLC circuit

 By applying Kirchhoff’s Law, this system could be modelled by three equations as follows:

The given conditions are:

* the capacitor is pre-charged at time t = 0 with qc(0) = 500 nC
* no initial current flows through the inductor at time t = 0
* R = 250Ω; C = 3 μF; L = 650mH:

Runge-Kutta 4th Order Method

Runge-Kutta Methods are widely used to solve 1st order ODE’s. In this report, the fourth-order Runge-Kutta 3/8 algorithm is implemented in Matlab to solve the RLC system. The 3/8 method is a fourth order Runge-Kutta method for approximating the solution of the initial value problem y'(x) = f(x,y); y(x0) = y0 which evaluates the integrand, f(x,y), four times per step. For step i+1,

yi+1 = yi + 1/8 ( k1 + 3\*k2 + 3\*k3 + k4 ), where

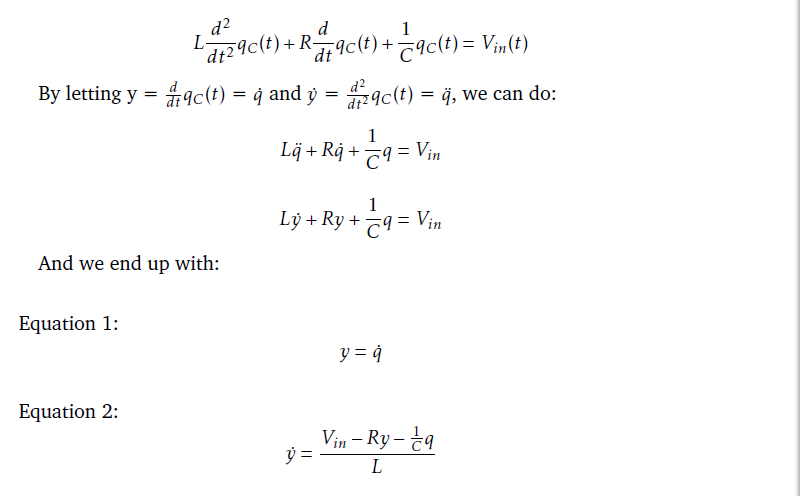
k1 = h\*f(xi, yi),

k2 = h\*f(xi + h / 3, yi + k1 / 3 ),

k3 = h\*f(xi + 2\*h / 3, yi - k1 / 3 + k2 ),

k4 = h\*f(xi + h, yi + k1 - k2 + k3 ),

and xi = x0 + i\*h.

However, at this point the system is a 2nd order ODE and it needs to be reconstructed to a coupled form.

Using the Runge-Kutta 3/8 algorithm and the coupled equation, the matlab function called **RK4second.m** and a matlab script called **RLC script.m** are established:

The function call includes arguments xi, yi and ti and returns xi+1 and yi+1

% reference from http://www.mymathlib.com/diffeq/runge-kutta/runge\_kutta\_3\_8.html

%

%yi+1 = yi + 1/8 ( k1 + 3 k2 + 3 k3 + k4 ),

%where

%k1 = hf(xi, yi),

%k2 = hf(xi + h / 3, yi + k1 / 3 ),

%k3 = hf(xi + 2 h / 3, yi - k1 / 3 + k2 ),

%k4 = hf(xi + h, yi + k1 - k2 + k3 ),

%and xi = x0 + i h

function [xii,yii] = RK4second(xi,yi,t,h,f1,f2)

k1 = h\*f2(xi, yi, t);

k2 = h\*f2(xi+h/3, yi+k1/3, t);

k3 = h\*f2(xi+2\*h/3, yi-k1/3+k2, t);

k4 = h\*f2(xi+h, yi+k1-k2+k3, t);

yii = yi + 1/8\*(k1 + 3\*k2 + 3\*k3 + k4);

xii = xi + h\*f1(xi, yi, t);

end

The script below is used to simulate the RLC system and test the funtion under different input voltages.

function RLC\_Script ()

close all;

clear;

clc;

%initailise parameters

R = 250; %resistance

C = 3e-6; %capacitance

L = 650e-3; %inductance

h = 0.00001; %stepsize

tf = 0.5; %final time

N = round(tf/h); %number of steps

q = zeros(1, N); %charge

i = zeros(1, N); %current dqc/dt

t = zeros(1, N); %x-axis

q(1) = 500e-9; %intial charge

i(1) = 0; %zero initial current

t(1) = 0; %start at time 0

Vout = zeros(1, N); %output voltage

%input

%case 1

%5V dc

% Vin = @(t)5\*heaviside(t);

%case 2

% impluse signal with delay

%Vin = @(t)5\*exp(-(t^2)/(3e-6));

%case 3

%Square Wave with freq 5Hz, 100Hz, 500Hz

% f=500;

% Vin = @(t)5\*square(2\*pi\*f\*t);

%case 4

%%Sine Wave with freq 5Hz, 100Hz, 500Hz

f =500;

Vin = @(t)5\*sin(2\*pi\*f\*t);

%coupled first order ODEs from calculation

f1 = @(q, i, t)i;

f2 = @(q, i, t)(Vin(t) - R\*i - q/C )/L;

for k = 1 : N - 1

t(k + 1) = t(k) + h;

[q(k + 1), i(k + 1)] = RK4second(q(k), i(k), t(k), h, f1, f2);

Vout(k) = R\*i(k);

end

plot(t,Vout);

hold on;

fplot(Vin,'--');

xlim([0,0.1]);

ylim([-6,6]);

xlabel('Time(s)');

ylabel('Vout(V)');

end

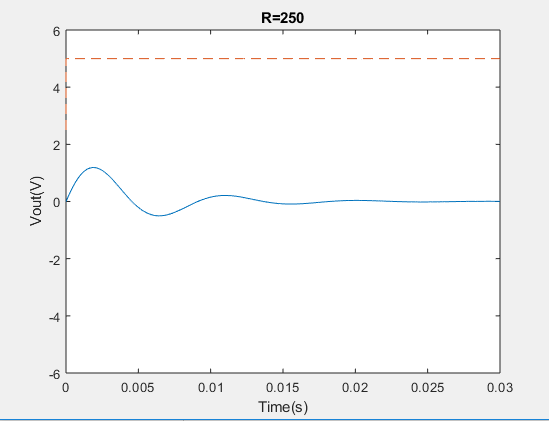
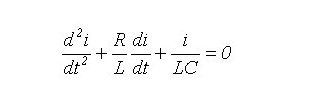
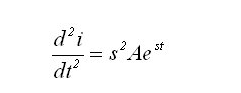
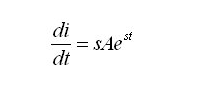
Different Input Voltages

Figure 19. Output for Step signal with amplitude of 5V

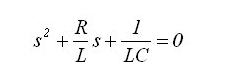
The output has a transient behaviour due to the capacitor and the inductor. The steady state is zero voltage, which means no current flow in the circuit due to the end of the charge processing for the capacitor. Furthermore, we could determine that the system undergoes an under-damping.

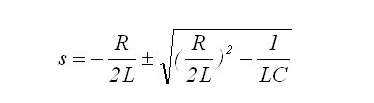
 We have the equation



Now let

Hence:



The last equation is a quadratic equation in terms of s. The roots of this equation are:

We know that

* If the determinant is greater than zero, there are two real roots and the circuit is said to be over-damped.
* If the determinant is negative, there are two complex roots and the circuit is said to be under-damped.
* If the determinant equals to zero, there is only one root and the circuit is said to be critically damped.

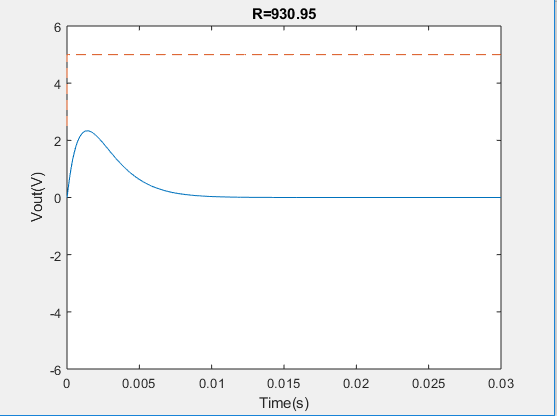
In this case the determinant is negative, therefore the output response is under-damped. We change the resistance of the resistor to obtain the other cases. From our calculations, setting R to 930.95 could achieve the critical damping and any value greater than 930.95 would get the result for over-damping. Here is the experiment figure:

Figure 20.

It can be easily discerned from the graph that there is no oscillation in this case.

Impulsive signal with decay

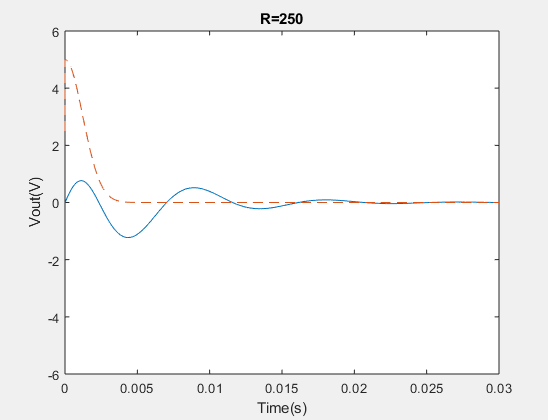
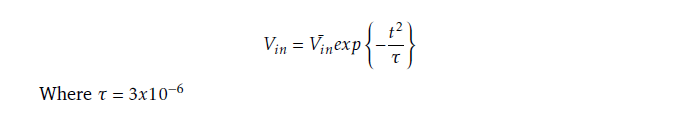


Figure 21.

The input voltage can be described as follow

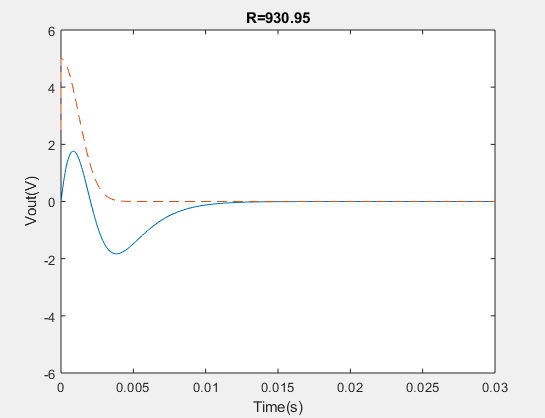
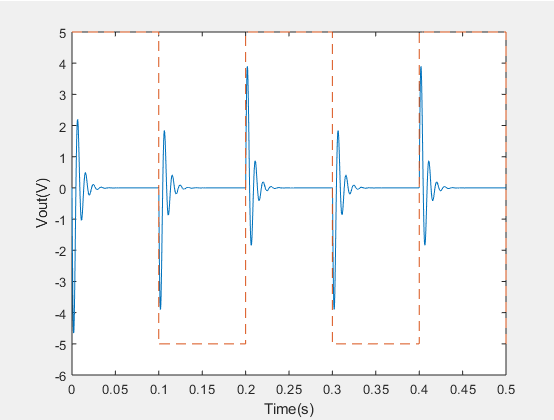
Compared with the step signal, the exponential decay is of the order , which leads to a very quick decay for the input signal. Although the impulse is of infinite amplitude, it decays to zero in an infinitesimal period of time and the output only rises for a bit. Besides, as in the previous example, the system is underdamped. Setting R to 930.35 could make a critical damping.

Figure 22.

Square wave with different frequencies

For this part, three square waves of 5Hz, 100H and 500Hz are tested and the graphs for each output are shown below:



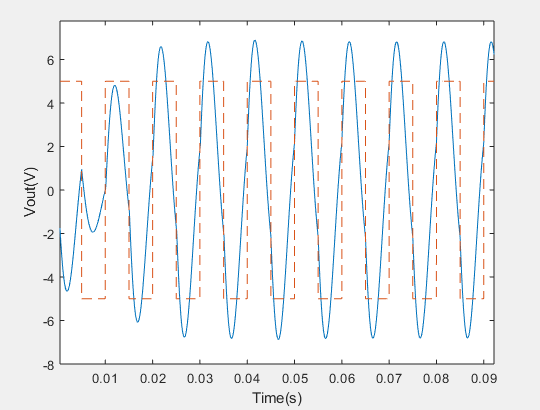
Figure 23. Output of 5Hz

Figure 24. Output of 100Hz

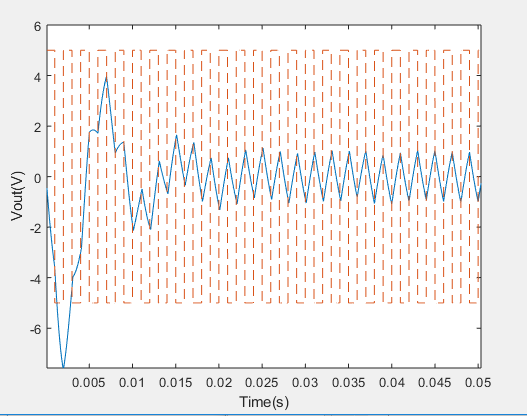
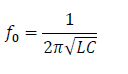


Figure 25. Output of 500Hz

When we are discussing periodic inputs, we should always consider the resonant frequency for the RLC circuit.



From the equation

We can get the resonant frequency for this circuit, 113.97Hz. Thus, when the input frequency is 5 Hz, which is much smaller than 113.97Hz, the circuit does not experience a resonance and the output at each square-wave edge has a similar behaviour to the step function. What’s more, the output could reach a steady state similarly.

When the input frequency is 100Hz, which is close to 114Hz, we could easily observe a resonant behaviour. The amplitude of the output voltage is greater than the input amplitude.

When the input frequency is 500Hz, which is much larger than 114Hz, the output is almost a sawtooth function. This is because of the fact that the output does not have enough time to reach steady state when the input square wave switches voltage.

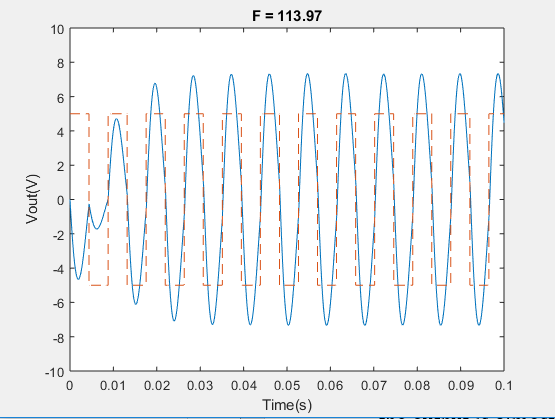
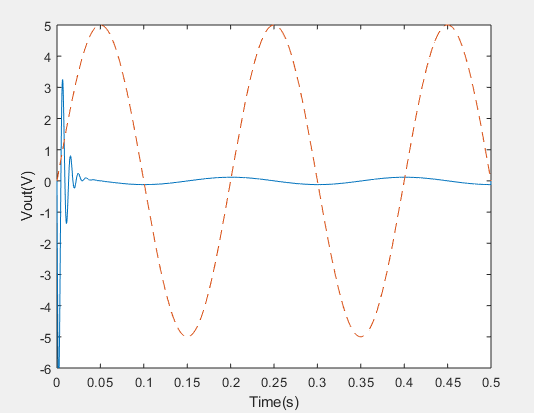
The graph below shows the behaviour of the output voltage at resonant frequency. The amplitude of Vout is at maximum value.

Figure 26. Behaviour of Output Voltage

Sine wave with different frequencies

For this part, three sine waves of 5Hz, 100H and 500Hz are tested and the pictures of each output are shown below:



500Hz

Figure 27. Output of 5Hz

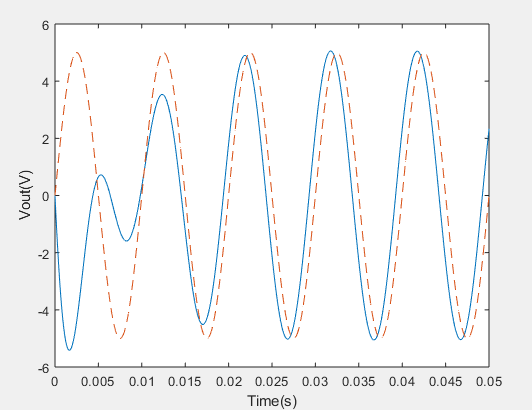


Figure 28. Output of 100Hz

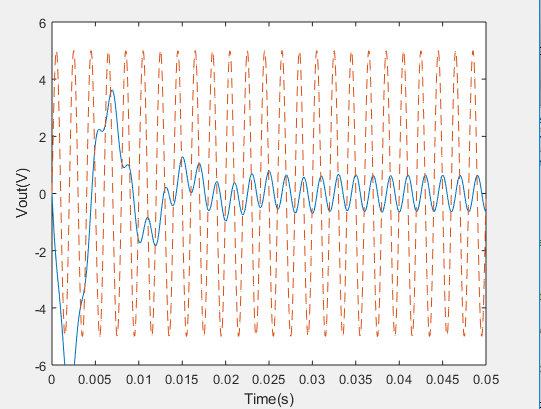


Figure 29. Output of 500Hz

As in the previous section, the resonant frequency is about 114Hz.

The first input is the sinusoid wave with frequency 5Hz which is much less than the resonance frequency. After it reaches a steady state, we observed that the output voltage amplitude is much smaller than the input voltage amplitude. However, the frequencies are the same.

When the input frequency is 100Hz (close to the resonant frequency) the resonance happens, and the output waveform becomes similar to the input signal.

When the input frequency is 500Hz, much larger than 114Hz, we see that the output is similar to a sawtooth function as the input switches voltage quickly. Nonetheless, the output at steady state is still a sinusoid wave with the same frequency of the input wave.

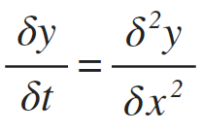
**Exercise 4**

Background for the method which is called the finite differences for the PDE

In mathematics, the finite difference methods are kind of methods which using the approximating method to solve the differential equations. According to the definition for the derivatives, we could use the difference between two adjacent values to approximate the first derivatives. Alternatively, FDM could convert the ODE/PDE to the linear equations, which is kind of typical numerical analysis.

In general, the finite difference is the method which uses approximation to get the result of derivatives. As the method is an approximation, the error from the approximation to the actual result depends on the interval of the arguments.

The problem of Exercise 4:

In this exercise we are asked to solve the 1-D heated equation which could be written as the

This gives the zero boundary conditions y(0, t) = y(1, t) = 0, and initial condition y(x, 0) = y0(x).

//the mathematical proof for the Finite difference

Firstly, we set the interval of argument of x to be h, and total N segments which means N = 1/h, we can easily generate the following results:

X0 = 0, x1=h, x2= 2h, xsn=1;

Alternatively, if we do the same thing with the argument of t, discretize time into equal segments of length k:

T0 = x, t1 = k, t2 = 2k

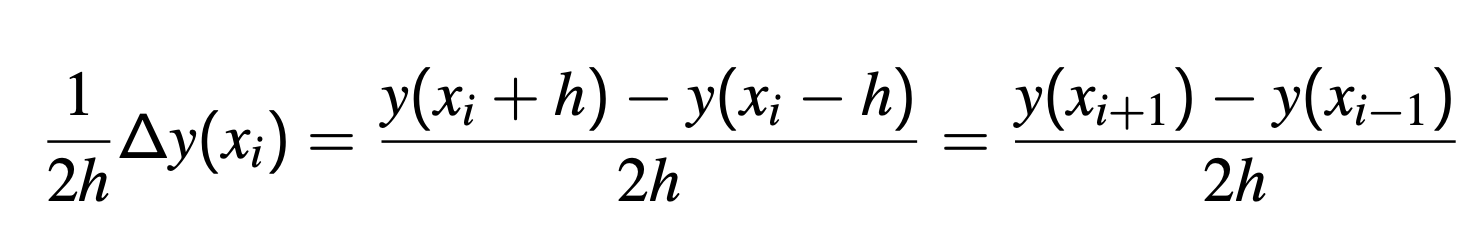
Now we need to go back to the pervious knowledge about the finite difference for ODE,

From the first year, the definition of derivative is:

图片包含 物体, 时钟

描述已自动生成

When we divide both sides by 2h, the expression would become:



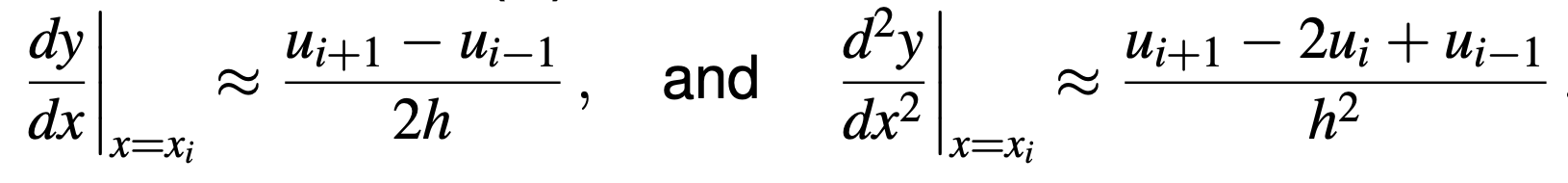
As we defined,

图片包含 物体

描述已自动生成 for j from 0 to N

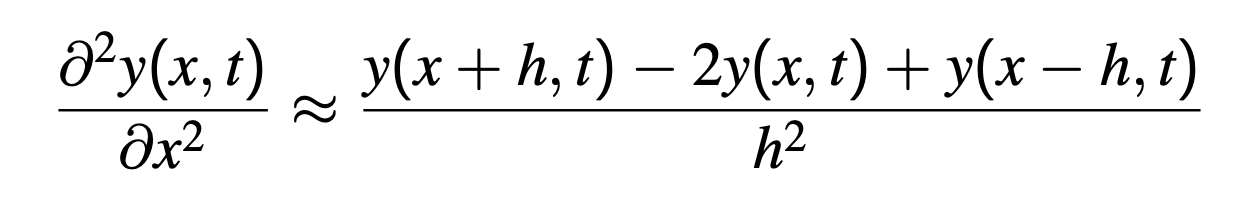
Obviously, 图片包含 物体

描述已自动生成, so the pervious expression would become



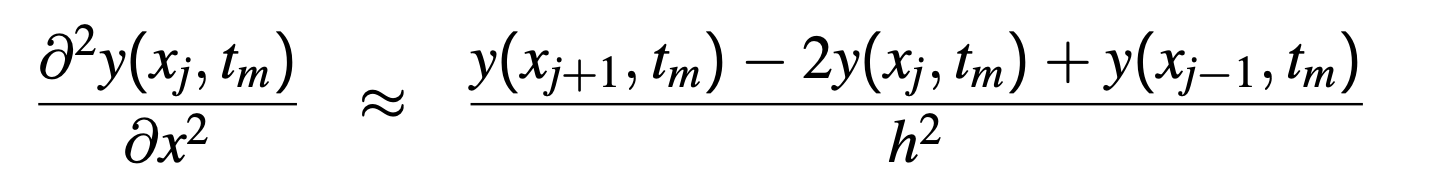
as we set ui equal to yi.

Up to now, we have gotten the expression of the second order derivatives. Eventually, we could write the expression for the partial derivatives, which is:



As a result, when we set the x = xj, and t= tm, we could derive that x+h = xj+1

x-h = xj-1; the y in expression could be substituted into:



Which is 图片包含 物体

描述已自动生成

Similarly, we could use the same principle to derive the expression of left-hand side of the heat function, which is:

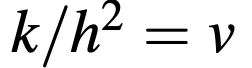
图片包含 物体

描述已自动生成

Finally, when we get these two expressions for both left hand side and right hand side, we could easily write the equation：

图片包含 物体

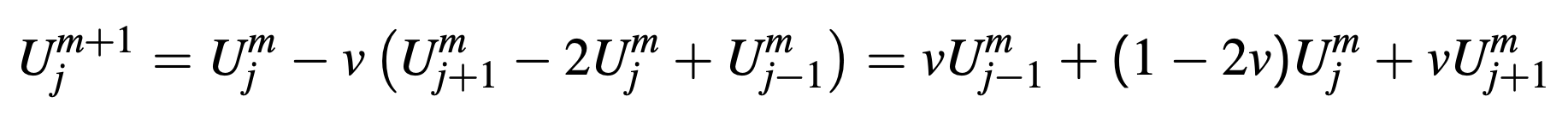
描述已自动生成

As the value of h and k are constant, we could set the , and the equation could be rearranged into:

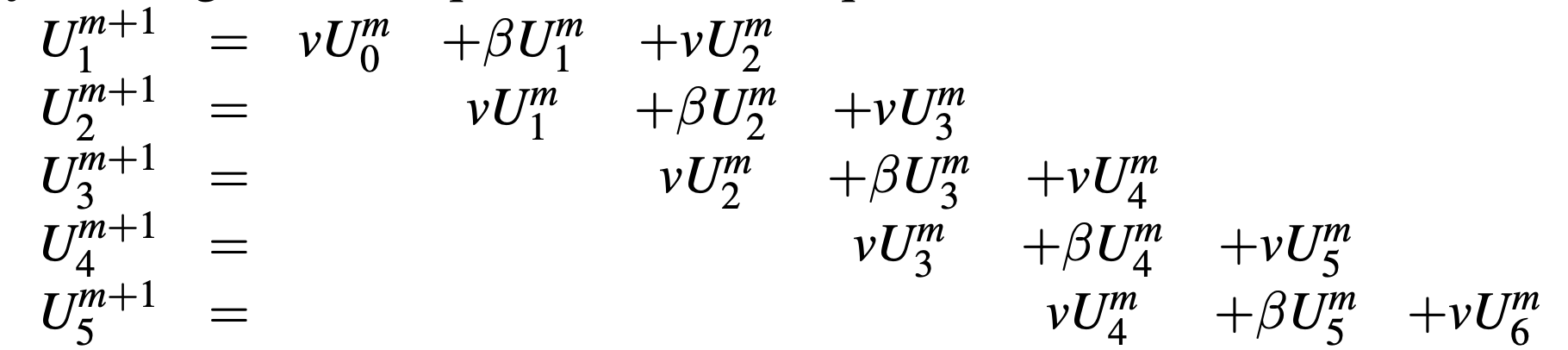
图片包含 物体

描述已自动生成

The equation could be rewritten into another form, which is:



So we could set the function to be matrix when we set N=6, the whole set of 5 function could be written into



Where 图片包含 家具

描述已自动生成.

Now the matrix equation could be written into:

图片包含 物体, 时钟, 天空

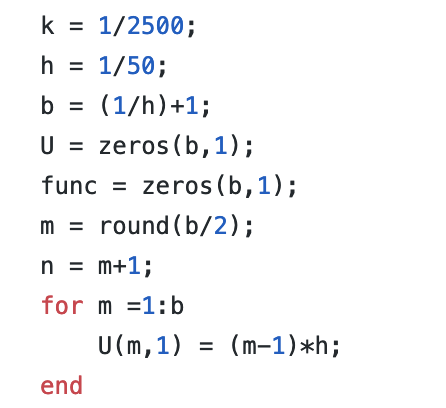
描述已自动生成

Where we could get the next value of y from the previous value.

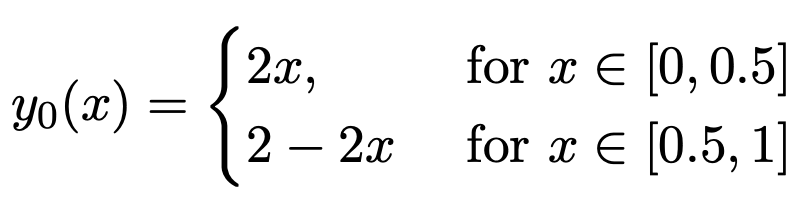
The implementation for the finite difference by code:

First of all，we set the value of the intervals for two different arguments. In the code we set N to be the segments of the equal length of x, so h=1/N. Similarly, we set k to the number of segments of the equal length of t, where 1/k would be the small increase for the argument t.

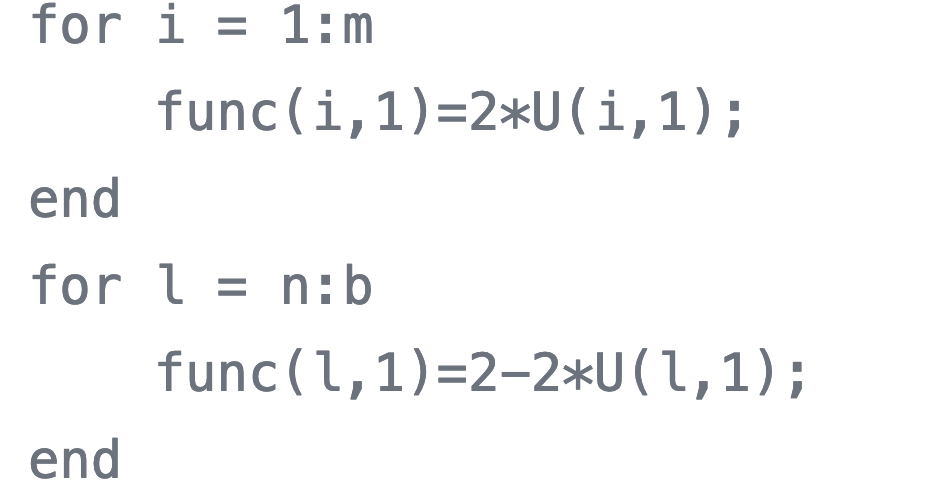
We should create the vector which is also the (N+1 x 1) matrix of the argument of x and we apply the test function to set the initial condition for the y0 where the value at x is 0. The vector called U is the set of the x:



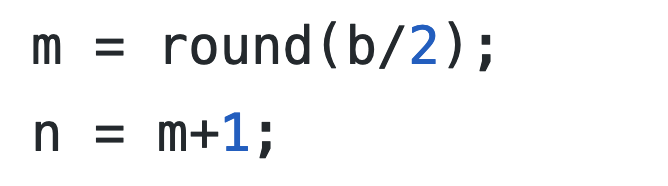
The vector called func is the set of y; we set the value of y according to the test function. For instance, when the test function is:



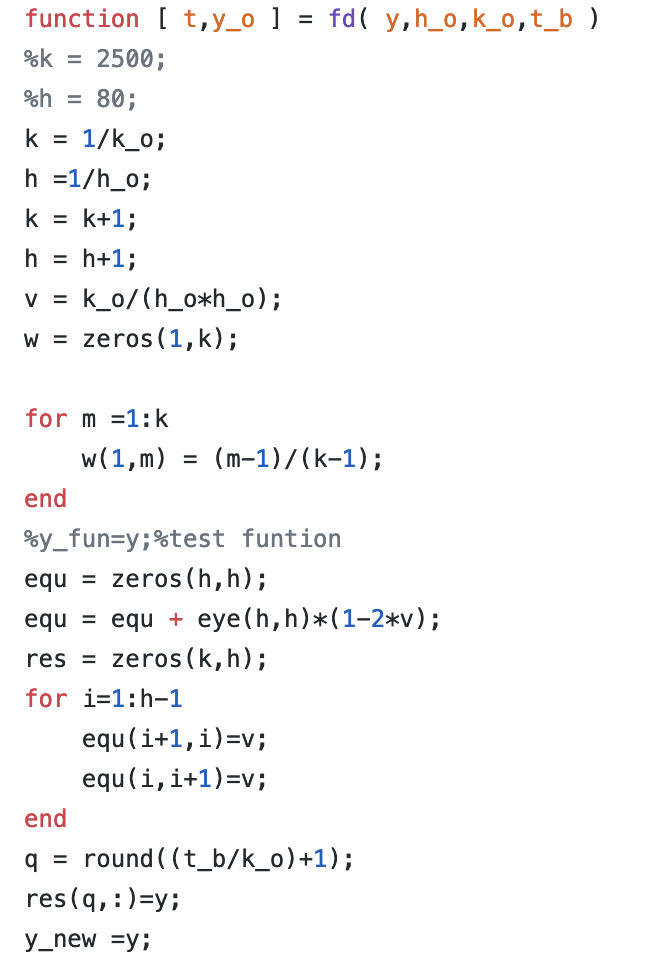
The for loop to set the value of y:



Where m and n are:

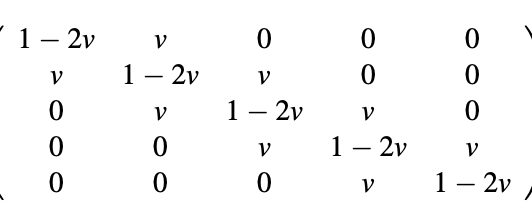


Then we apply the method we have proved perviously:



In the finite difference method, we get the value of y for each x, and each y is depending on the previous y. In order to get their value, we also need the matrix to be multiplied to the previous vector. In the code we call the matrix ‘equ’.

The matrix has a dimension of (h x h). The main goal of setting the matrix to be in this form is:

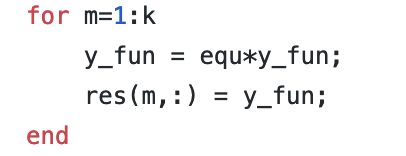


First, we set the matrix to be the diagonal matrix and then we multiply the matrix by the value of 1-2v where v is equal to k/h\*h. One of the most significant things we need to mention here is that the value of v should be less than 0.5 which could keep the tool matrix converge, if the tool matrix diverges, the multiplication for the y value would diverge, which means the elements in the matrix would become infinite. The next step is to set the elements next to diagonal to be v; the for loop is used to set each corresponding position.

The equ matrix remains constant during the calculating process, and we could use the for loop to calculate the value of y with the corresponding x. We now define the equ matrix as the tool function.

For each iteration we put the vector results into a new matrix which stores the corresponding results for the calculation and the first column would be the initial condition for y.

The code is shown below:



And eventually, we get the results for the Y and we plot the 3d graph for the y.

4.The results of testing:

As required, we set different test functions to test cases.

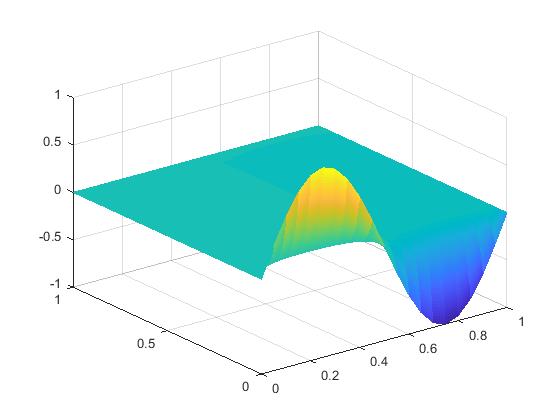
First y=sin(2\*pi\*x), for k=1/10000, h=1/50 and the 3Dgraph is shown:

Figure 30.

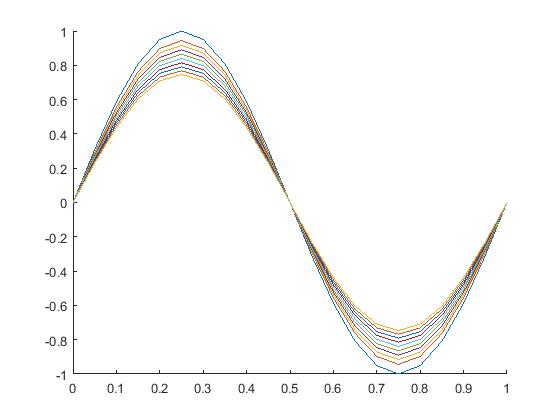
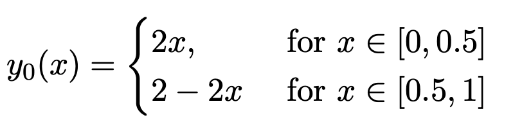
and then we take the cross-section of the graph:

Figure 31.



Secondly, the test function has been set with

Here are the 3D graph and cross-section graph:

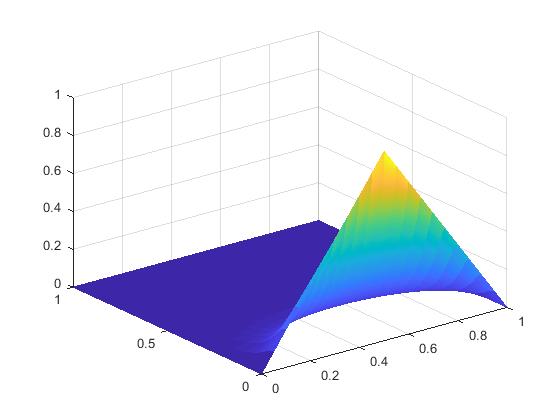


Figure 32.

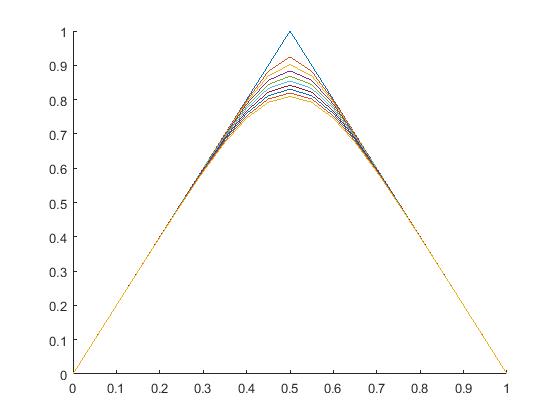


Figure 33.

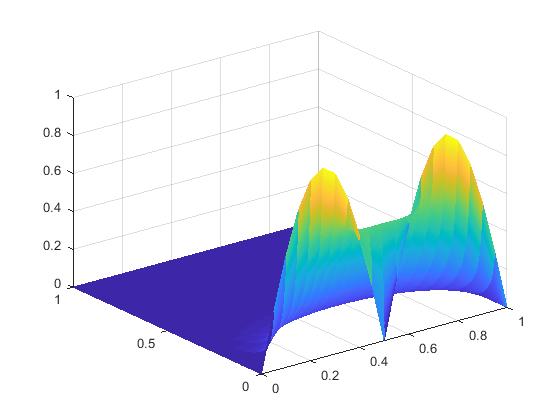
The last test function is the absolute value for sine function:

Figure 34.

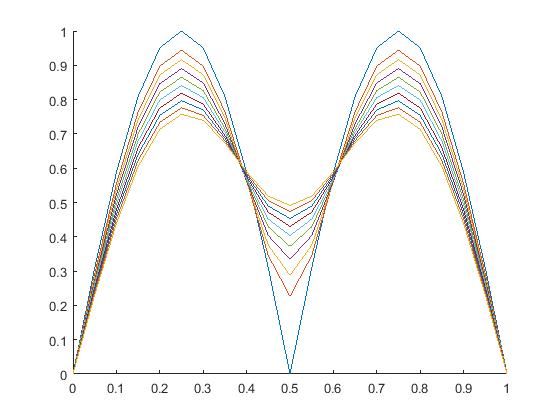
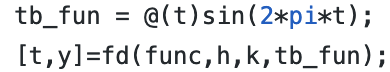


Figure 35.

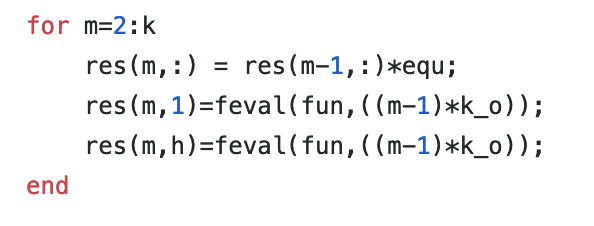
Exercise 5

The optimization for the code

In the exercise 5, we have to transfer the code into a more general case, which means that we need to set other values instead of 0 to the zeros condition, however, one of the important things that needs to be mentioned is that the two-boundary conditions should remain the same. As a result, we changed the code in the MATLAB into the following:



Where tb\_fun is the function for the boundary conditions, we set the variable in the fd function, also in the function file we should change the for loop of setting the value for the first and last elements into the value of the boundary conditions in each column:



And the first test function in this exercise is 

The results of the 3D plot graph and cross-sectional graph are shown:

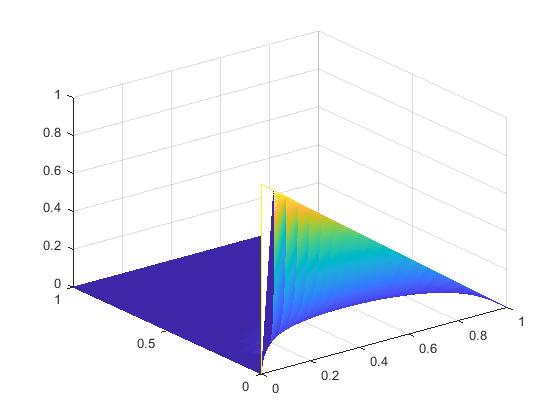


Figure 36.

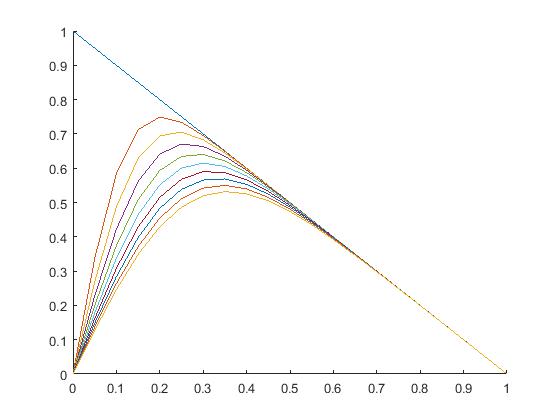


Figure 37.

图片包含 配件, 雨伞

描述已自动生成And the other test function is y0 = cos(pi\*x\*0.5), the result has shown:

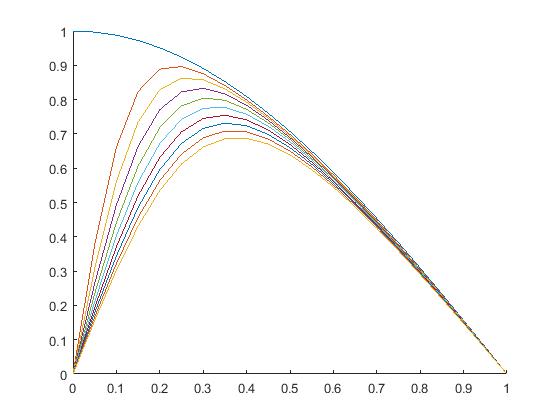
Figure 38.

Figure 39.

In conclusion, the boundary conditions are different, however, we set both of them to be zero.

Alternatively, we could also change the boundary conditions into the sine wave function or any other function which ensures that the value at x=0 equals to the value at x=1;

For instance, we could set the boundary into the sin(2\*pi\*t), which matches the conditions y = 0 at x = 1 and x = 0.

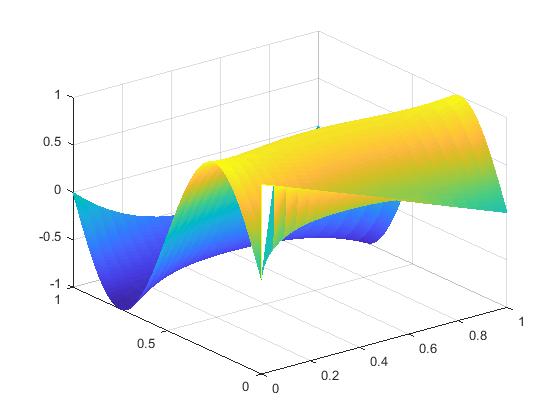
As a result, the graph is shown below and the result matches with the expectation.

Figure 40.

Exercise 6

Using Taylor’s theorem:

which approximates using its derivatives at point . Replacing with to obtain a forward approximation, and with , gives:

Which simplifies to:

Similarly, using :

The negative h gives negative odd powers and positive even powers.

For small h we can omit terms of order 3 and hence we replace it with

Subtracting equation (2) from equation (1) and rearranging to make the subject:

The LHS is the exact solution of , which is approximated by , with an error of .Hence order of error 2 , .

b)

Considering forward and backward approximation again.

Adding equation (1) and equation (2) and then making the subject of the equation:

The LHS is the exact solution of , which is approximated by , with a truncation error of . Hence order of error 2 , .

*Reference:*

<https://en.wikipedia.org/wiki/Finite_difference_method>

https://bb.imperial.ac.uk/bbcswebdav/pid-1506664-dt-content-rid-4914901\_1/courses/DSS-EE2\_08-18\_19/Slides\_numerics%281%29.pdf