Imperial College London

Coursework Group 18

Imperial College London

Department of Electrical and Electronic Engineering

Mathematics II: Numerical Analysis

Authors
Erk Bayar (01060692), Henry Eshbaugh (01075992), Koral Hassan (01096803), Paul Streli (01103106), Chengdong Sun (01115007)

Date: March 15, 2017

Contents

List of Figures

1. Exercise 1: RL circuit

1.1. The RL circuit

The RL circuit is made up of a resistor connected in series with an inductor; therefore the resistor and inductor share the same current $i_L(t)$. The circuit shown in Figure 1 is a high pass filter, which are usually built using capacitors as they are more easily manufactured and deviate less from their ideal component models.

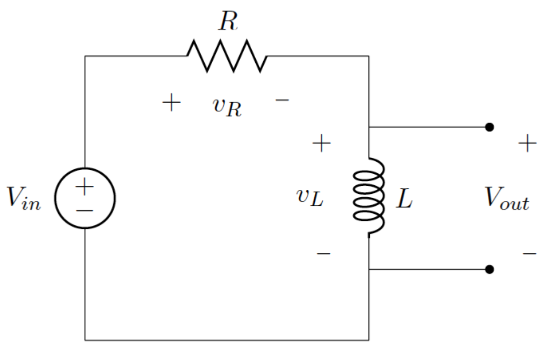


Figure 1: RL-Circuit

The RL circuit of interest can be described by the following equation:

$$V_L(t) + V_R(t) = V_{in}(t)$$

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

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

The state of the circuit is described by the inductor current $i_L(t)$, and the input voltage $V_{\text{in}}(t)$.

The output voltage V_{out} of the circuit is the voltage across the inductor L, which could be obtained by:

$$V_{\text{out}} = V_{\text{in}}(t) - Ri_L(t) \tag{3}$$

The values of the components are given as:

$$R = 0.5\Omega \quad L = 1.5\,\text{mH}$$

There is an initial condition stating that the current through the inductor at time $t = 0$ is $i_L(0) = 0$. This implies that there is no voltage across the resistor at $t = 0$.

1.2. Implementation of second-order Runge-Kutta methods

Referring to the notation used in the Mathematics II course, **heuns.m** (Appendix A.1), **midpoint.m** (Appendix A.2), and **ralston.m** (Appendix A.3) are MATLAB functions that calculate y_i+1, x_i+1 and t_i+1 for previously-computed values y_i, x_i and t_i, hence yielding an approximate numerical solution to an ordinary differential equation. Heun's method, the midpoint method, and Ralston's method are all second order Runge-Kutta methods, with different scaling factors.

$$
\begin{aligned}
y_{i+1} &= y_i + h\phi(x_i, y_i, h) \\
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + p_1 h, y_i + q_{11} k_1 h) \\
\phi &= a k_1 + b k_2
\end{aligned}
$$

The second-order Runge-Kutta method is implemented with the following code.

```
func = @(t,iL) (feval(Vin,t) - R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)

N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
% set up arrays
t = zeros(1,N);
iL=zeros(1,N);
Vout=zeros(1,N); %set up arrays
Vout(1) = feval(Vin,ti);% calculate initial value of Vout
t(1)=ti;
iL(1)=iL0;%set initial values of t_0, and iL at t_0
```

The script starts by defining a function for evaluating $\frac{\mathrm{d}}{\mathrm{d}t}i_L(t)$ as given in Equation (1), then calculates the number of steps N. Three empty arrays of length N are created to store t, i_L and V_{out}.

```
for j=1:N-1 % loop for N steps
    ttemp = t(j);
    iLtemp = iL(j); %temporary names
    grad1 = feval(func, ttemp,iLtemp);  % gradient at t, iL
    iLp = iLtemp + q11*h*grad1; % calculate iL predictor
    grad2 = feval(func, ttemp+p1*h,iLp); % gradient at t+p1*h, iL+q11k1h
    iL(j+1) = iLtemp + h*(a*grad1 + b*grad2); % next value of iL ...
        calculated from previous values of t,iL
    t(j+1) = ttemp+h; % increase t by stepsize
    Vout(j+1) = feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
end
```

For each segment of step-size h, the method firstly evaluates the gradient at (x_i, y_i), k_1, which is then fed into the calculation of k_2, evaluated at different amount of increments given by $x + p_1 h$ and $y + q_{11} k_1 h$. After k_1 and k_2 have been calculated, the increment is computed and the next value of y is calculated from previous values of x and y, using different scaling factors a and b, which depend on the method employed. V_{out} is then calculated using Equation (2) and stored in the V_{out} array. This is repeated N times in order to estimate the solution for an ordinary differential equation.

$V_{in}, i_{L0}, h, R, L, t_i$, and t_f are passed to **heuns.m**, **midpoint.m**, and **ralston.m**, where t and i_L are the x_i and y_i (in the sense that the solution of the ODE maps time to voltage), h is the step-size, t_i and t_f are the starting time and final time, R is the value of the resistor, and L is the value of the inductor.

The above code is used in common for all **heuns.m**, **midpoint.m**, and **ralston.m**, with a change of scaling factors p_1, q_{11}, a, b.

From the Mathematics II course notes, the scaling factors a, b, p_1, and q_{11} are chosen so that they agree with the following equations obtained from Taylor series:

$$\begin{aligned} a + b &= 1 \\ bp_1 &= \frac{1}{2} \\ bq_{11} &= \frac{1}{2} \end{aligned}$$

For **Heun's method**,

$$a = \frac{1}{2}, b = \frac{1}{2}, p_1 = 1, q_{11} = 1$$

For the **Midpoint method**,

$$a = 0, b = 1, p_1 = \frac{1}{2}, q_{11} = \frac{1}{2}$$

For **Ralston's method** [4],

$$a = \frac{1}{3}, b = \frac{2}{3}, p_1 = \frac{3}{4}, q_{11} = \frac{3}{4}$$

Heuns_script.m (Appendix A.4), **Midpoint_script.m** (Appendix A.5), **Ralstons_script.m** (Appendix A.6) are constructed as follows:

```
%set up initial conditions
iL0=0;
ti=0;

%define component values
R=0.5;
L=0.0015;

Vina = 5.5;
Vin=@(t) Vina*exp(0); %define input signal as function of time
figure
Vout = feval(Vin,ti)-R*iL0;
subplot(3,4,nn);
plot(ti,Vout); % plot initial condition
```

The code starts with defining the initial conditions, defining the component values, and defining input voltage. Then plots the initial condition.

```
h=10e-7; % set step-size
tf=0.04; % set final value of t
[t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
    using heun's method
plot(t,Vout); % plot Vout against t
title('Heuns Vin=5.5V')
xlabel('Time [t]') % x-axis label
ylabel('Vout [V]') % y-axis label
```

Next, the step size and final value of t is chosen. The parameters are passed to scripts implementing each method, which in turn return output voltage and time arrays. Then calls heuns.m/midpoint.m/ralston.m depends on what method to use and they return arrays of output voltage and time. Finally it plots output voltage against time.

The code above only shows Heun's method with the step signal input as an example. The MATLAB code for the other methods can be found in Appendix (Appendix A.2, A.3). It will plot the output voltage obtained using **heuns.m**, **midpoint.m**, or **ralston.m** against time for a stated step-size and interval size.

1.3. Analytical solution

The RL-circuit can be solved analytically. This can be done using the Laplace transform.

Starting from the second order differential equation describing the RL-circuit:

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

The Laplace transform is applied:

$$L(\overline{i_L}(s) - i_L(0)) + R\overline{i_L}(s) = \overline{V_{in}}(s)$$

With algebraic manipulation, an expression for $\overline{i_L}(s)$ can be found:

$$\overline{i_L}(s) = \frac{\overline{V_{in}}(s) + Li_L(0)}{Ls + R} \tag{4}$$

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

$$\overline{V_{out}}(s) = \frac{\overline{V_{in}}(s) - R(\overline{V_{in}}(s) + Li_L(0))}{Ls + R} \tag{5}$$

In the following sections, the inverse Laplace transforms of the preceding two equations will be obtained for particular component values.

1.3.1. Step signal with amplitude $\overline{V_{in}} = 5.5V$

The first input is a step signal with amplitude $\overline{V_{in}} = 5.5V$. Because the observed time interval does not include the negative time axis, the step signal looks like a DC voltage with amplitude 5.5V. These conditions are implemented by the following MATLAB code.

```
Vina = 5.5;
Vin=@(t) Vina*exp(0);    %define input signal as function of time
```

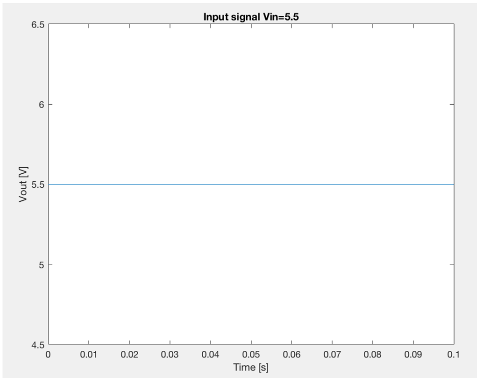
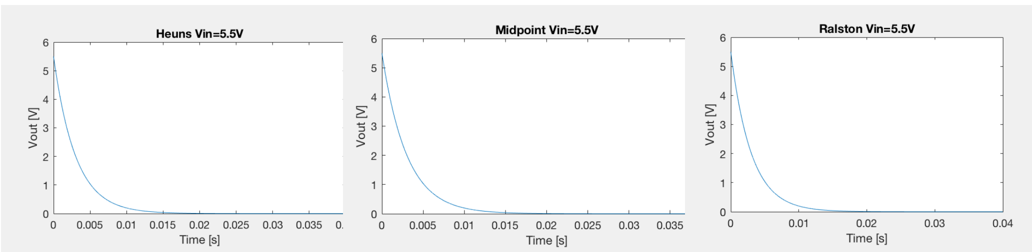


Figure 2: $\overline{V_{in}} = 5.5$

The input jumps up to 5.5V at $t = 0$ instantaneously and remains at this level for all positive t.

Since the inductor will act as a short circuit for DC-voltages, eventually all the voltage will be dropped across the resistor and no voltage will be dropped across the inductor. Therefore the steady state value of V_{out} should be 0V.

After executing the MATLAB script, several plots are obtained.



(a) Heun's method (b) Midpoint method (c) Ralston's method

Figure 3: V_{out} against time when $\overline{V_{in}} = 5.5V$

This displays the exponential characteristic of a high-pass filter step response, as expected. In order to find an analytical solution, firstly, the Laplace transform of the step signal is obtained:

$$\overline{V_{in}}(s) = \frac{5}{s}$$

Then $\overline{V_{in}}(s)$, the initial conditions and the components values are plugged into Equation (4). Using partial fraction decomposition, the inverse Laplace transform is found.

$$\overline{i_L}(s) = \frac{\overline{V_{in}}(s) + Li_L(0)}{Ls + R}$$

$$\overline{i_L}(s) = \frac{5.5}{0.0015s^2 + 0.5s}$$

$$\overline{i_L}(s) = \frac{11}{x} - \frac{11}{x + \frac{11}{x+\frac{1000}{3}}}$$

$$i_L(t) = 11(1 - e^{-\frac{1000}{3}t})u(t)$$

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

$$V_{out}(t) = 5.5u(t) - 5.5(1 - e^{-\frac{1000}{3}t})u(t)$$

$$V_{out}(t) = 5.5(e^{-\frac{1000}{3}t})u(t) \tag{6}$$

When the input is a step signal the output voltage decays exponentially with time described by the following equation. $V_{out} = \overline{V_{in}}e^{-\frac{R}{L}t}$ $V_{out}(t) = 5.5(e^{-\frac{1000}{3}t})$ This is

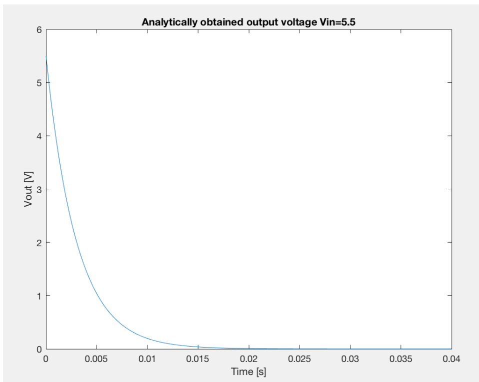


Figure 4: Analytically obtained output voltage when $\overline{V_{in}} = 5.5V$

because the current i_L equals zero initially. The current flowing through the inductor will increase by $i_L = \frac{1}{L}\int_{t_0}^{t} V_L(t)dt + i_L(0)$. As V_{out} is given by $V_{in}(t) - Ri_L(t)$, initially $V_{out} = V_{in}$. Since i_L gets bigger as t gets larger, V_{out} decays to 0.

The shape of the output voltage is almost identical between the three methods and they all look similar to the analytically obtained solution. This indicates that **heuns.m**, **midpoint.m**, and **ralston.m** are functioning correctly.

1.3.2. Impulse signal and decay

$V_{in} = \overline{V_{in}}e^{-\frac{t^2}{\tau}}$ with $\overline{V_{in}} = 3.5V$ and $\tau = 160(\mu s)^2$

This is implemented as the following MATLAB function:

```
Vina = 3.5;
tau = 160e-12;
Vin=@(t) Vina*exp(-t^2/tau);
```

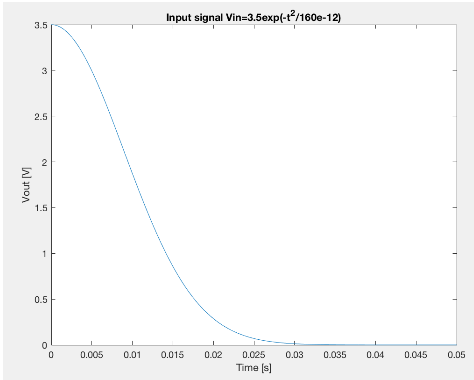
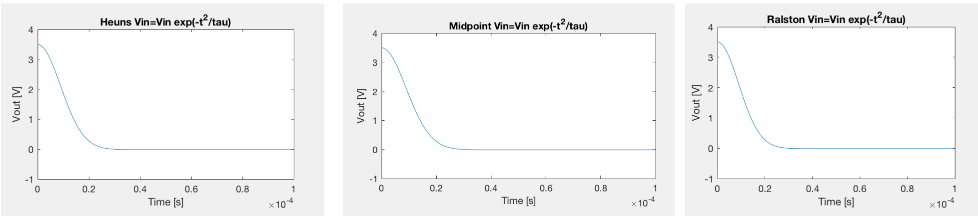


Figure 5: $V_{in} = 3.5e^{-\frac{t^2}{160(\mu)^2}}$

At $t = 0$, the input jumps to 3.5V and quickly decays to the steady state value 0V. Initially, there is no current flowing in the circuit and no voltage is dropped across the resistor. All the input voltage is dropped across the inductor. Eventually, the input voltage value approaches 0V and the change in the input signal will decay. Thus, the inductor will act like a short circuit and the output voltage will reach 0V.



(a) Heun's method (b) Midpoint method (c) Ralston's method

Figure 6: V_{out} against time when $V_{in} = 3.5e^{-\frac{t^2}{160(\mu)^2}}$

The output voltage in Figure 24 dropped to a very slightly negative value and subsequently approaches 0V.

$V_{in} = \overline{V_{in}}e{-}\frac{t}{\tau}$ with $\overline{V_{in}} = 3.5V$ and $\tau = 160\mu s$

This is implemented in the following MATLAB:

```
Vina = 3.5;
tau = 160e-6;
Vin=@(t) Vina*exp(-t/tau);
```

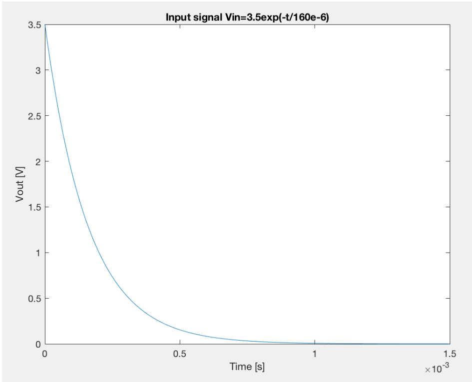
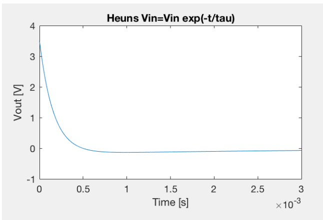
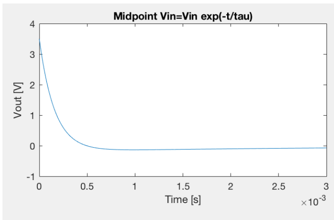


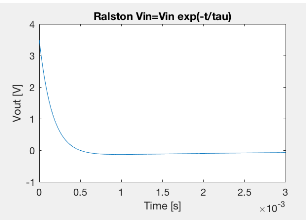
Figure 7: $V_{in} = 3.5e^{-\frac{t}{160\mu}}$



(a) Heun's method



(b) Midpoint method



(c) Ralston's method

Figure 8: V_{out} against time when $V_{in} = 3.5e^{-\frac{t}{160\mu}}$

In order to find an analytical solution, first, the Laplace transform of the step signal is obtained:

$$V_{in}(t) = 3.5e^{-\frac{t}{160\mu}}$$

$$\overline{V_{in}}(s) = \frac{3.5}{s + \frac{1}{160\mu}}$$

Then $\overline{V_{in}}(s)$, the initial conditions and the components values are plugged into Equation(4). Using partial fraction decomposition, we find the inverse Laplace transform.

$$\overline{i_L}(s) = \frac{\overline{V_{in}}(s) + Li_L(0)}{Ls + R}$$

$$\overline{i_L}(s) = \frac{3.5}{0.0015s^2 + \frac{79}{8}s + 3125}$$

$$\overline{i_L}(s) = \frac{0.394366}{x + \frac{100}{3}} - \frac{0.394366}{x + 6250}$$

$$i_L(t) = 0.394366(e^{-\frac{100}{3}t} - e^{-\frac{1}{160\mu}t})$$

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

$$V_{out}(t) = 3.697183e^{-\frac{1}{160\mu}t} - 0.197183e^{-\frac{100}{3}t} \tag{7}$$

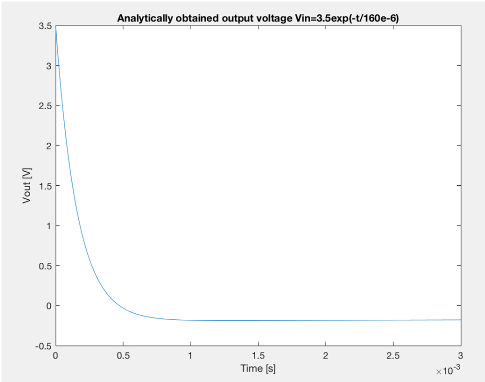


Figure 9: Analytically obtained output voltage when $V_{in} = 3.5e^{-\frac{t}{160\mu}}$

The analytically-obtained output in Figure 9 has a similar shape as the results obtained from the numerical methods shown in Figure 8.

1.3.3. Sine wave input

The sine wave input is implemented with the following code. The code and input graph are only shown for sinusoidal input with amplitude 4.5 and period $20\mu s$.

```
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sin(2*pi*t/T);
```

The value of T is changed to give sinusoids with different periods.

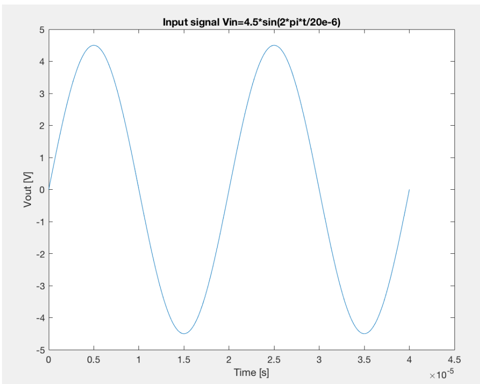


Figure 10: $V_{in} = 4.5sin(\frac{2\pi t}{20\mu})$

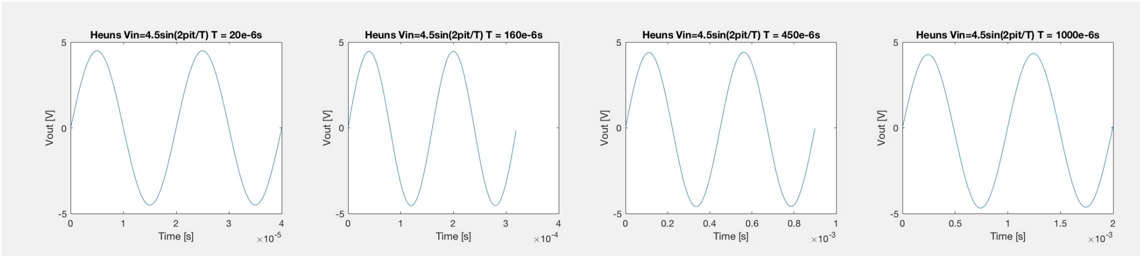


Figure 11: Heun's method $V_{in} = 4.5sin(\frac{2\pi t}{T})$

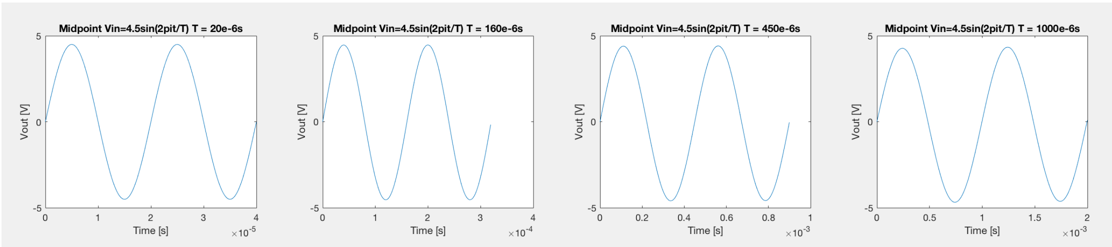


Figure 12: Midpoint method $V_{in} = 4.5sin(\frac{2\pi t}{T})$

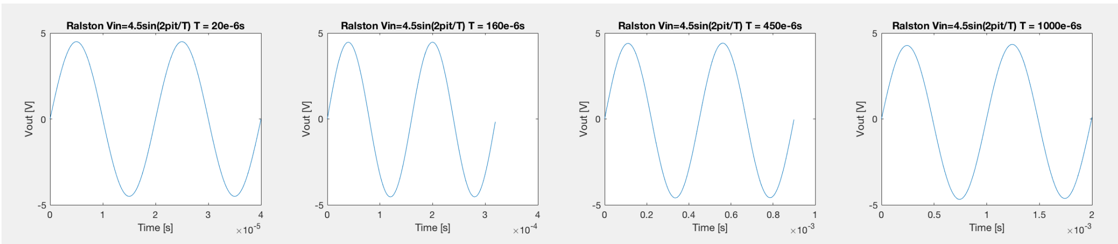


Figure 13: Heun's method $V_{in} = 4.5sin(\frac{2\pi t}{T})$

As expected, the input signal shows a sine wave with amplitude 4.5 V and a period of 20μs.

The steady state response of the output sine wave is expected to be shaped by the transfer function.

$$H(j\omega) = \frac{V_{out}}{V_{in}}$$
$$H(j\omega) = \frac{j\omega L}{R + j\omega L} \tag{8}$$

For sine wave input with amplitude 4.5V and period 20μs, $H(j\frac{2\pi}{20\mu})$ is evaluated with the corresponding component values. Then, the gain and the phase-shift of the transfer function are obtained:

$$H(j\frac{2\pi}{20\mu}) = \frac{j(\frac{2\pi}{20\times10^{-6}})1.5\times10^{-3}}{0.5 + j(\frac{2\pi}{20\times10^{-6}})(1.5\times10^{-3})} = 1.0 + 0.00106i$$

$$|H(j\frac{2\pi}{20\mu})| = 1.000$$

$$\angle H(j\frac{2\pi}{20\mu}) = 0.0608°$$

For output voltage, we expect a sine wave with $\overline{V} = 4.5 \times 1 = 4.5V$ and with a positive phase shift of 0.0608°.

Next, a sine wave input with amplitude 4.5V and period 160μs:

$$H(j\frac{2\pi}{160\mu}) = 1.0 + 0.00849i$$

$$|H(j\frac{2\pi}{160\mu})| = 1.000$$

$$\angle H(j\frac{2\pi}{160\mu}) = 0.486°$$

For output voltage, we expect a sine wave with $\overline{V} = 4.5 \times 1 = 4.5V$ and with a positive phase shift of 0.486°.

Following the same method, a sine wave input with amplitude 4.5V and period 450μs:

$$H(j\frac{2\pi}{450\mu}) = 0.999 + 0.0239i$$

$$|H(j\frac{2\pi}{450\mu})| = 0.9997$$

$$\angle H(j\frac{2\pi}{450\mu}) = 1.368°$$

For output voltage, expect a sine wave with $\overline{V} = 4.5 \times 0.9997 = 4.499V$ and with a positive phase shift of 0.486°.

For sine wave input with amplitude 4.5V and period 1000μs:

$$H(j\frac{2\pi}{1000\mu}) = 0.997 + 0.0529i$$

$$|H(j\frac{2\pi}{1000\mu})| = 0.9986$$

$$\angle H(j\frac{2\pi}{1000\mu}) = 3.037°$$

For output voltage, we expect a sine wave with $\overline{V} = 4.5 \times 0.9986 = 4.49V$ and with a positive phase shift of 3.037°.

These results could be observed in Figures 11, 12 and 13. The decrease in amplitude and phase lag are sufficiently small that the output voltage is essentially equal to the input voltage.

1.3.4. Square wave input

The square wave input is implemented with the following code:

```
Vina = 4.5;
T= 20e-6;
Vin=@(t) Vina*square(2*pi*t/T);
```

The value of T is changed to yield square waves with different periods.

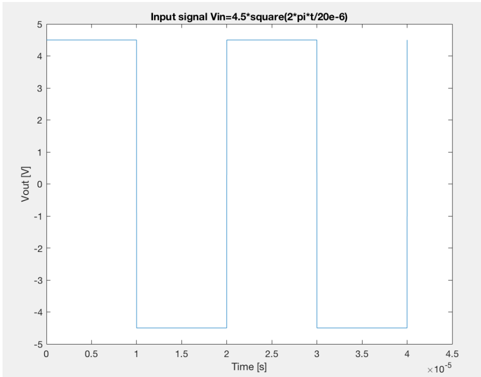


Figure 14: $V_{in} = 4.5\text{square}(\frac{2\pi t}{20\mu})$

The output voltage against time graph is plotted:

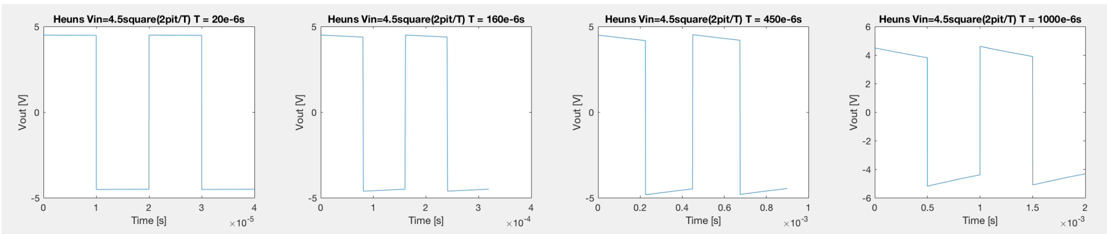


Figure 15: Heun's method $V_{in} = 4.5\text{square}(\frac{2\pi t}{T})$

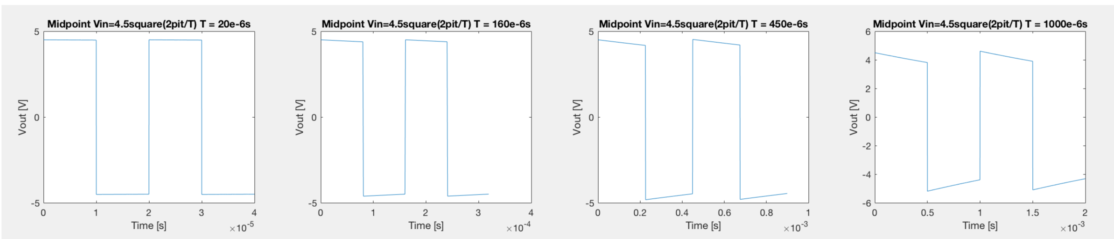


Figure 16: Midpoint method $V_{in} = 4.5\text{square}(\frac{2\pi t}{T})$

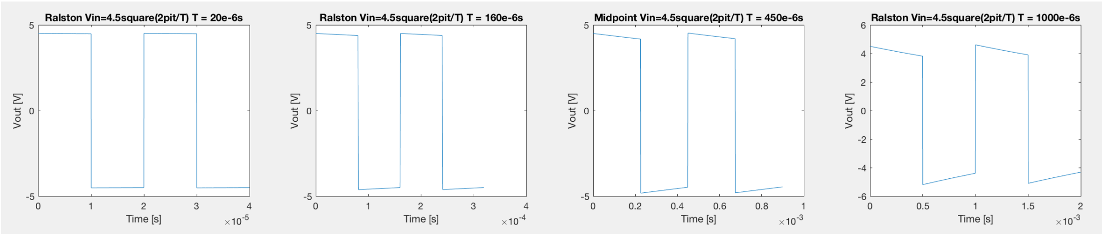


Figure 17: Ralston's method $V_{in} = 4.5\text{square}(\frac{2\pi t}{T})$

When a small-period square wave is the input, the output looks identical to the input waveform. However, as the period of the square wave increases, the output changes shape and the slope becomes non-zero. This is because for a step input, the output decays to zero exponentially, but since the period of the input is very small, the gradient appears linear. As the period increases, V_{out} has more time to decrease and the change becomes better observable. Hence, there appears to be a greater gradient for lower frequencies.

The time constant is given by $\tau = \frac{L}{R} = 30ms$ for this RL-circuit. For a small period like 20µs, the period is much smaller than the time constant, so the gradient is essentially vanishing. At higher periods like 1000µs, the value is much closer to the time constant, and so the exponential decay is more readily observed.

1.3.5. Sawtooth wave input

The sawtooth wave input is implemented with the following code:

```
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T);
```

The value of T is changed to simulate square waves with different periods.

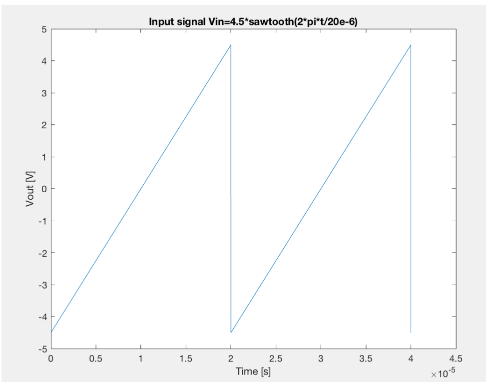


Figure 18: $V_{in} = 4.5 \text{ sawtooth} \left(\frac{2\pi t}{20\mu}\right)$

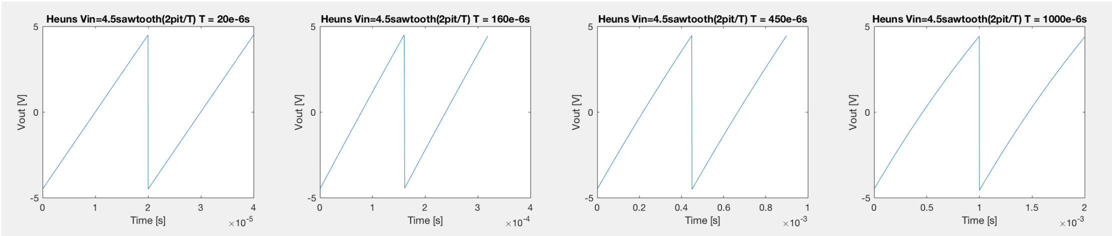


Figure 19: Heun's method $V_{in} = 4.5$ sawtooth$(\frac{2\pi t}{T})$

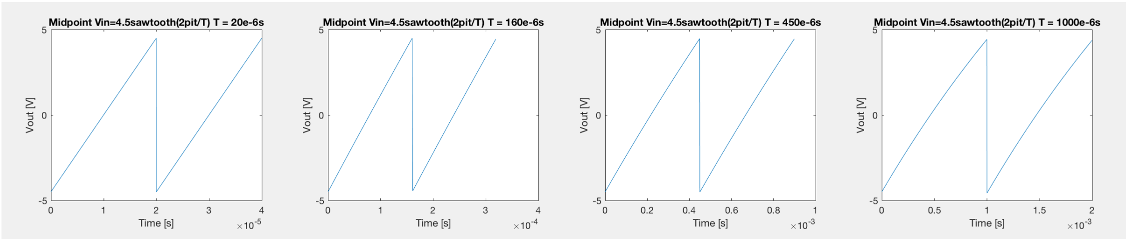


Figure 20: Midpoint method $V_{in} = 4.5$ sawtooth$(\frac{2\pi t}{T})$

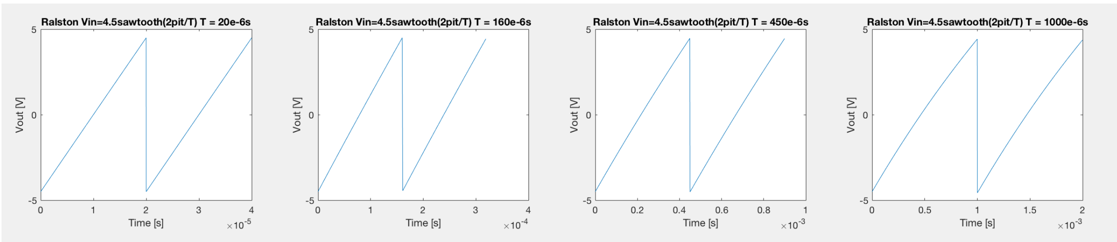


Figure 21: Heun's method $V_{in} = 4.5$ sawtooth$(\frac{2\pi t}{T})$

When a sawtooth wave with very small period (compared to τ) is fed into the circuit, the output looks identical to the input waveform. For sawtooth wave inputs with longer periods, the output starts to slightly curve. This happens as the period of the input becomes comparable the time constant of the circuit, resulting in the circuit trying to restore its "natural state" of $V_L = 0$.

2. Exercise 2: Error analysis

$$V_{in} = \overline{V_{in}} cos(\frac{2\pi}{T}t)$$

Where $\overline{V_{in}} = 6V$, $T = 150\mu s$

2.1. Exact solution of the ODE

In order to calculate the exact solution of the ODE, start with the ODE that describes RL-circuit.

$$\begin{aligned}
L\frac{d}{dt}i_L(t) + Ri_L(t) &= V_{in}(t) \\
\frac{d}{dt}i_L(t) + \frac{R}{L}i_L(t) &= \frac{1}{L}V_{in}(t) \\
\mu(t) = e^{\int \frac{R}{L}dt} &= e^{\frac{R}{L}t} \\
e^{\frac{R}{L}t}\frac{d}{dt}i_L(t) + e^{\frac{R}{L}t}\frac{R}{L}i_L(t) &= \frac{1}{L}e^{\frac{R}{L}t}V_{in}(t) \\
\frac{d}{dt}(e^{\frac{R}{L}t}i_L(t)) &= \frac{1}{L}e^{\frac{R}{L}t}V_{in}(t) \\
&= \frac{1}{L}e^{\frac{R}{L}t}Acos(\frac{2\pi}{T}t) \\
e^{\frac{R}{L}t}i_L(t) &= \int \frac{1}{L}e^{\frac{R}{L}t}Acos(\frac{2\pi}{T}t)dt + c \\
i_L(t) &= e^{-\frac{R}{L}t}(\frac{1}{L}A\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt + c)
\end{aligned} \tag{9}$$

Calculate $\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt$ using integration by parts.

$$\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt = e^{\frac{R}{L}t}\frac{T}{2\pi}sin(\frac{2\pi}{T}t) - \frac{RT}{2\pi L}\int e^{\frac{R}{L}t}sin(\frac{2\pi}{T}t)dt$$

Calculate $\int e^{\frac{R}{L}t}sin(\frac{2\pi}{T}t)dt$ using integration by parts.

$$\int e^{\frac{R}{L}t}sin(\frac{2\pi}{T}t)dt = -e^{\frac{R}{L}t}\frac{T}{2\pi}cos(\frac{2\pi}{T}t) + \frac{RT}{2\pi L}\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt$$

Plug the expression for $\int e^{\frac{R}{L}t}sin(\frac{2\pi}{T}t)dt$ in the $\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt$ and rearrange.

$$\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt = e^{\frac{R}{L}t}\frac{T}{2\pi}sin(\frac{2\pi}{T}t)dt + \frac{RT}{2\pi L}e^{\frac{R}{L}t}\frac{T}{2\pi}cos(\frac{2\pi}{T}t) - \frac{R^2T^2}{4\pi^2L^2}\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt$$

$$\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt = \frac{4\pi^2L^2(e^{\frac{R}{L}t}\frac{T}{2\pi}sin(\frac{2\pi}{T}t)dt + \frac{RT^2}{4\pi^2L}e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t))}{4\pi^2L^2 + R^2T^2}$$

$$= \frac{2\pi TL^2 e^{\frac{R}{L}t} sin(\frac{2\pi}{T}t)dt + LRT^2 cos(\frac{2\pi}{T}t)}{4\pi^2L^2 + R^2T^2}$$

Then plug back to the i_L Equation (9),

$$i_L(t) = e^{-\frac{R}{L}t}(\frac{1}{L}A\int e^{\frac{R}{L}t}cos(\frac{2\pi}{T}t)dt + c)$$

$$= \frac{2A\pi TLsin(\frac{2\pi}{T}t) + ART^2cos(\frac{2\pi}{T}t)}{4\pi^2L^2 + R^2T^2} + ce^{-\frac{R}{L}t}$$

Finally calculate c from the initial condition given.

$$i_L(0) = \frac{ART^2cos(\frac{2\pi}{T}t)}{4\pi^2L^2 + R^2T^2} + c = 0$$

$$c = -\frac{ART^2}{4\pi^2L^2 + R^2T^2}$$

$$i_L(t) = \frac{2A\pi TLsin(\frac{2\pi}{T}t) + ART^2cos(\frac{2\pi}{T}t)}{4\pi^2L^2 + R^2T^2} - \frac{ART^2}{4\pi^2L^2 + R^2T^2}e^{-\frac{R}{L}t} \tag{10}$$

2.2. Error analysis

error_script.m(Appendix **??**) starts by defining the initial conditions, and constant values.

```
iL0=0; % set initial values
ti=0;
R=0.5; % set constant values
L=0.0015;
A=6;
Vina = 6;
T= 150e-6;
Vin=@(t) Vina*cos(2*pi*t/T); % set Vin
```

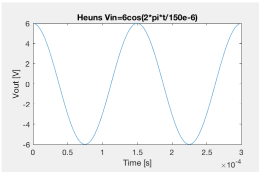
Next, plot the output voltage using the second-order Runge-Kutta methods.

```
tf=0.0003; % set final value of t
h=10e-7; % set step-size
[t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf);
figure
subplot(3,2,1);
plot(t,Vout); % plot heuns Vout against t
title('Heuns Vin=6cos(2*pi*t/150e-6)')
xlabel('Time [s]') % x-axis label
ylabel('Vout [V]') % y-axis label
```

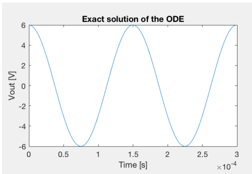
Then plot the exact solution that has been calculated previously in Equation (10).

```
iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout for exact iL
subplot(3,2,2);
plot(t,Vout_exact); %plot exact solution
    title('Exact solution of the ODE')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
error=abs(Vout_exact-Vout); % calculate maximum error over range of x
subplot(3,2,[3,4]);
plot(t,error); % plot error against t
    title('Error against time')
    xlabel('Time [s]') % x-axis label
    ylabel('Error [V]') % y-axis label
```

A step size of $h = 10^{-7}$ was used.

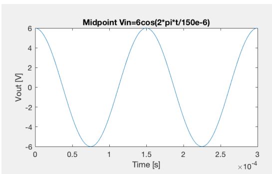


(a) Heun's method

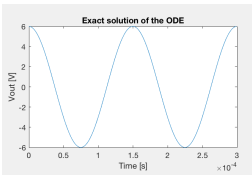


(b) Exact solution

Figure 22: V_{out} against time, $V_{in} = 6cos(\frac{2\pi}{150\mu}t)$

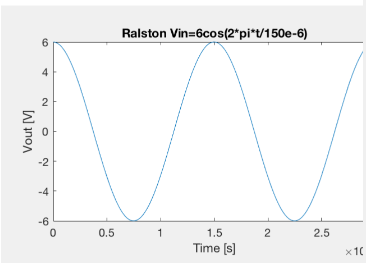


(a) Midpoint method

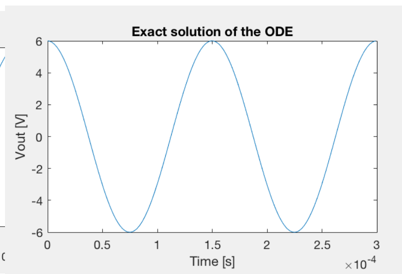


(b) Exact solution

Figure 23: V_{out} against time, $V_{in} = 6cos(\frac{2\pi}{150\mu}t)$



(a) Ralston's method



(b) Exact solution

Figure 24: V_{out} against time, $V_{in} = 6cos(\frac{2\pi}{150\mu}t)$

Comparing the shapes of the graphs obtained from the three methods and the exact ODE solution, we find that they have the same shape, which proves that the three methods are functioning correctly and predicting the solution of the ODE.

```
c=((A*R*T^2)/(4*pi^2*L^2+R^2*T^2)); %c for exact solution
subplot(3,2,[5,6]);
h = zeros(1,10);     %initialize arrays
errororder = zeros(1,10);
count = 1;  %initialize count

    hi=1e-9;hh=1e-9;hf=1e-7;                    % set initial step-size, ...
        increment in step-size and final step-size value
    h=hi:hh:hf;
    Nh=round((hf-hi)/hh)+1;            % number of steps=(interval size ...
        of step-size)/(increment in step-size)
```

The MATLAB code is implemented as follows: Initialize arrays for h and "errororder", then initialize a counter variable. Set the initial step-size, increment in step-size and final step-size value, then calculate the number of steps.

```
    for count=1:Nh
        [t,Vout]=heuns(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
        iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
        Vout_exact=feval(Vin,t)-R*iL_exact; % calculate exact solution ...
            as array
        errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
            maximum error over range of x
        hold on;
    end
    hold off;
    plot(log(h),log(errororder)); % plot log log graph
    gradheuns=polyfit(log(h), log(errororder),1); % calculate gradient

    title('log of maximum error against log of h')
    ylabel('log error max') % x-axis label
    xlabel('log h') % y-axis label
```

Iterate from 1 to the number of steps calculated and obtain the arrays of V_{out} for each value of h. The code calculates the maximum error by subtracting the V_{out}-array from the exact V_{out}-array, calculated previously in Equation (10). Eventually, it finds the maximum absolute value of the array, which is equal to the maximum error. Then it plots the maximum error as a point on the graph and repeats the same process for all step-sizes h. Finally, a log-log graph is plotted and the gradient of the log-log graph is obtained.

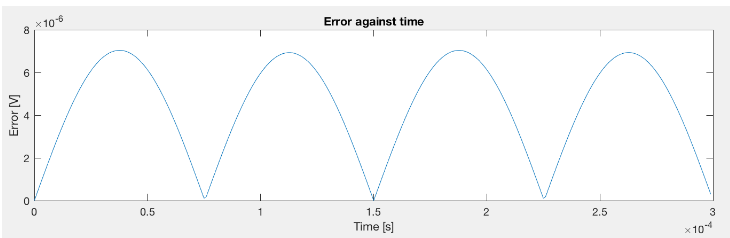


Figure 25: Heun's method error against time

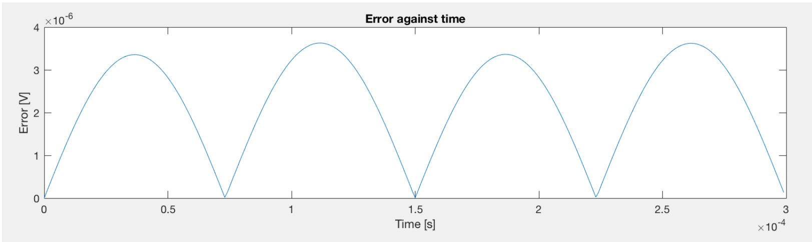


Figure 26: Midpoint method error against time

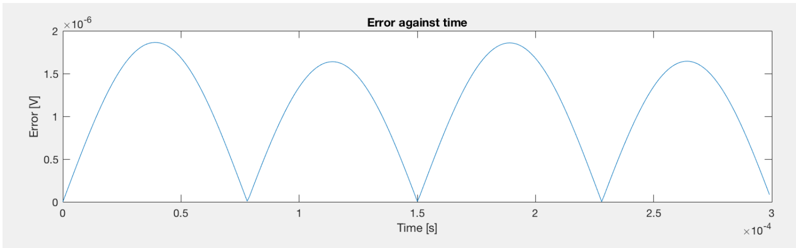


Figure 27: Ralston's method error against time

Comparing the amplitude of the error, we find that Heun's method has a magnitude of 7×10^{-6}, the midpoint method has a magnitude of 3.4×10^{-6} and Ralston's method has a magnitude of 1.8×10^{-6}. Thus, Ralston's method has the smallest error amplitude at step size $h = 10^{-7}$, indicating that Ralston's method gives the smallest truncation error.

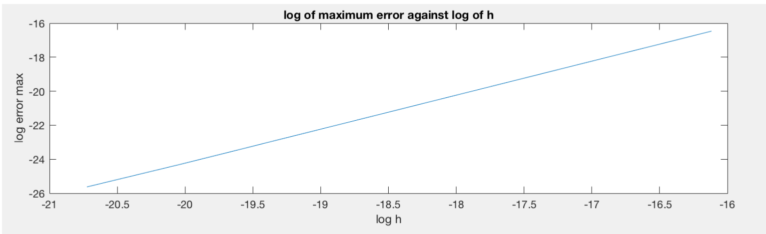


Figure 28: Heun's method log maxerror against $\log h$

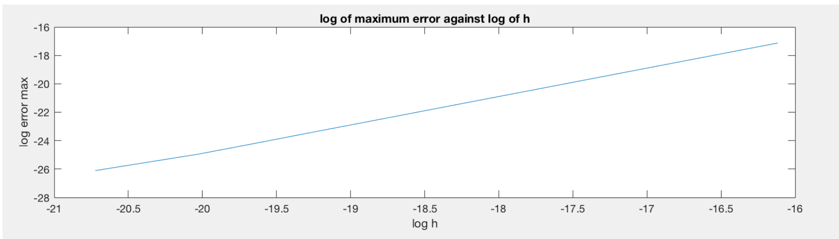


Figure 29: Midpoint method log maxerror against $\log h$

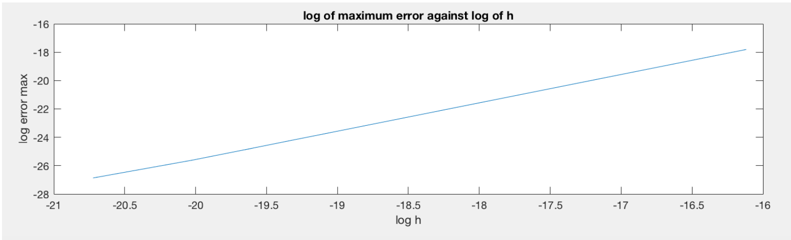


Figure 30: Ralston's method log(maxerror) against $\log h$

The log-log plot shows that by using a smaller step size, the maximum size of error also decreases. The gradient of the line was 1.9976 for Heun's method, 1.9900 for the Midpoint method, and 1.9934 for Ralston's method. Because the plot is of the order of maximum error against the order of step-size, the gradient of 2 means that all three 2nd order Runge-Kutta methods have an error of order $O(h^2)$.

3. Exercise 3: RLC circuit

3.1. RK4

Referring to the notation used in the Mathematics *II* course, **RK4second.m** is a MATLAB function that calculates y_{i+1}, x_{i+1} and t_{i+1} for a given y_i, x_i and t_i, using the *Runge-Kutta 3/8* algorithm. This means that the algorithm is used to approximate the values of a function x and its derivative y as we increase its independent variable t by a small step h. This is possible as the relationship between x, y and the argument t is known in form of a second order differential equation.

h, t_i, x_i, y_i, and two arbitrary functions are passed as input arguments to RK4second. The first function $funcx$ states the relationship between the derivative of x ($\dot{x}$=y) at a point t_i and the values t_i, x_i and y_i. The second function $funcy$ states the relationship between the derivative of y at a point t_i and the values t_i, x_i and y_i.

Given this information, the MATLAB function follows the *Runge-Kutta 3/8* algorithm, which obtains the increment functions ϕ_x and ϕ_y to evaluate the change in x and y:

$$x_{i+1} = x + h \cdot \phi_x;$$

$$y_{i+1} = y + h \cdot \phi_y.$$

The increment functions are given as follows:

$$\phi_x = (k_{1x} + 3k_{2x} + 3k_{3x} + k_{4x})/8;$$

$$\phi_y = (k_{1y} + 3k_{2y} + 3k_{3y} + k_{4y})/8,$$

where

$$k_{1x} = funcx(t_i, x_i, y_i);$$

$$k_{1y} = funcy(t_i, x_i, y_i);$$

$$k_{2x} = funcx(t_i + \frac{h}{3},\ x_i + \frac{h}{3}k_{1x},\ y_i + \frac{h}{3}k_{1y});$$

$$k_{2y} = funcy(t_i + \frac{h}{3},\ x_i + \frac{h}{3}k_{1x},\ y_i + \frac{h}{3}k_{1y});$$

$$k_{3x} = funcx(t_i + \frac{2}{3}h,\ x_i - \frac{1}{3}hk_{1x} + hk_{2x},\ y_i - \frac{1}{3}hk_{1y} + hk_{2y});$$

$$k_{3y} = funcy(t_i + \frac{2}{3}h,\ x_i - \frac{1}{3}hk_{1x} + hk_{2x},\ y_i - \frac{1}{3}hk_{1y} + hk_{2y});$$

$$k_{4x} = funcx(t_i + h,\ x_i + hk_{1x} - hk_{2x} + hk_{3x},\ y_i + hk_{1y} - hk_{2y} + hk_{3y});$$

$$k_{4y} = funcy(t_i + h,\ x_i + hk_{1x} - hk_{2x} + hk_{3x},\ y_i + hk_{1y} - hk_{2y} + hk_{3y}).$$

The following scaling factors are used:

$$a = \frac{1}{8}, \quad b = \frac{3}{8}, \quad c = \frac{3}{8}, \quad d = \frac{1}{8}$$

$$p_1 = \frac{1}{3}, \quad p_2 = \frac{2}{3}, \quad p_3 = 1$$

$$q_{11} = \frac{1}{3}, \quad q_{21} = -\frac{1}{3}, \quad q_{22} = 1, \quad q_{31} = 1, \quad q_{32} = -1, \quad q_{33} = 1.$$

These were obtained by looking at the Butcher tableau for the Runge-Kutta 3/8 algorithm [2] (see Figure 31).

$$\begin{array}{c|cccc}
0 & & & & \\
1/3 & 1/3 & & & \\
2/3 & -1/3 & 1 & & \\
1 & 1 & -1 & 1 & \\
\hline
 & 1/8 & 3/8 & 3/8 & 1/8
\end{array}$$

Figure 31: Butcher tableau for *Runge-Kutta 3/8* [6]

The MATLAB function starts by calculating the values of k_1. As the higher-order coefficients are evaluated at $\frac{1}{3}h$ increments of t_1, the values of x and y at these points rely on predictions that use the previous obtained coefficients. Therefore, the order of the evaluation of the coefficients is important. After all k-values have been calculated, the increment functions are computed, which are eventually used to find x_{i+1} and y_{i+1}.

When the function terminates, RK4second returns x_{i+1} and y_{i+1} (x_n and y_n).

RK4second.m is implemented with the following MATLAB code:

```
function [xn, yn] = RK4second(funcx, funcy, h, ti, xi, yi)
%RK4second computes y(i+1) and x(i+1) using the Runge-Kutta 3/8 algorithm
%   xn refers to x(i+1) and yn refers to y(i+1)
%   funcx computes the derivative of x (dx/dt) at a point (ti, xi, yi)
%   funcy computes the derivative of y (dy/dt) at a point (ti, xi ,yi)

%calculate coefficients (predicted gradients) at ti, ti+h/3, ti+2h/3, ti+h
%using Runge-Kutta 3/8
k1x = feval(funcx, ti, xi, yi);
k1y = feval(funcy, ti, xi, yi);
k2x = feval(funcx, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
k2y = feval(funcy, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
k3x = feval(funcx, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
k3y = feval(funcy, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
k4x = feval(funcx, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
k4y = feval(funcy, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);

%obtain phix and phiy by taking weighted average of obtained gradients
phix = (k1x + 3*k2x + 3*k3x + k4x)/8;
phiy = (k1y + 3*k2y + 3*k3y + k4y)/8;

%use phi-values as approximated gradients for x and y
xn = xi + h*phix; %calculate x(i+1)
yn = yi + h*phiy; %calculate y(i+1)
end
```

3.2. RLC circuit

The RLC_Circuit script finds the solution to a second order differential equation representing an RLC-circuit. RLC-circuits have the ability to resonate with a sinusoidal input signal and are sometimes used as band-pass filters. As all components are connected in series, they carry the same current $i(t)$.

The RLC circuit of interest (see Figure 32) can be described by the following equation:

$$\begin{aligned} v_L(t) + v_R(t) + v_C(t) &= V_{in}(t) \\ L\frac{d}{dt}i_L(t) + R\,i_L(t) + \frac{1}{C}\int_0^t i_L(t)\,\mathrm{d}t &= V_{in}(t) \\ L\frac{d^2}{dt^2}q_C(t) + R\frac{d}{dt}q_C(t) + \frac{1}{C}q_C(t) &= V_{in}(t) \end{aligned} \tag{11}$$

The state of the circuit is described by charge on the capacitor q_C, which is a function of time and depends on the input voltage $V_{in}(t)$.

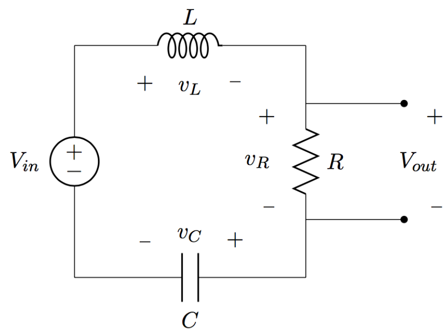


Figure 32: RLC-Circuit

The output of the circuit is the voltage across the resistor R (V_{out}) which is proportional to the derivative of $q_C(t)$:

$$V_{out} = R\,i_R = R\,i_C = R\frac{d}{dt}q_C(t) \tag{12}$$

The values of the components are given as:

$$R = 280\,\Omega, \quad C = 4\,\mu\mathrm{F}, \quad L = 600\,\mathrm{mH}$$

Moreover, there are initial conditions stating that the capacitor is initially charged with $q_C(0) = 500\,\mathrm{nC}$ and that there is no current running through the resistor at $t = 0$ ($i_L(0) = \frac{d}{dt}q_C(0) = 0\,\mathrm{A}$). Thus, there is also no voltage across the resistor.

The MATLAB function **RLC_script.m** starts by setting up the initial conditions, the component values, and the step-size as well as the time interval, in which the function is going to be evaluated. The step-size h will define the size of the error. A smaller h will lead to more calculation steps, and thus a longer simulation time, but will yield a result with a smaller error.

```
%The RLC_script calculates the voltage across R (Vout) for a given ...
    input signal(Vin)
%The RLC Circuit script is finding the solution to a second order ...
    differential equation representing an RLC-circuit

%set up initial conditions
q0 = 500*10^(-9); %[C]; capacitor charge at t=0
i0 = 0; %current at t=0
t0 = 0; %set up starting time
h = 0.000001; %[s]; set up step-size for Runge-Kutta 3/8
tf = 0.06; %[s]; define endpoint of time-interval

%define component values
R = 280; %resistance equals 280 Ohm
C = 4*10^(-6); %Capacitor value is 4 microFarad
L=600*10^(-3); %Inductance is 600 milliHenry
```

In the next step, the second order differential equation will be expressed as a system of two coupled first-order equations. As the current through a capacitor is defined as the derivative of its charge, the variable $i_L(t)$ $(= \frac{d}{dt}q_C(t))$ is introduced and will be used instead of $\frac{d}{dt}q_C(t)$.

The state of the RLC-circuit can be described by a second-order differential equation:

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

This equation is expressed as two coupled first-order equations:

$$L\frac{d}{dt}i_L(t) + Ri_L(t) + \frac{1}{C}q_C(t) = V_{in}(t) \tag{13}$$

$$i_L(t) = \frac{d}{dt}q_C(t) \tag{14}$$

The MATLAB function $funcq$ calculates the derivative of $q_C(t)$ (see Equation (14)). Even though $\dot{q}_C(t)$ is equal to $i(t)$, the function $funcq$ is expressed as a function of all three variables t, q and i. This generalizes the script to cases with two coupled first-order equations that are functions of all three variables.

$$funcq(t,\, i,\, q) = i$$

The second function $funci$ obtains the derivative of $i(t)$. $\dot{i}(t)$ is used to evaluate the change in i as t increases. $funci$ is a function of all three variables t, q and i and implements Equation (13):

$$funci(t,\, i,\, q) = (V_{in}(t) - R\,i - \frac{1}{C}\,q)/L$$

$V_{in}(t)$ is the input voltage. It is a function of t only and is created by a voltage source. Its exact shape is defined by the input signals specified in Exercise 3.

The MATLAB code creates the three functions mentioned above.

```
funcvin = @(t) 5; %define input signal (step-input, tf=0.06) as ...
    function of time

%the other input functions with their corresponding tf-value are stated
%below:
%funcvin = @(t) 5*exp(-t^2/(3*10^-6)); (impulse, tf=0.06)
%funcvin = @(t) 5*square(2*pi*t*5); (square, f=5Hz, tf=0.5)
%funcvin = @(t) 5*square(2*pi*t*110); (square, f=110Hz, tf=0.05)
%funcvin = @(t) 5*square(2*pi*t*500); (square, f=500Hz, tf=0.03)
%funcvin = @(t) 5*sin(2*pi*t*5); (sine, f=5Hz, tf=0.5)
%funcvin = @(t) 5*sin(2*pi*t*110); (sine, f=110Hz, tf = 0.05)
%funcvin = @(t) 5*sin(2*pi*t*500); (sine, f=500Hz, tf = 0.035)

%set up coupled first-order equations
funcq = @(t, q, i) i; %gradient of q at time t (=i(t))
funci = @(t, q, i)   (feval(funcvin, t) - R*i - 1/C * q)/L; %funci ...
    calculates di/dt at time t
```

Then, we calculate the number of steps N needed to go from the starting point (t_i) to the end of the time interval (t_f) in steps of h. As h does not necessarily divide $t_f - t_i$ without a remainder, the result of the division is rounded to obtain an integer number of steps. This means that after N steps, the function might not be evaluated exactly at t_f. However, as h gets smaller the last evaluated point in time approaches t_f.

```
N = round((tf-t0)/h); %calculate number of steps to reach tf
```

For the input functions specified in Exercise 3, t_f was sometimes changed in order to see all the important features of the output signal. The exact values of t_f are stated in the **RLC_script.m**. A small value of $h = 0.000001$ was chosen for all input signals in order to obtain an output signal with small error. Note that this may require significant computing power to be at the user's disposal.

Three empty arrays (zero valued) of length N are created to store t, q and i each. The first entry of each array is set by its corresponding initial condition.

```
%set up arrays to store results
q = zeros(1,N);
i = zeros(1,N);
t = zeros(1,N);

%first element of each array is equal to corresponding initial condition
q(1) = q0;
i(1) = i0;
t(1) = t0;
```

In the next step, a for loop is used to iterate from 1 to $N-1$. During each iteration q_{j+1} and i_{j+1} at $t_{j+1} = t + h$ are evaluated with the RK4second function, that uses $funcq\,(\widehat{=}\,funcx)$, $funci\,(\widehat{=}\,funcy)$, $h\,(\widehat{=}\,h)$, $t_j\,(\widehat{=}\,t)$, $q_j\,(\widehat{=}\,x)$ and $i_j\,(\widehat{=}\,y)$ as input arguments. The output arguments of RK4second are stored at the right positions of the arrays.

```
%use for-loop to iterate through arrays
%RK4second uses Runge-Kutta-3/8 algorithm to calculate next values for q
%and i as t is increased by h after each iteration
for j = 1 : N-1
   [q(j+1),i(j+1)] = RK4second(funcq, funci, h, t(j), q(j), i(j));
   t(j+1) = t(j) + h;
end
```

Once the for-loop has terminated we will be able to find the output voltage across the resistor R.

Using Ohm's Law (see Equation (12)), the output voltage is obtained:

$$V_{out} = i\,R$$

Finally, we plot the input signal, the capacitor charge, and the output voltage as a function of time between t_i and t_f.

```
vout = i*R; %obtain Vout(t)(voltage across R) using Ohms Law
vin = arrayfun(funcvin, t); %calculate Vin(t)

figure;
plot(t, q); %plot q(t) as a function of t
title('Capacitor Charge (q_{C}(t))');
xlabel('Time [s]');
ylabel('Charge [C]');

figure;
plot(t, vout); %plot Vout(t) as a function of t
title('Output Voltage (V_{out}(t)=v_{R}(t))');
xlabel('Time [s]');
ylabel('Voltage [V]');


figure;
plot(t, vin); %plot Vin(t) as a function of t
title('Input Signal (V_{in}(t))');
xlabel('Time [s]');
ylabel('Voltage [V]');
```

RLC_script.m as a whole can be found in the Appendix B.2.

3.3. Output voltage for different input signals

3.3.1. Analytical solution

The RLC-circuit can be solved analytically as well. This can be done using the Laplace transform.

A second-order differential equation describes the state of the RLC-circuit:

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

The Laplace transform is applied on both sides:

$$L(\bar{q}_C(s)s^2 - q_C(0)s) + R(\bar{q}_C(s)s - q_C(0)) + \frac{1}{C}\bar{q}_C(s) = \overline{V}_{in}(s).$$

By algebraic manipulation, an expression for $\bar{q}_C(s)$ can be found:

$$\bar{q}_C(s) = \frac{LCq_C(0)s + RCq_C(0) + C\overline{V}_{in}(s)}{LCs^2 + RCs + 1}. \tag{15}$$

After specifying the initial condition $q_C(0)$ and the applied input signal $V_{in}(t)$, this expression can be evaluated. Using the inverse Laplace transform, it is possible to obtain $q_C(t)$ and its derivative $i_L(t)$, thereby determining $V_{out}(t)$. $\overline{V}_{out}(s)$ can be also found directly from $\bar{q}_C(s)$ by applying the derivative Laplace-transform rule and multiplying by R. This yields:

$$\overline{V}_{out}(s) = Rs\frac{LCq_C(0)s + RCq_C(0) + C\overline{V}_{in}(s)}{LCs^2 + RCs + 1} - R\,q_C(0) \tag{16}$$

3.3.2. Step Signal

The first input signal is a step signal with amplitude $V_{in} = 5\,\text{V}$. It is implemented in the RLC_Circuit script as a function of time.

```
funcvin = @(t)5; %define input signal as function of time
```

After executing the MATLAB script, several plots are obtained.

The first plot shows the input signal as a function of time between t_i and t_f (see Figure 33). As the observed time interval does not include the negative time axis, the step signal looks like a DC-voltage with amplitude $5\,\text{V}$. However, in fact the input has jumped from $0\,\text{V}$ to $5\,\text{V}$ at $t = 0\,\text{s}$.

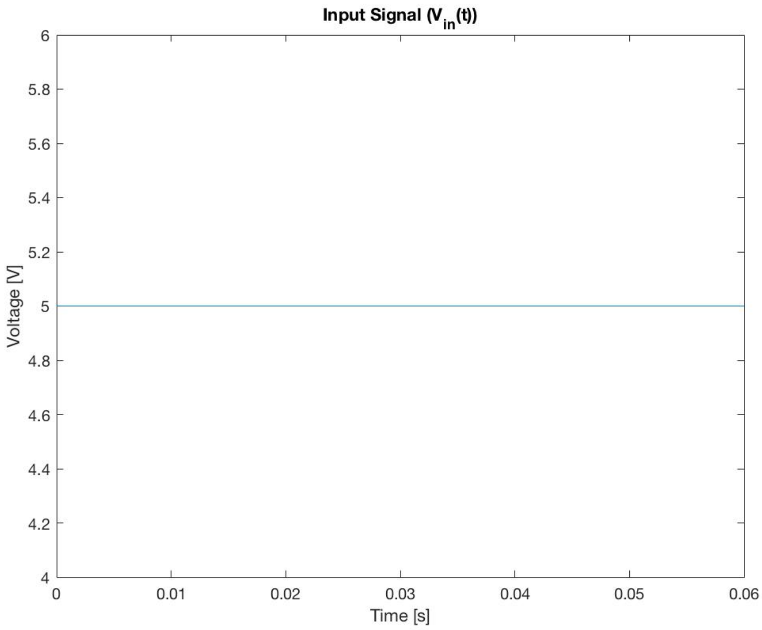


Figure 33: Step Signal with amplitude 5 V

In the second plot, the capacitor charge $q_C(t)$ is drawn as a function of time (see Figure 34). The graph starts at $q_C(0) = 0.5\,\mu\text{C}$ and eventually stabilises around its steady-state value of $20\,\mu\text{C}$.

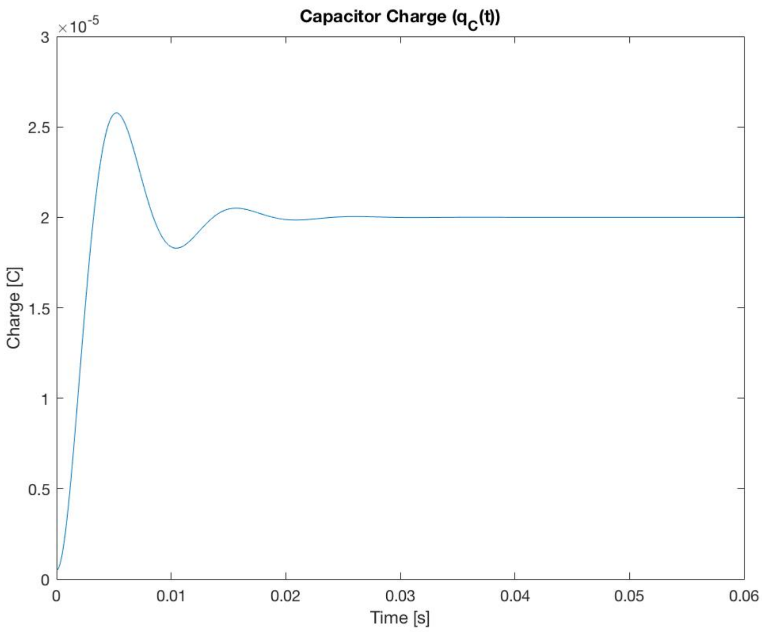


Figure 34: Response of capacitor charge for input step signal

The third plot shows the output voltage as a function of time (see Figure 35). The output voltage is the derivative of the capacitor charge scaled by the resistor value.

Since a capacitor will act as an open circuit for DC-voltages, all the input voltage will eventually drop across the capacitor and there will be no current running through the circuit. Thus, there will be no voltage drop across the resistor. The steady-state value of V_{out} is therefore 0 V. This can be easily confirmed by looking at Figure 35. The voltage drop of 5 V across the capacitor will determine the $q_C(t)$ value of 20 µF seen in Figure 34. This follows from the formula $Q = CV$.

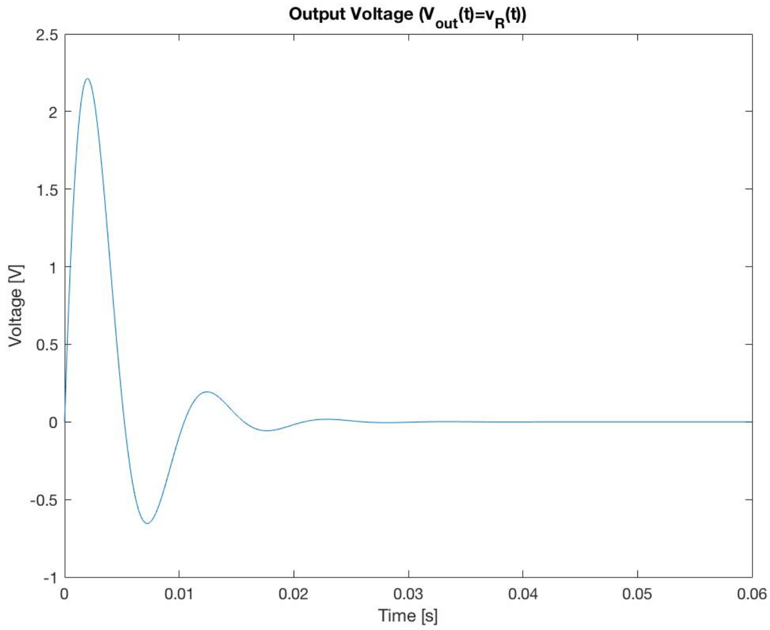


Figure 35: Output voltage for input step signal

The initial behaviour of the transients is more difficult to explain using simple intuition. In order to check that the code works as expected and that the obtained results are correct, the analytical solution of $q_C(t)$ and $V_{out}(t)$ for the input step signal were obtained.

Firstly, the Laplace transform of the step signal is obtained:

$$\overline{V}_{in}(s) = \frac{5}{s}$$

Then $\overline{V}_{in}(s)$, the initial conditions, and the components values are plugged into Equation (15). The inverse Laplace transform is then obtained after using partial fraction decomposition and minor rearrangement.

$$\bar{q}_C(s) = \frac{3s^2 + 1400s + 5 \times 10^7}{2 \times 10^6\, s\,(3s^2 + 1400s + 1.25 \times 10^6)} =$$

$$= \frac{-117s}{2 \times 10^6\,(3s^2 + 1400s + 1.25 \times 10^6)} - \frac{-273}{10^4\,(3s^2 + 1400s + 1.25 \times 10^6)} + \frac{1}{50000s}$$

Finally, the inverse Laplace transform is applied to obtain $q_C(t)$.

$$q_C(t) = 2{\times}10^{-5} - 1.95{\times}10^{-5} \exp\left\{\frac{-700t}{3}\right\} \cos\left(\frac{100\sqrt{326}}{3}t\right) - 7.56{\times}10^{-6} \exp\left\{\frac{-700t}{3}\right\} \sin\left(\frac{100\sqrt{326}}{3}t\right)$$

$i_L(t)$ is obtained by taking the derivative of the expression above and $V_{out}(t)$ follows by multiplying by R.

$$i(t) = 2.154\times10^{-8}\exp\left\{\frac{-700t}{3}\right\}\cos\left(\frac{100\sqrt{326}}{3}t\right)+1.35\times10^{-2}\exp\left\{\frac{-700t}{3}\right\}\sin\left(\frac{100\sqrt{326}}{3}t\right)$$

$$V_{out}(t) = 6.031\times10^{-6}\exp\left\{\frac{-700t}{3}\right\}\cos\left(\frac{100\sqrt{326}}{3}t\right)+3.78\exp\left\{\frac{-700t}{3}\right\}\sin\left(\frac{100\sqrt{326}}{3}t\right)$$

Plotting the analytically-obtained $V_{out}(t)$ (see Figure 36), one can find the same graph as in Figure 35. This proves that the MATLAB script and its corresponding function **RK4second.m** are working correctly.

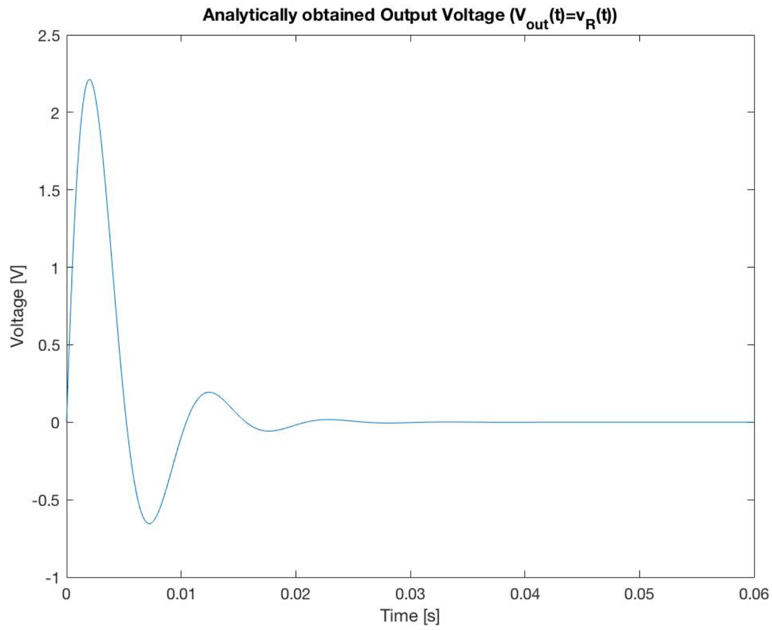


Figure 36: Analytically obtained output voltage for input step signal

Following the control lecture about system responses [3], the shape of the transient response for $q_C(t)$ and $V_{out}(t)$ are as expected. Assuming that the overshoot and rise time of $q_C(t)$ are determined by the dominant poles of $\bar{q}_C(s)$ (solution to $3s^2 + 1400s + 1.25\times10^6 = 0$), $\bar{q}_C(s)$ is found to be an underdamped system. This conclusion comes from the fact that the two poles have a negative real part and are complex conjugates ($p_1 = \frac{-700}{3} + \frac{100\sqrt{326}}{3}i$, $p_2 = \frac{-700}{3} - \frac{100\sqrt{326}}{3}i$). The overshoot is determined by the angle of p_1 and p_2 in the complex plane $\alpha = \arctan\left(\frac{-Re(p_1)}{Im(p_1)}\right)$ and the settling time, which is defined to be the time to reach a 2%-tube around the steady-state value. It is given by $\frac{4}{Re(p_1)}$. For $q_C(t)$ α is 68.808°, which yields to an overshoot of around 30% of the steady-state value. The settling time is around $0.017s$. The overshoot in Figure 34 is close to this expected value. The actual settling time seems to be a little bit bigger than expected. This inaccuracy is caused by the initial assumption of a system with two complex conjugate poles only.

3.3.3. Impulsive signal with decay

An impulsive signal with exponential decay is applied to the RLC-circuit. The input voltage is defined by the following equation, where $\overline{V}_{in} = 5\,\mathrm{V}$ and $\tau = 3(\mathrm{ms})^2 = 3\mathrm{\mu s}$:

$$V_{in}(t) = \overline{V}_{in} \exp\left\{-\frac{t^2}{\tau}\right\}$$

This is implemented as the following MATLAB function:

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

The graph for the input signal is a quickly-decaying second order exponential function (see Figure 37). At $t = 0$, it jumps to 5 V and then approaches its steady-state value of 0 V, simulating a quick and brief change in the input conditions.

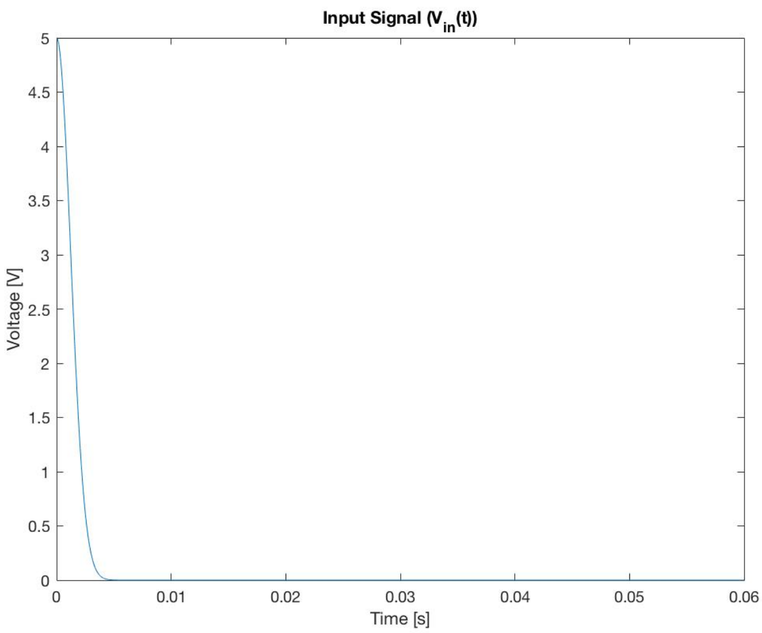


Figure 37: Impulsive signal with decay

The plot showing the capacitor charge (see Figure 38) starts from its initial position of 500 nC. As the input voltage eventually reaches 0V, there will be no voltage across the capacitor. Thus, there is no electric field causing a charge difference between the two capacitor plates. Therefore, after an initial ringing transient behaviour caused by the change of the input signal, $q_C(t)$ will approach 0 C.

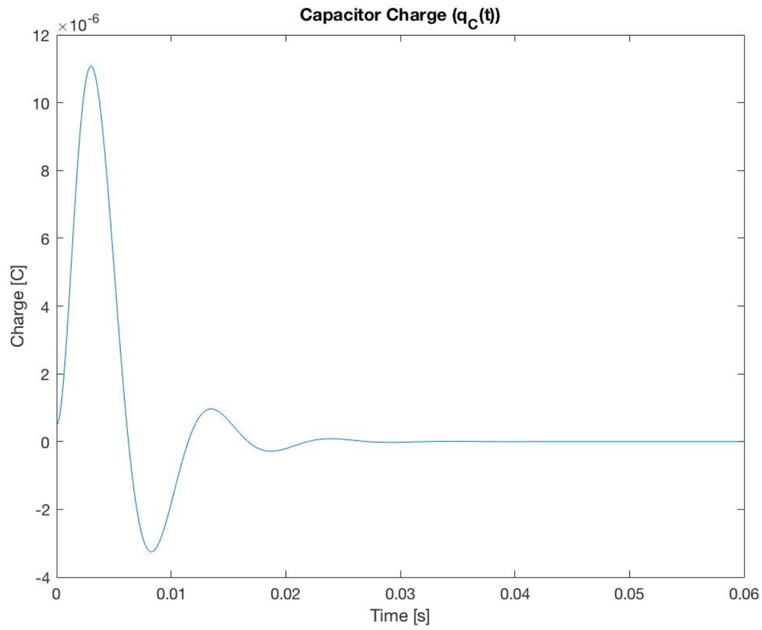


Figure 38: Capacitor charge for input impulsive signal with decay

The output voltage graph is showing ringing transient oscillation with decaying amplitude (see Figure 39). This damped oscillation is caused by the exchange of energy between the inductor and the capacitor. A sudden rise in the input voltage puts a higher voltage across the inductor, which leads to an increase in current ($\frac{d}{dt}i_L(t) = \frac{v_L}{L}$). The current builds up a magnetic field inside the inductor and increases q_C and the voltage across the capacitor. As q_C rises, the capacitor stores energy in form of an electric field. As the input voltage fades, the magnetic field of the inductor will start to decay, forcing i_L to continue to flow, while shrinking in amplitude. When i_L eventually reaches $0\,\mathrm{A}$, the electric field across the capacitor will have reached its maximum. Then, the electric field will decline, causing current to flow in the opposite direction. In return, this will build up a magnetic field inside the inductor with reverse polarity. The amplitude of the oscillation is declining as there is a continuos loss of energy in form of heat inside the resistor.

The steady state value of the output voltage will be equal to $0\,\mathrm{V}$. As the change in the input signal is getting smaller and its absolute value is approaching $0\,\mathrm{V}$, the oscillation will die down and the capacitor will act as an open circuit. Eventually, there will be no potential voltage difference causing charges to flow. Thus, there will be no current flowing through the resistor and the output voltage will reach $0\,\mathrm{V}$. The exact analytical solution of the impulsive signal using the Laplace transform is difficult to find, as there does not exist a closed form solution for the Laplace transform of the input function. However, as it is a fast decaying signal with a peek at $t = 0$s, it can be approximated by a Dirac delta function. Therefore, the output voltage is looking similar to the unit impulse of the system. This is found as:

$$\boldsymbol{L}(\delta(t)) = 1$$

Putting this together with the initial conditions and the component values into Equation (16), the following expression is found for $\overline{V}_{out}(s)$:

$$\overline{V}_{out}(s) = \frac{175 \times (8s - 1)}{3s^2 + 1400s + 1250000}$$

Taking the Laplace inverse, the unit impulse response is found as:

$$V_{out}(t) = \frac{1400}{3} \exp\left\{\frac{-700t}{3}\right\} \cos\left(\frac{100\sqrt{326}}{3} t\right) - \frac{39221}{12\sqrt{326}} \sin\left(\frac{100\sqrt{326}}{3} t\right) \exp\left\{\frac{-700t}{3}\right\}$$

This can be plotted using matlab with the following command (see Figure 40):

```
impulse([1400 -175], [3 1400 1250000]);
```

Comparing the two graphs, one can find that both are varying with the corner frequency of the transfer function of the system. This is also the corresponding frequency of the dominant complex conjugate pole pair of the system. Moreover, they have a similar settling time, which is defined by the real part of the complex conjugate pole pair. However, since the Dirac delta function has an amplitude approaching infinity, the amplitude of the unit impulse response is much bigger than that of the obtained output signal. This high amplitude also forces the unit impulse response to immediately start from a non-zero value.

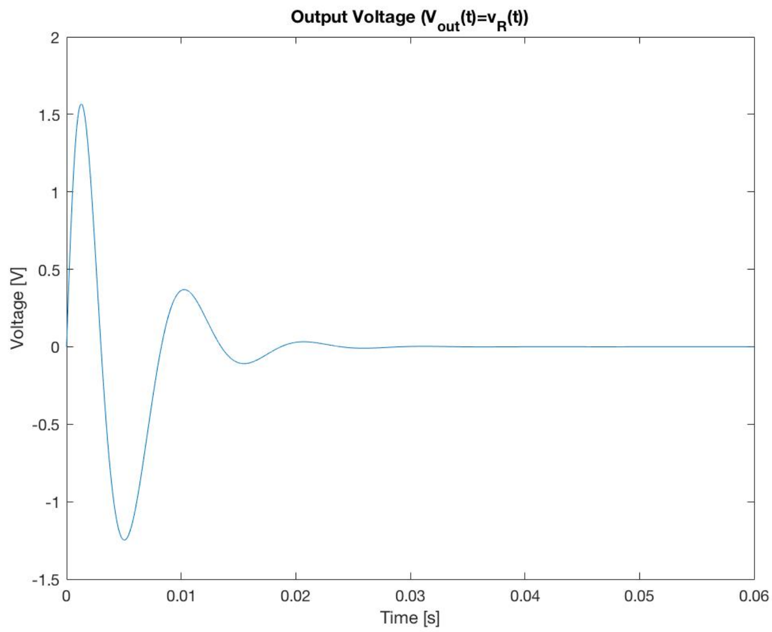


Figure 39: Output voltage for input impulsive signal with decay

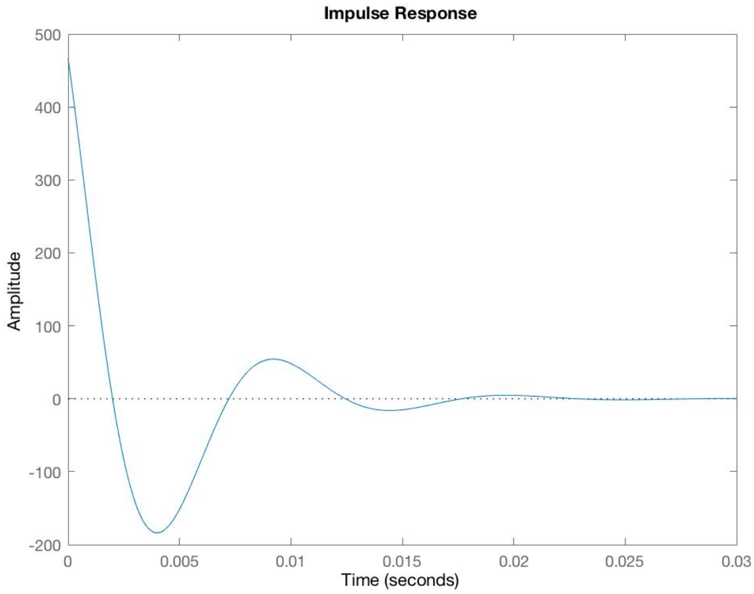


Figure 40: Unit Impulse Response of the RLC-circuit

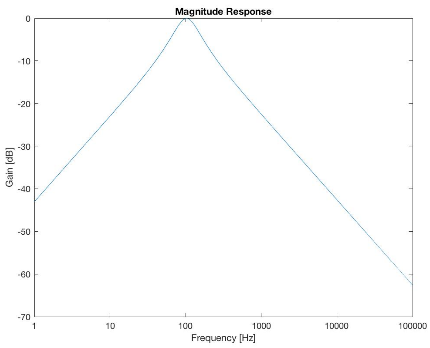
3.3.4. Square Waves with different frequencies

In this subsection, the input to the RLC-circuit is a square wave with amplitude 5 V. As the frequency of the square wave is varied, the corresponding output will take on different shapes. A square wave can be split up into a sum of sine waves using the Fourier series expansion (see Equation (18)). The RLC-Circuit will act as a second order bandpass-filter and attenuate frequency components higher and lower than the corner frequency $f_C = \frac{\sqrt{\frac{1}{LC}}}{2\pi} = 102.73\text{Hz}$. While an inductor will act like a short circuit for low frequency signals, the capacitor will have a high impedance and thus most of the input voltage will drop across it. For high frequencies the exact opposite happens and the voltage across the inductor increases. At f_C, the two impedances cancel out and all the voltage drops across the resistor, whose characteristics are frequency independent. This behaviour can be described by the transfer function $H(j\omega)$.

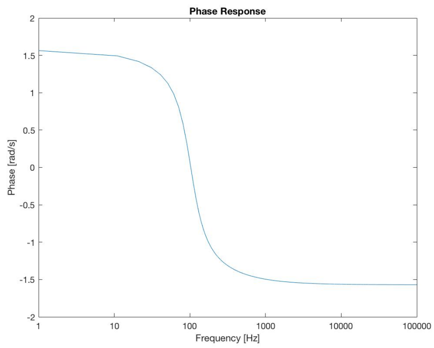
Using the potential divider rule, $H(j\omega)$ is found as:

$$H(j\omega) = \frac{R}{R + j\omega L + \frac{1}{j\omega C}} = \frac{RCj\omega}{LC(j\omega)^2 + RCj\omega + 1} \tag{17}$$

Plotting the gain and the phase against different input frequencies will confirm the theory stated above (see Figure 41a and Figure 41b):



(a) Magnitude Response



(b) Phase Response

Figure 41: Frequency Response of the RLC-Circuit

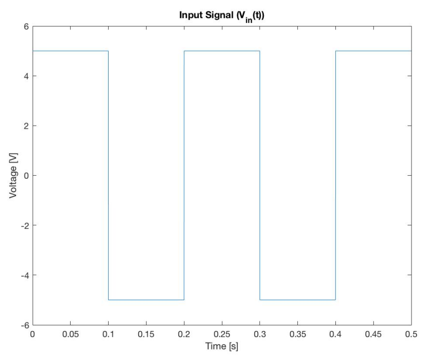
The Fourier series of a square wave with frequency f and amplitude 5 V has the form of:

$$5 \times \text{square}(2\pi f t) = \frac{20}{\pi} \sum_{n=1,3,5....}^{\infty} \frac{1}{n} \sin\left(n 2\pi f t\right) \tag{18}$$

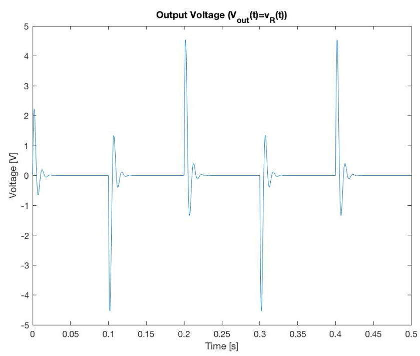
The first square wave used as input has a frequency of 5 Hz (see Figure 42a):

```
funcvin = @(t)5*square(2*pi*t*5);
```

In order to be able to observe all important changes in the output waveform, the endpoint of the time interval is increased to $t_f = 0.5\,\mathrm{s}$. For this input, the output consist of a periodic signal with $f = 5\,\mathrm{Hz}$ (see Figure 42b). As the period of the square wave is much smaller than the settling time for a step input, the output looks like the sum of several step input responses shifted in time. The time between transition is long enough in order for the system to reach the steady state position of 0 V (DC-voltage is all dropped across the capacitor). However, as the change in amplitude at each transition is 10 V, the amplitude of the ringing is twice as big as in subsection 3.3.2. This is true, except for the first ringing as the system is starting from 0 V and the first voltage step has a magnitude of 5 V. Moreover, now there are also negative going transitions from 5 V to -5 V, which cause a "negative" oscillation.



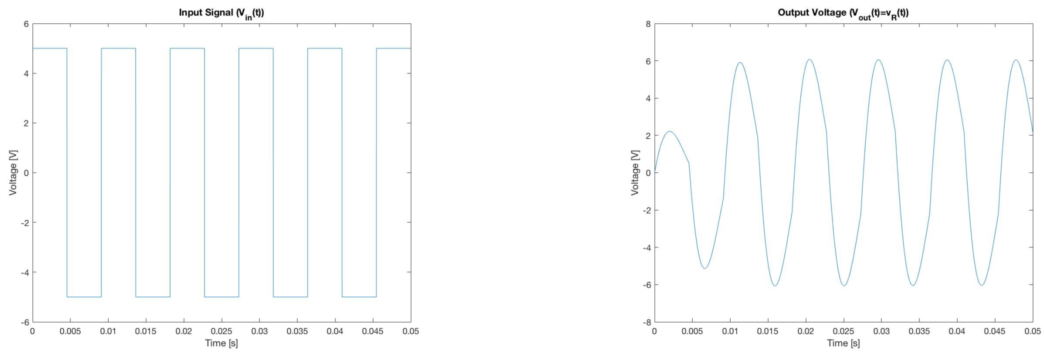
(a) Input square wave with $f = 5\,\mathrm{Hz}$



(b) Corresponding Output Waveform

Figure 42: RLC-Circuit response to square wave of $f = 5\,\mathrm{Hz}$

The second input square wave has a frequency of $f = 110\,\mathrm{Hz}$ and an amplitude of $5\,\mathrm{V}$ (see Figure 43a). t_f is set to $0.05\,\mathrm{s}$. As the frequency of the first harmonic ($110\,\mathrm{Hz}$) is close to the corner frequency, it will be only slightly attenuated. With increasing order, the attenuation of the system for a harmonic term gets bigger (see Figure 41a). Thus, their contribution to the final output signal becomes more negligible. This is why, an only slightly distorted sine wave of frequency $110\,\mathrm{Hz}$ can be seen as the steady state output (see Figure 43b). The time constant of the transient behaviour and the frequency of the transient oscillation are equal to the ones seen in subsection 3.3.2, as these two values depend on the system itself (Laplace transform denominator). In this case, the period of the input sine wave and its frequency are similar in size to the transient response. Thus, there appears to be no sharp change in voltage.

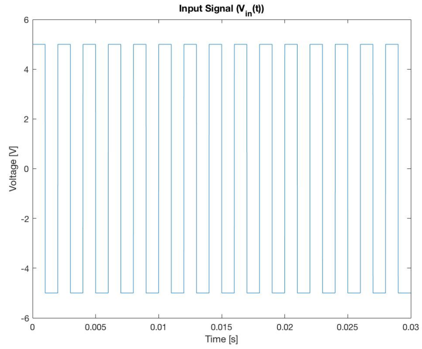


(a) Input square wave with $f = 110\,\mathrm{Hz}$

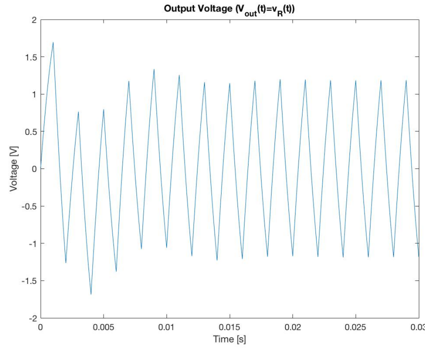
(b) Corresponding Output Waveform

Figure 43: RLC-Circuit response to square wave of $f = 110\,\mathrm{Hz}$

The last square wave input has a frequency of $f = 500\,\mathrm{Hz}$ (see Figure 44a). In order to see the whole transient response t_f was set to $0.03\,\mathrm{V}$. As the frequency of the square wave is much greater than the frequency and the time constant of the transient oscillation, the system has not much time to respond to the change. This is why, the output voltage is sharply rising from its steady state maximum and minimum with a frequency of $500\,\mathrm{Hz}$ (see Figure 44b). Before the output oscillation is able to reach its maximum after a positive going input transition, there is already another change in the input. This leads to a sharp peak in the output voltage and a very steep transition in order to follow the input. Moreover, the steady state amplitude of the waveform is decreased to around $1.2\,\mathrm{V}$. This was expected as all the harmonics will be attenuated by the RLC-circuit. The initial transient oscillation frequency due to the initial conditions is smaller than the output frequency and therefore appears like a time varying offset.



(a) Input square wave with $f = 500\,\mathrm{Hz}$



(b) Corresponding Output Waveform

Figure 44: RLC-Circuit response to square wave of $f = 500\,\mathrm{Hz}$

3.3.5. Sine Waves with different frequencies

Three sine waves with an amplitude of $5\,\mathrm{V}$ and frequencies of $5\,\mathrm{Hz}$, $110\,\mathrm{Hz}$ and $500\,\mathrm{Hz}$ are applied to the circuit. The steady-state output of a sine wave to a circuit with passive components can be analytically obtained using the transfer function ($H(j\omega)$) of the circuit. The output will always be a sine wave with the same frequency as the input signal. However, there might be a change in amplitude and phase.

A MATLAB expression is found for the first sine wave input with $f = 5\,\mathrm{Hz}$:

```
funcvin = @(t)5*sin(2*pi*t*5);
```

t_f is equal to $= 0.5\,\mathrm{s}$.

As expected, the input signal shows a sine wave with amplitude 5 V and a period of $0.2\,\mathrm{s}$ (see Figure 45a).

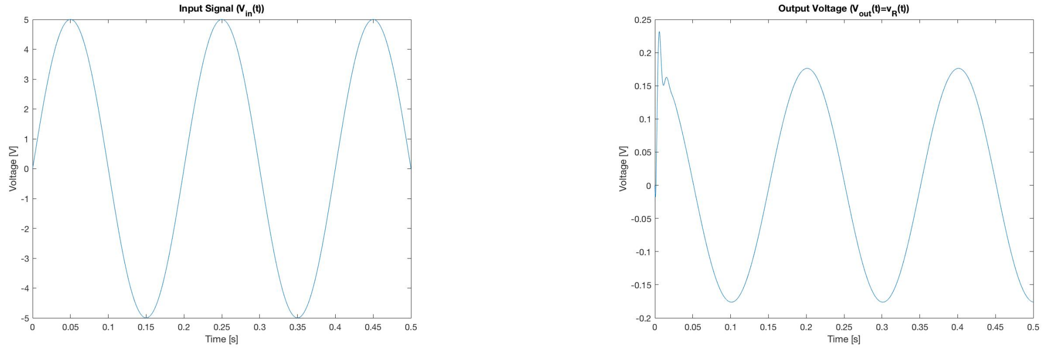
The steady state response of the output sine wave is expected to be shaped by the transfer function. $H(j2\pi5)$ is evaluated with the corresponding component values. Then, the gain and the phase-shift are obtained from the transfer function:

$$H(j2\pi5) = \frac{280 \times 4 \times 10^{-6} \times j2\pi5}{600 \times 10^{-3} \times 4 \times 10^{-6}\ (j2\pi5)^2 + 280 \times 4 \times 10^{-6} \times j2\pi5 + 1} = 1.24{\times}10^{-3}{+}3.52{\times}10^{-2}\, i$$

$$|H(j2\pi5)| = 0.0352$$

$$\angle H(j2\pi5) = 87.98°$$

After the initial transients have died down, one would expect a sine wave with $\overline{V} = 5 \times 0.0352 = 0.176\,\mathrm{V}$ and with a positive phase shift of almost 90°. That is exactly what can be observed in Figure 45b. After the initial overshoot, the system reaches its steady state after a similar settling time as in subsection 3.3.2 and varies with the parameters stated above.



(a) Input sine wave with $f = 5\,\text{Hz}$

(b) Corresponding Output Waveform

Figure 45: RLC-Circuit response to sine wave of $f = 5\,\text{Hz}$

The next input is a sine wave with $f = 110\,\text{Hz}$ and $\overline{V} = 5\,\text{V}$, which is described using the following MATLAB function:

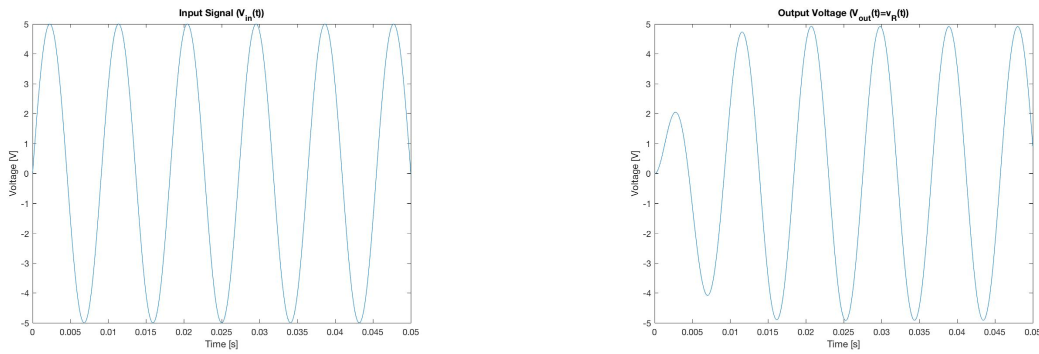
```
funcvin = @(t)5*sin(2*pi*t*110);
```

t_f is changed to $0.05\,\text{s}$. The input signal is shown in Figure 46a.

For this frequency, the following gain and phase shift are expected:

$$H(j2\pi 110) = 0.965 - 0.183\,i$$
$$|H(j2\pi 110)| = 0.9826$$
$$\angle H(j2\pi 110) = -10.713°$$

The steady-state response in Figure 46b clearly shows this behaviour. The amplitude is $4.913\,\text{V}$ ($= 5 \times 0.9826$) and there is a small phase lag compared to the input signal. $110\,\text{Hz}$ is close to the corner frequency $f_C = \frac{\sqrt{\frac{1}{LC}}}{2\pi} = 102.73\text{Hz}$, which is the resonance frequency, where the gain has its maximum of 1. The initial transient behaviour reveals that the sine wave is continuously increasing in amplitude, as the time constant and the period of the initial transient oscillation are of similar magnitude to the period of the output sine wave.



(a) Input sine wave with $f = 110\,\text{Hz}$

(b) Corresponding Output Waveform

Figure 46: RLC-Circuit response to sine wave of $f = 110\,\text{Hz}$

The final input signal consists of a sine wave with $f = 500\,\text{Hz}$ (see Figure 47a). t_f is further decreased to 0.035 s. The sine wave is described through the following MATLAB function:

```
funcvin = @(t)5*sin(2*pi*t*500);
```

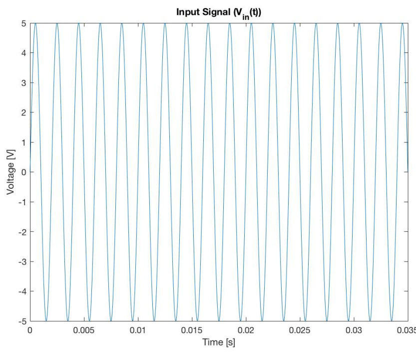
The transfer function for $f = 500\,\text{Hz}$ has the following properties:

$$H(j2\pi500) = 2.35 \times 10^{-2} - 0.151\,i$$

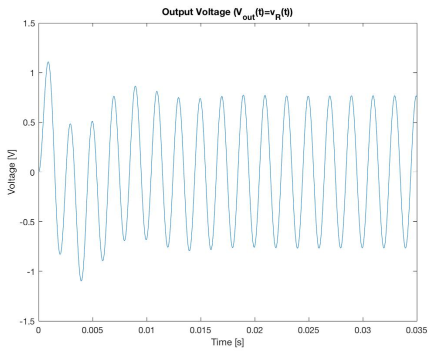
$$|H(j2\pi500)| = 0.153$$

$$\angle H(j2\pi500) = -81.18°$$

The output waveform for an input sine wave of frequency 500 Hz and amplitude of 5 V is expected to lag behind the input sine wave by 81.18°. The amplitude will equal 0.766 V ($= 5 \times 0.153$). Plotting the graph for the output waveform confirms this behaviour (see Figure 47b). As the frequency of the input signal grows larger than of the transient oscillations, the initial transient behaviour looks like a varying offset for the output sine wave.



(a) Input sine wave with $f = 500\,\text{Hz}$



(b) Corresponding Output Waveform

Figure 47: RLC-Circuit response to sine wave of $f = 500\,\text{Hz}$

4. Exercise 4: Diffusion of heat along a wire

4.1. The heat equation, and obtaining a calculable expression

We consider the heat of a wire as a function of space and time. The wire is taken to be sufficiently thin that only a single spacial dimension is of significance.

The one-dimensional heat equation is [1]

$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2}. \tag{19}$$

We proceed using the finite-differences method; the terms on the LHS and RHS are replaced with approximations obtained from the Taylor series of y [**riley1998mathematical**] and truncated to the linear term. In the limit as the number of spacial divisions N_x and the number of temporal divisions N_t grows large, we recover the (expected) behavior of the analytical solution to the equation (which may not exist). In the case of the one-dimensional heat equation, we partition space and time into a grid U, where $U_x^t = y(x,t)$, where y is the solution to Equation (19). The wire has length $L_{\max} = 1.0$ and is simulated for a duration of time $T_{\max} = 1.0$, yielding boundaries on U. The *initial condition* is the heat of the wire for time $t = 0$, $U_x^0 = f_1(x)$. The *boundary conditions* represent the heat at each end of the wire as a function of time, $U_0^t = f_2(t)$ and $U_{L_{\max}}^t = f_3(t)$. The *increment* is defined as $\Delta x = L_{\max}/N_x$ in space and $\Delta t = T_{\max}/N_t$ in time. The equation generated from applying the approximations obtained from the Taylor series is

$$\frac{U_x^{t+1} - U_x^t}{\Delta t} = \frac{U_{x+1}^t - 2U_x^t + U_{x-1}^t}{(\Delta x)^2}, \tag{20}$$

which clearly becomes Equation (19) in the limit as $\Delta x, \Delta t \to 0$. Rearranging for U_x^{t+1}, we obtain

$$U_x^{t+1} = U_x^t + \frac{\Delta t}{(\Delta x)^2}\left(U_x^{t+1} - 2U_x^t + U_x^{t-1}\right) \tag{21}$$

Let $r = k\frac{\Delta t}{(\Delta x)^2}$ represent the thermal conductivity of the medium[1]; then, we obtain an implementable expression for U_x^{t+1} :

$$U_x^{t+1} = (1-2r)U_x^t + rU_{x+1}^t + rU_{x-1}^t. \tag{22}$$

4.2. Implementation of the finite-differences method

For the full implementation, see Appendix C.

To begin the MATLAB code, we initialize a number of constants relating to these conditions:

[1] k is a material-dependent factor; without loss of generality, we took $k = 1$ in the earlier working. In practice, $r \leq 0.5$ as a stability consideration; later, we take $k = 0.25$.

```
L = 1.;        % wire length
T = 1.;        % max simulation time
Nt = 2500;       % number of timesteps
Nx = 50;         % number of spacial divisions
dt = T / Nt; % increment through time
dx = L / Nx; % increment through space
```

Naming the variables dx and dt may appear to be somewhat of an abuse of notation; read these as *delta-x* and *delta-t*, rather than as differentials. This notation is preferred because the names of the variables suggest the dimension (space or time) that each increments, as opposed to the perhaps more 'traditional' names h and k.

We now determine the thermal conductivity parameter for some arbitrary choice of k. For stability, we must ensure that $r \leq \frac{1}{2}$; our choices of N_t, N_x, L, and T provide $r = k$. We choose $k = 1/4$, guaranteeing stability. A warning is provided to the user if $r > \frac{1}{2}$.

```
r = 0.25 * dt / (dx*dx);
if r > 0.5
    disp('warning: for stability, r ≤ 1/2');
    r
end
```

Anonymous ('lambda') functions are initialized to represent initial and boundary conditions. This is relatively straightforward. Unfortunately, in MATLAB, lambdas are not *closures*[2]; this means that variables in the scope of the declaration of the lambda will not implicitly be captured by the lambda. Hence, the user must explicitly pass each value used by the lambda in the function invocation.

Many initial conditions were used; these declarations are left here as commented-out code.

```
initialcond = @(x, L, Nx) abs(sin(2*pi*x/(Nx+1)));
% initialcond = @(x, L, Nx) sin(2 * pi * x / (Nx+1) / L);
% initialcond = @(x, L, Nx) triangularPulse(0.0, L, x/(Nx+1));
% initialcond = @(x, L, Nx) sinc(6 * pi * ((x - (Nx+1)/2) / (Nx+1)));
% initialcond = @(x, L, Nx) -12.0;

leftbound = @(t, Nt) sin(2*pi*t/Nt);
% leftbound = @(t, Nt) -sin(2*pi*t/Nt);
% leftbound = @(t, Nt) 0.0;

rightbound = @(t, Nt) sin(2*pi*t/Nt);
% rightbound = @(t, Nt) 0.0;
```

An array of x-values is created to serve as the spatial axis. This is relatively premature in the sense that this array is not needed until the computation has finished, and the

[2]Philosophically, some might view this as a good idea; the notion of "spooky action at a distance" in codebases refers to the ability to break one subsystem by making a seemingly innocuous change to another subsystem. Not including closures will mean that changes to global variables will not disrupt the behavior of the lambda in an unclear way. Hence, while some prefer the convenience of closures, others prefer this relatively hygienic approach.

data is ready for display; however, placing the line of code here minimizes the number of loops needed.

More importantly, the initial conditions are established in this loop. This is done by invoking the lambda function declared previously.

```
for i = 1:Nx+1
    x(i) = (i-1)*dx;
    u(i,1) = initialcond(i, L, Nx);
end
```

At this point, boundary conditions are established. These may be established independently for either end of the wire and are allowed to vary with time. In this case, the heat at either end of the wire is given by $\sin\left(\frac{2\pi t}{T_{\max}}\right)$. At this point, a time axis is also initialized, as with the spatial axis declared in the earlier loop.

```
for t = 1:Nt+1
    u(1,t) = leftbound(t, Nt);
    u(Nx+1, t) = rightbound(t, Nt);
    time(t) = (t-1) * dt;
end
```

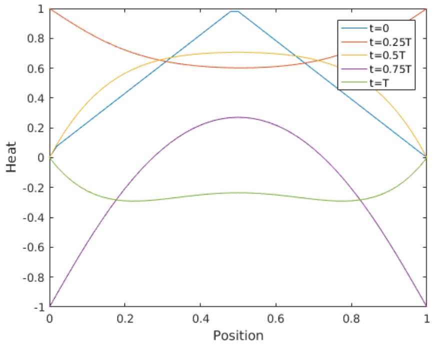
Finally, we iterate over all position and time to compute values for U_x^t with Equation (22).

```
for t=1:Nt
    for i = 2:Nx
        u(i, t+1) = (1-2*r) * u(i,t) + r*u(i+1, t) + r*u(i-1,t);
    end
end
```

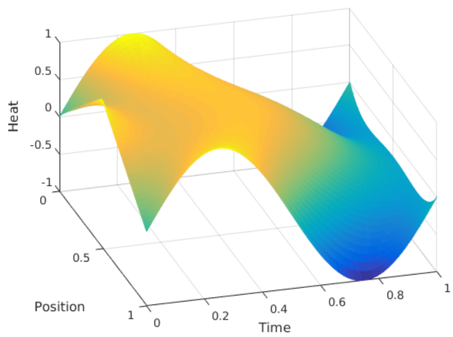
Once computed, the solution can be displayed trivially with MATLAB's built-in functions (note the necessary cast to an integer type of sufficient width to allow a large range of N_t):

```
figure(1)
plot(x,u(:,int32(0.00*Nt)+1),'-', ...
     x,u(:,int32(0.25*Nt)),'-',   ...
     x,u(:,int32(0.50*Nt)),'-',   ...
     x,u(:,int32(0.75*Nt)),'-',   ...
     x,u(:,int32(Nt)),'-')
legend('t=0', 't=0.25T', 't=0.5T', 't=0.75T', 't=T')
figure(2)
mesh(x,time,u')
```

The resulting plots are shown in Figures 48 and 49.



(a) Heat distribution along the wire for selected values of t.



(b) A mesh plot, viewed from an angle that makes clear the tent function initial conditions and sinusoidal time-varying boundary conditions.

Figure 48: Heat of a wire for time-varying boundary conditions. The initial condition is a triangular pulse; boundary conditions are computed along a single period of a sine wave.

Figures 48 and 49 show temperature variation along the wire through time, as boundary conditions change according to the function $\sin\left(\frac{2\pi t}{T}\right)$, where $T = 1$ is the highest value of t computed.

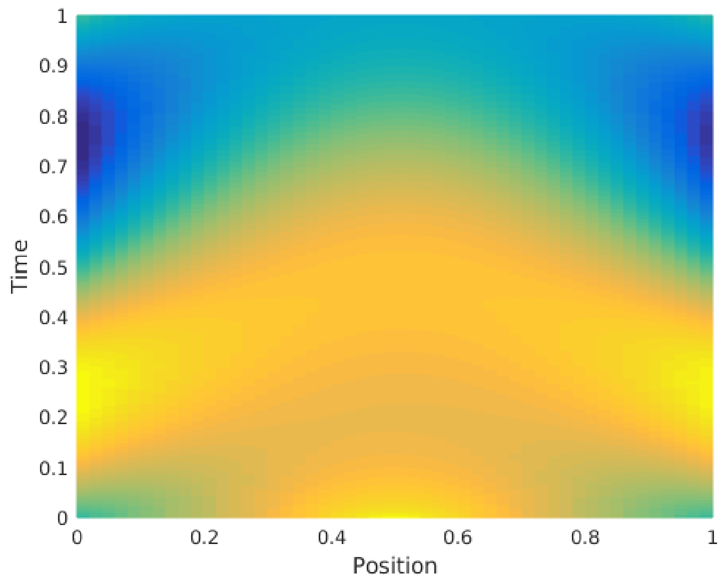


Figure 49: Heat distribution along the wire through time.

Figure 48a shows heat distribution along the wire for selected values of t. At $t = 0$, the initial heat distribution (a tent function) is observable. At $t = 0.25T_{\max}$, the boundary values of heat go to unity, and the ends of the wire act as a source of heat; as the ends were heating up, the middle cools, leading to a concave-up graph. Because of a symmetrical initial heat distribution and identical boundary conditions on either end of the wire, there is symmetry about $L_{\text{max}}/2$. By $t = 0.5T_{\text{max}}$, the boundary conditions have returned to zero, and heat once again flows from the middle of the wire to the ends. At $t = 0.75T_{\text{max}}$, the boundary conditions reach their minimum values, and the ends of the wire act as a sink for heat. The temperature rises once again, such that at $t = T_{\text{max}}$, the ends of the wire source heat. Points of inflection are notable along the graph, as the middle cools more slowly than the edges, as noticeable from the sharp gradient of the graph at $t = 0.75T_{\text{max}}$ [5]. Heat flow is more clearly visualized in Figure 49, where equipotentials vary linearly through time - i.e., heat flows at a constant speed.

Naturally, the left and right boundary conditions are independent of each other. In this case, the left boundary condition was the additive inverse of the other. The initial condition is the magnitude of a sine wave, as given in Equation (23c).

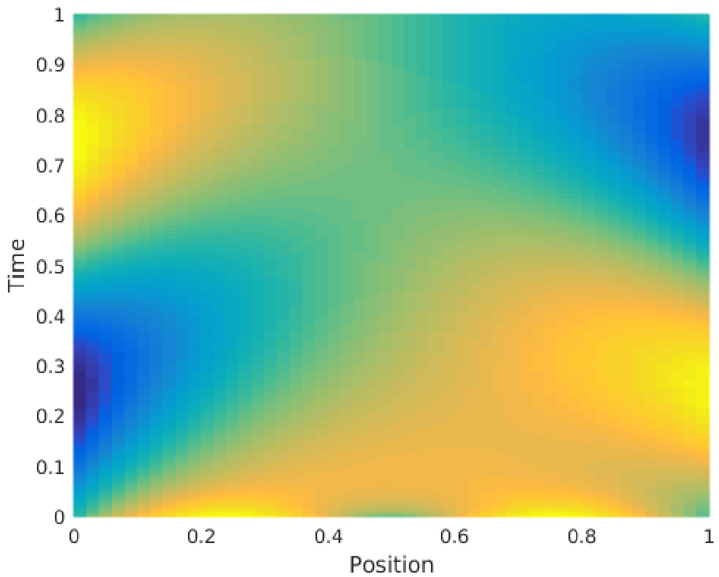
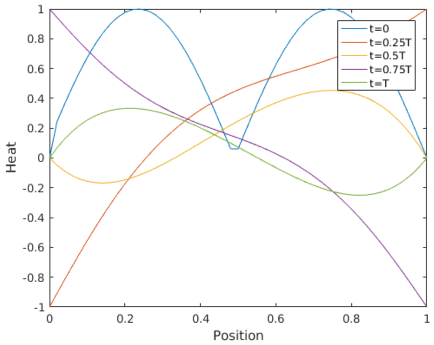
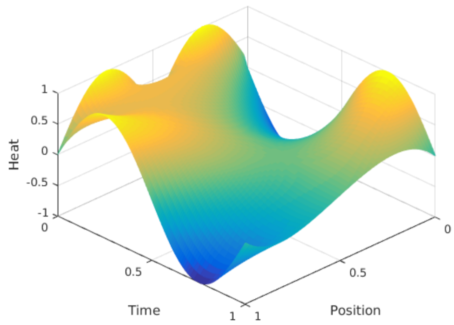


Figure 50: Heat distribution along the wire for unbalanced time-varying boundary conditions.



(a) Heat distribution along the wire with asymmetric time-varying boundary conditions, plotted for particular values of t.



(b) Another view of (a).

Figure 51: A view of the mesh shown in Figure reffig:unbalancedboundaryoverhead, displayed from the side.

4.3. Solutions for various initial conditions

The above procedure demonstrates a functioning script for calculating solutions to the one-dimensional heat equation with arbitrary initial conditions and time-varying boundary conditions. Solutions are requested for time-invariant zero boundary conditions and a number of initial conditions. The latter two, Equation (23d) and Equation (23e), were arbitrarily chosen. Solutions were obtained for the following initial heat distributions:

$$f(x) = \begin{cases} 2x & x < 0.5 \\ 2 - 2x & 0.5 < x \leq 1 \end{cases} \tag{23a}$$

$$f(x) = \sin\left(\frac{2\pi x}{L_{\max}}\right) \tag{23b}$$

$$f(x) = \left|\sin\left(\frac{2\pi x}{L_{\max}}\right)\right| \tag{23c}$$

$$f(x) = \operatorname{sinc}\left(\frac{12\pi x}{L_{\max}}\right) \tag{23d}$$

$$f(x) = -12, \quad 0 < x < 1 \tag{23e}$$

4.3.1. The triangular pulse

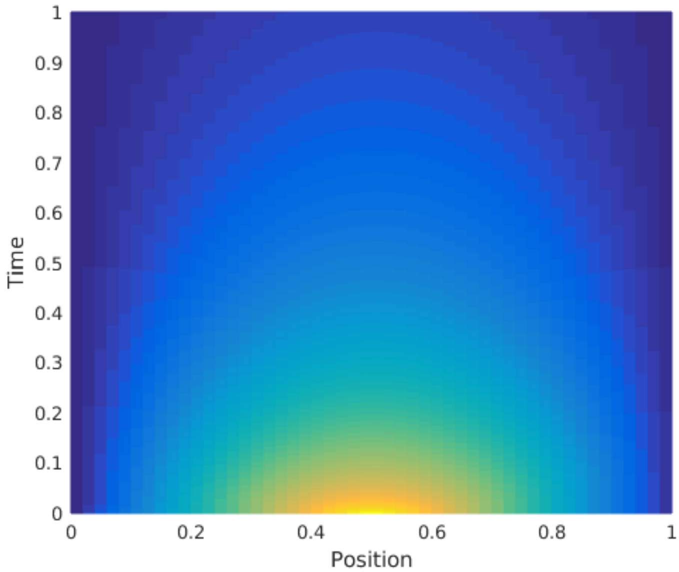
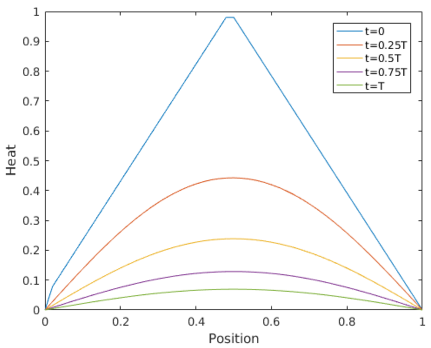
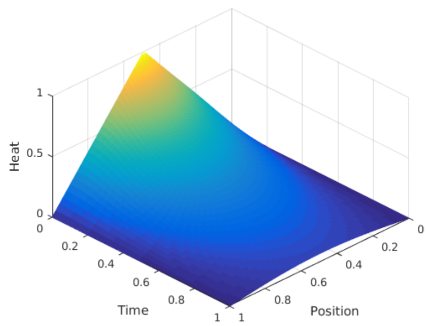


Figure 52: Heat distribution along the length of the wire for initial conditions given by Equation (23a).

Heat diffuses outwards, down the gradient on either side. While initially there is no particular point losing heat faster than another, points towards the extreme ends of the axis will tend to equilibrate faster, yielding second-order effects. Noticeably, heat distribution along the wire forms a parabola for $t > 0$.



(a) Temperature of a wire with initial distribution given by Equation (23a) for selected values of t.



(b) A side view of the mesh shown in Figure 52.

Figure 53: Heat in the wire. The initial condition is a triangular pulse.

4.3.2. A single period of a sinusoid

The sinusoid has a sort of antisymmetry; the local maximum will act as a heat source, while the local minimum will act as a heat sink. The end result is a rapidly-decaying sine wave. Noticeably, by $t = 0.25T$, while the heat is clearly distributed sinusoidally with no phase shift, the amplitude has diminished greatly. This is caused by the long, steep downwards slope between the two local extrema, which moves the heat quickly and causes the wire to very rapidly equilibrate.

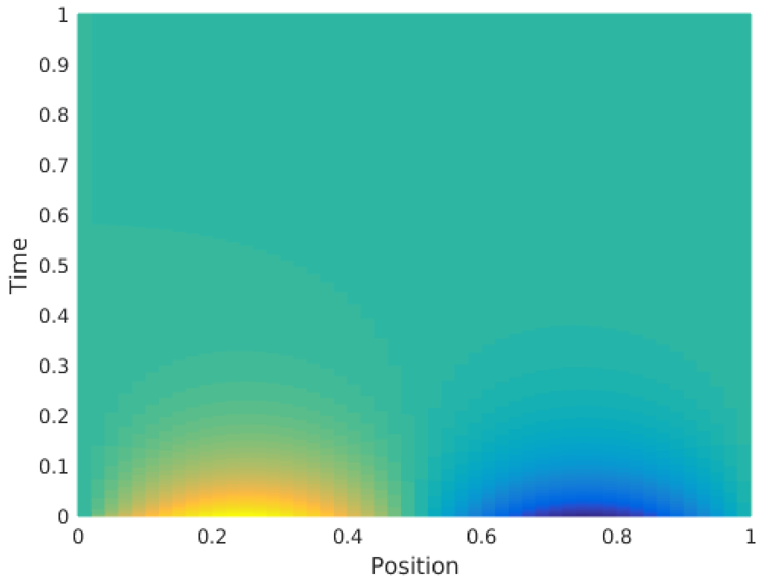
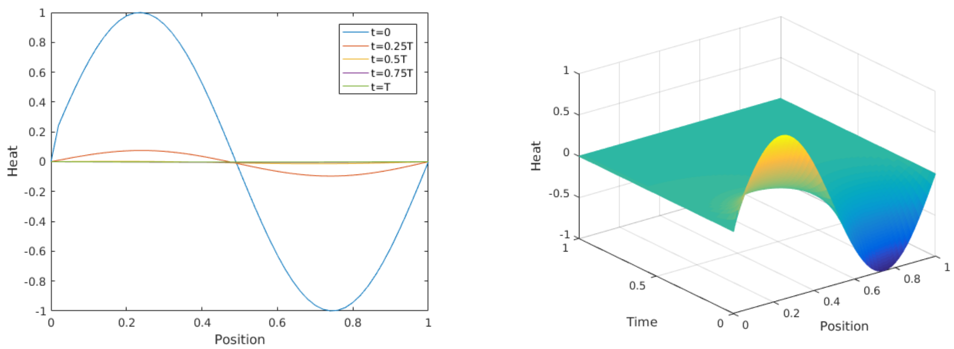


Figure 54: Heat distribution along the length of the wire for initial conditions given by Equation (23b).



(a) Temperature of a wire with initial distribution given by Equation (23b) for selected values of t.

(b) Side view of the mesh for Equation (23b).

Figure 55: Plots for Equation (23b).

4.3.3. The absolute value of one period of a sinusoid

In the case of Equation (23c), the inward gradients of the two initial heat concentrations cause the distribution to move inwards, merging into one larger concentration, which then diffuses outwards.

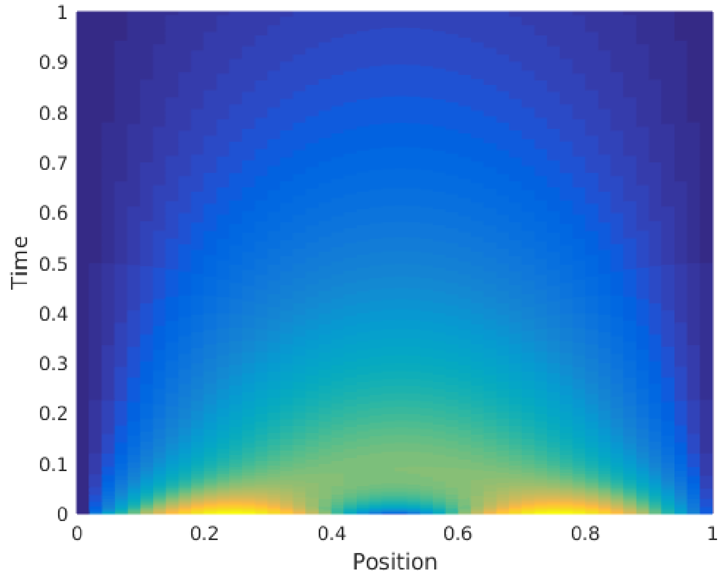
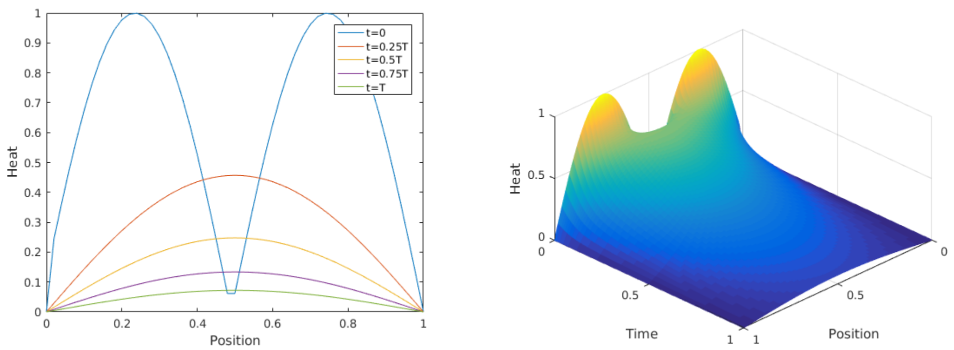


Figure 56: Heat distribution along the length of the wire for initial conditions given by Equation (23c).



(a) Temperature of a wire with initial distribution given by Equation (23c) for selected values of t.

(b) Another view of the mesh from Figure 56.

Figure 57: Heat distribution over a wire for initial conditions given by Equation (23e).

4.3.4. The sinc function

In the case of the sinc function, the behavior appeared quite granular without an increase in the number of spatial divisions. Thus, we took $k = 0.25/25$ and set $N_x = 250$; $r = 0.25$ as before, which provides the following:

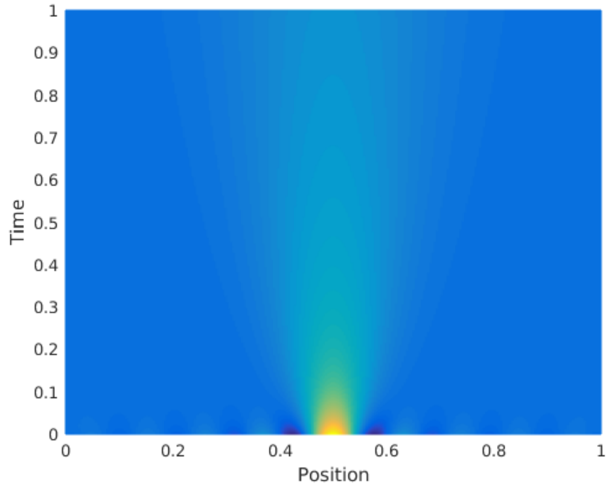
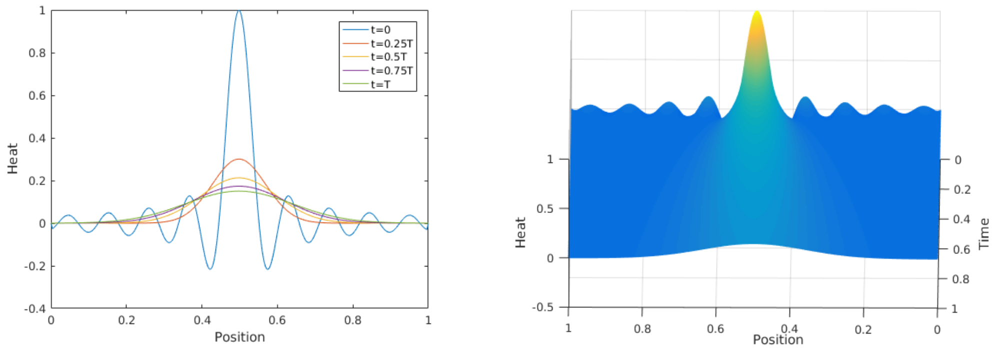


Figure 58: Heat distribution along the length of the wire for initial conditions given by Equation (23d).



(a) Temperature of a wire with initial distribution given by Equation (23d) for selected values of t.

(b) Another view of Figure 58, tilted to make the smaller details of the initial conditions more readily apparent.

Figure 59: Another view of (a).

The sinc function smooths itself out quickly, as there are many instances of high slope along the curve, which cancel each other out rapidly. Heat is concentrated in the center (driven there by "inward" gradients), and spreads out slowly over time.

4.3.5. The potential well

Equation (23e) acts as a potential well; the boundaries are held at zero, while the initial condition is a uniform heat distribution of substantially lower temperature. Heat flows into the wire over time, faster at the sharp gradient near the boundary at the beginning, then into the center of the wire. The heat at any point along the wire looks somewhat like logistic growth, as can be seen in Figure 61b.

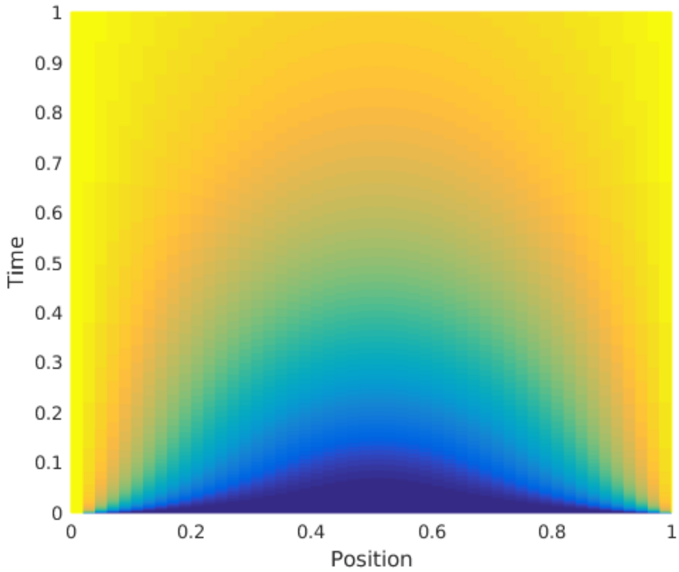
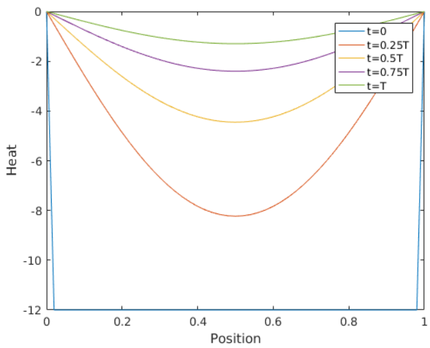
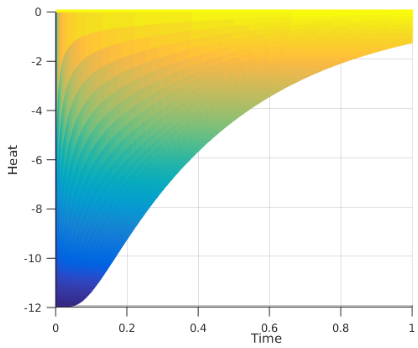


Figure 60: Heat distribution along the length of the wire for initial conditions given by Equation (23d).



(a) Temperature of a wire with initial distribution given by Equation (23d) for selected values of t.



(b) Another view of Figure 60, from the side, highlighting a roughly exponential increase in heat at each position along the length of the wire.

Figure 61: Mesh plots of Equation (23e).

References

[1] John Rozier Cannon. *The one-dimensional heat equation*. 23. Cambridge University Press, 1984.

[2] Peter Deuflhard and Folkmar Bornemann. *Scientific computing with ordinary differential equations*. Vol. 42. Springer Science & Business Media, 2012.

[3] I. Jaimoukha. *System responses*. `https://bb.imperial.ac.uk/bbcswebdav/pid-1015508-dt-content-rid-3468439_1/courses/DSS-EE2\_06-16\_17/Control\_SS\_Examinable\%281\%29.pdf`. Accessed: February 25th, 2017. 2017.

[4] Autar Kaw. *Runge-Kutta 2nd Order Method for Ordinary Differential Equations*. `https://www.saylor.org/site/wp-content/uploads/2011/11/ME205-8.3-TEXT.pdf`. Accessed: February 27th, 2017. 2011.

[5] Frank Kreith and William Z Black. *Basic heat transfer*. Harper & Row New York, 1980.

[6] Editors of Wikipedia. *Runge-Kutta methods*. `https://en.wikipedia.org/wiki/Runge\x{fffd}\x{fffd}\x{fffd}Kutta\_methods`. Accessed: March 11th, 2017. 2017.

A. Full code listing for the RL circuit

A.1. heuns.m

```
function [t,Vout] = heuns(Vin,iL0,h,R,L,ti,tf)
%The heuns.m finds the solution to an ordinary differential equation ... representing an RL-circuit and calculates Vout

a=1/2; % set scaling factor for heuns method
b=1/2;
p1=1;
q11=1;

func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)

N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
Vout(1) = feval(Vin,ti);% calculate initial value of Vout
t(1)=ti; iL(1)=iL0; %set initial values of t_0, and iL at t_0
for j=1:N-1 % loop for N steps
    ttemp=t(j);iLtemp=iL(j); %temporary names
    grad1=feval(func, ttemp,iLtemp);  % gradient at t,iL
    iLp=iLtemp+q11*h*grad1; % calculate iL predictor
    grad2=feval(func, ttemp+p1*h,iLp); % gradient at t+p1*h, iL+q11k1h
    iL(j+1)=iLtemp+h*(a*grad1 + b*grad2); % next value of iL ... calculated from previous values of t,iL
    t(j+1)=ttemp+h; % increase t by stepsize
    Vout(j+1)=feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
end
```

A.2. midpoint.m

```
function [t,Vout] = midpoint(Vin,iL0,h,R,L,ti,tf)
%The midpoint.m finds the solution to an ordinary differential equation
%representing an RL-circuit and calculates Vout

a=0;     % set scaling factor for midpoint method
b=1;
p1=1/2;
q11=1/2;

func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)

N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
Vout(1) = feval(Vin,ti);% calculate initial value of Vout
t(1)=ti; iL(1)=iL0; %set initial values of t_0 and iL at t_0
for j=1:N-1 % loop for N steps
    ttemp=t(j);iLtemp=iL(j); %temporary names
    grad1=feval(func, ttemp,iLtemp);  % gradient at x
    iLp=iLtemp+q11*h*grad1; % calculate iL predictor
    grad2=feval(func, ttemp+p1*h,iLp); % gradient at x+h
    iL(j+1)=iLtemp+h*(a*grad1 + b*grad2); % next value of iL ... calculated from previous values of t,iL
```

```
    t(j+1)=ttemp+h; % increase t by stepsize
    Vout(j+1)=feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
end
```

A.3. ralston.m

```
function [t,Vout] = ralston(Vin,iL0,h,R,L,ti,tf)
%The ralston.m finds the solution to an ordinary differential equation ...
    representing an RL-circuit and calculates Vout

a=1/3;  % set scaling factor for ralston method
b=2/3;
p1=3/4;
q11=3/4;

func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)

N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
Vout(1) = feval(Vin,ti);% calculate initial value of Vout
t(1)=ti; iL(1)=iL0; %set initial values of t_0 and iL at t_0
for j=1:N-1 % loop for N steps
    ttemp=t(j);iLtemp=iL(j); %temporary names
    grad1=feval(func, ttemp,iLtemp);  % gradient at x
    iLp=iLtemp+q11*h*grad1; % calculate iL predictor
    grad2=feval(func, ttemp+p1*h,iLp); % gradient at x+h
    iL(j+1)=iLtemp+h*(a*grad1 + b*grad2); % next value of iL ...
        calculated from previous values of t,iL
    t(j+1)=ttemp+h; % increase t by stepsize
    Vout(j+1)=feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
end
```

A.4. heuns_script.m

```

%set up initial conditions
iL0=0;
ti=0;

%define component values
R=0.5;
L=0.0015;

%there are 15 different inputs
for n=1:15 %define all 15 Vin
if(n<4)
    if(n==1)
        Vina = 5.5;
        Vin=@(t) Vina*exp(0); %define input signal as function of time
        figure
    end
    if(n==2)
        Vina = 3.5;
```

```
        tau = 160e-12;
        Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ...
            of time
    end
    if(n==3)
        Vina = 3.5;
        tau = 160e-6;
        Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ...
            time
    end
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,2,n);
    plot(ti,Vout); % plot initial condition
end
if((n>3)&&(n<16))
    if (n==4)
        figure
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==5)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==6)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==7)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==8)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==9)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==10)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==11)
        Vina = 4.5;
```

```
        T= 1000e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==12)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==13)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==14)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==15)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    nn=n-3;
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,4,nn);
    plot(ti,Vout); % plot initial condition
end

if(n==1) %plots all 15 Vout against time
    h=10e-7; % set step-size
    tf=0.04; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,2,1);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=5.5V')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==2)

    h=10e-7; % set step-size
    tf=0.0001; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,2,2);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=Vin exp(-t^2/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==3)
```

```
    h=10e-7; % set step-size
    tf=0.003; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,2,3);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=Vin exp(-t/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==4)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,1);
    plot(t,Vout); % plot t against Vout
    title('Heuns Vin=4.5sin(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==5)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,2);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sin(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==6)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,3);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sin(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==7)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,4);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sin(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
```

```
if(n==8)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,5);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5square(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==9)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,6);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5square(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==10)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,7);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5square(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==11)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,8);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5square(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==12)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,9);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sawtooth(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
```

```
if(n==13)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,10);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sawtooth(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==14)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,11);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sawtooth(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==15)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
        using heun's method
    subplot(3,4,12);
    plot(t,Vout); % plot Vout against t
    title('Heuns Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end


end
```

A.5. midpoint_script.m

```
%set up initial conditions
iL0=0;
ti=0;

%define component values
R=0.5;
L=0.0015;

for n=1:15 %defines all 15 Vin
if(n<4)
    if(n==1)
        Vina = 5.5;
        Vin=@(t) Vina*exp(0); %define input signal as function of time
        figure
    end
```

```
    if(n==2)
        Vina = 3.5;
        tau = 160e-12;
        Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ... of time
    end
    if(n==3)
        Vina = 3.5;
        tau = 160e-6;
        Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ... time
    end
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,2,n);
    plot(ti,Vout); % plot initial condition
end
if((n>3)&&(n<16))
    if (n==4)
        figure
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ... of time
    end
    if (n==5)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ... of time
    end
    if (n==6)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ... of time
    end
    if (n==7)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ... of time
    end
    if (n==8)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==9)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==10)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ... function of time
    end
```

```
    if (n==11)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==12)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==13)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==14)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==15)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
            function of time
    end
    nn=n-3;
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,4,nn);
    plot(ti,Vout); % plot initial condition
end

if(n==1)   %plots all 15 Vout against time
    h=10e-7; % set step-size
    tf=0.04; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,2,1);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=5.5V')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==2)

    h=10e-7; % set step-size
    tf=0.0001; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,2,2);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=Vin exp(-t^2/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
```

```

if(n==3)

    h=10e-7; % set step-size
    tf=0.003; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,2,3);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=Vin exp(-t/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==4)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,1);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sin(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==5)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,2);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sin(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==6)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,3);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sin(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==7)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,4);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sin(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
```

```
end

if(n==8)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,5);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5square(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==9)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,6);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5square(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==10)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,7);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5square(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==11)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,8);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5square(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==12)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,9);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sawtooth(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
```

```
end

if(n==13)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,10);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sawtooth(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==14)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,11);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sawtooth(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==15)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using midpoint method
    subplot(3,4,12);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

end
```

A.6. ralston_script.m

```
%set up initial conditions
iL0=0;
ti=0;

%define component values
R=0.5;
L=0.0015;

%there are 15 different inputs
for n=1:15  %define 15 Vin
if(n<4)
    if(n==1)
        Vina = 5.5;
        Vin=@(t) Vina*exp(0); %define input signal as function of time
```

```
        figure
    end
    if(n==2)
        Vina = 3.5;
        tau = 160e-12;
        Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ...
            of time
    end
    if(n==3)
        Vina = 3.5;
        tau = 160e-6;
        Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ...
            time
    end
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,2,n);
    plot(ti,Vout); % plot initial condition
end
if((n>3)&&(n<16))
    if (n==4)
        figure
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==5)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==6)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==7)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
            of time
    end
    if (n==8)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==9)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
            function of time
    end
    if (n==10)
        Vina = 4.5;
        T= 450e-6;
```

```
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==11)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==12)
        Vina = 4.5;
        T= 20e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==13)
        Vina = 4.5;
        T= 160e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==14)
        Vina = 4.5;
        T= 450e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ... function of time
    end
    if (n==15)
        Vina = 4.5;
        T= 1000e-6;
        Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ... function of time
    end
    nn=n-3;
    Vout = feval(Vin,ti)-R*iL0;
    subplot(3,4,nn);
    plot(ti,Vout); % plot initial condition
end


if(n==1) %plots all 15 Vout against time
    h=10e-7; % set step-size
    tf=0.04; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ... t using ralston method
    subplot(3,2,1);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=5.5V')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==2)

    h=10e-7; % set step-size
    tf=0.0001; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ... t using ralston method
    subplot(3,2,2);
    plot(t,Vout); % plot Vout against t
```

```
    title('Ralston Vin=Vin exp(-t^2/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==3)

    h=10e-7; % set step-size
    tf=0.003; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,2,3);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=Vin exp(-t/tau)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==4)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,1);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sin(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==5)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,2);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sin(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==6)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,3);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sin(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==7)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,4);
```

```
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sin(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==8)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,5);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5square(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==9)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,6);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5square(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==10)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,7);
    plot(t,Vout); % plot Vout against t
    title('Midpoint Vin=4.5square(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==11)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,8);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5square(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==12)
    h=10e-9; % set step-size
    tf=0.00004; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,9);
```

```
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sawtooth(2pit/T) T = 20e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==13)
    h=10e-7; % set step-size
    tf=0.00032; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,10);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sawtooth(2pit/T) T = 160e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==14)
    h=10e-7; % set step-size
    tf=0.0009; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,11);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sawtooth(2pit/T) T = 450e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

if(n==15)
    h=10e-7; % set step-size
    tf=0.002; % set final value of t
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
        t using ralston method
    subplot(3,4,12);
    plot(t,Vout); % plot Vout against t
    title('Ralston Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end

end
```

A.7. error_script.m

```
iL0=0; % set initial values
ti=0;
R=0.5; % set constant values
L=0.0015;
A=6;
Vina = 6;
T= 150e-6;
Vin=@(t) Vina*cos(2*pi*t/T); % set Vin
c=((A*R*T^2)/(4*pi^2*L^2+R^2*T^2)); %c for exact solution

```

```
tf=0.0003; % set final value of t

for n=1:3

if(n==1)
    h=10e-7; % set step-size
    [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf);
    figure
    subplot(3,2,1);
    plot(t,Vout); % plot heuns Vout against t
    title('Heuns Vin=6cos(2*pi*t/150e-6)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
if(n==2)
    h=10e-7; % set step-size
    [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf);
    figure
    subplot(3,2,1);
    plot(t,Vout); % plot heuns Vout against t
    title('Midpoint Vin=6cos(2*pi*t/150e-6)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
if(n==3)
    h=10e-7; % set step-size
    [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf);
    figure
    subplot(3,2,1);
    plot(t,Vout); % plot heuns Vout against t
    title('Ralston Vin=6cos(2*pi*t/150e-6)')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
end
iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout for exact iL
subplot(3,2,2);
plot(t,Vout_exact); %plot exact solution
    title('Exact solution of the ODE')
    xlabel('Time [s]') % x-axis label
    ylabel('Vout [V]') % y-axis label
error=abs(Vout_exact-Vout); % calculate maximum error over range of x
subplot(3,2,[3,4]);
plot(t,error); % plot error against t
    title('Error against time')
    xlabel('Time [s]') % x-axis label
    ylabel('Error [V]') % y-axis label

subplot(3,2,[5,6]);
h = zeros(1,10);    %initialize arrays
errororder = zeros(1,10);
count = 1;  %initiallize count



if(n==1)
    hi=1e-9;hh=1e-9;hf=1e-7;                  % set initial step-size, ...
        increment in step-size and final step-size value
    h=hi:hh:hf;
```

```
    Nh=round((hf-hi)/hh)+1;     % number of steps=(interval size of ...
        step-size)/(increment in step-size)
    for count=1:Nh
        [t,Vout]=heuns(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
        iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
        Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
            exact solution iL
        errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
            maximum error over range of x
        hold on;
    end
    hold off;
    plot(log(h),log(errororder));
    gradheuns=polyfit(log(h), log(errororder),1);
end

if(n==2)
    hi=1e-9;hh=1e-9;hf=1e-7;                 % set initial step-size, ...
        increment in step-size and final step-size value
    h=hi:hh:hf;
    Nh=round((hf-hi)/hh)+1;      % number of steps=(interval size of ...
        step-size)/(increment in step-size)
    for count=1:Nh
        [t,Vout]=midpoint(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
        iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
        Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
            exact solution iL
        errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
            maximum error over range of x
        hold on;
    end
    hold off;
    plot(log(h),log(errororder));
    gradmidpoint=polyfit(log(h), log(errororder),1);
end

if(n==3)
    hi=1e-9;hh=1e-9;hf=1e-7;                 % set initial step-size, ...
        increment in step-size and final step-size value
    h=hi:hh:hf;
    Nh=round((hf-hi)/hh)+1;      % number of steps=(interval size of ...
        step-size)/(increment in step-size)
    for count=1:Nh
        [t,Vout]=ralston(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
        iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
/(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
        Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
            exact solution iL
        errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
            maximum error over range of x
        hold on;
    end
    hold off;
    plot(log(h),log(errororder)); % plot log log graph
    gradralston=polyfit(log(h), log(errororder),1);% calculate gradient
end
    title('log of maximum error against log of h')
    ylabel('log error max') % x-axis label
```

```
    xlabel('log h') % y-axis label
end
```

B. Full Code listing for RLC-circuit (Exercise3)

B.1. RK4second.m

```
function [xn, yn] = RK4second(funcx, funcy, h, ti, xi, yi)
%RK4second computes y(i+1) and x(i+1) using the Runge-Kutta 3/8 algorithm
%   xn refers to x(i+1) and yn refers to y(i+1)
%   funcx computes the derivative of x (dx/dt) at a point (ti, xi, yi)
%   funcy computes the derivative of y (dy/dt) at a point (ti, xi ,yi)

%calculate coefficients (predicted gradients) at ti, ti+h/3, ti+2h/3, ti+h
%using Runge-Kutta 3/8
k1x = feval(funcx, ti, xi, yi);
k1y = feval(funcy, ti, xi, yi);
k2x = feval(funcx, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
k2y = feval(funcy, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
k3x = feval(funcx, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
k3y = feval(funcy, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
k4x = feval(funcx, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
k4y = feval(funcy, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);

%obtain phix and phiy by taking weighted average of obtained gradients
phix = (k1x + 3*k2x + 3*k3x + k4x)/8;
phiy = (k1y + 3*k2y + 3*k3y + k4y)/8;

%use phi-values as approximated gradients for x and y
xn = xi + h*phix; %calculate x(i+1)
yn = yi + h*phiy; %calculate y(i+1)
end
```

B.2. RLC_script.m

RLC_script.m is written with the following matlab code:

```
%The RLC_script calculates the voltage across R (Vout) for a given ...
    input signal(Vin)
%The RLC Circuit script is finding the solution to a second order ...
    differential equation representing an RLC-circuit

%set up initial conditions
q0 = 500*10^(-9); %[C]; capacitor charge at t=0
i0 = 0; %current at t=0
t0 = 0; %set up starting time
h = 0.000001; %[s]; set up step-size for Runge-Kutta 3/8
tf = 0.06; %[s]; define endpoint of time-interval

%define component values
R = 280; %resistance equals 280 Ohm
```

```
C = 4*10^(-6); %Capacitor value is 4 microFarad
L=600*10^(-3); %Inductance is 600 milliHenry

funcvin = @(t) 5; %define input signal (step-input, tf=0.06) as ...
    function of time

%the other input functions with their corresponding tf-value are stated
%below:
%funcvin = @(t) 5*exp(-t^2/(3*10^-6)); (impulse, tf=0.06)
%funcvin = @(t) 5*square(2*pi*t*5); (square, f=5Hz, tf=0.5)
%funcvin = @(t) 5*square(2*pi*t*110); (square, f=110Hz, tf=0.05)
%funcvin = @(t) 5*square(2*pi*t*500); (square, f=500Hz, tf=0.03)
%funcvin = @(t) 5*sin(2*pi*t*5); (sine, f=5Hz, tf=0.5)
%funcvin = @(t) 5*sin(2*pi*t*110); (sine, f=110Hz, tf = 0.05)
%funcvin = @(t) 5*sin(2*pi*t*500); (sine, f=500Hz, tf = 0.035)

%set up coupled first-order equations
funcq = @(t, q, i) i; %gradient of q at time t (=i(t))
funci = @(t, q, i)   (feval(funcvin, t) - R*i - 1/C * q)/L; %funci ...
    calculates di/dt at time t

N = round((tf-t0)/h); %calculate number of steps to reach tf

%set up arrays to store results
q = zeros(1,N);
i = zeros(1,N);
t = zeros(1,N);

%first element of each array is equal to corresponding initial condition
q(1) = q0;
i(1) = i0;
t(1) = t0;

%use for-loop to iterate through arrays
%RK4second uses Runge-Kutta-3/8 algorithm to calculate next values for q
%and i as t is increased by h after each iteration
for j = 1 : N-1
   [q(j+1),i(j+1)] = RK4second(funcq, funci, h, t(j), q(j), i(j));
   t(j+1) = t(j) + h;
end

vout = i*R; %obtain Vout(t)(voltage across R) using Ohms Law
vin = arrayfun(funcvin, t); %calculate Vin(t)

figure;
plot(t, q); %plot q(t) as a function of t
title('Capacitor Charge (q_{C}(t))');
xlabel('Time [s]');
ylabel('Charge [C]');

figure;
plot(t, vout); %plot Vout(t) as a function of t
title('Output Voltage (V_{out}(t)=v_{R}(t))');
xlabel('Time [s]');
ylabel('Voltage [V]');


figure;
plot(t, vin); %plot Vin(t) as a function of t
title('Input Signal (V_{in}(t))');
```

```
xlabel('Time [s]');
ylabel('Voltage [V]');
```

C. Full code listing for the finite differences method

C.1. finite_script.m

```
L = 1.;       % wire length
T = 1.;       % max simulation time
Nt = 2500;    % number of timesteps
Nx = 50;      % number of spacial divisions
dt = T / Nt; % increment through time
dx = L / Nx; % increment through space

% conductivity parameter
r = 0.25 * dt / (dx*dx);
if r > 0.5
    % von neumann stability criterion has been violated
    disp('warning: for stability, r ≤ 1/2');
    r
end

% lambdas to compute initial wire heat distribution
initialcond = @(x, L, Nx) abs(sin(2*pi*x/(Nx+1)));
% initialcond = @(x, L, Nx) sin(2 * pi * x / (Nx+1) / L);
% initialcond = @(x, L, Nx) triangularPulse(0.0, L, x/(Nx+1));
% initialcond = @(x, L, Nx) sinc(6 * pi * ((x - (Nx+1)/2) / (Nx+1)));
% initialcond = @(x, L, Nx) -12.0;

% lambdas to compute heat at edge of wire for x=0
leftbound = @(t, Nt) sin(2*pi*t/Nt);
% leftbound = @(t, Nt) -sin(2*pi*t/Nt);
% leftbound = @(t, Nt) 0.0;

% lambdas to compute heat at edge of wire for x=L
rightbound = @(t, Nt) sin(2*pi*t/Nt);
% rightbound = @(t, Nt) 0.0;

% compute initial conditions and make an x-axis for plotting
for i = 1:Nx+1
    x(i) = (i-1)*dx;
    u(i,1) = initialcond(i, L, Nx);
end

% compute boundary conditions and make a time-axis for plotting
for t = 1:Nt+1
    u(1,t) = leftbound(t, Nt);
    u(Nx+1, t) = rightbound(t, Nt);
    time(t) = (t-1) * dt;
end

% go-time; iterate over entire matrix
for t=1:Nt
    for i = 2:Nx
        % directly from the expression we obtained
```

```
        u(i, t+1) = (1-2*r) * u(i,t) + r*u(i+1, t) + r*u(i-1,t);
    end
end

figure(1)
% plot snapshots of heat distribution for [0, 0.25, 0.5, 0.75, 1] * T
% even time spacing means this plot indicates how fast heat is diffusing
% too many more or fewer would mean the plot would be cluttered, IMO
plot(x,u(:,int32(0.00*N_t)+1),'-', ...
     x,u(:,int32(0.25*N_t)),'-',   ...
     x,u(:,int32(0.50*N_t)),'-',   ...
     x,u(:,int32(0.75*N_t)),'-',   ...
     x,u(:,int32(N_t)),'-')
legend('t=0', 't=0.25T', 't=0.5T', 't=0.75T', 't=T')
figure(2)
% 3d plot, with a space and a time axis; I prefer this for showing ...
    variation,
% _especially_ with an overhead view
mesh(x,time,u')
```