

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

1.	Exerc	cise 1: RL circuit	6
	1.1.	The RL circuit	6
	1.2.	Implementation of second-order Runge-Kutta methods	$\overline{7}$
	1.3.	Analytical solution	9
		1.3.1. Step signal with amplitude $\overline{V_{in}} = 5.5V$	10
		1.3.2. Impulse signal and decay	11
		1.3.3 Sine wave input	14
		1.3.4 Square wave input	17
		1.3.5. Sawtooth wave input \ldots \ldots \ldots \ldots \ldots \ldots	18
2	F		20
Ζ.	Exerc	cise 2: Error analysis	20
	2.1.	Exact solution of the ODE	20
	2.2.	Error analysis	21
3.	Exerc	cise 3: RLC circuit	26
	3.1.	RK4	26
	3.2.	RLC circuit	28
	3.3.	Output voltage for different input signals	31
		3.3.1. Analytical solution	31
		3.3.2 Step Signal	32
		3.3.3 Impulsive signal with decay	36
		3.3.4 Square Wayes with different frequencies	39
		3.3.5 Sine Waves with different frequencies	42
		5.5.6. Since waves with different frequencies	74
4.	Exerc	cise 4: Diffusion of heat along a wire	45
4.	Exerc 4.1.	cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45
4.	Exerc 4.1. 4.2.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 45
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 45 50
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wireThe heat equation, and obtaining a calculable expressionImplementation of the finite-differences methodSolutions for various initial conditions4.3.1. The triangular pulse	45 45 45 50 51
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expressionImplementation of the finite-differences methodSolutions for various initial conditions4.3.1. The triangular pulse4.3.2. A single period of a sinusoid	45 45 50 51 52
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53
4.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wireThe heat equation, and obtaining a calculable expressionImplementation of the finite-differences methodSolutions for various initial conditions4.3.1. The triangular pulse4.3.2. A single period of a sinusoid4.3.3. The absolute value of one period of a sinusoid4.3.4. The sinc function4.3.5. The potential well	45 45 50 51 52 53 53 54
4. Re	Exerci 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 54 56
4. Re	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 54 56
4. Re A.	Exerc 4.1. 4.2. 4.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 54 56 57
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 54 56 57 57
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression Implementation of the finite-differences method Solutions for various initial conditions 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well Code listing for the RL circuit heuns.m midpoint.m	 45 45 50 51 52 53 53 54 56 57 57 57 57
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2. A.3.	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression Implementation of the finite-differences method Solutions for various initial conditions 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well Code listing for the RL circuit heuns.m midpoint.m midpoint.m	45 45 50 51 52 53 53 54 56 57 57 57 58
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2. A.3. A.4.	cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 53 54 56 57 57 57 58 58
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2. A.3. A.4. A.5.	cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression	45 45 50 51 52 53 53 54 56 57 57 57 58 58 58 63
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2. A.3. A.4. A.5. A.6.	cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression Implementation of the finite-differences method Solutions for various initial conditions 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well code listing for the RL circuit heuns.m midpoint.m ralston.m midpoint_script.m midpoint_script.m	45 45 50 51 52 53 53 54 56 57 57 57 58 58 63 68
4. Re A.	Exerc 4.1. 4.2. 4.3. ferenc Full c A.1. A.2. A.3. A.4. A.5. A.6. A.7.	cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression Implementation of the finite-differences method Solutions for various initial conditions 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well ces code listing for the RL circuit heuns.m midpoint.m ralston.m heuns_script.m midpoint_script.m ralston_script.m	45 45 50 51 52 53 53 54 56 57 57 57 58 63 68 73
4. Re A.	Exerci 4.1. 4.2. 4.3. ference Full c A.1. A.2. A.3. A.4. A.5. A.6. A.7. Full C	Cise 4: Diffusion of heat along a wire The heat equation, and obtaining a calculable expression Implementation of the finite-differences method Solutions for various initial conditions Solutions for various initial conditions 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well 4.3.6. The potential well 4.3.7. The sinc function 4.3.8. The sinc function 4.3.9. The potential well 4.3.1. The triangular pulse 4.3.2. A single period of a sinusoid 4.3.3. The absolute value of one period of a sinusoid 4.3.4. The sinc function 4.3.5. The potential well 4.3.6. The potential well 4.3.7. The since for the RL circuit heuns.m midpoint.m ralston.m heuns_script.m ralston_script.m ralston_script.m error_script.m code listing for RLC-circuit (Exercise3)	45 45 50 51 52 53 53 54 56 57 57 57 58 63 68 73 76

	B.2. RLC_script.m	76
С.	Full code listing for the finite differences method	

List of Figures

1.	RL-Circuit	6
2.	$\overline{V_{in}} = 5.5 \dots \dots$	10
3.	V_{out} against time when $\overline{V_{in}} = 5.5V$	10
4.	Analytically obtained output voltage when $\overline{V_{in}} = 5.5V$	11
5.	$V_{in} = 3.5e^{-\frac{t^2}{160(\mu)^2}}$	12
6.	V_{out} against time when $V_{in} = 3.5e^{-\frac{t^2}{160(\mu)^2}}$	12
7.	$V_{in} = 3.5e^{-\frac{\iota}{160\mu}} \dots $	13
8.	V_{out} against time when $V_{in} = 3.5e^{-\frac{t}{160\mu}}$	13
9.	Analytically obtained output voltage when $V_{in} = 3.5e^{-\frac{\iota}{160\mu}}$	14
10.	$V_{in} = 4.5 \sin(\frac{2\pi t}{20\mu}) \dots \dots \dots \dots \dots \dots \dots \dots \dots $	14
11.	Heun's method $V_{in} = 4.5 sin(\frac{2\pi t}{T})$	15
12.	Midpoint method $V_{in} = 4.5 \sin(\frac{2\pi t}{T})$	15
13.	Heun's method $V_{in} = 4.5 sin(\frac{2\pi t}{T})^{2}$	15
14.	$V_{in} = 4.5 \text{square}(\frac{2\pi t}{20\mu}) \dots \dots \dots \dots \dots \dots \dots \dots \dots $	17
15.	Heun's method $V_{in} = 4.5$ square $(\frac{2\pi t}{T})$	17
16.	Midpoint method $V_{in} = 4.5$ square $(\frac{2\pi t}{T})$	17
17.	Ralston's method $V_{in} = 4.5$ square $\left(\frac{2\pi t}{T}\right)$	18
18.	$V_{in} = 4.5 \text{ sawtooth} \left(\frac{2\pi t}{20\mu}\right) \dots \dots$	18
19.	Heun's method $V_{in} = 4.5$ sawtooth $(\frac{2\pi t}{T})$	19
20.	Midpoint method $V_{in} = 4.5$ sawtooth $\left(\frac{2\pi t}{T}\right)$	19
21.	Heun's method $V_{in} = 4.5$ sawtooth $\left(\frac{2\pi t}{T}\right)^{-1}$	19
22.	V_{out} against time, $V_{in} = 6cos(\frac{2\pi}{150\mu}t)^{-1}$	22
23.	V_{out} against time, $V_{in} = 6\cos(\frac{2\pi}{150\mu}t)$	22
24.	V_{out} against time, $V_{in} = 6\cos(\frac{2\pi}{150\mu}t)$	22
25.	Heun's method error against time	23
26.	Midpoint method error against time	24
27.	Ralston's method error against time	24
28.	Heun's method log maxerror against log h	24
29.	Midpoint method log maxerror against $\log h$	24
30.	Ralston's method log(maxerror) against log h	25
31.	Butcher tableau for Runge-Kutta $3/8$ [6]	27
32.	RLC-Circuit	28
33.	Step Signal with amplitude 5 V	33
34.	Response of capacitor charge for input step signal	33
35.	Output voltage for input step signal	34
36.	Analytically obtained output voltage for input step signal	35
37.	Impulsive signal with decay	36
38.	Capacitor charge for input impulsive signal with decay	37
39.	Output voltage for input impulsive signal with decay	38
40.	Unit Impulse Response of the RLC-circuit	39
41.	Frequency Response of the RLC-Circuit	40
42.	RLC-Circuit response to square wave of $f = 5 \text{ Hz} \dots \dots \dots \dots$	40
43.	RLC-Circuit response to square wave of $f = 110 \text{ Hz} \dots \dots \dots \dots$	41
44.	RLC-Circuit response to square wave of $f = 500 \text{Hz} \dots \dots \dots \dots$	42

45.	RLC-Circuit response to sine wave of $f = 5 \text{ Hz}$	43
46.	RLC-Circuit response to sine wave of $f = 110 \text{Hz}$	43
47.	RLC-Circuit response to sine wave of $f = 500 \text{Hz}$	44
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.	48
49.	Heat distribution along the wire through time.	48
50.	Heat distribution along the wire for unbalanced time-varying boundary	
	conditions	49
51.	A view of the mesh shown in Figure reffig:unbalancedboundaryoverhead,	
	displayed from the side.	50
52.	Heat distribution along the length of the wire for initial conditions given	
	by Equation (23a). \ldots \ldots \ldots \ldots \ldots \ldots \ldots	51
53.	Heat in the wire. The initial condition is a triangular pulse	51
54.	Heat distribution along the length of the wire for initial conditions given	
	by Equation $(23b)$	52
55.	Plots for Equation (23b).	52
56.	Heat distribution along the length of the wire for initial conditions given	
	by Equation (23c). \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	53
57.	Heat distribution over a wire for initial conditions given by Equation (23e).	53
58.	Heat distribution along the length of the wire for initial conditions given	
	by Equation $(23d)$	54
59.	Another view of (a).	54
60.	Heat distribution along the length of the wire for initial conditions given	
	by Equation (23d). \ldots \ldots \ldots \ldots \ldots \ldots \ldots	55
61.	Mesh plots of Equation $(23e)$	55

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:

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

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

$$d \qquad V_{in}(t) - Ri_L(t)$$
(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 $V_{in}(t)$.

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

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

The values of the components are given as:

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

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.

$$y_{i+1} = y_i + h\phi(x_i, y_i, h)$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + p_1h, y_i + q_{11}k_1h)$$
$$\phi = ak_1 + bk_2$$

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

```
1 func = @(t,iL) (feval(Vin,t) - R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)
2
3 N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
4 % set up arrays
5 t = zeros(1,N);
6 iL=zeros(1,N);
7 Vout=zeros(1,N); % set up arrays
8 Vout(1) = feval(Vin,ti); % calculate initial value of Vout
9 t(1)=ti;
10 iL(1)=iL0; % set initial values of t_0, and iL at t_0
```

The script starts by defining a function for evaluating $\frac{d}{dt}i_L(t)$ 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
1
       ttemp = t(j);
\mathbf{2}
       iLtemp = iL(j); %temporary names
3
      grad1 = feval(func, ttemp, iLtemp); % gradient at t, iL
4
       iLp = iLtemp + q11*h*grad1; % calculate iL predictor
5
6
       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 ...
7
           calculated from previous values of t, iL
       t(j+1) = ttemp+h; % increase t by stepsize
8
       Vout(j+1) = feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
9
10
  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 $y + q_{11}k_1h$. 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:

$$a + b = 1$$
$$bp_1 = \frac{1}{2}$$
$$bq_{11} = \frac{1}{2}$$

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:

```
1 %set up initial conditions
2 iL0=0;
3 ti=0;
4
5 %define component values
6 R=0.5;
7 L=0.0015;
8
9 Vina = 5.5;
10 Vin=@(t) Vina*exp(0); %define input signal as function of time
11 figure
12 Vout = feval(Vin,ti)-R*iL0;
13 subplot(3,4,nn);
14 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.

```
1 h=10e-7; % set step-size
2 tf=0.04; % set final value of t
3 [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
using heun's method
4 plot(t,Vout); % plot Vout against t
5 title('Heuns Vin=5.5V')
6 xlabel('Time [t]') % x-axis label
7 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 $V_{out} = V_{in}(t) - Ri_L(t)$,

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





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.



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.

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

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

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

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

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

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

$$V_{out}(t) = 5.5(e^{-\frac{1000}{3}t})u(t)$$
(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: 1 Vina = 3.5; 2 tau = 160e-12; 3 Vin=@(t) Vina*exp(-t^2/tau);



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.



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:

1 Vina = 3.5; 2 tau = 160e-6; 3 Vin=@(t) Vina*exp(-t/tau);



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



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.

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

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

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

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

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

$$V_{out}(t) = 3.697183e^{-\frac{1}{160\mu}t} - 0.197183e^{-\frac{100}{3}t}$$
(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 $20\mu s$.

1 Vina = 4.5; 2 T= 20e-6; 3 Vin=@(t) Vina*sin(2*pi*t/T);

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



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



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



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



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

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

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

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

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

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

$$|H(j\frac{2\pi}{20\mu})| = 1.000$$
$$\angle H(j\frac{2\pi}{20\mu}) = 0.0608^{\circ}$$

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 160μ s:

$$H(j\frac{2\pi}{160\mu}) = 1.0 + 0.00849i$$
$$|H(j\frac{2\pi}{160\mu})| = 1.000$$
$$\angle H(j\frac{2\pi}{160\mu}) = 0.486^{\circ}$$

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 450μ s:

$$H(j\frac{2\pi}{450\mu}) = 0.999 + 0.0239i$$
$$|H(j\frac{2\pi}{450\mu})| = 0.9997$$
$$\angle H(j\frac{2\pi}{450\mu}) = 1.368^{\circ}$$

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 1000μ s:

$$H(j\frac{2\pi}{1000\mu}) = 0.997 + 0.0529i$$
$$|H(j\frac{2\pi}{1000\mu})| = 0.9986$$
$$\angle H(j\frac{2\pi}{1000\mu}) = 3.037^{\circ}$$

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:

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

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



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



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

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:

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

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





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



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



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

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.

$$L\frac{d}{dt}i_{L}(t) + Ri_{L}(t) = V_{in}(t)$$

$$\frac{d}{dt}i_{L}(t) + \frac{R}{L}i_{L}(t) = \frac{1}{L}V_{in}(t)$$

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

$$e^{\frac{R}{L}t}\frac{d}{dt}i_{L}(t) + e^{\frac{R}{L}}t\frac{R}{L}i_{L}(t) = \frac{1}{L}e^{\frac{R}{L}t}V_{in}(t)$$

$$\frac{d}{dt}(e^{\frac{R}{L}t}i_{L}(t)) = \frac{1}{L}e^{\frac{R}{L}t}V_{in}(t)$$

$$= \frac{1}{L}e^{\frac{R}{L}t}A\cos(\frac{2\pi}{T}t)$$

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

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

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

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

Then plug back to the i_L Equation (9),

$$i_L(t) = e^{-\frac{R}{L}t} \left(\frac{1}{L}A \int e^{\frac{R}{L}t} \cos(\frac{2\pi}{T}t)dt + c\right)$$
$$= \frac{2A\pi T L \sin(\frac{2\pi}{T}t) + ART^2 \cos(\frac{2\pi}{T}t)}{4\pi^2 L^2 + R^2 T^2} + ce^{-\frac{R}{L}t}$$

Finally calculate c from the initial condition given.

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

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

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

2.2. Error analysis

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

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

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

```
1 tf=0.0003; % set final value of t
2 h=10e-7; % set step-size
3 [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf);
4 figure
5 subplot(3,2,1);
6 plot(t,Vout); % plot heuns Vout against t
7 title('Heuns Vin=6cos(2*pi*t/150e-6)')
8 xlabel('Time [s]') % x-axis label
9 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))
1
  /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
\mathbf{2}
  Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout for exact iL
3
  subplot(3,2,2);
4
  plot(t,Vout_exact); %plot exact solution
5
       title('Exact solution of the ODE')
6
       xlabel('Time [s]') % x-axis label
7
       ylabel('Vout [V]') % y-axis label
8
  error=abs(Vout_exact-Vout); % calculate maximum error over range of x
9
  subplot(3,2,[3,4]);
10
  plot(t,error); % plot error against t
11
       title('Error against time')
12
       xlabel('Time [s]') % x-axis label
13
       ylabel('Error [V]') % y-axis label
14
```

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





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



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

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<sup>2</sup>)/(4*pi<sup>2</sup>*L<sup>2</sup>+R<sup>2</sup>*T<sup>2</sup>)); %c for exact solution
2
  subplot(3,2,[5,6]);
  h = zeros(1, 10);
                          %initialize arrays
3
  errororder = zeros(1, 10);
4
  count = 1; %initialize count
5
6
7
       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;
8
       Nh=round((hf-hi)/hh)+1;
                                              % number of steps=(interval size ...
9
           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.

```
1
       for count=1:Nh
           [t,Vout]=heuns(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
2
           iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
3
  /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
4
           Vout_exact=feval(Vin,t)-R*iL_exact; % calculate exact solution ...
5
               as array
           errororder(count) = max(abs(Vout_exact-Vout)); % calculate ...
6
               maximum error over range of x
7
           hold on;
8
       end
       hold off;
9
       plot(log(h), log(errororder)); % plot log log graph
10
       gradheuns=polyfit(log(h), log(errororder),1); % calculate gradient
11
12
       title('log of maximum error against log of h')
13
       ylabel('log error max') % x-axis label
14
       xlabel('log h') % y-axis label
15
```

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 $\log(\max ror)$ 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 MAT-LAB 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

$$\begin{aligned} k_{1x} &= funcx(t_i, x_i, y_i);\\ k_{1y} &= funcy(t_i, x_i, y_i);\\ k_{2x} &= funcx(t_i + \frac{h}{3}, x_i + \frac{h}{3}k_{1x}, y_i + \frac{h}{3}k_{1y});\\ k_{2y} &= funcy(t_i + \frac{h}{3}, x_i + \frac{h}{3}k_{1x}, y_i + \frac{h}{3}k_{1y});\\ k_{3x} &= funcx(t_i + \frac{2}{3}h, x_i - \frac{1}{3}hk_{1x} + hk_{2x}, y_i - \frac{1}{3}hk_{1y} + hk_{2y});\\ k_{3y} &= funcy(t_i + \frac{2}{3}h, x_i - \frac{1}{3}hk_{1x} + hk_{2x}, y_i - \frac{1}{3}hk_{1y} + hk_{2y});\\ k_{4x} &= funcx(t_i + h, x_i + hk_{1x} - hk_{2x} + hk_{3x}, y_i + hk_{1y} - hk_{2y} + hk_{3y});\\ k_{4y} &= funcy(t_i + h, x_i + hk_{1x} - hk_{2x} + hk_{3x}, y_i + hk_{1y} - hk_{2y} + hk_{3y}). \end{aligned}$$

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).



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:

```
1 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
2
      xn refers to x(i+1) and yn refers to y(i+1)
3 %
      funcx computes the derivative of x (dx/dt) at a point (ti, xi, yi)
4
  8
  8
      funcy computes the derivative of y (dy/dt) at a point (ti, xi, yi)
5
6
  %calculate coefficients (predicted gradients) at ti, ti+h/3, ti+2h/3, ti+
7
8 %using Runge-Kutta 3/8
9 k1x = feval(funcx, ti, xi, yi);
10 kly = feval(funcy, ti, xi, yi);
11 k2x = feval(funcx, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
12 k2y = feval(funcy, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
13 k3x = feval(funcx, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
14 k3y = feval(funcy, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
15 k4x = feval(funcx, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
16 k4y = feval(funcy, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
17
18 %obtain phix and phiy by taking weighted average of obtained gradients
19 phix = (k1x + 3 k2x + 3 k3x + k4x)/8;
  phiy = (k1y + 3 k2y + 3 k3y + k4y)/8;
20
21
22 %use phi-values as approximated gradients for x and y
23 xn = xi + h*phix; %calculate x(i+1)
24 yn = yi + h*phiy; %calculate y(i+1)
25 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:

$$v_{L}(t) + v_{R}(t) + v_{C}(t) = V_{in}(t)$$

$$L\frac{d}{dt}i_{L}(t) + Ri_{L}(t) + \frac{1}{C}\int_{0}^{t}i_{L}(t) dt = V_{in}(t)$$

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

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

The values of the components are given as:

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

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.

```
1 %The RLC_script calculates the voltage across R (Vout) for a given ...
      input signal(Vin)
2 %The RLC Circuit script is finding the solution to a second order ...
      differential equation representing an RLC-circuit
3
4 %set up initial conditions
5 q0 = 500*10^(-9); %[C]; capacitor charge at t=0
6 i0 = 0; % current at t=0
7 t0 = 0; %set up starting time
8 h = 0.000001; %[s]; set up step-size for Runge-Kutta 3/8
  tf = 0.06; %[s]; define endpoint of time-interval
9
10
11 %define component values
12 R = 280; %resistance equals 280 Ohm
13 C = 4*10<sup>(-6)</sup>; %Capacitor value is 4 microFarad
14 L=600*10<sup>(-3)</sup>; %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:

$$L\frac{d}{dt}i_{L}(t) + Ri_{L}(t) + \frac{1}{C}q_{C}(t) = V_{in}(t)$$
(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):

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

 $V_{in}(t)$ 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.

```
1 funcvin = @(t) 5; %define input signal (step-input, tf=0.06) as ...
      function of time
2
3 %the other input functions with their corresponding tf-value are stated
4 %below:
5 %funcvin = @(t) 5*exp(-t^2/(3*10^-6)); (impulse, tf=0.06)
  funcvin = Q(t) 5 + square(2 + pi + t + 5); (square, f=5Hz, tf=0.5)
6
  %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)
10 %funcvin = @(t) 5*sin(2*pi*t*110); (sine, f=110Hz, tf = 0.05)
11 %funcvin = @(t) 5*sin(2*pi*t*500); (sine, f=500Hz, tf = 0.035)
12
13 %set up coupled first-order equations
14 funcq = @(t, q, i) i; %gradient of q at time t (=i(t))
15 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 .

1 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.

```
1 %set up arrays to store results
2 q = zeros(1,N);
3 i = zeros(1,N);
4 t = zeros(1,N);
5
6 %first element of each array is equal to corresponding initial condition
7 q(1) = q0;
8 i(1) = i0;
9 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 $funcq (\triangleq funcx)$, $funci (\triangleq funcy)$, $h (\triangleq h)$, $t_j (\triangleq t)$, $q_j (\triangleq x)$ and $i_j (\triangleq y)$ as input arguments. The output arguments of RK4second are stored at the right positions of the arrays.

```
1 %use for-loop to iterate through arrays
2 %RK4second uses Runge-Kutta-3/8 algorithm to calculate next values for q
3 %and i as t is increased by h after each iteration
4 for j = 1 : N-1
5  [q(j+1),i(j+1)] = RK4second(funcq, funci, h, t(j), q(j), i(j));
6  t(j+1) = t(j) + h;
7 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 .

```
1 vout = i*R; %obtain Vout(t)(voltage across R) using Ohms Law
2 vin = arrayfun(funcvin, t); %calculate Vin(t)
3
4 figure;
5 plot(t, q); %plot q(t) as a function of t
6 title('Capacitor Charge (q_{C}(t))');
7 xlabel('Time [s]');
8 ylabel('Charge [C]');
9
10 figure;
11 plot(t, vout); %plot Vout(t) as a function of t
12 title('Output Voltage (V_{out}(t)=v_{R}(t))');
13 xlabel('Time [s]');
14 ylabel('Voltage [V]');
15
16
17 figure;
18 plot(t, vin); %plot Vin(t) as a function of t
19 title('Input Signal (V_{in}(t))');
20 xlabel('Time [s]');
21 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:

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

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

$$\bar{q}_C(s) = \frac{LCq_C(0)s + RCq_C(0) + C\bar{V}_{in}(s)}{LCs^2 + RCs + 1}.$$
(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 $\overline{q}_C(s)$ by applying the derivative Laplace-transform rule and multiplying by R. This yields:

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

3.3.2. Step Signal

The first input signal is a step signal with amplitude $V_{in} = 5$ V. It is implemented in the RLC_Circuit script as a 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 V. However, in fact the input has jumped from 0 V to 5 V at t = 0 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 μ 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.

$$i(t) = 2.154 \times 10^{-8} \exp\left\{\frac{-700t}{3}\right\} \cos\left(\frac{100\sqrt{326}}{3}t\right) + 1.35 \times 10^{-2} \exp\left\{\frac{-700t}{3}\right\} \sin\left(\frac{100\sqrt{326}}{3}t\right)$$
$$V_{out}(t) = 6.031 \times 10^{-6} \exp\left\{\frac{-700t}{3}\right\} \cos\left(\frac{100\sqrt{326}}{3}t\right) + 3.78 \exp\left\{\frac{-700t}{3}\right\} \sin\left(\frac{100\sqrt{326}}{3}t\right)$$

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 $3s^2 + 1400s + 1.25 \times 10^6 = 0$), $\bar{q}_C(s)$ 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 $q_C(t) \alpha$ 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 $\overline{V}_{in} = 5 \text{ V}$ and $\tau = 3(\text{ms})^2 = 3\mu\text{s}$:

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

This is implemented as the following MATLAB function:

funcvin =
$$Q(t) 5 \exp(-t^2/(3 \times 10^{-6}));$$

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 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 continuous loss of energy in form of heat inside the resistor.

The steady state value of the output voltage will be equal to 0V. As the change in the input signal is getting smaller and its absolute value is approaching 0V, 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 0V. 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 = 0s, 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):

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 $f_C = \frac{\sqrt{\frac{1}{LC}}}{2\pi} = 102.73$ 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 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:

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

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



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:

$$5 \times \text{square}(2\pi ft) = \frac{20}{\pi} \sum_{n=1,3,5...}^{\infty} \frac{1}{n} \sin(n2\pi ft)$$
(18)

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

funcvin =
$$Q(t)$$
 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 $t_f = 0.5$ s. For this input, the output consist of a periodic signal with f = 5 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 Hz and an amplitude of 5 V (see Figure 43a). t_f is set to 0.05 s. As the frequency of the first harmonic (110 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 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 $f = 110 \,\text{Hz}$ (b) Corresponding Output Waveform

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

The last square wave input has a frequency of f = 500 Hz (see Figure 44a). In order to see the whole transient response t_f was set to 0.03 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 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 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 $f = 500 \,\text{Hz}$

(b) Corresponding Output Waveform

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

3.3.5. Sine Waves with different frequencies

Three sine waves with an amplitude of 5 V and frequencies of 5 Hz, 110 Hz and 500 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 Hz:

funcvin = $Q(t) 5 \cdot sin(2 \cdot pi \cdot t \cdot 5);$

 t_f is equal to = 0.5 s.

As expected, the input signal shows a sine wave with amplitude 5 V and a period of 0.2 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:

$$H(j2\pi5) = \frac{280 \times 4 \times 10^{-6} \times j2\pi5}{600 \times 10^{-3} \times 4 \times 10^{-6} (j2\pi5)^2 + 280 \times 4 \times 10^{-6} \times j2\pi5 + 1} = 1.24 \times 10^{-3} + 3.52 \times 10^{-2} i$$
$$|H(j2\pi5)| = 0.0352$$
$$\angle H(j2\pi5) = 87.98^{\circ}$$

After the initial transients have died down, one would expect a sine wave with $\overline{V} = 5 \times 0.0352 = 0.176$ 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.



Figure 45: RLC-Circuit response to sine wave of f = 5 Hz

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

$$funcvin = Q(t) 5 * sin(2 * pi * t * 110);$$

 t_f is changed to 0.05 s. The input signal is shown in Figure 46a.

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

$$H(j2\pi 110) = 0.965 - 0.183 i$$
$$|H(j2\pi 110)| = 0.9826$$
$$\angle H(j2\pi 110) = -10.713^{\circ}$$

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 Hz is close to the corner frequency $f_C = \frac{\sqrt{\frac{1}{LC}}}{2\pi} = 102.73$ 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:

$$funcvin = Q(t) 5 * sin(2 * pi * t * 500);$$

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

$$H(j2\pi500) = 2.35 \times 10^{-2} - 0.151 i$$
$$|H(j2\pi500)| = 0.153$$
$$\angle H(j2\pi500) = -81.18^{\circ}$$

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 $0.766 \text{ V} (= 5 \times 0.153)$. 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 $f = 500 \,\mathrm{Hz}$

(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]

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

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

Let $r = k \frac{\Delta t}{(\Delta x)^2}$ represent the thermal conductivity of the medium¹; then, we obtain an implementable expression for U_x^{t+1} :

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

¹k 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.

```
1 L = 1.; % wire length
2 T = 1.; % max simulation time
3 Nt = 2500; % number of timesteps
4 Nx = 50; % number of spacial divisions
5 dt = T / Nt; % increment through time
6 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 $r > \frac{1}{2}$.

```
1 r = 0.25 * dt / (dx*dx);
2 if r > 0.5
3 disp('warning: for stability, r ≤ 1/2');
4 r
5 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)));
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)));
4
  \% initialcond = @(x, L, Nx) - 12.0;
5
6
  leftbound = @(t, Nt) sin(2*pi*t/Nt);
7
  % leftbound = @(t, Nt) -sin(2*pi*t/Nt);
8
9
 % leftbound = @(t, Nt) 0.0;
10
 rightbound = @(t, Nt) sin(2*pi*t/Nt);
11
 % rightbound = @(t, Nt) 0.0;
12
```

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

²Philosophically, 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.

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 $\sin\left(\frac{2\pi t}{T_{\text{max}}}\right)$. At this point, a time axis is also initialized, as with the spatial axis declared in the earlier loop.

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

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):

```
1 figure(1)
2 plot(x,u(:,int32(0.00*Nt)+1),'-', ...
       x,u(:,int32(0.25*Nt)),'-',
3
                                     . . .
       x,u(:,int32(0.50*Nt)),'-',
4
                                     . . .
       x,u(:,int32(0.75*Nt)),'-',
5
                                     . . .
6
       x,u(:,int32(Nt)),'-')
7 legend('t=0', 't=0.25T', 't=0.5T', 't=0.75T', 't=T')
8
 figure(2)
9 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 $\sin(\frac{2\pi t}{T})$, 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 $t = 0.25T_{\text{max}}$, 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 $L_{\rm max}/2$. By $t = 0.5T_{\rm max}$, the boundary conditions have returned to zero, and heat once again flows from the middle of the wire to the ends. At $t = 0.75T_{\rm max}$, 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 $t = T_{\rm max}$, 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 $t = 0.75T_{\rm max}$ [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).



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:

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

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

$$f(x) = \left| \sin\left(\frac{2\pi x}{L_{\max}}\right) \right|$$
(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 dis- (b) Side view of the mesh for Equation tribution given by Equation (23b) for (23b). selected values of t.

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 = 0.25as 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}Kutta_methods. Accessed: March 11th, 2017. 2017.

A. Full code listing for the RL circuit

A.1. heuns.m

```
1 function [t,Vout] = heuns(Vin,iL0,h,R,L,ti,tf)
2 %The heuns.m finds the solution to an ordinary differential equation ...
      representing an RL-circuit and calculates Vout
3
4 a=1/2; % set scaling factor for heuns method
5 b=1/2;
6 p1=1;
7 q11=1;
8
9 func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)
10
11 N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
12 t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
13 Vout(1) = feval(Vin,ti);% calculate initial value of Vout
14 t(1)=ti; iL(1)=iL0; %set initial values of t_0, and iL at t_0
15 for j=1:N-1 % loop for N steps
       ttemp=t(j);iLtemp=iL(j); %temporary names
16
      grad1=feval(func, ttemp, iLtemp); % gradient at t, iL
17
      iLp=iLtemp+q11*h*grad1; % calculate iL predictor
18
19
      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 ...
20
           calculated from previous values of t, iL
      t(j+1)=ttemp+h; % increase t by stepsize
21
       Vout(j+1)=feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
22
23 end
```

A.2. midpoint.m

```
1 function [t,Vout] = midpoint(Vin,iL0,h,R,L,ti,tf)
2 %The midpoint.m finds the solution to an ordinary differential equation
3 %representing an RL-circuit and calculates Vout
4
5 a=0;
           % set scaling factor for midpoint method
6 b=1;
7 p1=1/2;
8 q11=1/2;
10 func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)
11
12 N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
13 t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
14 Vout(1) = feval(Vin,ti);% calculate initial value of Vout
15 t(1)=ti; iL(1)=iL0; %set initial values of t_0 and iL at t_0
16 for j=1:N-1 % loop for N steps
17
       ttemp=t(j);iLtemp=iL(j); %temporary names
18
      grad1=feval(func, ttemp, iLtemp); % gradient at x
      iLp=iLtemp+q11*h*grad1; % calculate iL predictor
19
      grad2=feval(func, ttemp+p1*h,iLp); % gradient at x+h
20
      iL(j+1)=iLtemp+h*(a*grad1 + b*grad2); % next value of iL ...
21
           calculated from previous values of t, iL
```

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

A.3. ralston.m

```
1 function [t,Vout] = ralston(Vin,iL0,h,R,L,ti,tf)
2 %The ralston.m finds the solution to an ordinary differential equation ...
      representing an RL-circuit and calculates Vout
3
4 a=1/3;
          % set scaling factor for ralston method
5 b=2/3;
6 p1=3/4;
7 g11=3/4;
8
9 func=@(t,iL) (feval(Vin,t)-R*iL)/L; % LiL'=Vin-R*iL -> iL'=f(t,iL)
10
11 N=round((tf-ti)/h); % number of steps=(interval size)/(step size)
12 t=zeros(1,N); iL=zeros(1,N); Vout=zeros(1,N); %set up arrays
13 Vout(1) = feval(Vin,ti);% calculate initial value of Vout
14 t(1)=ti; iL(1)=iL0; %set initial values of t_0 and iL at t_0
15 for j=1:N-1 % loop for N steps
       ttemp=t(j);iLtemp=iL(j); %temporary names
16
       grad1=feval(func, ttemp, iLtemp); % gradient at x
17
      iLp=iLtemp+q11*h*grad1; % calculate iL predictor
18
       grad2=feval(func, ttemp+p1*h,iLp); % gradient at x+h
19
       iL(j+1)=iLtemp+h*(a*grad1 + b*grad2); % next value of iL ...
20
          calculated from previous values of t, iL
       t(j+1)=ttemp+h; % increase t by stepsize
21
22
      Vout(j+1)=feval(Vin,t(j+1))-R*iL(j+1);%calculate Vout
23 end
```

A.4. heuns_script.m

```
1
2 %set up initial conditions
3 iL0=0;
4 ti=0;
5
6 %define component values
7 R=0.5;
8 L=0.0015;
9
10 %there are 15 different inputs
11 for n=1:15 %define all 15 Vin
12 if (n<4)
       if(n==1)
13
           Vina = 5.5;
14
           Vin=@(t) Vina*exp(0); %define input signal as function of time
15
           figure
16
17
       end
       if(n==2)
18
19
           Vina = 3.5;
```

```
tau = 160e - 12;
20
            Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ...
^{21}
                of time
22
       end
       if(n==3)
23
            Vina = 3.5;
24
            tau = 160e-6;
25
            Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ...
26
                time
       end
27
       Vout = feval(Vin,ti)-R*iL0;
28
       subplot(3,2,n);
29
       plot(ti,Vout); % plot initial condition
30
31 end
   if((n>3)&&(n<16))
32
       if (n==4)
33
            figure
34
            Vina = 4.5;
35
            T= 20e-6;
36
            Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
37
                of time
       end
38
       if (n==5)
39
            Vina = 4.5;
40
            T = 160e - 6;
41
            Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
42
               of time
       end
43
       if (n==6)
44
            Vina = 4.5;
45
            T= 450e-6;
46
            Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
47
                of time
48
       end
49
       if (n==7)
50
            Vina = 4.5;
            T= 1000e-6;
51
            Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
52
                of time
       end
53
       if (n==8)
54
            Vina = 4.5;
55
            T = 20e - 6;
56
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
57
                function of time
       end
58
       if (n==9)
59
60
            Vina = 4.5;
            T= 160e-6;
61
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
62
                function of time
       end
63
       if (n==10)
64
            Vina = 4.5;
65
            T= 450e-6;
66
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
67
                function of time
68
       end
       if (n==11)
69
            Vina = 4.5;
70
```

```
T= 1000e-6;
71
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
72
                function of time
        end
73
        if (n==12)
74
            Vina = 4.5;
75
            T= 20e-6;
76
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
77
                function of time
        end
78
        if (n==13)
79
            Vina = 4.5;
80
            T = 160e - 6;
81
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
82
                function of time
83
        end
        if (n==14)
84
            Vina = 4.5;
85
            T= 450e-6;
86
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
87
                function of time
88
        end
        if (n==15)
89
            Vina = 4.5;
90
            T= 1000e-6;
91
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
92
                function of time
        end
93
        nn=n-3;
94
        Vout = feval(Vin,ti)-R*iL0;
95
        subplot(3,4,nn);
96
        plot(ti,Vout); % plot initial condition
97
   end
98
99
   if(n==1) %plots all 15 Vout against time
100
        h=10e-7; % set step-size
101
        tf=0.04; % set final value of t
102
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
103
            using heun's method
        subplot(3,2,1);
104
        plot(t,Vout); % plot Vout against t
105
        title('Heuns Vin=5.5V')
106
        xlabel('Time [s]') % x-axis label
107
        ylabel('Vout [V]') % y-axis label
108
109 end
110
111 if(n==2)
112
113
        h=10e-7; % set step-size
        tf=0.0001; % set final value of t
114
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
115
            using heun's method
        subplot(3,2,2);
116
        plot(t,Vout); % plot Vout against t
117
        title('Heuns Vin=Vin exp(-t^2/tau)')
118
        xlabel('Time [s]') % x-axis label
119
120
        ylabel('Vout [V]') % y-axis label
121 end
122
123 if (n==3)
```

```
124
        h=10e-7; % set step-size
125
        tf=0.003; % set final value of t
126
127
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
            using heun's method
128
        subplot(3,2,3);
        plot(t,Vout); % plot Vout against t
129
        title('Heuns Vin=Vin exp(-t/tau)')
130
        xlabel('Time [s]') % x-axis label
131
132
        ylabel('Vout [V]') % y-axis label
133 end
134
   if(n==4)
135
136
        h=10e-9; % set step-size
137
        tf=0.00004; % set final value of t
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
138
            using heun's method
        subplot(3,4,1);
139
        plot(t,Vout); % plot t against Vout
140
        title('Heuns Vin=4.5sin(2pit/T) T = 20e-6s')
141
        xlabel('Time [s]') % x-axis label
142
        ylabel('Vout [V]') % y-axis label
143
144 end
145
146 if (n==5)
        h=10e-7; % set step-size
147
        tf=0.00032; % set final value of t
148
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
149
            using heun's method
        subplot(3,4,2);
150
        plot(t,Vout); % plot Vout against t
151
        title('Heuns Vin=4.5sin(2pit/T) T = 160e-6s')
152
        xlabel('Time [s]') % x-axis label
153
154
        ylabel('Vout [V]') % y-axis label
155 <mark>end</mark>
156
   if(n==6)
157
        h=10e-7; % set step-size
158
        tf=0.0009; % set final value of t
159
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
160
            using heun's method
161
        subplot(3,4,3);
        plot(t,Vout); % plot Vout against t
162
        title('Heuns Vin=4.5sin(2pit/T) T = 450e-6s')
163
        xlabel('Time [s]') % x-axis label
164
        ylabel('Vout [V]') % y-axis label
165
166 end
167
168
   if(n==7)
        h=10e-7; % set step-size
169
        tf=0.002; % set final value of t
170
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
171
            using heun's method
        subplot(3,4,4);
172
        plot(t,Vout); % plot Vout against t
173
174
        title('Heuns Vin=4.5sin(2pit/T) T = 1000e-6s')
        xlabel('Time [s]') % x-axis label
175
        ylabel('Vout [V]') % y-axis label
176
177 end
178
```

```
179 if (n==8)
        h=10e-9; % set step-size
180
        tf=0.00004; % set final value of t
181
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
182
            using heun's method
183
        subplot(3,4,5);
        plot(t,Vout); % plot Vout against t
184
        title('Heuns Vin=4.5square(2pit/T) T = 20e-6s')
185
        xlabel('Time [s]') % x-axis label
186
        ylabel('Vout [V]') % y-axis label
187
188 end
189
   if(n==9)
190
        h=10e-7; % set step-size
191
        tf=0.00032; % set final value of t
192
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
193
            using heun's method
        subplot(3,4,6);
194
        plot(t,Vout); % plot Vout against t
195
        title('Heuns Vin=4.5square(2pit/T) T = 160e-6s')
196
        xlabel('Time [s]') % x-axis label
197
        ylabel('Vout [V]') % y-axis label
198
199 end
200
201 if (n==10)
        h=10e-7; % set step-size
202
        tf=0.0009; % set final value of t
203
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
204
            using heun's method
        subplot(3,4,7);
205
        plot(t,Vout); % plot Vout against t
206
        title('Heuns Vin=4.5square(2pit/T) T = 450e-6s')
207
208
        xlabel('Time [s]') % x-axis label
209
        ylabel('Vout [V]') % y-axis label
210 end
211
212 if(n==11)
        h=10e-7; % set step-size
213
        tf=0.002; % set final value of t
214
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
215
            using heun's method
216
        subplot(3,4,8);
        plot(t,Vout); % plot Vout against t
217
        title('Heuns Vin=4.5square(2pit/T) T = 1000e-6s')
218
        xlabel('Time [s]') % x-axis label
219
220
        ylabel('Vout [V]') % y-axis label
221 end
222
223
   if(n==12)
        h=10e-9; % set step-size
224
        tf=0.00004; % set final value of t
225
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
226
            using heun's method
        subplot(3,4,9);
227
        plot(t,Vout); % plot Vout against t
228
        title('Heuns Vin=4.5sawtooth(2pit/T) T = 20e-6s')
229
        xlabel('Time [s]') % x-axis label
230
        ylabel('Vout [V]') % y-axis label
231
232 end
233
```

```
234 if (n==13)
        h=10e-7; % set step-size
235
236
        tf=0.00032; % set final value of t
237
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
            using heun's method
        subplot(3,4,10);
238
        plot(t,Vout); % plot Vout against t
239
        title('Heuns Vin=4.5sawtooth(2pit/T) T = 160e-6s')
240
        xlabel('Time [s]') % x-axis label
241
242
        ylabel('Vout [V]') % y-axis label
243 end
244
245 if(n==14)
        h=10e-7; % set step-size
246
247
        tf=0.0009; % set final value of t
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
248
            using heun's method
        subplot(3,4,11);
249
        plot(t,Vout); % plot Vout against t
250
        title('Heuns Vin=4.5sawtooth(2pit/T) T = 450e-6s')
251
        xlabel('Time [s]') % x-axis label
252
        ylabel('Vout [V]') % y-axis label
253
254 end
255
256 if(n==15)
        h=10e-7; % set step-size
257
        tf=0.002; % set final value of t
258
        [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and t ...
259
            using heun's method
        subplot(3,4,12);
260
        plot(t,Vout); % plot Vout against t
261
        title('Heuns Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
262
263
        xlabel('Time [s]') % x-axis label
264
        ylabel('Vout [V]') % y-axis label
265 end
266
267
268 end
```

A.5. midpoint_script.m

```
1 %set up initial conditions
2 iL0=0;
3 ti=0;
4
5 %define component values
6 R=0.5;
7 L=0.0015;
8
9 for n=1:15 %defines all 15 Vin
10 if (n<4)
11
       if(n==1)
           Vina = 5.5;
12
           Vin=@(t) Vina*exp(0); %define input signal as function of time
13
           figure
14
       end
15
```

```
if(n==2)
16
           Vina = 3.5;
17
           tau = 160e - 12;
18
           Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ...
19
               of time
20
       end
       if(n==3)
21
           Vina = 3.5;
22
           tau = 160e-6;
23
           Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ...
24
               time
25
       end
       Vout = feval(Vin,ti)-R*iL0;
26
27
       subplot(3,2,n);
^{28}
       plot(ti,Vout); % plot initial condition
29 end
  if((n>3)&&(n<16))
30
       if (n==4)
31
           figure
32
           Vina = 4.5;
33
34
           T= 20e-6;
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
35
               of time
       end
36
37
       if (n==5)
           Vina = 4.5;
38
           T= 160e-6;
39
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
40
               of time
       end
41
       if (n==6)
42
           Vina = 4.5;
43
           T= 450e-6;
44
45
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
               of time
46
       end
       if (n==7)
47
           Vina = 4.5;
48
           T= 1000e-6;
49
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
50
               of time
       end
51
       if (n==8)
52
           Vina = 4.5;
53
           T= 20e-6;
54
           Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
55
               function of time
56
       end
       if (n==9)
57
           Vina = 4.5;
58
           T= 160e-6;
59
           Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
60
                function of time
       end
61
       if (n==10)
62
           Vina = 4.5;
63
64
           T= 450e-6;
           Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
65
                function of time
       end
66
```

```
if (n==11)
67
            Vina = 4.5;
68
            T = 1000e - 6;
69
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
70
                function of time
71
        end
        if (n==12)
72
            Vina = 4.5;
73
            T = 20e - 6;
74
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
75
                function of time
        end
76
        if (n==13)
 77
            Vina = 4.5;
 78
            T = 160e - 6;
 79
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
80
                function of time
        end
81
        if (n==14)
82
            Vina = 4.5;
83
            T= 450e-6;
84
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
85
                function of time
        end
86
        if (n==15)
87
88
            Vina = 4.5;
            T= 1000e-6;
89
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
90
                function of time
        end
91
        nn=n-3;
92
        Vout = feval(Vin,ti)-R*iL0;
93
        subplot(3,4,nn);
94
95
        plot(ti,Vout); % plot initial condition
96 end
97
              %plots all 15 Vout against time
98
   if(n==1)
        h=10e-7; % set step-size
99
        tf=0.04; % set final value of t
100
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
101
            t using midpoint method
102
        subplot(3,2,1);
        plot(t,Vout); % plot Vout against t
103
        title('Midpoint Vin=5.5V')
104
        xlabel('Time [s]') % x-axis label
105
106
        ylabel('Vout [V]') % y-axis label
107 end
108
109 if (n==2)
110
        h=10e-7; % set step-size
111
        tf=0.0001; % set final value of t
112
        [t,Vout]=midpoint (Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
113
            t using midpoint method
        subplot(3,2,2);
114
115
        plot(t,Vout); % plot Vout against t
116
        title('Midpoint Vin=Vin exp(-t^2/tau)')
        xlabel('Time [s]') % x-axis label
117
        ylabel('Vout [V]') % y-axis label
118
119 end
```

```
120
   if(n==3)
121
122
123
        h=10e-7; % set step-size
        tf=0.003; % set final value of t
124
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
125
            t using midpoint method
        subplot(3,2,3);
126
        plot(t,Vout); % plot Vout against t
127
        title('Midpoint Vin=Vin exp(-t/tau)')
128
        xlabel('Time [s]') % x-axis label
129
        ylabel('Vout [V]') % y-axis label
130
131 end
132
133 if(n==4)
        h=10e-9; % set step-size
134
        tf=0.00004; % set final value of t
135
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
136
            t using midpoint method
137
        subplot(3,4,1);
138
        plot(t,Vout); % plot Vout against t
        title('Midpoint Vin=4.5sin(2pit/T) T = 20e-6s')
139
        xlabel('Time [s]') % x-axis label
140
        ylabel('Vout [V]') % y-axis label
141
142 end
143
144 if(n==5)
        h=10e-7; % set step-size
145
        tf=0.00032; % set final value of t
146
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
147
            t using midpoint method
        subplot(3,4,2);
148
        plot(t,Vout); % plot Vout against t
149
150
        title('Midpoint Vin=4.5sin(2pit/T) T = 160e-6s')
        xlabel('Time [s]') % x-axis label
151
        ylabel('Vout [V]') % y-axis label
152
153 end
154
   if(n==6)
155
        h=10e-7; % set step-size
156
        tf=0.0009; % set final value of t
157
158
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using midpoint method
        subplot(3,4,3);
159
        plot(t,Vout); % plot Vout against t
160
        title('Midpoint Vin=4.5sin(2pit/T) T = 450e-6s')
161
162
        xlabel('Time [s]') % x-axis label
163
        ylabel('Vout [V]') % y-axis label
164 end
165
   if(n==7)
166
        h=10e-7; % set step-size
167
        tf=0.002; % set final value of t
168
169
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using midpoint method
170
        subplot(3,4,4);
171
        plot(t,Vout); % plot Vout against t
        title('Midpoint Vin=4.5sin(2pit/T) T = 1000e-6s')
172
        xlabel('Time [s]') % x-axis label
173
        ylabel('Vout [V]') % y-axis label
174
```

```
175 end
176
   if(n==8)
177
        h=10e-9; % set step-size
178
        tf=0.00004; % set final value of t
179
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
180
            t using midpoint method
        subplot(3,4,5);
181
        plot(t,Vout); % plot Vout against t
182
        title('Midpoint Vin=4.5square(2pit/T) T = 20e-6s')
183
        xlabel('Time [s]') % x-axis label
184
        ylabel('Vout [V]') % y-axis label
185
186 end
187
   if(n==9)
188
        h=10e-7; % set step-size
189
        tf=0.00032; % set final value of t
190
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
191
            t using midpoint method
192
        subplot(3,4,6);
        plot(t,Vout); % plot Vout against t
193
        title('Midpoint Vin=4.5square(2pit/T) T = 160e-6s')
194
        xlabel('Time [s]') % x-axis label
195
        ylabel('Vout [V]') % y-axis label
196
197 end
198
   if(n==10)
199
        h=10e-7; % set step-size
200
        tf=0.0009; % set final value of t
201
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
202
            t using midpoint method
        subplot(3,4,7);
203
        plot(t,Vout); % plot Vout against t
204
205
        title('Midpoint Vin=4.5square(2pit/T) T = 450e-6s')
        xlabel('Time [s]') % x-axis label
206
        ylabel('Vout [V]') % y-axis label
207
208 end
209
210 if (n==11)
        h=10e-7; % set step-size
211
        tf=0.002; % set final value of t
212
213
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using midpoint method
        subplot(3,4,8);
214
        plot(t,Vout); % plot Vout against t
215
        title('Midpoint Vin=4.5square(2pit/T) T = 1000e-6s')
216
217
        xlabel('Time [s]') % x-axis label
218
        ylabel('Vout [V]') % y-axis label
219 end
220
   if (n==12)
221
        h=10e-9; % set step-size
222
        tf=0.00004; % set final value of t
223
224
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using midpoint method
225
        subplot(3,4,9);
226
        plot(t,Vout); % plot Vout against t
227
        title('Midpoint Vin=4.5sawtooth(2pit/T) T = 20e-6s')
        xlabel('Time [s]') % x-axis label
228
        ylabel('Vout [V]') % y-axis label
229
```

```
230 end
231
   if(n==13)
232
233
        h=10e-7; % set step-size
        tf=0.00032; % set final value of t
234
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
235
            t using midpoint method
        subplot(3,4,10);
236
        plot(t,Vout); % plot Vout against t
237
        title('Midpoint Vin=4.5sawtooth(2pit/T) T = 160e-6s')
238
        xlabel('Time [s]') % x-axis label
239
        ylabel('Vout [V]') % y-axis label
240
241 end
242
243 if (n==14)
        h=10e-7; % set step-size
244
        tf=0.0009; % set final value of t
245
246
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using midpoint method
        subplot(3,4,11);
247
        plot(t,Vout); % plot Vout against t
248
        title('Midpoint Vin=4.5sawtooth(2pit/T) T = 450e-6s')
249
        xlabel('Time [s]') % x-axis label
250
        ylabel('Vout [V]') % y-axis label
251
252 end
253
254 if(n==15)
        h=10e-7; % set step-size
255
        tf=0.002; % set final value of t
256
        [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
257
            t using midpoint method
        subplot(3,4,12);
258
259
        plot(t,Vout); % plot Vout against t
260
        title('Midpoint Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
        xlabel('Time [s]') % x-axis label
261
        ylabel('Vout [V]') % y-axis label
262
263 end
264
265 end
```

A.6. ralston_script.m

```
1 %set up initial conditions
2 iL0=0;
3 ti=0;
4
5 %define component values
6 R=0.5;
7 L=0.0015;
8
9 %there are 15 different inputs
10 for n=1:15 %define 15 Vin
11 if (n<4)
       if(n==1)
12
           Vina = 5.5;
13
           Vin=@(t) Vina*exp(0); %define input signal as function of time
14
```

```
figure
15
       end
16
17
       if(n==2)
           Vina = 3.5;
18
           tau = 160e - 12;
19
           Vin=@(t) Vina*exp(-t^2/tau); %define input signal as function ...
20
               of time
       end
21
       if(n==3)
22
           Vina = 3.5;
23
            tau = 160e-6;
24
           Vin=@(t) Vina*exp(-t/tau); %define input signal as function of ...
25
               time
26
       end
       Vout = feval(Vin,ti)-R*iL0;
27
       subplot(3,2,n);
28
       plot(ti,Vout); % plot initial condition
29
30 end
31 if((n>3)&&(n<16))
       if (n==4)
32
           figure
33
           Vina = 4.5;
34
           T= 20e-6;
35
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
36
               of time
37
       end
       if (n==5)
38
           Vina = 4.5;
39
           T= 160e-6;
40
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
41
               of time
       end
42
       if (n==6)
43
44
           Vina = 4.5;
45
           T= 450e-6;
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
46
               of time
       end
47
       if (n==7)
48
           Vina = 4.5;
49
           T = 1000e - 6;
50
           Vin=@(t) Vina*sin(2*pi*t/T); %define input signal as function ...
51
               of time
       end
52
       if (n==8)
53
           Vina = 4.5;
54
55
           T= 20e-6;
           Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
56
               function of time
       end
57
       if (n==9)
58
           Vina = 4.5;
59
           T = 160e - 6;
60
           Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
61
                function of time
62
       end
63
       if (n==10)
           Vina = 4.5;
64
           T= 450e-6;
65
```

```
Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
66
                function of time
        end
67
        if (n==11)
68
            Vina = 4.5;
69
            T = 1000e - 6;
70
            Vin=@(t) Vina*square(2*pi*t/T); %define input signal as ...
71
                function of time
72
        end
        if (n==12)
73
            Vina = 4.5;
74
            T= 20e-6;
 75
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
76
                function of time
77
        end
        if (n==13)
78
            Vina = 4.5;
79
            T= 160e-6;
80
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
81
                function of time
82
        end
        if (n==14)
83
            Vina = 4.5;
84
            T= 450e-6;
85
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
86
                function of time
        end
87
        if (n==15)
88
            Vina = 4.5;
89
            T = 1000e - 6;
90
            Vin=@(t) Vina*sawtooth(2*pi*t/T); %define input signal as ...
91
                function of time
        end
92
93
        nn=n-3;
        Vout = feval(Vin,ti)-R*iL0;
94
95
        subplot(3,4,nn);
        plot(ti,Vout); % plot initial condition
96
97
   end
98
99
   if (n==1) %plots all 15 Vout against time
100
101
        h=10e-7; % set step-size
        tf=0.04; % set final value of t
102
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
103
            t using ralston method
104
        subplot(3,2,1);
105
        plot(t,Vout); % plot Vout against t
        title('Ralston Vin=5.5V')
106
        xlabel('Time [s]') % x-axis label
107
        ylabel('Vout [V]') % y-axis label
108
109 end
110
111 if(n==2)
112
        h=10e-7; % set step-size
113
114
        tf=0.0001; % set final value of t
115
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
        subplot(3,2,2);
116
117
        plot(t,Vout); % plot Vout against t
```

```
title('Ralston Vin=Vin exp(-t^2/tau)')
118
        xlabel('Time [s]') % x-axis label
119
120
        ylabel('Vout [V]') % y-axis label
121 end
122
123 if (n==3)
124
        h=10e-7; % set step-size
125
        tf=0.003; % set final value of t
126
127
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
        subplot(3,2,3);
128
        plot(t,Vout); % plot Vout against t
129
        title('Ralston Vin=Vin exp(-t/tau)')
130
        xlabel('Time [s]') % x-axis label
131
        ylabel('Vout [V]') % y-axis label
132
133 end
134
135 if (n==4)
136
        h=10e-9; % set step-size
137
        tf=0.00004; % set final value of t
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
138
            t using ralston method
        subplot(3,4,1);
139
        plot(t,Vout); % plot Vout against t
140
        title('Ralston Vin=4.5sin(2pit/T) T = 20e-6s')
141
        xlabel('Time [s]') % x-axis label
142
        ylabel('Vout [V]') % y-axis label
143
144 end
145
146 if (n==5)
        h=10e-7; % set step-size
147
        tf=0.00032; % set final value of t
148
149
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
150
        subplot(3,4,2);
        plot(t,Vout); % plot Vout against t
151
        title('Ralston Vin=4.5sin(2pit/T) T = 160e-6s')
152
        xlabel('Time [s]') % x-axis label
153
        ylabel('Vout [V]') % y-axis label
154
155 end
156
   if(n==6)
157
        h=10e-7; % set step-size
158
        tf=0.0009; % set final value of t
159
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
160
            t using ralston method
161
        subplot(3,4,3);
162
        plot(t,Vout); % plot Vout against t
        title('Ralston Vin=4.5sin(2pit/T) T = 450e-6s')
163
        xlabel('Time [s]') % x-axis label
164
165
        ylabel('Vout [V]') % y-axis label
166 end
167
   if(n==7)
168
        h=10e-7; % set step-size
169
        tf=0.002; % set final value of t
170
171
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
172
        subplot(3,4,4);
```

```
plot(t,Vout); % plot Vout against t
173
        title('Ralston Vin=4.5sin(2pit/T) T = 1000e-6s')
174
        xlabel('Time [s]') % x-axis label
175
176
        ylabel('Vout [V]') % y-axis label
177 end
178
   if(n==8)
179
        h=10e-9; % set step-size
180
        tf=0.00004; % set final value of t
181
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
182
            t using ralston method
        subplot(3,4,5);
183
        plot(t,Vout); % plot Vout against t
184
        title('Ralston Vin=4.5square(2pit/T) T = 20e-6s')
185
        xlabel('Time [s]') % x-axis label
186
        ylabel('Vout [V]') % y-axis label
187
188 end
189
190 if (n==9)
191
        h=10e-7; % set step-size
192
        tf=0.00032; % set final value of t
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
193
            t using ralston method
        subplot(3,4,6);
194
        plot(t,Vout); % plot Vout against t
195
        title('Ralston Vin=4.5square(2pit/T) T = 160e-6s')
196
        xlabel('Time [s]') % x-axis label
197
        ylabel('Vout [V]') % y-axis label
198
199 end
200
_{201} if (n==10)
        h=10e-7; % set step-size
202
203
        tf=0.0009; % set final value of t
204
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
205
        subplot(3,4,7);
        plot(t,Vout); % plot Vout against t
206
        title('Midpoint Vin=4.5square(2pit/T) T = 450e-6s')
207
        xlabel('Time [s]') % x-axis label
208
        ylabel('Vout [V]') % y-axis label
209
210 end
211
212 if (n==11)
        h=10e-7; % set step-size
213
        tf=0.002; % set final value of t
214
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
215
            t using ralston method
216
        subplot(3,4,8);
217
        plot(t,Vout); % plot Vout against t
        title('Ralston Vin=4.5square(2pit/T) T = 1000e-6s')
218
        xlabel('Time [s]') % x-axis label
219
220
        ylabel('Vout [V]') % y-axis label
221 end
222
   if(n==12)
223
        h=10e-9; % set step-size
224
        tf=0.00004; % set final value of t
225
226
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
        subplot(3,4,9);
227
```
```
plot(t,Vout); % plot Vout against t
228
        title('Ralston Vin=4.5sawtooth(2pit/T) T = 20e-6s')
229
        xlabel('Time [s]') % x-axis label
230
231
        ylabel('Vout [V]') % y-axis label
232 end
233
234 if (n==13)
        h=10e-7; \% set step-size
235
        tf=0.00032; % set final value of t
236
237
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
        subplot(3,4,10);
238
        plot(t,Vout); % plot Vout against t
239
        title('Ralston Vin=4.5sawtooth(2pit/T) T = 160e-6s')
240
        xlabel('Time [s]') % x-axis label
241
        ylabel('Vout [V]') % y-axis label
242
243 end
244
245 if (n==14)
        h=10e-7; % set step-size
246
247
        tf=0.0009; % set final value of t
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
248
            t using ralston method
        subplot(3,4,11);
249
        plot(t,Vout); % plot Vout against t
250
        title('Ralston Vin=4.5sawtooth(2pit/T) T = 450e-6s')
251
        xlabel('Time [s]') % x-axis label
252
        ylabel('Vout [V]') % y-axis label
253
254 end
255
256 if (n==15)
        h=10e-7; % set step-size
257
258
        tf=0.002; % set final value of t
259
        [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf); %obtain arrays of Vout and ...
            t using ralston method
260
        subplot(3,4,12);
        plot(t,Vout); % plot Vout against t
261
        title('Ralston Vin=4.5sawtooth(2pit/T) T = 1000e-6s')
262
        xlabel('Time [s]') % x-axis label
263
        ylabel('Vout [V]') % y-axis label
264
265 end
266
267 end
```

A.7. error_script.m

```
1 iL0=0; % set initial values
2 ti=0;
3 R=0.5; % set constant values
4 L=0.0015;
5 A=6;
6 Vina = 6;
7 T= 150e-6;
8 Vin=@(t) Vina*cos(2*pi*t/T); % set Vin
9 c=((A*R*T^2)/(4*pi^2*L^2+R^2*T^2)); %c for exact solution
10
```

```
11 tf=0.0003; % set final value of t
12
13 for n=1:3
14
15 if (n==1)
      h=10e-7; % set step-size
16
       [t,Vout]=heuns(Vin,iL0,h,R,L,ti,tf);
17
       figure
18
       subplot(3,2,1);
19
20
       plot(t,Vout); % plot heuns Vout against t
       title('Heuns Vin=6cos(2*pi*t/150e-6)')
21
       xlabel('Time [s]') % x-axis label
22
       ylabel('Vout [V]') % y-axis label
23
24 end
25 if (n==2)
       h=10e-7; % set step-size
26
       [t,Vout]=midpoint(Vin,iL0,h,R,L,ti,tf);
27
       figure
28
       subplot(3,2,1);
29
       plot(t,Vout); % plot heuns Vout against t
30
       title('Midpoint Vin=6cos(2*pi*t/150e-6)')
31
       xlabel('Time [s]') % x-axis label
32
       ylabel('Vout [V]') % y-axis label
33
34 end
35 if (n==3)
      h=10e-7; % set step-size
36
       [t,Vout]=ralston(Vin,iL0,h,R,L,ti,tf);
37
       figure
38
       subplot(3,2,1);
39
       plot(t,Vout); % plot heuns Vout against t
40
       title('Ralston Vin=6cos(2*pi*t/150e-6)')
41
       xlabel('Time [s]') % x-axis label
42
43
       ylabel('Vout [V]') % y-axis label
44 end
45 iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
46 /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
47 Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout for exact iL
48 subplot(3,2,2);
49 plot(t,Vout_exact); %plot exact solution
       title('Exact solution of the ODE')
50
       xlabel('Time [s]') % x-axis label
51
       ylabel('Vout [V]') % y-axis label
52
53 error=abs(Vout_exact-Vout); % calculate maximum error over range of x
54 subplot(3,2,[3,4]);
55 plot(t,error); % plot error against t
       title('Error against time')
56
57
       xlabel('Time [s]') % x-axis label
       ylabel('Error [V]') % y-axis label
58
59
60 subplot(3,2,[5,6]);
                      %initialize arrays
61 h = zeros(1, 10);
62 errororder = zeros(1, 10);
63 count = 1; %initiallize count
64
65
66
67 if (n==1)
       hi=1e-9; hh=1e-9; hf=1e-7;
                                                 % set initial step-size, ...
68
           increment in step-size and final step-size value
       h=hi:hh:hf;
69
```

```
Nh=round((hf-hi)/hh)+1;
                                   % number of steps=(interval size of ...
70
            step-size)/(increment in step-size)
        for count=1:Nh
71
            [t,Vout]=heuns(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
72
            iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
73
   /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
74
            Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
75
                exact solution iL
            errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
76
                maximum error over range of x
            hold on;
77
        end
 78
        hold off;
 79
        plot(log(h), log(errororder));
 80
        gradheuns=polyfit(log(h), log(errororder),1);
 81
82 end
83
84 if (n==2)
        hi=1e-9; hh=1e-9; hf=1e-7;
                                                  % set initial step-size, ...
85
            increment in step-size and final step-size value
        h=hi:hh:hf:
86
                                   % number of steps=(interval size of ...
87
        Nh=round((hf-hi)/hh)+1;
           step-size)/(increment in step-size)
        for count=1:Nh
88
            [t,Vout]=midpoint(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
89
            iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
90
   /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
91
            Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
92
               exact solution iL
            errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
93
               maximum error over range of x
            hold on;
94
        end
95
96
        hold off;
97
        plot(log(h), log(errororder));
98
        gradmidpoint=polyfit(log(h), log(errororder),1);
99 end
100
101 if (n==3)
                                                  % set initial step-size, ...
       hi=1e-9; hh=1e-9; hf=1e-7;
102
           increment in step-size and final step-size value
103
        h=hi:hh:hf;
104
        Nh=round((hf-hi)/hh)+1;
                                     % number of steps=(interval size of ...
           step-size)/(increment in step-size)
        for count=1:Nh
105
            [t,Vout]=ralston(Vin,iL0,h(count),R,L,ti,tf);% call heuns.m
106
107
            iL_exact=((2*A*pi*T*L*sin((2*pi*t)/T)+A*R*T^2*cos((2*pi*t)/T))
108 /(4*pi^2*L^2+R^2*T^2))-(c*exp(-R*t/L)); %calculate exact solution
109
            Vout_exact=feval(Vin,t)-R*iL_exact; % calculate Vout using ...
                exact solution iL
            errororder(count)=max(abs(Vout_exact-Vout)); % calculate ...
110
                maximum error over range of x
            hold on;
111
        end
112
        hold off;
113
114
        plot(log(h), log(errororder)); % plot log log graph
115
        gradralston=polyfit(log(h), log(errororder),1);% calculate gradient
116 end
        title('log of maximum error against log of h')
117
       ylabel('log error max') % x-axis label
118
```

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

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

B.1. RK4second.m

```
1 function [xn, yn] = RK4second(funcx, funcy, h, ti, xi, yi)
2 %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)
3  ^{\diamond}
      funcx computes the derivative of x (dx/dt) at a point (ti, xi, yi)
4 %
5 %
      funcy computes the derivative of y (dy/dt) at a point (ti, xi ,yi)
7 %calculate coefficients (predicted gradients) at ti, ti+h/3, ti+2h/3, ti+h
8 %using Runge-Kutta 3/8
9 k1x = feval(funcx, ti, xi, yi);
10 kly = feval(funcy, ti, xi, yi);
11 k2x = feval(funcx, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
12 k2y = feval(funcy, ti + h/3, xi + h/3*k1x, yi + h/3*k1y);
13 k3x = feval(funcx, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
14 k3y = feval(funcy, ti + 2*h/3, xi - h/3*k1x+h*k2x, yi - h/3*k1y+h*k2y);
15 k4x = feval(funcx, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
16 k4y = feval(funcy, ti+h, xi+h*k1x-h*k2x+h*k3x, yi+h*k1y-h*k2y+h*k3y);
17
18 %obtain phix and phiy by taking weighted average of obtained gradients
19 phix = (k1x + 3 k2x + 3 k3x + k4x)/8;
20 phiy = (k1y + 3 k2y + 3 k3y + k4y)/8;
21
22 %use phi-values as approximated gradients for x and y
23 xn = xi + h*phix; %calculate x(i+1)
24 yn = yi + h*phiy; %calculate y(i+1)
25 end
```

B.2. RLC_script.m

RLC_script.m is written with the following matlab code:

```
1 %The RLC.script calculates the voltage across R (Vout) for a given ...
input signal(Vin)
2 %The RLC Circuit script is finding the solution to a second order ...
differential equation representing an RLC-circuit
3
4 %set up initial conditions
5 q0 = 500*10^(-9); %[C]; capacitor charge at t=0
6 i0 = 0; %current at t=0
7 t0 = 0; %set up starting time
8 h = 0.000001; %[s]; set up step-size for Runge-Kutta 3/8
9 tf = 0.06; %[s]; define endpoint of time-interval
10
11 %define component values
12 R = 280; %resistance equals 280 Ohm
```

```
13 C = 4 \times 10^{(-6)}; %Capacitor value is 4 microFarad
14 L=600*10<sup>(-3)</sup>; %Inductance is 600 milliHenry
15
16 funcvin = @(t) 5; %define input signal (step-input, tf=0.06) as ...
      function of time
17
18 %the other input functions with their corresponding tf-value are stated
19 %below:
20 %funcvin = Q(t) 5*exp(-t<sup>2</sup>/(3*10<sup>-6</sup>)); (impulse, tf=0.06)
21 %funcvin = @(t) 5*square(2*pi*t*5); (square, f=5Hz, tf=0.5)
22 %funcvin = @(t) 5*square(2*pi*t*110); (square, f=110Hz, tf=0.05)
23 %funcvin = @(t) 5*square(2*pi*t*500); (square, f=500Hz, tf=0.03)
24 %funcvin = @(t) 5*sin(2*pi*t*5); (sine, f=5Hz, tf=0.5)
25 %funcvin = @(t) 5*sin(2*pi*t*110); (sine, f=110Hz, tf = 0.05)
26 %funcvin = @(t) 5*sin(2*pi*t*500); (sine, f=500Hz, tf = 0.035)
27
28 %set up coupled first-order equations
29 funcq = @(t, q, i) i; %gradient of q at time t (=i(t))
30 funci = @(t, q, i) (feval(funcvin, t) - R*i - 1/C * q)/L; %funci ...
      calculates di/dt at time t
31
32 N = round((tf-t0)/h); %calculate number of steps to reach tf
33
34 %set up arrays to store results
35 q = zeros(1, N);
36 i = zeros(1, N);
37 t = zeros(1, N);
38
39 %first element of each array is equal to corresponding initial condition
40 q(1) = q0;
41 i(1) = i0;
42 t(1) = t0;
43
44 %use for-loop to iterate through arrays
45 %RK4second uses Runge-Kutta-3/8 algorithm to calculate next values for q
46 %and i as t is increased by h after each iteration
47 for j = 1 : N-1
     [q(j+1),i(j+1)] = RK4second(funcq, funci, h, t(j), q(j), i(j));
48
     t(j+1) = t(j) + h;
49
50 end
51
52 vout = i*R; %obtain Vout(t) (voltage across R) using Ohms Law
53 vin = arrayfun(funcvin, t); %calculate Vin(t)
54
55 figure;
56 plot(t, q); %plot q(t) as a function of t
57 title('Capacitor Charge (q_{C}(t))');
58 xlabel('Time [s]');
59 ylabel('Charge [C]');
60
61 figure;
62 plot(t, vout); %plot Vout(t) as a function of t
63 title('Output Voltage (V_{out}(t)=v_{R}(t))');
64 xlabel('Time [s]');
65 ylabel('Voltage [V]');
66
67
68 figure;
69 plot(t, vin); %plot Vin(t) as a function of t
70 title('Input Signal (V_{in}(t))');
```

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

C. Full code listing for the finite differences method

C.1. finite_script.m

```
1 L = 1.;
                % wire length
2 T = 1.;
                % max simulation time
3 \text{ Nt} = 2500;
                % number of timesteps
4 Nx = 50;
                % number of spacial divisions
5 dt = T / Nt; % increment through time
6 dx = L / Nx; % increment through space
7
8 % conductivity parameter
9 r = 0.25 * dt / (dx*dx);
10 if r > 0.5
       % von neumann stability criterion has been violated
11
12
       disp('warning: for stability, r < 1/2');</pre>
13
       r
14 end
15
16 % lambdas to compute initial wire heat distribution
17 initialcond = Q(x, L, Nx) abs(sin(2*pi*x/(Nx+1)));
18 % initial
cond = @(x, L, Nx) \sin(2 * pi * x / (Nx+1) / L);
19 % initialcond = @(x, L, Nx) triangularPulse(0.0, L, x/(Nx+1));
20 % initialcond = @(x, L, Nx) sinc(6 * pi * ((x - (Nx+1)/2) / (Nx+1)));
21 % initialcond = @(x, L, Nx) -12.0;
22
23 % lambdas to compute heat at edge of wire for x=0
24 leftbound = @(t, Nt) sin(2*pi*t/Nt);
25
   % leftbound = @(t, Nt) -sin(2*pi*t/Nt);
26 % leftbound = @(t, Nt) 0.0;
27
_{\rm 28} % lambdas to compute heat at edge of wire for x=L
29 rightbound = @(t, Nt) sin(2*pi*t/Nt);
30 % rightbound = @(t, Nt) 0.0;
31
32\, % compute initial conditions and make an x-axis for plotting
33 for i = 1:Nx+1
      x(i) = (i-1) * dx;
34
       u(i,1) = initialcond(i, L, Nx);
35
36 end
37
38 % compute boundary conditions and make a time-axis for plotting
39 for t = 1:Nt+1
40
      u(1,t) = leftbound(t, Nt);
41
      u(Nx+1, t) = rightbound(t, Nt);
       time(t) = (t-1) * dt;
42
43 end
44
45 % go-time; iterate over entire matrix
46 for t=1:Nt
       for i = 2:Nx
47
           % directly from the expression we obtained
48
```

```
49
          u(i, t+1) = (1-2*r) * u(i,t) + r*u(i+1, t) + r*u(i-1,t);
50
      end
51 end
52
53 figure(1)
54 % plot snapshots of heat distribution for [0, 0.25, 0.5, 0.75, 1] * T
_{55}\, % even time spacing means this plot indicates how fast heat is diffusing
_{56} % too many more or fewer would mean the plot would be cluttered, IMO
57 plot(x,u(:,int32(0.00*N_t)+1),'-', ...
        x,u(:,int32(0.25*N_t)),'-',
58
                                      . . .
        x,u(:,int32(0.50*N_t)),'-',
59
                                     . . .
       x,u(:,int32(0.75*N_t)),'-',
60
                                     . . .
61
       x,u(:,int32(N_t)),'-')
62 legend('t=0', 't=0.25T', 't=0.5T', 't=0.75T', 't=T')
63 figure(2)
_{64} % 3d plot, with a space and a time axis; I prefer this for showing ...
      variation,
65 % _especially_ with an overhead view
66 mesh(x,time,u')
```