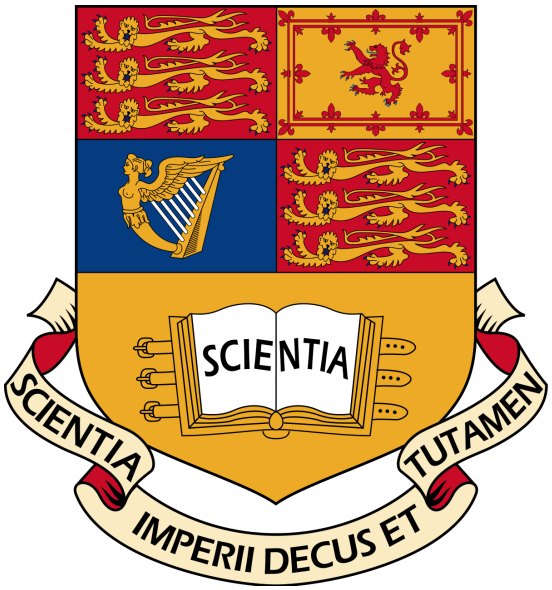
Department of Electrical and Electronic Engineering
The Imperial College of Science, Technology and Medicine

Numerical Analysis of Differential Equations

Group 25

Edward John Harriss (CID: 01523595)
Aditya Gupta (CID: 01533171)
Anushka Kulkarni (CID: 01567227)
Ludwig Jonsson (CID: 01520034)
Zhongjian Qiu (CID: 01517667)
Christoph Renschler (CID: 01580976)
Kimon Grigorakis (CID: 01535391)

Real Word Count: 5895



15-03-2020

Contents

0 Abstract

This coursework consists of three main parts. In the first part, different Runge-Kutta methods are used to model an RC circuit. In the second part, said methods are applied to RLC circuits. The third and last part implements the Relaxation method in order to model solutions to the Laplace equation.

During part one, we observed a low-pass filter usually used in NAB filters. The circuit cuts off any signals above 1.592kHz. The first Exercise was to use three methods; Heun's, midpoint and custom method. Once we had the numerically approximated solutions to the ODE, we compared our results to the exact solutions for error analysis, expressing the error between the two as a function of the chosen step-size and thus, we showed that the order of the global truncation error of the numerical solution is $O(h^2)$. We found that the filter behaved as expected for input signals of low frequency but for higher frequency inputs it acted as an integrator. For each input waveform, a transient and steady state output could be observed.

In part two we came across an RLC circuit acting as an band-pass filter. The purpose of this part was to approximate the output signal using classic 4th order Runge-Kutta method, for a variety of input signals.

In the third part, we applied the Relaxation method to numerically solve Laplace's equation on the Cartesian plane and model it. We analysed limitations of the relaxation method, and further explored how different resolutions, maximum errors and boundary conditions impact the performance of our implementation, both in terms of its accuracy and complexity. Furthermore, we analyzed how Successive Over-relaxation can improve the performance of our algorithm and how the optimal relaxation factor depends on the boundary conditions of the system. We found that a relaxation factor of 1.94 works consistently well for all systems, and that the optimal relaxation factor is lower for more complex boundary conditions.

1 Exercise 1 - Second Order Runge-Kutta

1.1 The Circuit

A low-pass filter takes an input signal and removes all high-frequency components such as noise. The RC circuit shown below (figure 1) is a low pass filter that implements this functionality. It removes frequencies above 1.592 kHz.

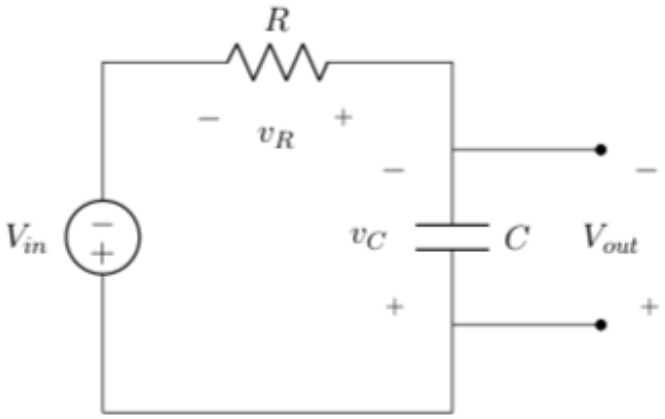


Figure 1: RC circuit acting as a low pass filter

The equations obtained after analysing this circuit are shown below (figure 2):

$$v_C(t) + v_R(t) = V_{in}(t)\,, \quad \frac{1}{C}\int_0^t i_C(t) + R\,i_C(t) = V_{in}(t)\,, \quad \frac{1}{C}\,q_C + R\,\frac{dq_C}{dt} = V_{in}(t)\,.$$

Figure 2: RC circuit equations for Vin

The equation that will be focused on in this exercise is the differential equation. In this equation, C is the capacitance, q_C is the charge across the capacitor as a function of time, R is the resistance, t is time, and V_{in} is the input voltage. The output voltage (V_{out}) is determined by

$$V_{out} = \frac{q_C(t)}{C}$$

.
The initial conditions were as follows:

- $q_C(0) = 500$ nC
- $R = 1000\ \Omega$
- $C = 100$ nF

Very slight differences were observed between different methods – Heun, midpoint and a third method of our own invention, so this report will only show the results obtained from Heun's method.

1.2 RK2 Methods

Runge-Kutta methods are numerical analysis methods that use the gradient of a function, given by its ODE, at a given point to estimate the gradient at further points and approximate the function. This can be represented by

$$y_{i+1} = y_i + h\phi(x_i, y_i, h)$$

where ϕ is called the increment function, representing the gradient of y on the interval $[x_i, x_{i+1}]$.
ϕ can be written as $a_1k_1 + a_2k_2 + a_3k_3 + ...$ but for the purpose of this exercise we are only considering RK2 methods. Thus,

$$\phi = ak_1 + bk_2$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + ph, y_i + qhk_1)$$

where the following three equations link a,b p and q:

$$a + b = 1$$
$$bp = \frac{1}{2}$$
$$bq = \frac{1}{2}$$

As this a system of three equations in four variables we can have infinitely many solutions and so we can pick values for the variables which satisfy the equations.

1.2.1 Heun's Method

Heun's method sets the values of both a and b to be $\frac{1}{2}$ and hence $p = q = 1$.
This gives the equation:

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

where:

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + h, y_i + hf(x_i, y_i)) = f(x_{i+1}, y_{i+1})$$

From this we can see that k_1 is the gradient at the start of the interval $[x_i, x_{i+1}]$ and k_2 is the gradient at the end of the interval as $y_i + hf(x_i, y_i)$ is the estimate for y_{i+1} using Euler's method. This is shown by the image below.

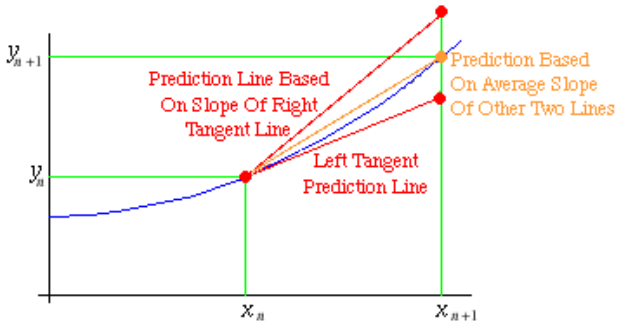


Figure 3: Explanation of Heun's Method

The left gradient is the gradient at the start point and the right gradient is a line with the same gradient as that at the end point of the interval but passes through the start point. Heun's method takes the average of these two gradients to give us a better estimate of the endpoint of the interval.

1.2.2 Midpoint Method

The Midpoint Method uses an a value of 0 and b value of 1 giving $p = q = \frac{1}{2}$
This gives:

$$y_{i+1} = y_i + hk_2$$

where:

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

This shows that the midpoint method uses the gradient at the midpoint of the interval $[x_i, x_{i+1}]$ to give a better estimate of the endpoint.

1.2.3 Our Method

The method was chosen at random where the value of a was a randomly chosen value between 0 and 1. In our case we chose a to be 0.3.

1.3 MATLAB Code

1.3.1 RK2

The MATLAB code used to implement the second order Runge-Kutta methods is shown below. The code shown is only that which implements the prediction and correction alogrithm. The full code can be found in the Appendix[5.2.3]

```
function [tout, qcout] = RK2(func, t_i, qc_i, t_f, h)
    for j = 1:N-1 % loop for N-1 steps
        k1 = feval( func, tout(j), qcout(j) );
        k2 = feval( func, (tout(j) + p*h), (qcout(j) + q*k1*h) );
        qcout(j+1) = qcout(j) + h*(a*k1 + b*k2);
        tout(j+1) = tout(j) + h;
    end
end
```

The RK2 function takes as input parameters the ODE (in the form $y' = f(x, y)$), the initial value of t, the initial value of q_C, the final value of t and finally the step size h. The method of approximation is set by setting the values of a and b and calculating the number of steps using the final value of t (the code for both of which is shown in the Appendix) . The for loop is run for the number of intervals calculated. For each iteration, the value of k_1 and k_2 is calculated and using these, and the previous value of y, the next value of y is determined. The values of x and the corresponding y values are outputted from the function as an array.

1.3.2 RK2 Script

The MATLAB code which executes the RK2.m code for different input signals is shown below. The script calls the RK2 function for each input and this generates the output voltage signal which is then plotted. The full code can be found in Appendix[5.2.3]

```
t_i = 0; % set initial value of t_0
q_i = 500e-9; % set q_initial condition q at t_0
h = 0.00001; % set t step-size
t_f = 0.01; % stop here
R = 1000; % resistance
C = 100e-9; % capacitance


%*****************Step_Signal 2.5V*****************
func = @(t, q) 2.5/R*heaviside(t) - 1/(R*C)*q;
%
```

```
[tout, qout] = RK2(func, t_i, q_i, t_f, h);
plot(tout, qout, 'b')
%************************************************
```

1.4 Step signal

The input signal passed to the RC circuit is a unit step function with amplitude 2.5V. The observed Vout signal can be explained through both the transient and steady state response of the circuit. To obtain this we solve the ODE.

$$\frac{dq_C}{dt} + \frac{q_C(t)}{RC} = \frac{V_{in}}{R}$$

where

$$V_{in}(t) = 2.5u(t)$$

This gives an integrating factor of:

$$e^{\int \frac{1}{RC} dt} = e^{\frac{t}{RC}}$$

Multiplying both sides with the integrating factor and solving gives:

$$e^{\frac{t}{RC}} \frac{dq_C}{dt} + e^{\frac{t}{RC}} \frac{q_C(t)}{RC} = e^{\frac{t}{RC}} \frac{2.5}{R}$$

$$\frac{d}{dt}(q_C e^{\frac{t}{RC}}) = e^{\frac{t}{RC}} \frac{2.5}{R}$$

$$e^{q_C \frac{t}{RC}} = \frac{2.5}{R} \int e^{\frac{t}{RC}} dt$$

$$q_C e^{\frac{t}{RC}} = \frac{2.5}{R}(RCe^{\frac{t}{RC}} + K)$$

$$q_C = \frac{2.5}{R}(RC + Ke^{\frac{-t}{RC}})$$

$$V_{out} = \frac{q_C}{C} = \frac{2.5}{RC}(RC + Ke^{\frac{-t}{RC}})$$

We know the voltage across the capacitor cannot change abruptly giving:

$$V_{out}(0^+) = V_{out}(0^-)$$

The capacitor is charged to 500 nF at $t = 0$

$$V_{out}(0^-) = \frac{500 \times 10^{-9}}{100 \times 10^{-9}} = 5V$$

$$5 = \frac{2.5}{RC}(RC + Ke^{\frac{-t}{RC}})$$

$$5 = 2.5 + \frac{2.5}{RC} Ke^0 \quad giving\ K = RC$$

This gives us:

$$V_{out} = \frac{2.5}{RC}(RC + RCe^{\frac{-t}{RC}})$$

$$V_{out} = 2.5(1 + e^{\frac{-t}{RC}})$$

The following figures show the exact (blue) solution compared to the graphs obtained using Heun's method (red).

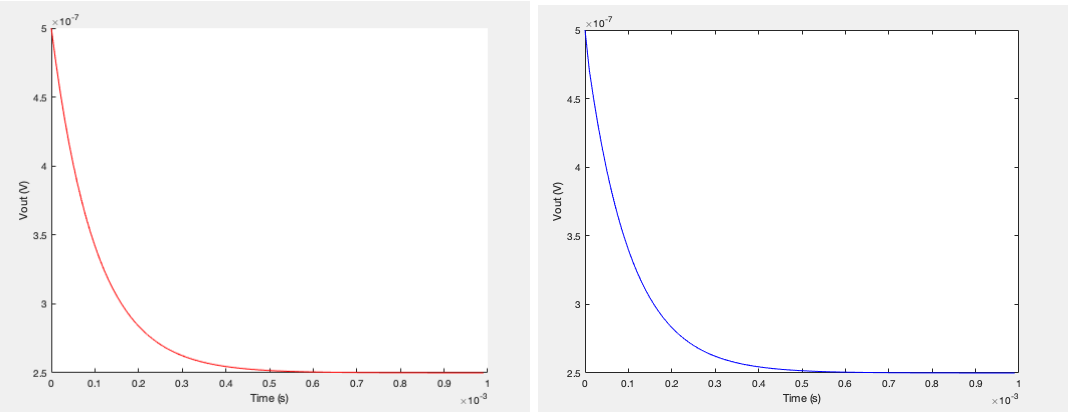


Figure 4: Exact solution (red), Heun's method (blue) for the step signal of 2.5V

By changing the value of R, the time constant of the circuit can be changed and thus the output voltage signal will be changed accordingly. The results of these modifications are shown below (figure 5). By increasing R to 5000Ω, the value of RC (which is the time constant) is increased. This means the coefficient of t in the decaying exponential term $e^{\frac{-t}{RC}}$ will be smaller and therefore the output voltage will reach its steady state value more slowly hence a larger transient term. Conversely, decreasing R to 100 Ω makes the coefficient of t a larger negative number. This results in a much faster response and the transient dies out much more quickly.

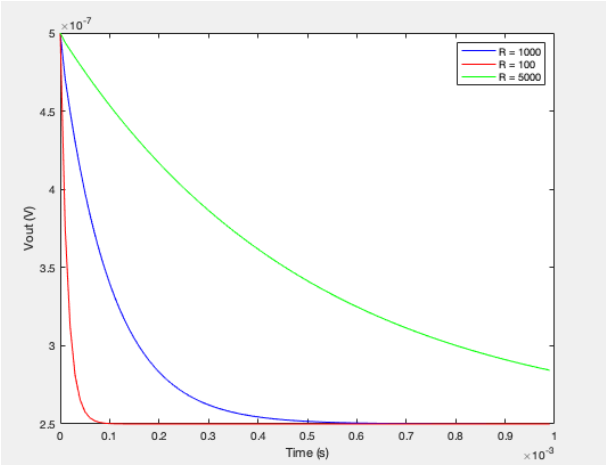


Figure 5: Comparison with different values of R

1.5 Impulsive signal and Decay signal

Passing exponential signals through a low pass filter is fairly straightforward – it cuts off high frequencies, and retains its decaying shape. Two different exponential signals were passed through the RC circuit with time constant $\tau = 0.001$:

1. Impulsive signal
$$V_{in} = 2.5e^{\frac{-t^2}{\tau}}$$
2. Decay signal
$$V_{in} = 2.5e^{\frac{-t}{\tau}}$$

It can be seen that the impulsive signal has a $-t^2$ component, which decreases very quickly as t increases, causing the signal to approach 1 when t = 0.0001. The initial drop from 5V to 2.5V is due to the fact that the voltage of

a capacitor cannot change instantly. It stops at 2.5V, which indicates that it is voltage of the input signal (5V is the initial voltage of the capacitor). After this transient, the graph has a steady state decay to 1. For the decay signal, however, the transient is not visible at $\tau = 0.001$. This is because the as the value of $-t$ does not decrease as rapidly when t increases as it does for $-t^2$. This results in a smooth exponential graph that decays to 0. This is shown in the figures below. To explain the observations for the decay signal, we can obtain a closed form solution for the output voltage.

$$\frac{dq_C}{dt} + \frac{q_C(t)}{RC} = \frac{V_{in}}{R}$$

where

$$V_{in}(t) = 2.5e^{\frac{-t}{\tau}}$$

This gives an integrating factor of:

$$e^{\int \frac{1}{RC} dt} = e^{\frac{t}{RC}}$$

Multiplying both sides with the integrating factor and solving gives:

$$e^{\frac{t}{RC}} \frac{dq_C}{dt} + e^{\frac{t}{RC}} \frac{q_C(t)}{RC} = e^{\frac{t}{RC}} \frac{2.5e^{\frac{-t}{\tau}}}{R} = \frac{2.5e^0}{R} = \frac{2.5}{R}$$

$$\frac{d}{dt}(q_C e^{\frac{t}{RC}}) = \frac{2.5}{R}$$

$$q_C e^{\frac{t}{RC}} = \int \frac{2.5}{R} dt$$

$$q_C e^{\frac{t}{RC}} = \frac{2.5}{R}(t + K)$$

$$q_C = \frac{2.5te^{\frac{-t}{RC}}}{R} + Ke^{\frac{-t}{RC}}$$

$$V_{out} = \frac{2.5te^{\frac{-t}{RC}}}{RC} + Ke^{\frac{-t}{RC}}$$

We know the voltage across the capacitor cannot change abruptly giving:

$$V_{out}(0^+) = V_{out}(0^-)$$

The capacitor is charged to 500 nF at $t = 0$

$$V_{out}(0^-) = \frac{500 \times 10^{-9}}{100 \times 10^{-9}} = 5V$$

$$5 = \frac{2.5(0)e^0}{RC} + Ke^0 \ \ giving \ \ K = 5$$

Hence:

$$V_{out} = \frac{2.5te^{\frac{-t}{RC}}}{RC} + 5e^{\frac{-t}{RC}}$$

The analytical solution shows that in the steady state, the output voltage decays to 0. This can be shown using limits.
The steady state value is given by:

$$\lim_{t\to\infty} V_{out} = \lim_{t\to\infty} (\frac{2.5te^{\frac{-t}{RC}}}{RC} + 5e^{\frac{-t}{RC}})$$

$$\lim_{t\to\infty} V_{out} = \lim_{t\to\infty}(\frac{2.5te^{\frac{-t}{RC}}}{RC}) + \lim_{t\to\infty}(5e^{\frac{-t}{RC}})$$

We can evaluate the limits separately

For the first limit:

$$\lim_{t\to\infty}(\frac{2.5te^{\frac{-t}{RC}}}{RC}) = \lim_{t\to\infty}(\frac{2.5t}{RCe^{\frac{t}{RC}}}) = \frac{"0"}{0}$$

We need to use L'Hoptial's rule:

$$\lim_{t\to\infty}(\frac{\frac{d}{dt}(2.5t)}{\frac{d}{dt}(RCe^{\frac{t}{RC}})}) = \lim_{t\to\infty}(\frac{2.5}{e^{\frac{t}{RC}}}) = 0$$

For the second limit:

$$\lim_{t\to\infty}(5e^{\frac{-t}{RC}}) = 0$$

Therefore in the steady state:

$$\lim_{t\to\infty} V_{out} = \lim_{t\to\infty}(\frac{2.5te^{\frac{-t}{RC}}}{RC}) + \lim_{t\to\infty}(5e^{\frac{-t}{RC}})$$

$$\lim_{t\to\infty} V_{out} = 0 + 0 = 0$$

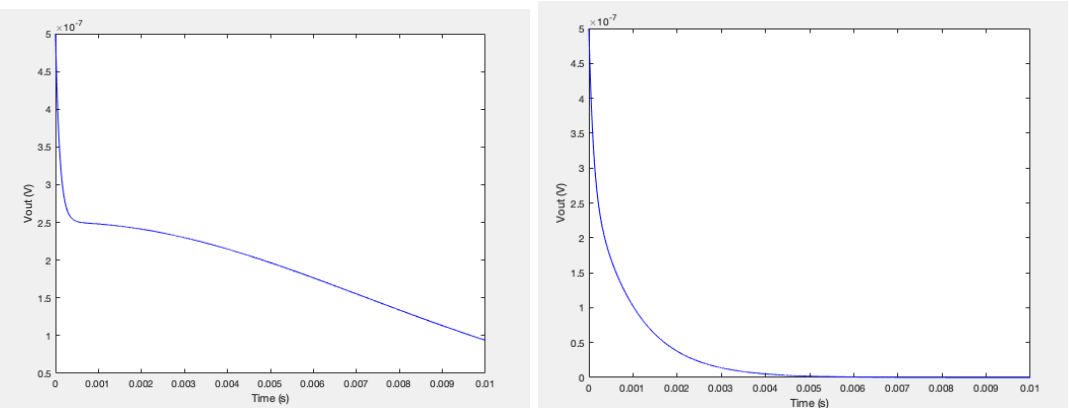


Figure 6: Heun's method for the Impulsive signal (left) and decay signal (right) with $\tau = 0.0001$

However, if the value of τ is increased to 0.01, the transient can be observed for the decay signal as well. This can be seen in the figure below:

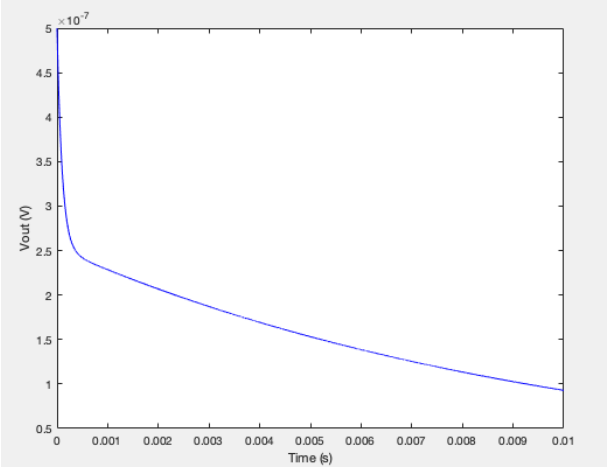


Figure 7: Decay signal with $\tau = 0.01$ – transient is visible now

1.6 Sine wave

The RC circuit is provided input sine waves with periods $T = 10\ \mu$s, $T = 100\ \mu$s, $T = 500\ \mu$s, $T = 1000\ \mu$s. The output voltages for each input voltage above are shown below. As mentioned before, the initial capacitor voltage is 5V. For a sinusoidal input, the steady state output is also a sinusoidal wave but with a different phase and amplitude. To obtain both steady state and transient responses, we can solve the ODE and obtain an analytical expression for V_{out}.

Lets start by writing the ODE:

$$\frac{dq_C}{dt} + \frac{q_C}{RC} = \frac{V_{in}(t)}{R}$$

where

$$V_{in}(t) = 5sin(\omega t) \quad (\omega = \frac{2\pi}{T})$$

This gives an integrating factor of:

$$e^{\int \frac{1}{RC} dx} = e^{\frac{t}{RC}}$$

Multiplying the ODE by the integrating factor gives:

$$e^{\frac{t}{RC}} \frac{dq_C}{dt} + e^{\frac{t}{RC}} \frac{q_C}{RC} = e^{\frac{t}{RC}} \frac{sin(\omega t)}{R}$$

$$\frac{d}{dt}(q_C e^{\frac{t}{RC}}) = e^{\frac{t}{RC}} \frac{5sin(\omega t)}{R}$$

$$q_C e^{\frac{t}{RC}} = \int e^{\frac{t}{RC}} \frac{5sin(\omega t)}{R}\ dt$$

Let

$$I = \int e^{\frac{t}{RC}} sin(\omega t)\ dt$$

Solving this using integration by parts gives:

$$I = RCe^{\frac{t}{RC}} sin(\omega t) - RC\omega \int e^{\frac{t}{RC}} 5cos(\omega t)$$

Further solving the cosine integral by parts gives:

$$I = RCe^{\frac{t}{RC}} sin(\omega t) - RC\omega[RCe^{\frac{t}{RC}} cos(\omega t) + RC\omega \int e^{\frac{t}{RC}} \frac{5sin(\omega t)}{R} + K]$$

Substituting in I and rearranging gives:

$$I = RCe^{\frac{t}{RC}} sin(\omega t) - RC\omega[RCe^{\frac{t}{RC}} cos(\omega t) + RC\omega I + K]$$

$$I = RCe^{\frac{t}{RC}} sin(\omega t) - (RC)^2 \omega e^{\frac{t}{RC}} cos(\omega t) - (RC\omega)^2 I + K$$

$$I[(RC\omega)^2 + 1] = RCe^{\frac{t}{RC}} sin(\omega t) - (RC)^2 \omega e^{\frac{t}{RC}} cos(\omega t) + K$$

$$I[(RC\omega)^2 + 1] = RCe^{\frac{t}{RC}} [sin(\omega t) - RC\omega cos(\omega t)] + K$$

$$I = \frac{RCe^{\frac{t}{RC}} [sin(\omega t) - RC\omega cos(\omega t)] + K}{[(RC\omega)^2 + 1]}$$

Hence:

$$q_C e^{\frac{t}{RC}} = \frac{5I}{R}$$

$$q_C = \frac{5I}{Re^{\frac{t}{RC}}}$$

$$V_{out}(t) = \frac{5I}{RCe^{\frac{t}{RC}}}$$

Substituting in I gives a final result of:

$$V_{out}(t) = \frac{5}{[(RC\omega)^2 + 1]}[sin(\omega t) - RC\omega cos(\omega t)] + Ke^{\frac{-t}{RC}}$$

This shows that in the steady state, the output waveform is a sum of a sine and cosine wave which is a sinusoidal waveform in itself with a different amplitude and phase to the input. The transient is a decaying exponential which will have a different amplitude based on the value of ω.

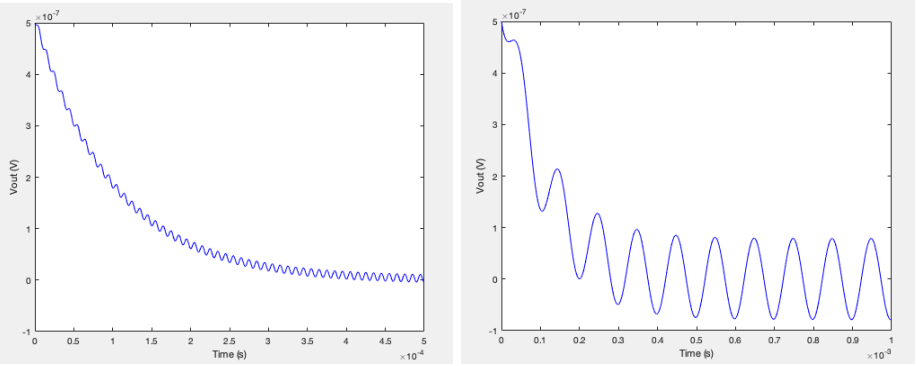


Figure 8: Heun's method for sine wave of 10μs (left) and 100μs (right)

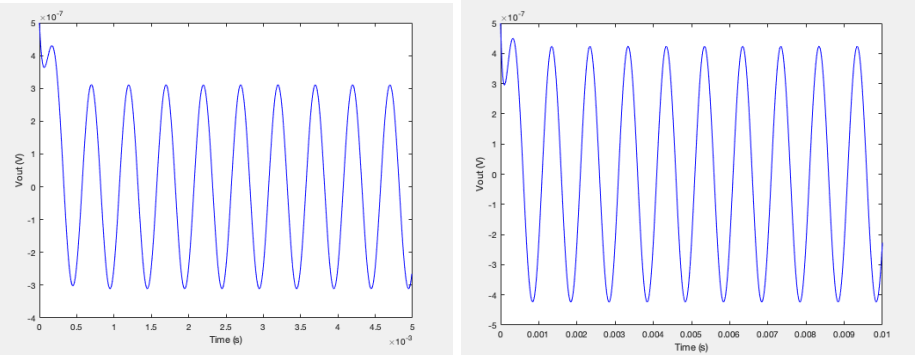


Figure 9: Heun's method for sine wave of 500μs (left) and 1000μs (right)

1.7 Square wave

The RC circuit is provided with square waves with periods $T = 10$ μs, $T = 100$ μs, $T = 500$ μs, $T = 1000$ μs and input voltage $V_{in} = 5$ V. The graphs below show the result of passing these waves through the low pass filter. As mentioned in the earlier sections, a capacitor cannot change its voltage instantly, and so a sudden change in the input voltage cannot be reflected in the output voltage. As a result, a transient can be observed in all of these waveforms, as the initial capacitor voltage is higher than the input voltage. Similarly, due to this property of the capacitor, the output waveform is not a square wave, but rather a triangular wave, because the voltage increases

and decreases gradually. The drop, however, is steeper than the increase, because the low pass filter cuts off high frequency components of the wave. The voltage drop across the capacitor alternates between $+V_{out}$ and $-V_{out}$, and hence the shape of the graph.

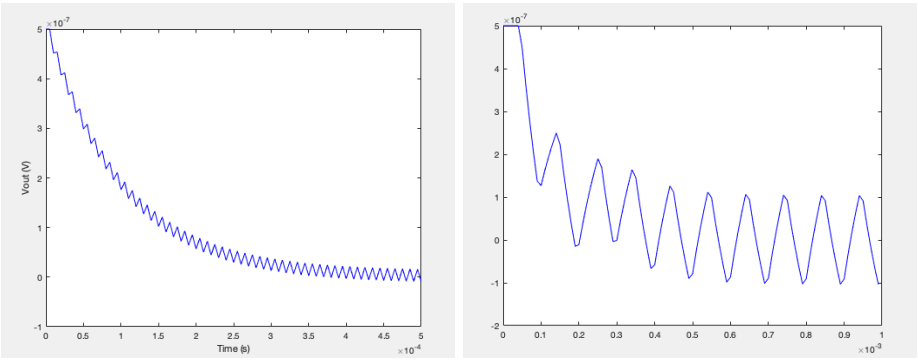


Figure 10: Heun's method for square wave of 10μs (left) and 100μs (right)

For a higher frequency square wave, the output waveform is a triangular wave. This observation can be explained using the transfer function of the circuit.
The transfer function of the circuit is given by:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{1 + RCs}$$

Using this we can say that at high frequencies, the 1 term in the denominator is very small compared to the RCs term therefore we can say:

$$\frac{V_{out}(s)}{V_{in}(s)} \approx \frac{1}{RCs}$$

Which is the transfer function of an integrator circuit.
We know that a square wave integrated gives the triangle wave and so our observations support the theory. The output wave form is shown in Figure 10.

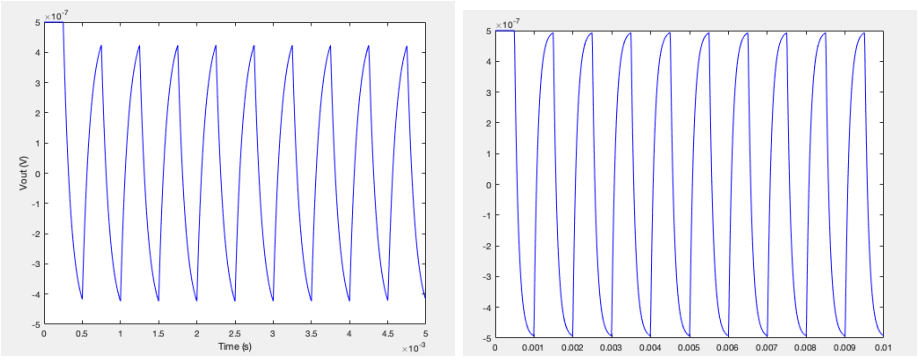


Figure 11: Heun's method for square wave of 500μs (left) and 1000μs (right)

1.8 Sawtooth wave

Similar to the square wave (discussed above), sawtooth waves with periods $T = 10$ μs, $T = 100$ μs, $T = 500$ μs, $T = 1000$ μs and input voltage $V_{in} = 5$ V were provided to the RC circuit. The waveforms obtained upon passing the signal through the low pass filter were similar to those of the square wave. A similar explanation can be used to understand this behaviour. The transient response is the result of the initial (capacitor) voltage being higher than the input voltage.

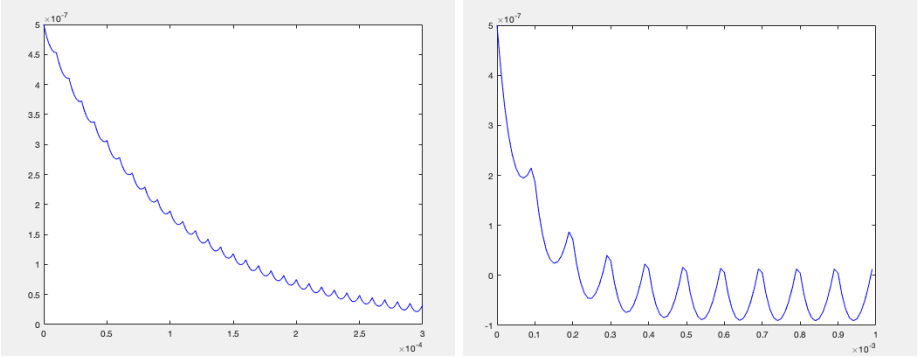


Figure 12: Heun's method for sawtooth wave of 10μs (left) and 100μs (right)

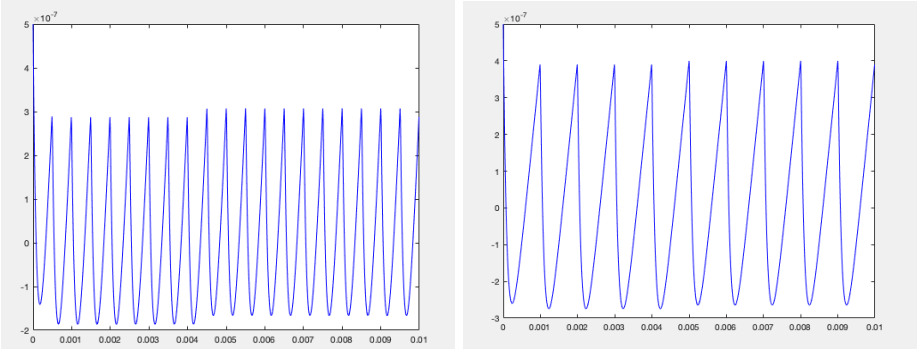


Figure 13: Heun's method for sawtooth wave of 500μs (left) and 1000μs (right)

2 Exercise 2 - Error Analysis

Please refer to the Appendix [5.2.3]for the full code for the error script

The constants used are:

$$R = 1000, C = 10E^{-7}, T = 10E^{-4}, V = 5, \omega = \frac{2\pi}{T}, q_{c_0} = 5E^{-7}, t_0 = 0, h = 0.0001, t_f = 0.001$$

It is worth mentioning that an error analysis can be performed only when the ODE can be solved analytically and an explicit exact solution for the ODE can be found, so that the numerically approximated solution can be compared to the exact solution. For this reason we performed an error analysis not only for the sinusoidal input signal but for the exponential decay input signal as well, for which an exact solution can be found in the above section.

2.1 RK2.m

The code for RK2.m which can be found in Appendix [5.2.3]is identical to the one was used in Exercise 1. The Second-Order Runge-Kutta (RK2) method is defined as a function with 6 parameters. The first takes in the ODE as a function of q_C and t, while the second and third parameters take in the initial values of 't' and '$q_C(t)$' respectively. The fourth parameter is the time step-size (i.e. h) between consecutive numerical approximation iterations and the fifth takes in the value of a, which is used to both determine which of the three RK2 method to be employed (i.e. Heun's, Midpoint and the random RK2 method of our choice), as well as to determine the values b, p, and q.

$$q_{c+1} = q_C + h(ak_1 + bk_2)$$

$$k_1 = \frac{dq_C}{dt}(t_i, q_{c_i}), k_2 = \frac{dq_C}{dt}(t_i + ph, q_{c_i} + qk_1h)$$

Which must satisfy:

$$a + b = 1$$
$$bp = \frac{1}{2}$$
$$bq = \frac{1}{2}$$

Where q_C and q are NOT the same variable.

These in turn define which of the infinite many Second-Order Runge-Kutta methods we employ. The last parameter tells our function at which interval we end our function and return the array of values it has calculated.

2.2 Sinusoidal Input Signal

2.2.1 Numerical and Exact Solution, Error

The input function to the circuit ODE:

$$\frac{dq_C(t)}{dt} = V_{in}(t) - \frac{1}{RC}q_C(t)$$

is:

$$V_{in} = Vcos(\omega t)$$

The code for error_script.m defines all the necessary constants including 'a', whose value determines which RK2 method will be employed by the RK2.m script to numerically approximate the solution of the ODE. The input function is then defined, which in this case is a cosine wave of period T and amplitude V. The RK2.m function is then invoked. Finally, the numerical solution is compared with the exact solution, which we obtained by analytically solving the ODE, for every value of t. The exact solution we retrieved was:

$$q_C(t) = \frac{e^{\frac{-t}{RC}}(C^2\omega^2R^2 + 2000000C^2\omega RVe^{\frac{t}{RC}}sin(\omega t) + 2000000CVe^{\frac{t}{RC}}cos(\omega t) - 2000000CV + 1)}{2000000(C^2\omega^2R^2 + 1)}$$

Please note that we are plotting $q_C(t)$ instead of V_{out} for simplicity. The only difference being that V_{out} is calculated as:

$$V_{out} = \frac{1}{C}q_C(t)$$

The plots will be thus scaled up by C. Scaling down the graph by C, one could very easily obtain the graph for V_{out}.

The error is then calculated as the exact value minus the numerical value:

$$Error = q_{c_{exact}} - q_{c_{numerical}}$$

The script then plots the numerical and exact solution as $q_C(t)$ (see graphs). Then, the script plots the error as a function of t as well.

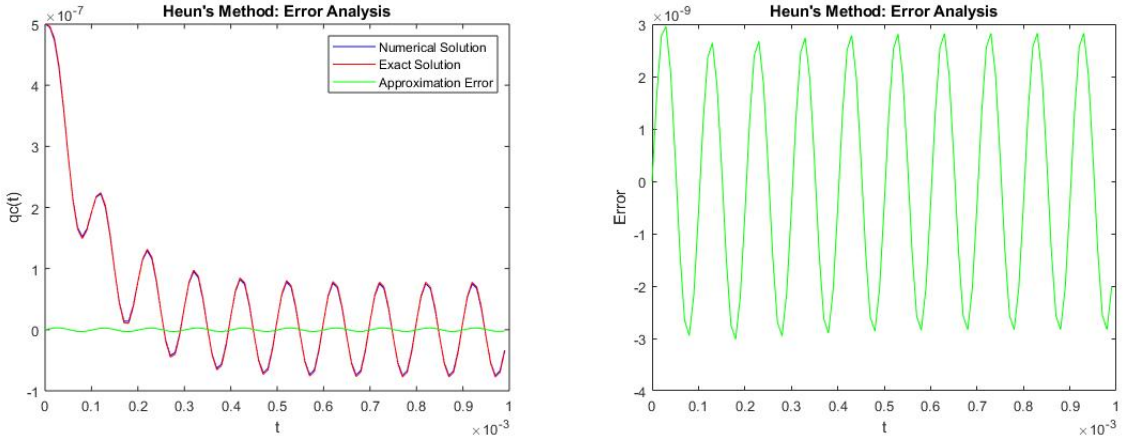


Figure 14: Heun's method

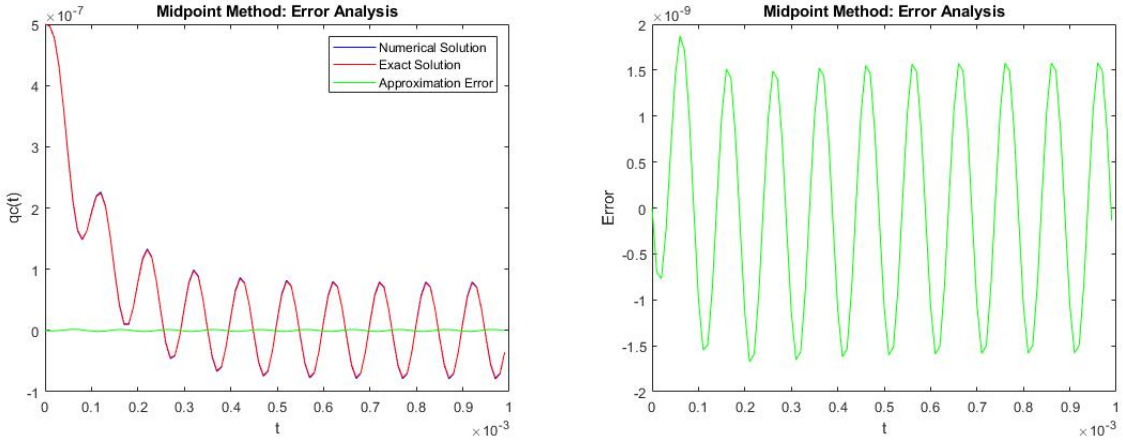


Figure 15: Midpoint method

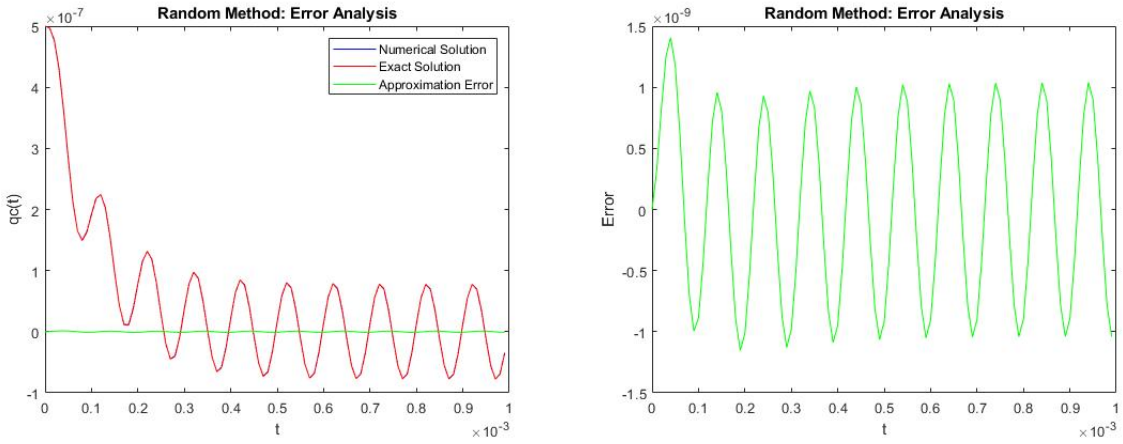


Figure 16: Random method

As we can see, our error as a function of t is a cosine wave for all three methods. Given that the numerical and exact function both represent cosine-characteristic waves, this is expected. We can also see that the selected parameter values (e.g. step size, time interval) were appropriate considering the order of magnitude of the generated error, which can easily be observed by comparing the numerical solution graph to its corresponding exact solution graph, for each RK2 method.

2.2.2 Logarithmic Error Plotting

The script then is used to calculate the error as a function of step-size. This is done by using for-loops. Each one will iterate between $i = [16, 25]$. The step-size chosen for each iteration will be:

$$h = 2^{-i}$$

The reason for starting at 16 and not a smaller number is because we found that small numbers made the error plot act unpredictably, making the error jump up and down. The reason for this is probably the fact that the closer the orders of magnitude of the step-size and the time interval size are, the greater the generated error. A larger initial value (i.e. greater than 15), and thus a smaller step-size, gave much more reliable numerical solutions.
For each step-size h we invoked the RK2.m function, getting different numerical solutions for each invocation. The error is then calculated as the maximum value of the absolute value of the exact solution minus the numerical approximation:

$$Error_{max} = max|q_{c_{exact}} - q_{c_{numerical}}|$$

Using a log-log plot, we plot this value on a graph for each h. After repeating this for 10 iterations, for each method, we get the following plot.

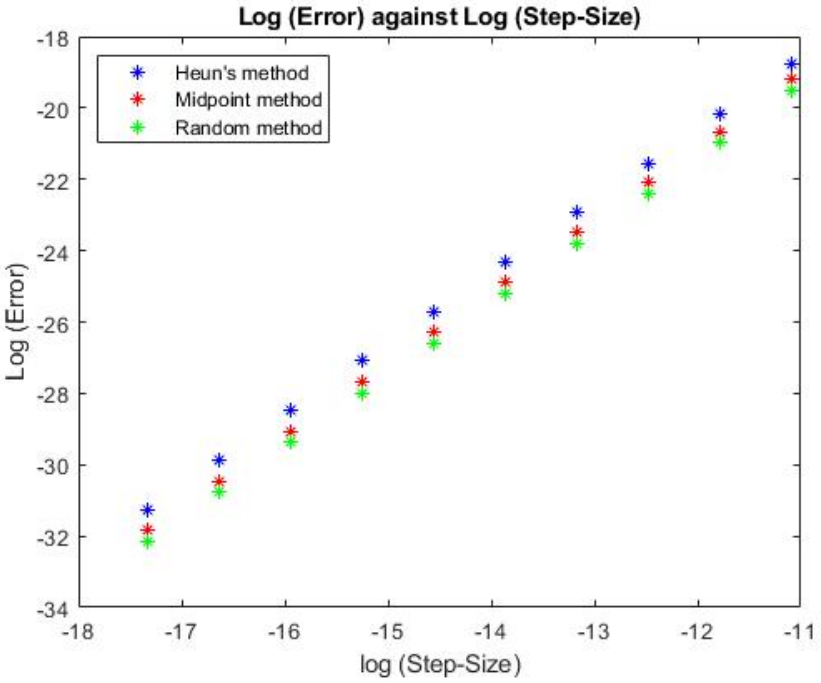


Figure 17: Plot of the log(error) vs log(step-size)

When calculating the truncation error, we are expecting to see $O(h^2)$ as the order of the global truncation error. This means that $O(h^3)$ would be the order of local truncation error, since we know that when the local truncation error is of order $O(h^n)$, the global truncation error will be of order $O(h^{n-1})$.
Given that the global error, which is the error plotted on the above graph, should represent a sort of parabola with h as the independent variable, we are expecting the log-log plots to be linear. To be more specific, we are expecting a linear curve of gradient 2 and different y-intercepts for each RK2 method. The reason for this is that when we say "the order of the global truncation error is $O(h^2)$", we mean that the order is proportional to h^2, meaning that h^2 is multiplied by any constant c. Thus, taking the logarithm of that would result in:

$$ln(Error_{max}) \approx c + 2ln(h)$$

Therefore, plotting in logarithmic scales this constant c becomes the y-intercept of the linear curve and 2 its gradient, which is exactly what we observe from the graph above. Hence, we can confirm that the global truncation error is indeed represented by our resulting log-log plotted errors.

Comparing the three methods' log-log plots against each other, we can also see that there are minor performance differences associated with each RK2 method used. Specifically, the random RK2 method seems to slightly outperform the other two, since the error generated is the lowest for every value of h in comparison with the other two RK2 methods. The Midpoint method follows and Heun's method the last one in terms of performance. However, we should note that the order of magnitude of the differences in performance (i.e. the differences in the error generated by each RK2 method) is 10^{-15}, and hence they are considered negligible.

2.2.3 Semi-log Error Plotting

The last for-loop of the script works just like the previous, except that instead of plotting the max error as a function of the step-size in logarithmic axes, it plots the max error as a function of the step-size using a semi-log plot, where only the step-size is in logarithmic scale.

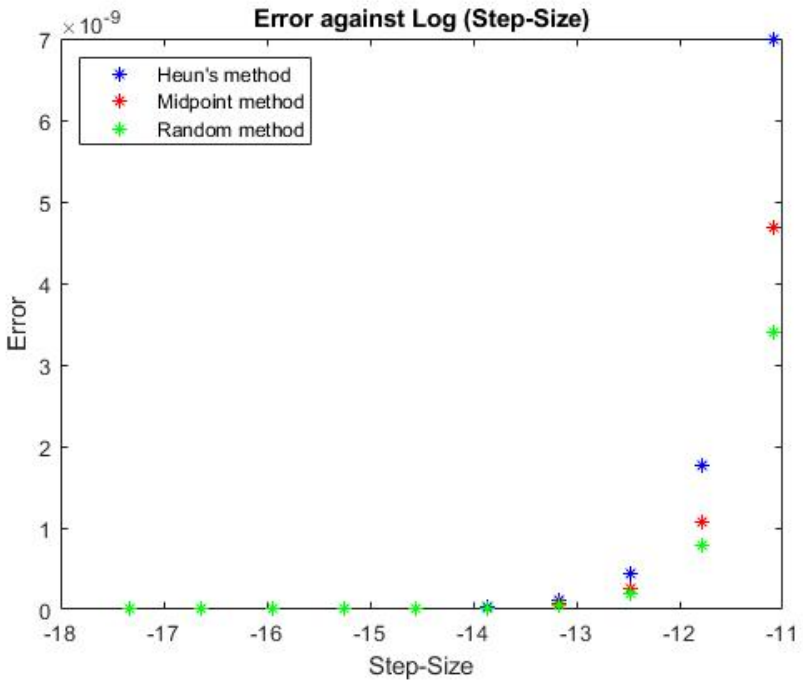


Figure 18: Plot of the error vs log(step-size)

As we can see from the graph above, the increase of the $Error_{max}$ (i.e. of the global truncation error) appears to be exponential. This is expected considering that:

$$Error_{max} \approx e^{ln(ch^2)} = e^{ln(c)+2ln(h)} = ce^{2ln(h)}$$

2.3 Exponential Decay Input Signal

2.3.1 Numerical and Exact Solution, Error

The input function to the circuit ODE:

$$\frac{dq_C(t)}{dt} = V_{in}(t) - \frac{1}{RC}q_C(t)$$

is:

$$V_{in} = Ve^{\frac{-t}{\tau}}$$

where

$$V = 2.5, \tau = RC$$

The exact solution we retrieved was:

$$q_C(t) = \frac{V}{R}te^{\frac{-t}{RC}} + 2Ve^{\frac{-t}{RC}}$$

As mentioned above, please note that we are plotting $q_C(t)$ instead of V_{out} for simplicity, meaning that the plots will be thus scaled up by C.

The error is again calculated as the exact value minus the numerical value:

$$Error = q_{c_{exact}} - q_{c_{numerical}}$$

The script then plots the numerical and exact solution as $q_C(t)$ (see graphs). Then, the script plots the error as a function of t as well.

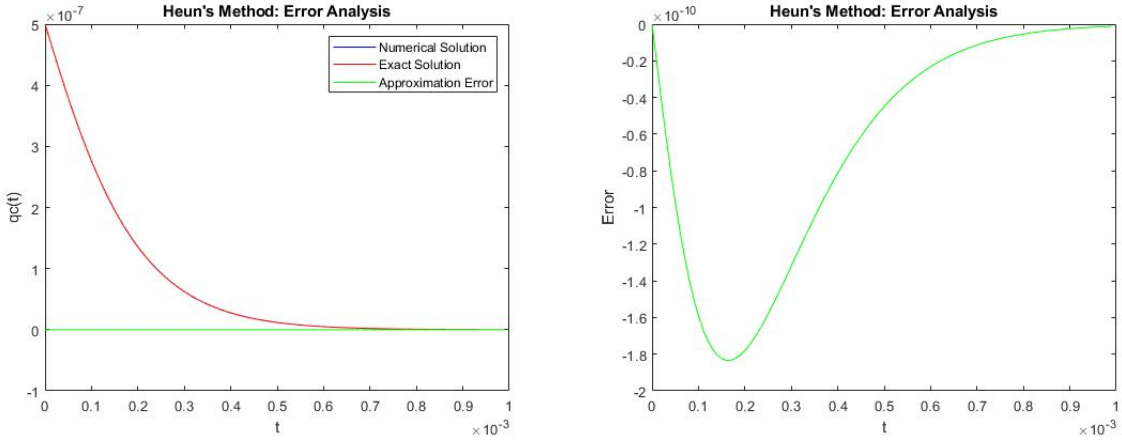


Figure 19: Heun's method

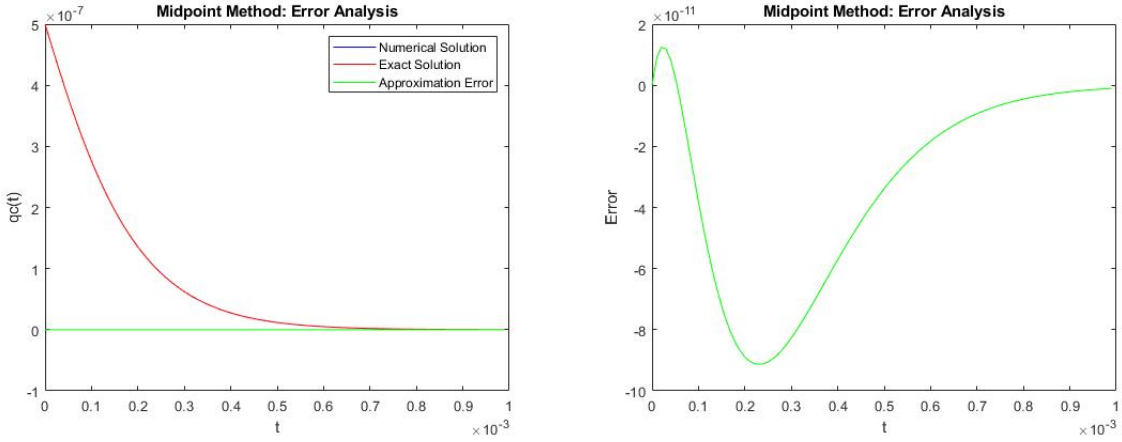


Figure 20: Midpoint method

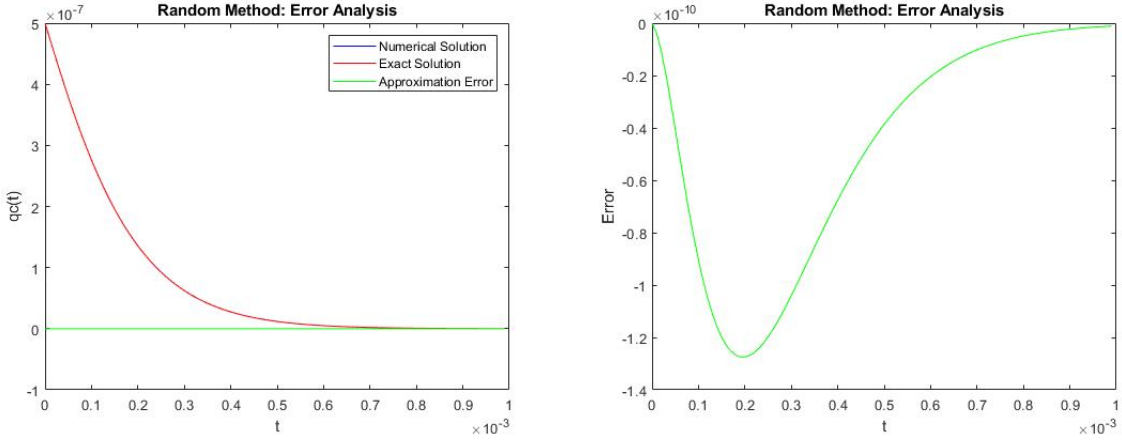


Figure 21: Random method

As we can see, our error as a function of t behaves in slightly different ways as we vary the parameters of the RK2 method. Specifically, the Midpoint method seems to produce a numerical solution that underestimates the exact solution initially, while the other two overestimate it. Yet, as t increases it is self-evident that all three methods have an error that tends to zero. We can also see that the order of magnitude of the error is relatively small compared

to both the numerical and exact solutions, which can also be easily seen by the fact that when both solutions are plotted together they are in fact very similar.

2.3.2 Logarithmic Error Plotting

The script is used to calculate the error as a function of step-size in exactly the same way as above, getting the following plot.

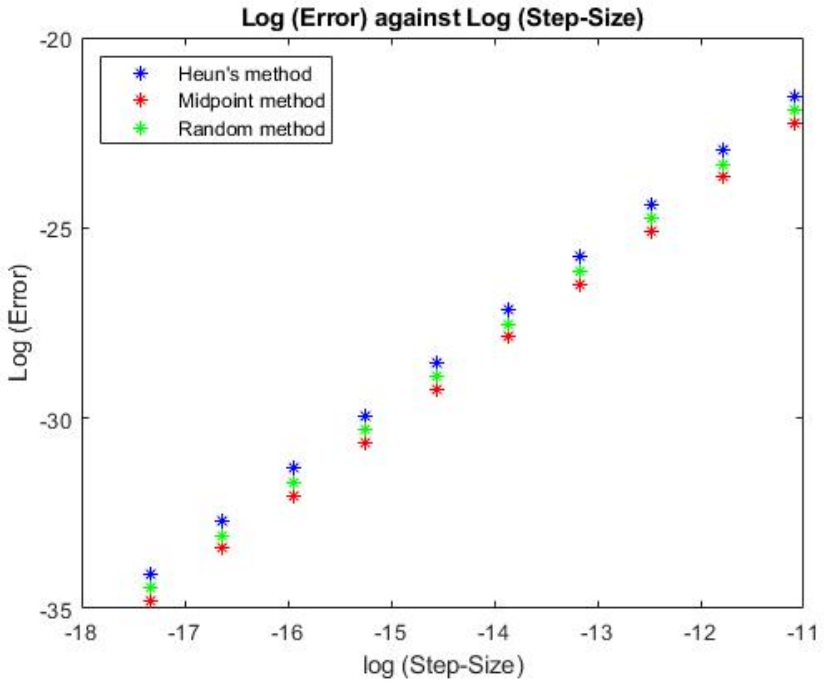


Figure 22: Plot of the log(error) vs log(step-size)

As mentioned above, plotting the error and the step size in logarithmic scales would generate a linear curve with gradient 2, which is exactly what we see, confirming that the global truncation error indeed can be represented by our resulting log-log plotted errors.

By comparing the three above linear graphs, we can once again see minor performance differences among the three used RK2 methods. It seems that the most efficient method (i.e. the method that produces the lowest errors for any step-size) is the Midpoint RK2 method, with the random method following and the Heun's method be the least efficient. It is important to note though that these differences in efficiency are so small, making them negligible. To be precise, their order of magnitude is 10^{-16}.

2.3.3 Semi-log Error Plotting

Using a semi-log plot, where only the step-size is in logarithmic scale, we can observe that indeed the truncation error is of magnitude $O(h^2)$, since its increase appears to be exponential as the respective graph for the sinusoidal input signal.

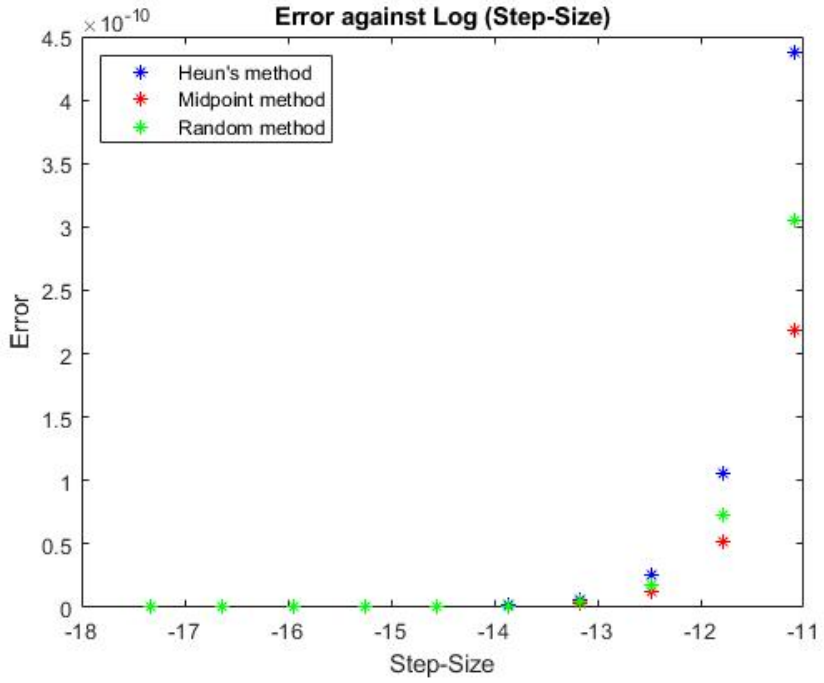


Figure 23: Plot of the error vs log(step-size)

3 Exercise 3 - Fourth Order Runge-Kutta

3.1 The Circuit

RLC circuits form a harmonic oscillator for current. The circuit shown below has all three components in series. Thus, one can find the governing differential equation using Kirchhoff's voltage law:

$$V_{in}(t) = V_r + V_I + V_c$$

The damping factor ζ describes the transient response of an RLC circuit. In this case, it is given by

$$\frac{R}{2}\sqrt{\frac{C}{L}}$$

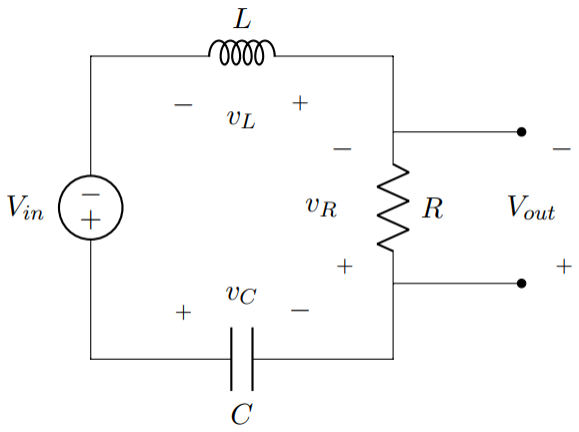


Figure 24: The RLC circuit

A damping factor $\zeta < 1$ corresponds to an under-damped transient, while $\zeta > 1$ and $\zeta = 1$ correspond to an over-damped and a critically damped transient respectively. The given values for resistance, capacitance, and inductance yield a damping factor of 0.95. This predicts the transient to be slightly under-damped.

$$Q = \frac{1}{R}\sqrt{\frac{C}{L}}$$

3.2 Runge-Kutta 4th Order Method

Runge-Kutta 4th order (RK4) is a numerical technique used to approximate first-order differential equations. It does this by combining Euler's method with an iterative model of taking four gradients between a set distance. These four approximations are weighted and averaged to give the final answer. The two figures below show a simple problem and the four approximations on it. The 4 equations are:

$$k_1 = f(t, y);$$

$$k_2 = f(\frac{t+h}{2}, \frac{y+h}{2})$$

$$k_3 = f(\frac{t+h}{2}, \frac{y+h}{2})$$

$$k_4 = f(t+h, y+h)$$

And they are weighted accordingly:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

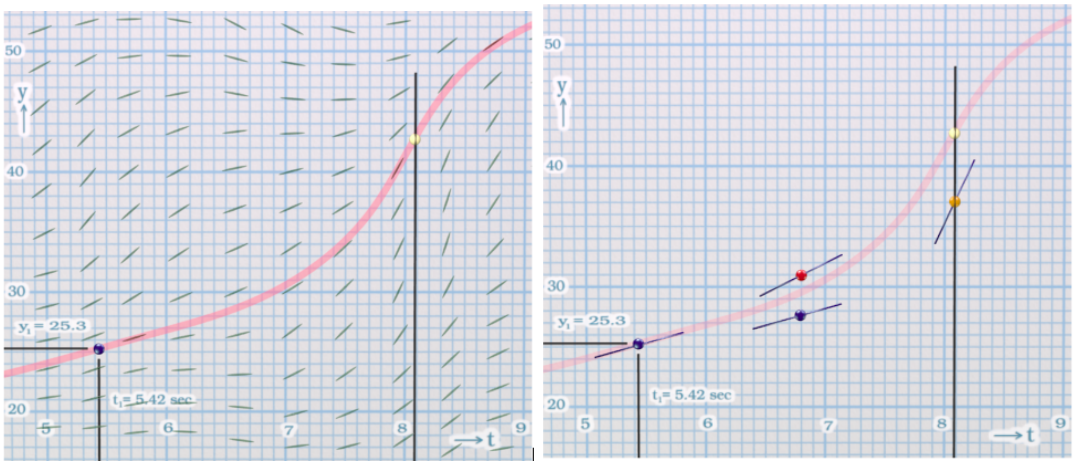


Figure 25: Some y'=f(t, y) with initial conditions (left) and RK4 approximations (right)

3.3 Implementation

3.3.1 RK4

RK4 is the implementation of classic Runge-Kutta 4th order. V_{out}, F_2, F_1, h, t, q_C, $q_C grad$ and N are all passed to the functions. Here it uses two sets of Runge Kutta equations for the approximations, K_n and M_n, unlike 1^{st} Order Differential Equations Runge-Kutta. The this method calls two functions, F_1 and F_2, derived from the circuit:

$$F_1 = q_C grad$$

$$F_2 = (V_{in}(t) - (q_C)/C - (R * q_C grad))/L$$

The equations for Runge-Kutta are shown below are shown below:

$$k_1 = F_1(t(i), q_C(i), q_C grad(i));$$

$$m_1 = F_2(t(i), q_C(i), q_C grad(i));$$

$$k_2 = F_1(t(i) + 0.5 * h, q_C(i) + (0.5 * h * k_1), q_C grad(i) + (0.5 * h * m1));$$
$$m_2 = F_2(t(i) + 0.5 * h, q_C(i) + (0.5 * h * k_1), q_C grad(i) + (0.5 * h * m1));$$
$$k_3 = F_1(t(i) + 0.5 * h, q_C(i) + (0.5 * h * k_2), q_C grad(i) + (0.5 * h * m2));$$
$$m_3 = F_2(t(i) + 0.5 * h, q_C(i) + (0.5 * h * k_2), q_C grad(i) + (0.5 * h * m2));$$

$$k_4 = F_1(t(i) + h, q_C(i) + (k_3 * h), q_C grad(i) + (m3 * h));$$
$$m_4 = F_2(t(i) + h, q_C(i) + (k_3 * h), q_C grad(i) + (m3 * h));$$

K_n represents the approximation for q_C and M_n the approximation for $q_C grad$.

3.3.2 RLC script

The RLC script is written to set up the variables for the representation of the RLC circuit. These also include the initial values for the time, charge and the derivative of the charge. In the script, the vectors are initialized with their first value without allocating their final size. Instead, the size of the vector is increased dynamically each time a value is added. This way, the time taken to run the script was greatly reduced as the vectors do not have to be accessed twice.

The RLC script also sets up the different functions for V_in and the coupled first order differential equations. By simply changing the value of n for $V_in(n)$, each input is changed to the respected functions (See Appendix [5.2.3]).

This method uses coupled first order differential equations using each equation to approximate itself. Runge-Kutta with a first order differential equation, uses the gradient function to approximate the true signal. With a second order differential equation, it uses the second derivative function to approximate the gradient function and then uses this approximation to make an approximation on the true signal. The two functions are as follows:

$$Y = q'$$

$$Y' = \frac{\frac{vin - Ry - 1}{cqc}}{L}$$

3.4 Step Signal

For this input we have a heaviside function with an amplitude of 5V. As seen in the graph, once the capacitor gets charged there is no further change in the potential difference, which thus tends to zero.

The resonance frequency of the circuit is given by $\frac{1}{\sqrt{LC}} \approx 690\frac{rad}{s} \approx 110Hz$. To find out if the system is over or under damped, we can use the attenuation; the speed of how quickly the transient response will die away once the input is removed. $\alpha = \frac{R}{2L} \approx 208$. Because α is much larger than the resonance frequency, the system is under-damped. From these two values we can also work out what the frequency of the graph should show.

$$\omega_{5V} = \sqrt{110^2 - 208^2} \approx 105.$$

This can be seen in the graph shown below.

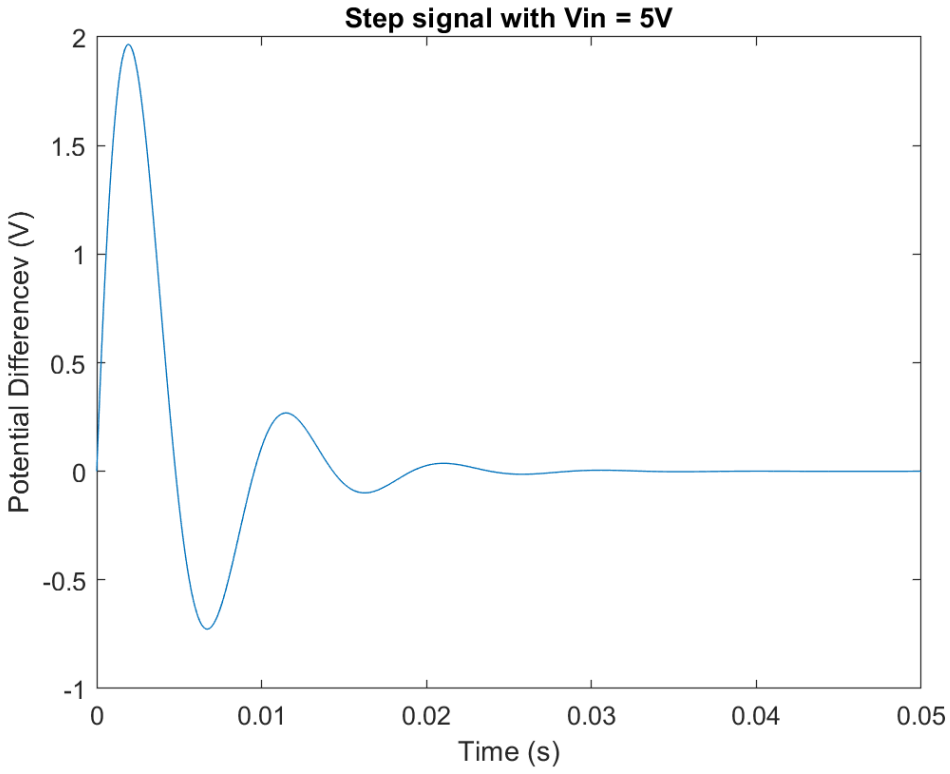


Figure 26: Step signal with input voltage 5V

3.5 Impulsive Signal with decay

The input for this system can be modelled with the equation:

$$V_{in} = 5 * exp(\frac{t^2}{\tau}) \; where \; \tau = 3 * 10^{-6}$$

With such a small τ, the signal decays very quickly, causing the output to do so as well. In comparison to the previous signal, the graph does not peak as high, while the frequency of the decaying signal is as predicted ≈ 105.

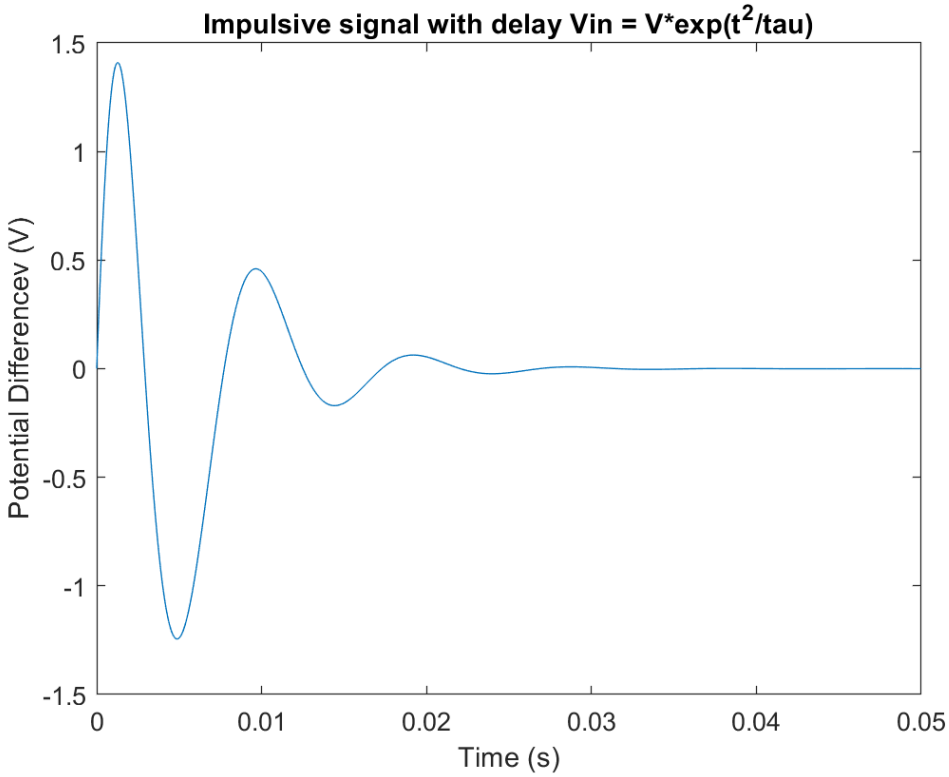


Figure 27: Impulsive Signal with decay

The next few signals are periodic signals. Each periodic signal was tested at 5Hz, 109Hz and 500Hz. A value of 109Hz was chosen because it closely resembles the resonance frequency of the system.

3.6 Square signal

For 5Hz there are two graphs, one long term and one more zoomed in. From looking at the zoomed in version, one can see the resemblance between that and the heaviside function. Due to changes in the voltage being so slow at 5Hz compared to the attenuation frequency, the signal dies out back to 0 after each change in frequency. At higher frequencies, the signal looks more like a saw tooth function with the potential difference being only 1V. However at 109Hz the signal looks more like the original after the first impulse but the amplitude goes up to 6V due the frequency being very similar to ω.

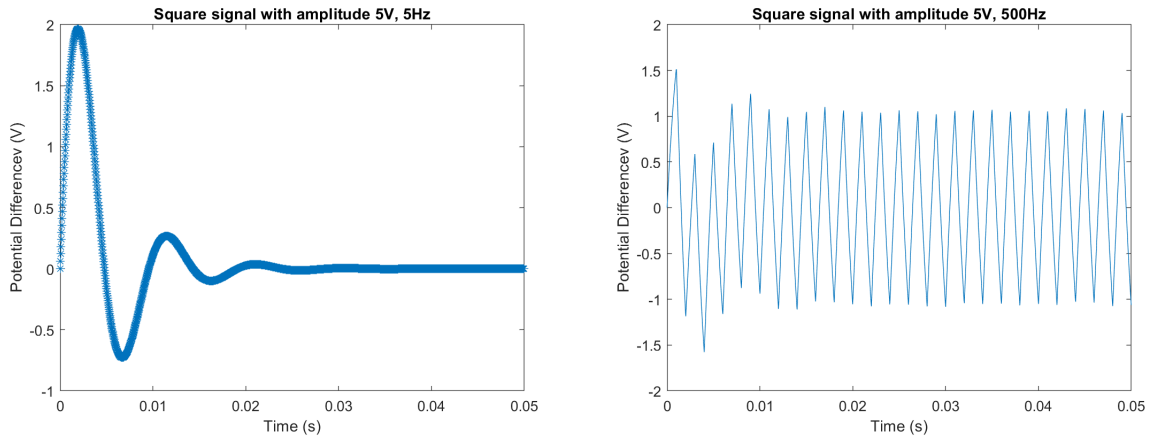


Figure 28: Square signal with amplitude 5V, frequency 5Hz (left) and frequency 500Hz (right)

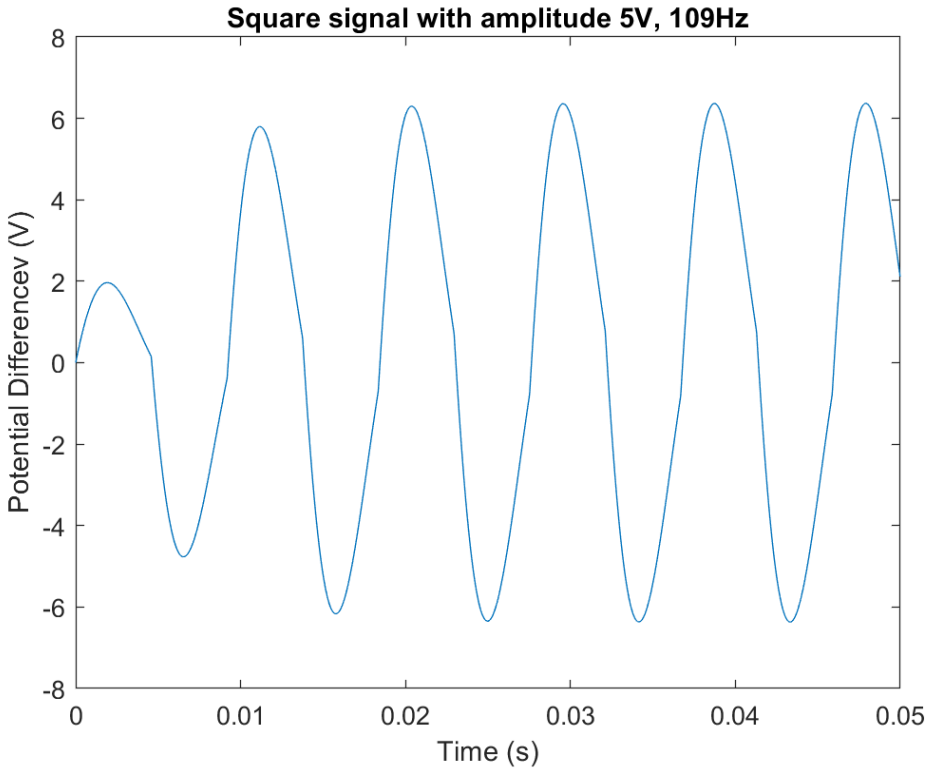


Figure 29: Square signal with amplitude 5V, frequency 109Hz (resembles resonance frequency of the system)

3.7 Sine wave

Here, the higher frequencies of 109Hz and 500Hz have a similar effect to the square wave function. The interesting part is at 5Hz. There are two graphs as before, one zoomed in and one not to show more detail. When the signal first starts it causes a spike like the other signals but then turns into the sine wave input (*Figure 30*). This is due to 5Hz ¡¡ , therefore the change each time unit is very small, making the circuit have no effect on the input.

If we work out the centre frequency for the band-pass filter, $\frac{1}{\sqrt{LC}} \approx 109Hz$. The bandwidth is $\frac{R}{L} \approx 66Hz$. So the lowest frequency is 43Hz and the highest is 175Hz. Therefore, the signal doesn't pass through the circuit correctly and has a spike at the beginning.

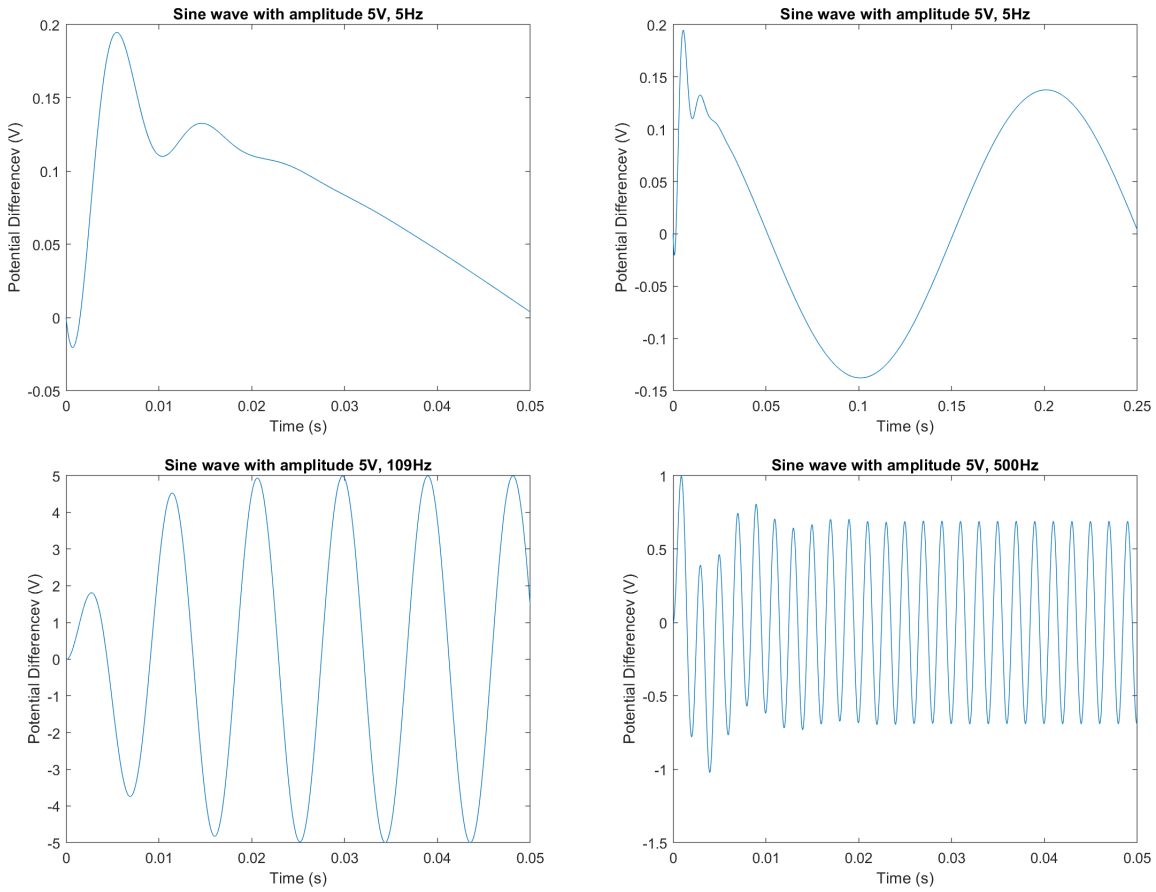


Figure 30: Sine signal with amplitude 5V, frequency 5Hz 0(top), 500Hz (left) and 109Hz (right)

3.8 Changing circuit values

After changing the resistance of the circuit to 10000, we took another plot at 500Hz. This time the bandwidth would now change to approx. 2653Hz. The plot below shows clearly how the signal passes through the circuit unaffected by the signal. We also change the resistance to 10. This is to see what affect a more severe bandwidth would have on the signal. As you can see the plot looks as the resistance of the wave is around 200Hz with a second frequency of the original 500Hz.

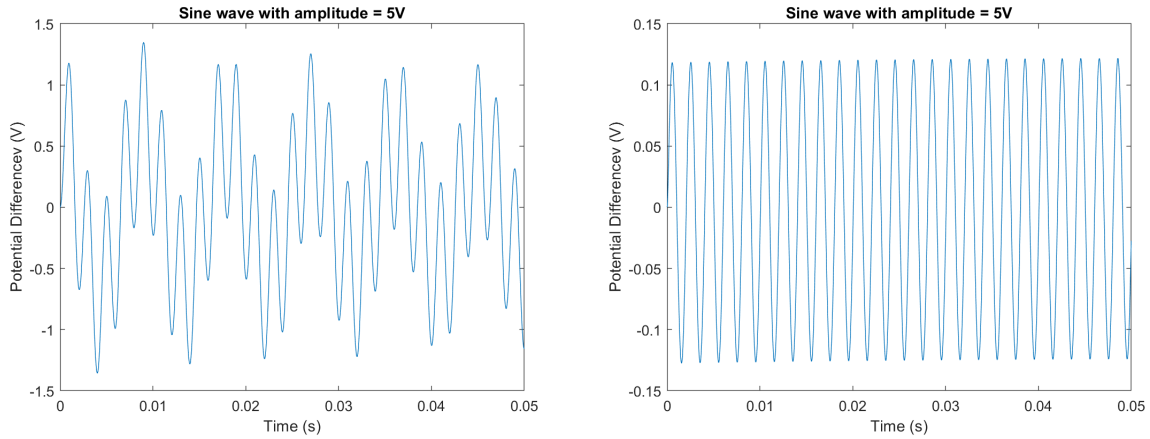


Figure 31: Sine signal with amplitude 5V and frequency 500Hz. Resistance $10\Omega(left) and 10000\Omega(right)$

4 Exercise 4 - Relaxation

4.1 Finite differences and the Laplace Equation

The Laplace equation in two dimensions is given by

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

and describes the curvature of a function **u** over the Cartesian plane with coordinates x and y. Since the Laplace equation is homogeneous, it relies solely on the boundary conditions imposed on the system, given which it can be solved computationally. To do so, the xy-plane is being divided into a grid with square size h, allowing a numeric approximation of the derivatives of our system using central differences.

Approximating the first derivative with the first central difference:

$$\frac{\partial u(x_i)}{h} = \frac{u(x_i + \frac{h}{2}) - u(x_i - \frac{h}{2})}{h} \approx \frac{du}{dx} \quad at\ x_i$$

Approximating the second derivative with the second central difference:

$$\frac{\partial(\partial u(x_i))}{h^2} = \frac{u(x_i + h) - 2u(x_i) + u(x_i - h)}{h^2} \approx \frac{d^2 u}{dx^2} \quad at\ x_i$$

Applying the same central differences to the y-dimension, indexed on the grid as j, the following shorthand will be used to denote the value of u at point u(x,y).

$$U_i^j = u(x_{i,j})$$

The resulting equation, together with the Laplace operator, define the value for every point on the grid based on its surrounding values.

$$U_i^j = \frac{1}{4}(U_{i+1}^j + U_{i-1}^j + U_i^{j+1} + U_i^{j-1})$$

4.2 Relaxation method

The relaxation method uses the above equation on a grid with relatively small square size h, together with the given boundary conditions, to recursively approximate the solution of the Laplace equation. More precisely, a residue r of every interior point is calculated in every iteration as the difference between the value at a point and the average value of its four neighbours. The residue is then added to the value at that point, decreasing on every iteration and recursively improving the approximation. This process terminates once it satisfies a terminating condition, whereto the residue for every interior point is less then a predefined maximum error ϵ.

```
while ~precise
    precise = true;
    for i = 2 : length(x)-1
        for j = 2 : length(y)-1 %for all interior points

            % Approximate U at i,j based on its residue
            r = ( U(i+1,j) + U(i-1,j) + U(i,j+1) + U(i,j-1) - 4*U(i,j)) / 4;
            U(i,j) = U(i,j) + r;

            % Terminating condition
            if abs(r) >= epsilon
                precise = false;
            end
        end
    end
end
```

For this implementation, all interior points of the grid are initialized as zero before the relaxation method is applied.

4.3 Application

Given the boundary conditions of a system, the Laplace equation can now be solved computationally.

$$u(0,y) = u(1,y) = y, \quad textfor\, 0 < y < 1;$$
$$u(x,0) = u(x,1) = x, \quad textfor\, 0 < x < 1$$

4.3.1 Varying h

By decreasing the grid square size h, one can achieve a higher resolution at the cost of heavier computation. The following graphs illustrate u(x,y) over the unit square for different values of h.

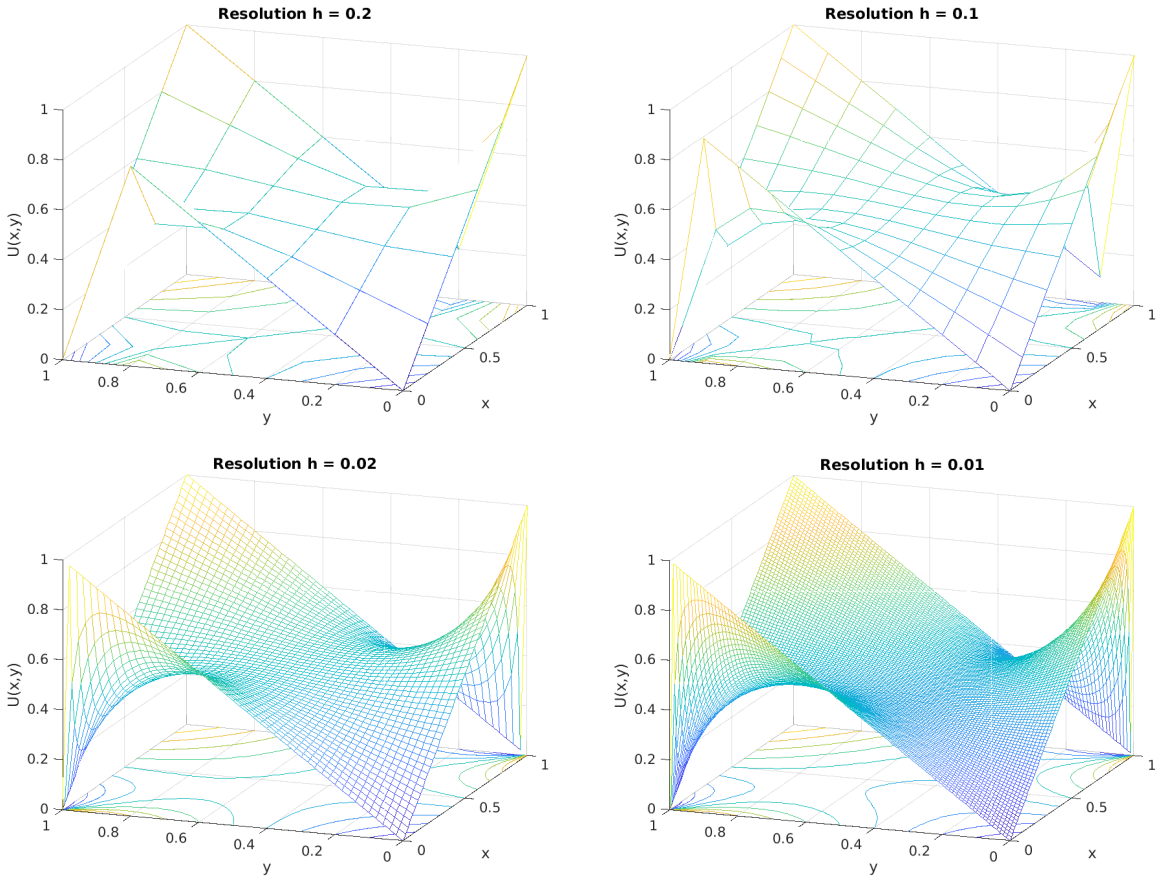


Figure 32: u(x,y) over unit square for h = 0.2 (top-left), 0.1 (top-right), 0.02 (bottom-left) and 0.01 (bottom-right).

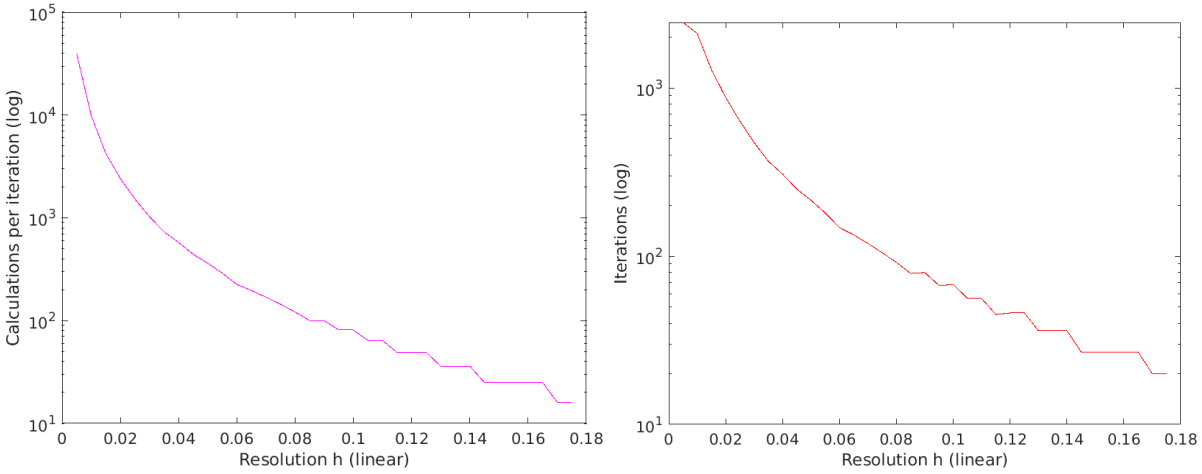


Figure 33: Calculations per iteration (left) and number of iterations (right) for different resolutions h

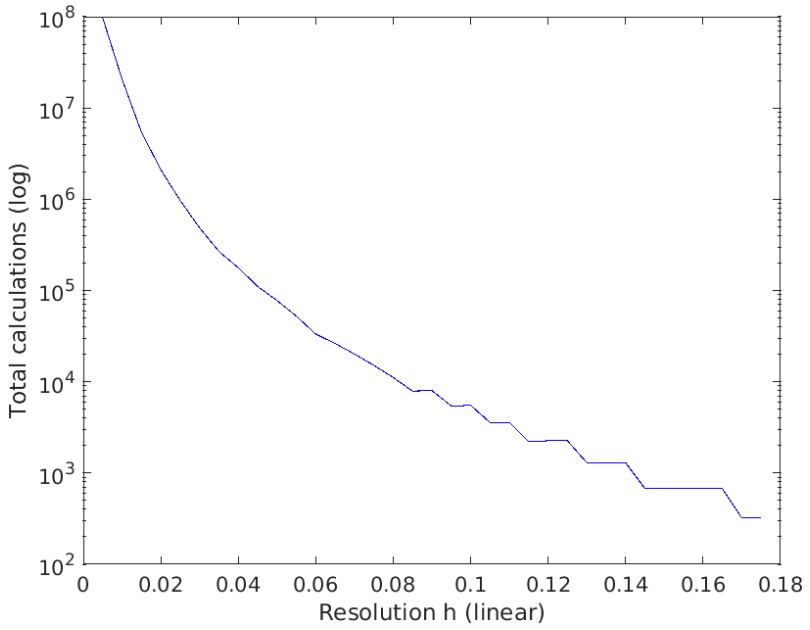


Figure 34: Total calculations (iterations * calculations) for different resolutions h

The residue has to be evaluated for every interior point on the unit square, leading to a partial complexity of $O(n^2)$ where $n = 1/h$. Furthermore, since the relaxation method recursively conveys information from the given corner sides to the middle of the grid, a finer grid will also require more iterations to similarly affect the centre point on the grid. More calculations per iterations together with more iterations yield an overall complexity which increases exponentially with a decreasing grid resolution h (*cp. Figure 32*). On the other side, low resolutions lead to a less precise graphical representation for non linear solutions to the Laplace equation. Similar to the Nyquist rate in signal processing, values of h bigger than $fracT2$ where T is the period of a signal, can lead to solutions that do not represent $U(x, y)$ at all (*cp. Figure 35*).

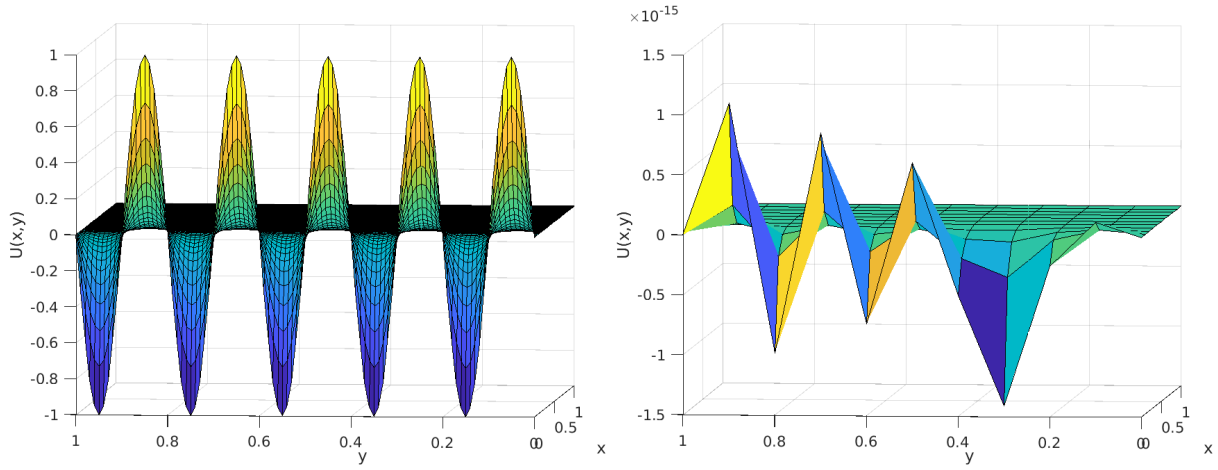


Figure 35: u(x,y) with high resolution h = 5000 (left) and low resolution h = 10 (right)

4.3.2 Varying ϵ

While h determines the resolution of the solution, ϵ, the maximum error, determines its accuracy (*see Ch. 5.2*).The below graphs display the same $u(x, y)$ for different values of ϵ.

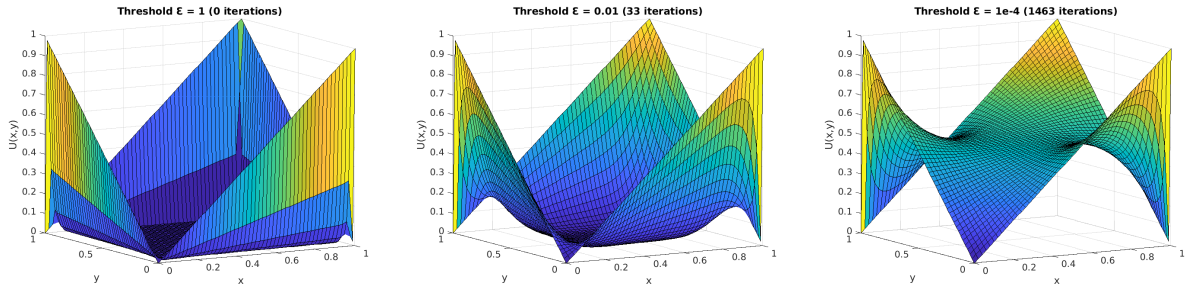


Figure 36: u(x,y) with different maximum error – left to right: ϵ = 0.1, 0.01, 1e-3, 1e-4

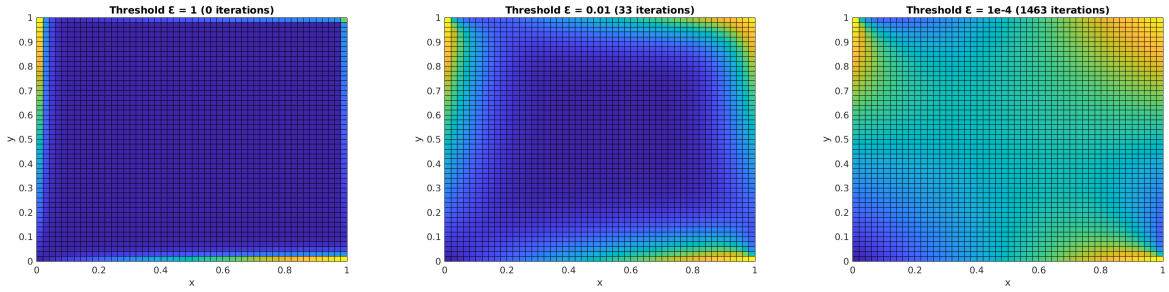


Figure 37: u(x,y), from above with different maximum error – left to right: ϵ = 1, 0.1, 0.001

Being it's terminating condition, ϵ determines the complexity of our implementation. Although the calculations in each iteration stay the same, the amount of iterations increases exponentially with a higher ϵ.
The implemented algorithm terminates once the residue is lower than ϵ for all interior points. For a high maximum error, the residue of points near the boundary conditions fall below ϵ much more quickly. Since the maximum error merely limits the difference between a point and its neighbours and does not represent the effective error of the evaluated solution, a high ϵ thus results in a close-zero residue that complies with the terminating condition. This leaves points near the centre near zero, as illustrated by the residue distributions in *Figure 38*.

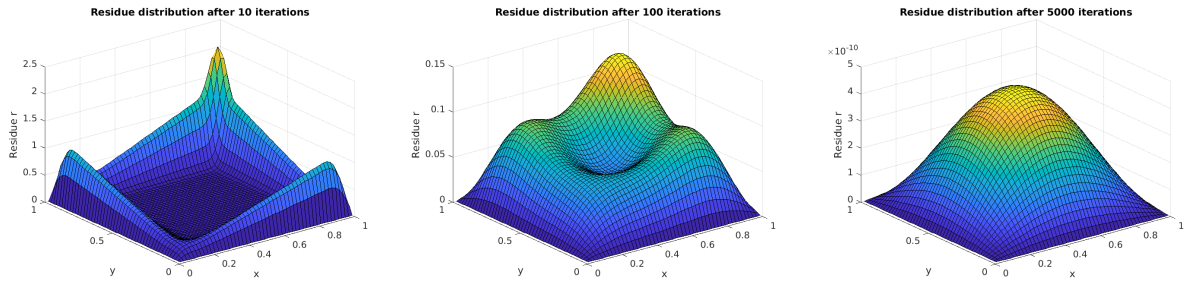


Figure 38: Residue distribution after (left to right): 10 iterations, 100 iterations, 5000 iterations

For sufficiently small ϵ, the maximum error is at the centre point of the grid, farthest away from any boundary conditions. Moreover, the residue at the the corner discontinuities U(1,0) and U(0,1) is and stays close-zero since the relaxation method simply evaluates it as the average values of the surrounding boundary conditions. This is the case for all discontinuities as long as they are on the edges of the grid. The following illustrations of the solution of the Laplace equation on the unit square with boundary conditions.

$$u(0,t) = u(1,t) = u(t,0) = u(t,1) = \begin{Bmatrix} 1, & 0 < t \leq 0.25; \\ 0, & 0.25 < t \leq 0.5; \\ 1, & 0.5 < t \leq 0.75; \\ 0, & 0.75 < t \leq 1; \end{Bmatrix}$$

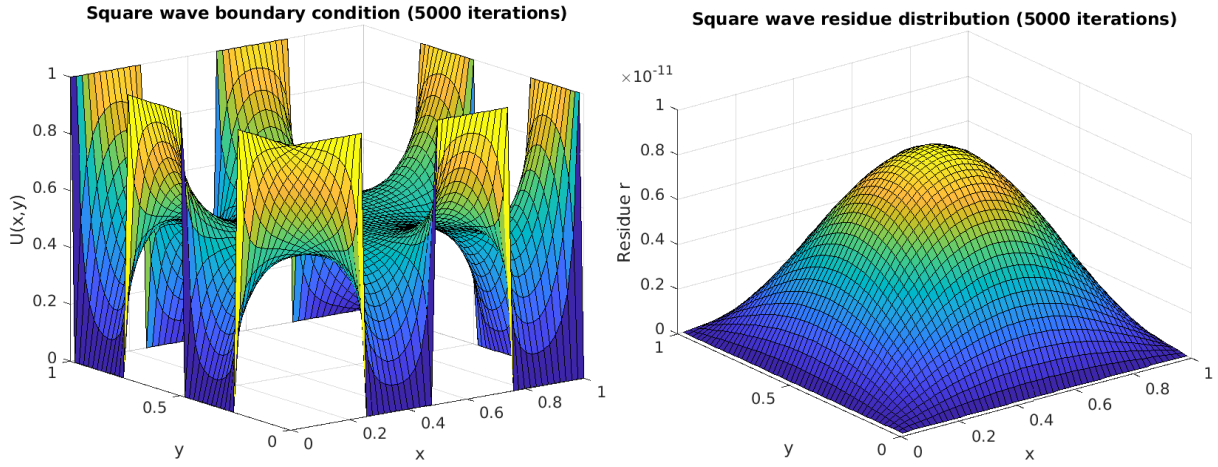


Figure 39: Solution of the Laplace equation on the unit square with discontinuities on all BC

4.3.3 Scaled boundary conditions

Figure 40 displays the solution for the Laplace equation on the unit square with boundary conditions defined as

$$u(0,y) = u(1,y) = 0,\ for\ 0 < y < 1;$$
$$u(x,0) = u(x,1) = c,\ where\ c \in \Re\ for\ 0 < x < 1$$

.

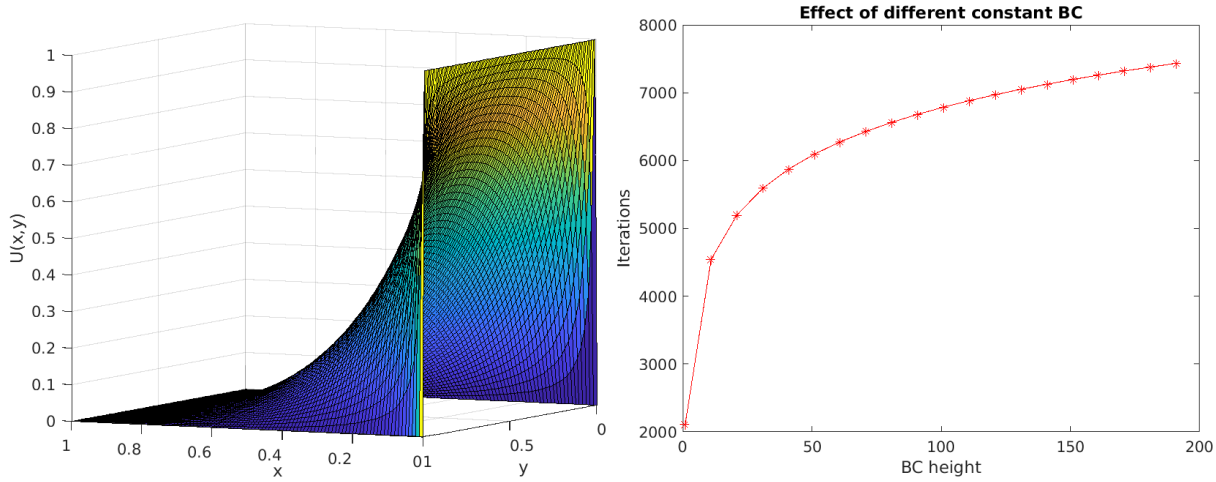


Figure 40: Solution of the Laplace equation on the unit square with constant boundary condition of varying height

Since the relaxation method evaluates the next approximation of a point as the average of its neighbours, the difference between two residues of the same point at subsequent iterations is steadily decreasing after every iteration. Starting with a higher difference between initial values of the interior points and the boundary conditions of the system, as is the case with a higher c, thus will lead to a logarithmic increase in iterations needed to satisfy the termination condition.

4.3.4 Sinusoidal boundary conditions

Figure 37 displays the solution to the Laplace equation with boundary conditions

$$u(0,t) = u(1,t) = u(t,0) = u(t,1) = sin(2\pi t),\ for\ 0 < t < 1$$

while the solution in *figure 38* has boundary conditions

$$u(0,t) = u(1,t) = u(t,0) = u(t,1) = -sin(2\pi t),\ for\ 0 < t < 1$$

.
Since both systems have finite boundary conditions, the relaxation method evaluates them without any problems. Applying the relaxation method to a system with boundary conditions

$$u(0,t) = u(1,t) = u(t,0) = u(t,1) = tan(2\pi t),\ for\ 0 < t < 1$$

on the other hand simply would not terminate since it would contain points with infinite value.
Interestingly, the residue distribution for *Figure 41* and *42* is of opposite sign, depicting the fact that the implemented relaxation method evaluates the interior points at every iteration from U_0^0 to U_1^1. A positive sine wave, being of higher value than the centre point, thus approximates the interior points in the centre from below, resulting in a negative residue. The opposite is the case for the negative sine wave, displaying what would happen if the algorithm would evaluate all interior points starting at U_1^1 instead.

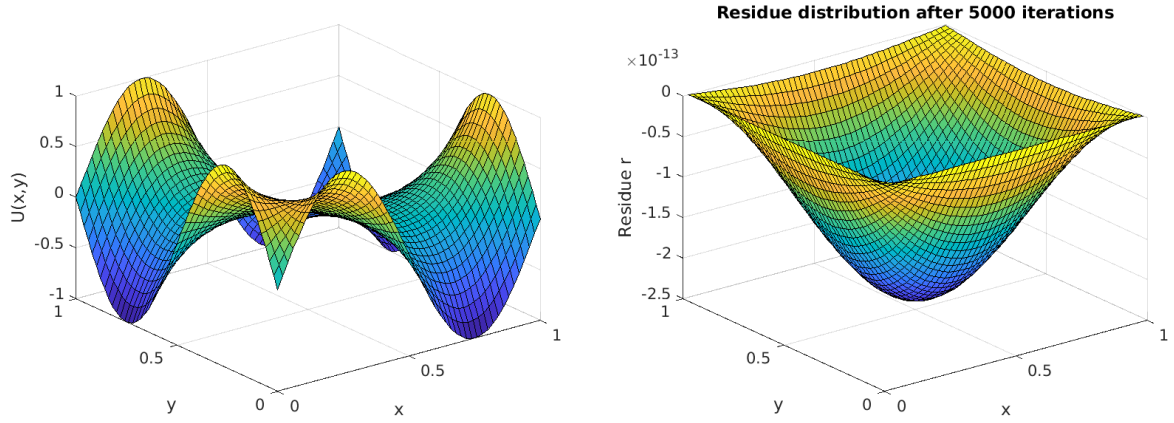


Figure 41: *Left:* Solution to the Laplace equation on the unit square for boundary conditions sin(x); *Right:* corresponding residue distribution after 5000 iterations

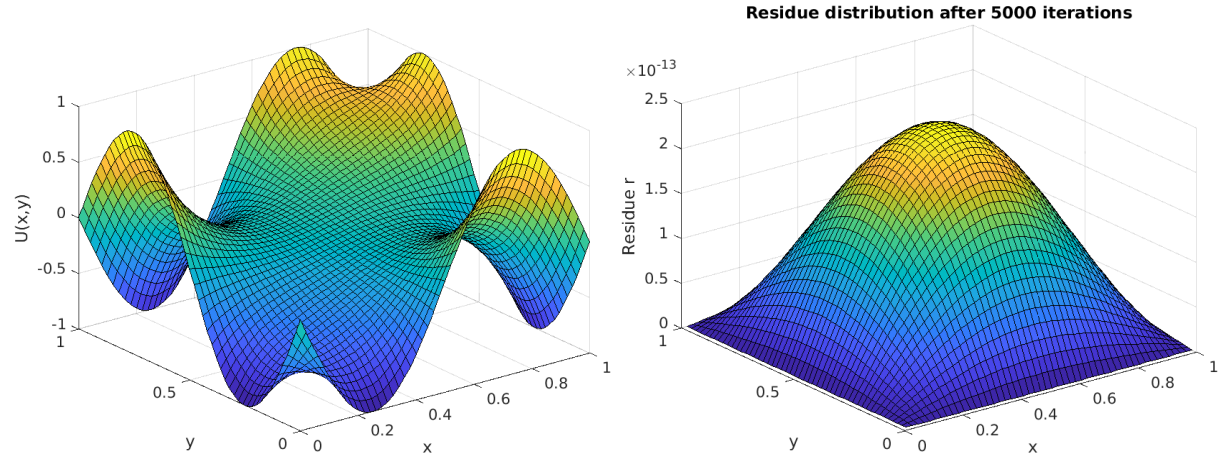


Figure 42: *Left:* Solution to the Laplace equation on the unit square for boundary conditions -sin(x); *Right:* corresponding residue distribution after 5000 iterations

5 Exercise 5 - Successive Over-Relaxation

In order to speed up the computation process, one can alter the weighting between the old value of some interior point and its residue when calculating its new value. The weighting of the residue is called the relaxation factor.

$$U^j_{i\ new} = U^j_{i\ old} + \alpha * r^j_i$$

While the normal relaxation method uses a relaxation factor of one, the method of Successive Over-Relaxation uses some value greater than one. This amplifies the effect the residue has on every update carried out on a point, reaching the terminating condition faster. Put another way, a higher relaxation factor can decrease the amount of iterations needed to find the solution of the Laplace equation without loosing its accuracy.

5.1 Range of relaxation factor α

The challenge of applying Successive Over-Relaxation to linear systems lies in finding the optimal relaxation factor. To start with, one can define the sensible range of α as

$$1 < \alpha < 2$$

where the lower limit is simply the relaxation factor used in the normal relaxation method. Using some $\alpha \leq 1$, also known as *Successive Under-Relaxation*, would decrease the impact of a points residue on its next approximated

value and thus require more iterations to converge to a solution with the same accuracy. As *Figure 43* shows, applying Successive Over-relaxation with a relaxation factor $\alpha \geq 2$ will never terminate, no matter the boundary conditions imposed on the system. Adding twice the difference between a point and its surroundings neighbours does not decrease the absolute residue but rather just flips the sign (*cp. Ch. 5.2*). Thus, the value of all interior points would never converge to the solution of the system, and the residue would never fall below the maximum error ϵ.

5.2 Finding the optimal relaxation factor α

Different boundary conditions predetermine the optimal relaxation factor with which the number of iterations until convergence are minimized. To establish criteria for determining the optimal relaxation factor α in the specified range, it thus makes sense to analyse how different isolated changes to the boundary conditions imposed on a system relate to changes to its optimal α.

5.2.1 Relating BC scale to α

As one scales the boundary conditions of a system by a constant factor, the optimal relaxation factor used to solve that system increases slightly from $\alpha = 1.92$ for boundary conditions $y(x) = 0.25x$ to $\alpha = 1.94$ for boundary conditions $y(x) = 32x$ (cp. *Figure 50)*. Hence, simply scaling a system does not effect its optimal relaxation factor much.

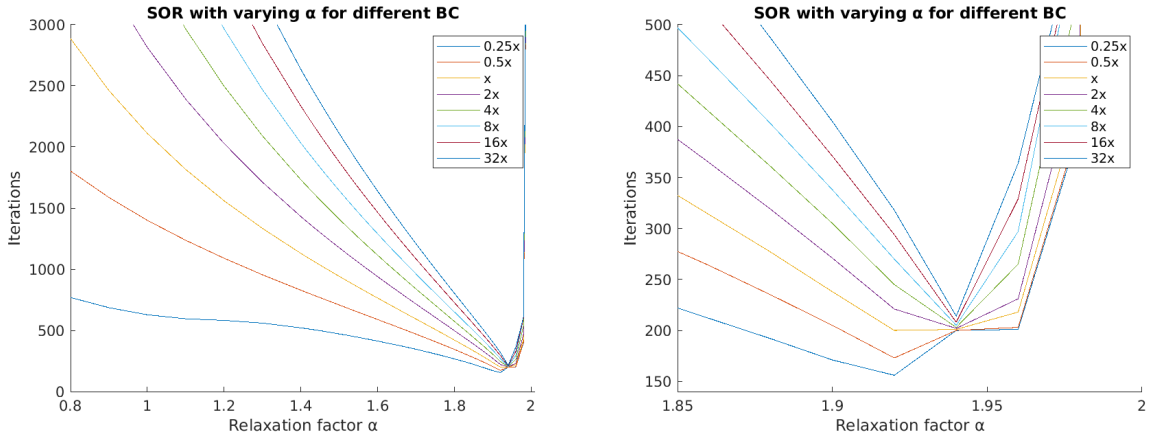


Figure 43: SOR with varying relaxation factors α for sinusoidal boundary conditions of different frequency (Zoomed in on the right).

5.2.2 Relating BC complexity to α

Next, the amount of iterations needed to solve a system that has simple polynomial boundary conditions of varying order using different α-values have been recorded (see *Figure 44*). The optimal relaxation factor to solve a system with higher order polynomial boundary conditions $y(x) = x^7$ is lower than the one for a system with simple boundary conditions $y(x) = x$; α being 1.9 and 1.92 respectively.

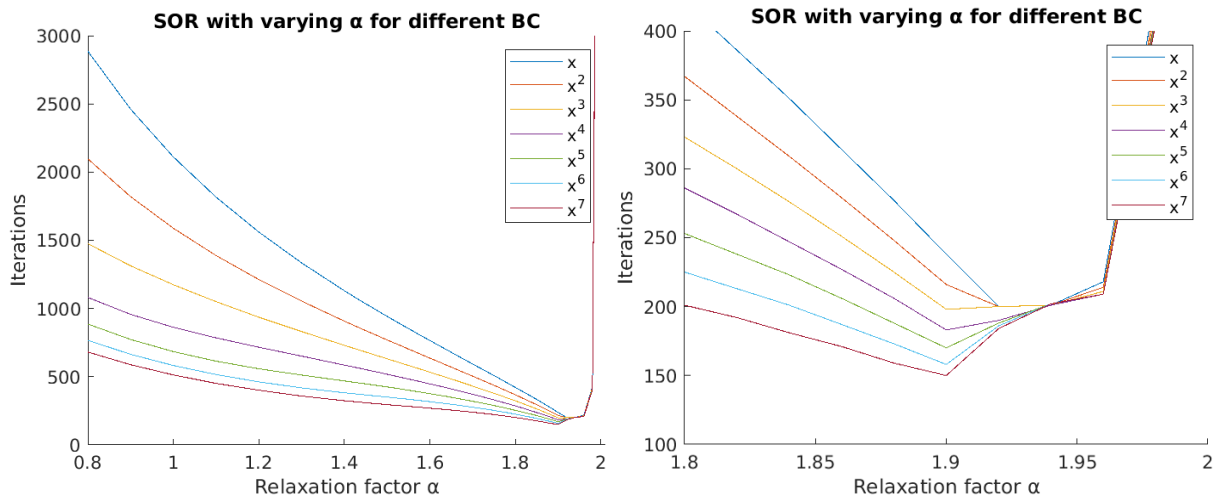


Figure 44: SOR with varying relaxation factors α for polynomial boundary conditions of different order (Zoomed in on the right).

While scaling the boundary conditions imposed on a system has only little effect on its optimal α, the complexity of the boundary conditions seem to be more significant. *Figure 45* compares systems with sinusoidal boundary conditions of varying frequency and the amount of iterations needed to solve them for different α-values. The system with boundary conditions $y(x) = sin(20\pi x)$ needs approximately 100 iterations to be solved using a regular relaxation factor $\alpha = 1$ while the system with boundary conditions $y(x) = sin(\pi x)$ requires close to 3000 iterations (cp. *Figure 45*. The zoomed in *Figure 45* illustrate that the first system is solved with a much lower relaxation factor of 1.4 compared to 1.94).

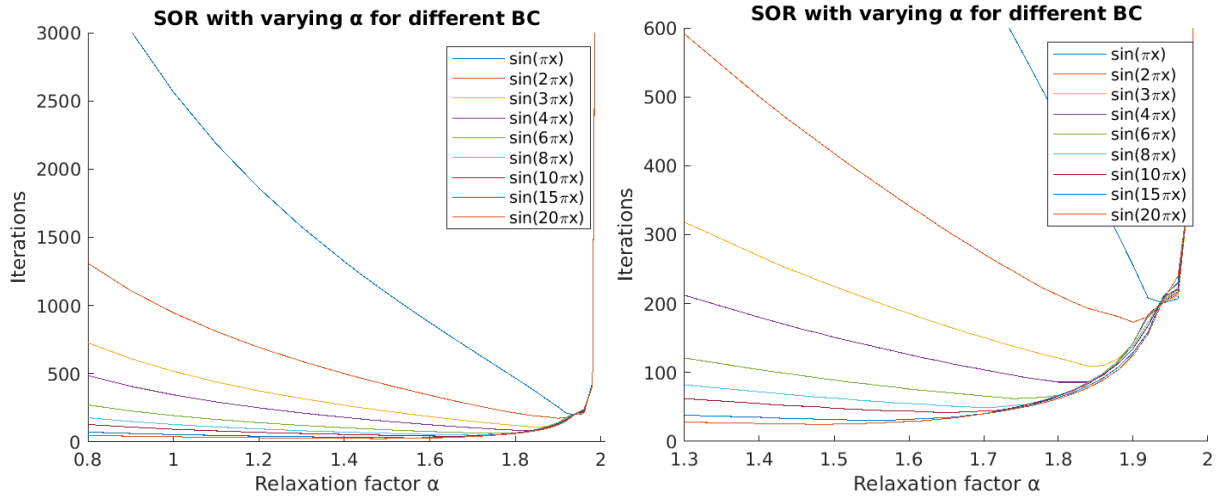


Figure 45: SOR with varying relaxation factors α for sinusoidal boundary conditions of different frequency (Zoomed in on the right).

5.2.3 Conclusions on optimal α

Generally, a system which can be solved to a certain accuracy with fewer iterations and using the normal relaxation method has a lower optimal relaxation factor. This holds for all tested cases, as illustrated by *Figures 43 to 45*.

Furthermore, while the optimal relaxation factor can differ a lot given different boundary conditions, Successive Over-Relaxation seems to work consistently well for a relaxation factor of $\alpha \approx 1.94$, solving any system to a certain accuracy after a similar amount of iterations that is merely depends on the resolution h and the maximum error ϵ rather than the boundary conditions imposed on the system. Changing both h and ϵ itself does not seem to impact the optimal relaxation factor at all.

All in all, if one has to choose a relaxation factor to solve the Laplace equation of some system without any more information about the exact boundary conditions, choosing $\alpha = 1.94$ seems to be a appropriate starting point. Given more information about the imposed boundary conditions, one can generally should choose a lower relaxation factor given more complex boundary conditions.

Appendix

RK2.m

```
function[x, y] = RK2(func, x0, y0, h, a, xf)

    b = 1 - a; % determining the constants the determines the RK method. all based
        on a.
    p = 0.5/b; % determining the constants the determines the RK method. all based
        on a.
    q = 0.5/b; % determining the constants the determines the RK method. all based
        on a.

    N = ceil((xf-x0)/h); %number of steps

    x = zeros(1, N); %create array populated with zeros. 1xN
    y = zeros(1, N); %create array populated with zeros. 1xN

    x(1) = x0; %Initialize first value to t=0
    y(1) = y0; %Initialize first value to t=0

    for i = 1:N-1 %iteration fun

        k1 = func(x(i), y(i)); %determine k1
        k2 = func( x(i) + p*h, y(i) + q*k1*h ); %determine k2

        x(i+1) = x(i) + h; %next x is obviously previous x + the step distance (h).
        y(i+1) = y(i) + h * (a*k1 + b*k2); %RK definition.

    end %end loop
end %end function
```

RK2_script.m

```
t_i = 0; % set initial value of t_0
q_i = 500e-9; % set q_initial condition q at t_0
h = 0.00001; % set t step-size
t_f = 0.01; % stop here
R = 1000; % resistance
C = 100e-9; % capacitance


%***************Heun's Method***************
a = 0.5;
%*******************************************

%**************Midpoint Method**************
% a = 0.0;
%*******************************************

%****************Random RK2*****************
% a = 0.3;
%*******************************************


%*****************Step_Signal 2.5V*****************
% func = @(t, q) 2.5/R*heaviside(t) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b')
% figure; % use this if you want the exact solution and the numerical on different figures
% %hold on; % use this if you want the exact solution and the numerical on the same figure
%
% qexact = 2.5 * C * ( 1 + exp(-tout/(R*C)) ); %calculate exact solution
% plot(tout, qexact, 'r')
%*************************************************


% %*****************Impulsive_Signal 2.5V*****************
% tau = 100;
% func = @(t, q) 2.5/R*exp(-t^2/tau) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b')
%
% There cannot be an exact solution for this Vin
% %*******************************************************


% %*****************Decay_Signal 2.5V*****************
% tau = 100;
% func = @(t, q) 2.5/R*exp(-t/tau) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b') %plot now values of x,y
%
% figure; % use this if you want the exact solution and the numerical on different
```

```
    figures
%%%hold on; % use this if you want the exact solution and the numerical on the
    same figure
%
% qexact = 2.5*tau*C/(tau - R*C)*(exp(-tout/tau) - exp(-tout/(R*C))) + q_i*exp(-
    tout/(R*C)); %calculate exact solution
% plot(tout, qexact, 'r')
% %************************************************


% %*****************Sine_Wave_Signal*****************
% period = 10e-6;
% period = 100e-6;
% period = 500e-6;
% period = 1000e-6;
% func = @(t, q) 5/R*sin(2*pi/period*t) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b') %plot now values of x,y
%
% figure; % use this if you want the exact solution and the numerical on different
    figures
%%hold on; % use this if you want the exact solution and the numerical on the same
    figure
%
% qexact = 2.5*C*period/(period^2 + (2*pi*R*C)^2)*(period*sin(2*pi/period*tout) -
    2*pi*C*R*cos(2*pi/period*tout)) + (q_i + (2.5*C^2*period*2*pi*R)/(period^2 + (2*
    pi*C*R)^2))*exp(-tout/(R*C)); %calculate exact solution
% plot(tout, qexact, 'r')
% %************************************************


% %*****************Squre_Wave_Signal*****************
% period = 10e-6;
% period = 100e-6;
% period = 500e-6;
% period = 1000e-6;
% func = @(t, q) 5/R*square(2*pi/period*t) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b')
%
% There cannot be an exact solution for this Vin
% %************************************************


% %*****************Sawtooth_Wave_Signal*****************
% period = 10e-6;
% period = 100e-6;
% period = 500e-6;
% period = 1000e-6;
% func = @(t, q) 5/R*sawtooth(2*pi/period*t) - 1/(R*C)*q;
%
% [tout, qout] = RK2(func, t_i, q_i, h, a, t_f);
% plot(tout, qout, 'b')
%
```

```
103 % There cannot be an exact solution for this Vin
104 % %*************************************************
```

error_script.m

```
clear;
R = 1e3; %resistance.
C = 100 * 1e-9; %capacitance.
qc0 = 500 * 1e-9; %charge at t0.
t0 = 0; %value of time when time is zero (obviously).
h = 0.00001; %step size.
tf = 0.001; %interval [0, 0.01].


%***************Heun's Method***************
a = 0.5;
%*******************************************

%**************Midpoint Method**************
% a = 0.0;
%*******************************************

%****************Random RK2*****************
% a = 0.3;
%*******************************************


%****************Sinusoidal Input Signal*****************
V = 5.0; %amplitude (voltage).
T = 100 * 1e-6; %period of T (1/T = frequency). 2pi*frequency = rotationshstighet (angular frequency?).
freq = 1/T; % frequency
w = 2 * pi * freq; %angular frequency
Vi = @(t) V * cos(w*t); %input cosine wave with given values.
%*********************************************************

%****************Decay Input Signal*****************
% V = 2.5;
% tau = 100 * 1e-6;
% Vi = @(t) V * exp(-t/tau);
%***************************************************

qcFunc = @(t, qc) ( Vi(t)/R ) - ( 1/(R*C) ) * qc; %we get the d/dt of qc (derivative).

[tout, qout] = RK2(qcFunc, t0, qc0, h, a, tf); %Numerical method

%****************Exact Solutions of the 2 Input Signal*****************
qcExact = ( exp(-tout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V * exp(tout/(R*C)) .* sin(w*tout) + 2000000 * C * V * exp(tout/(R*C)) .* cos(w*tout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
% qcExact = V * tout .* exp(-tout/(R*C)) / R + 2 * V * C * exp(-tout/(R*C));
%*************************************************************************

error = qcExact - qout; % get the error between the numerical and the exact one.

plot(tout, qout, 'b'); %plot the numerical graph and make sure it comes up and 'holds' so we can compare.
hold on;
plot(tout, qcExact, 'r'); %now plot the exact graph
```

```
hold on;
plot(tout, error, 'g'); %now plot the error
ylabel("qc(t)");
xlabel("t");
title("Heun's Method: Error Analysis");
% title("Midpoint Method: Error Analysis");
% title("Random Method: Error Analysis");
legend("Numerical Solution", "Exact Solution", "Approximation Error");
figure;

plot(tout, error, 'g'); %now plot the error
ylabel("Error");
xlabel("t");
title("Heun's Method: Error Analysis");
% title("Midpoint Method: Error Analysis");
% title("Random Method: Error Analysis");
figure;



%%%%    These first three for-loops are for doing the log-log plots
%%%%    of the errors vs step-size. The last three for-loop are doing
%%%%    the semi-log plot where only the step-size is in logarithmic scale.

%%%%     Heun's method Error Loglog
 a = 0.5;

for i = 16:25

    step = 2^(-i); %Trying different step sizes (h).
    [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);

    %*****************Exact Solutions of the 2 Input Signal******************
    qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
        * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
        cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
%     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
    %**********************************************************************

    error = max(abs(qcExact - eqout));
    p1 = plot(log(step), log(error), 'b*'); % doesn't seem to make a big difference
        doing log-log vs plot log.
    hold on; %10 iterations of plotting

end
hold on;

%%%     Midpoint method Error Loglog
a = 0.0;

for i = 16:25

    step = 2^(-i); %Trying different step sizes (h).
    [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);

    %*****************Exact Solutions of the 2 Input Signal******************
```

```
104     qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
            * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
            cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
105 %     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
106     %*********************************************************************
107
108     error = max(abs(qcExact - eqout));
109     p2 = plot(log(step), log(error), 'r*'); % doesn't seem to make a big difference
            doing log-log vs plot log.
110     hold on; %10 iterations of plotting
111
112 end
113 hold on;
114
115 %%%     Random method Error Loglog
116 a = 0.3;
117
118 for i = 16:25
119
120     step = 2^(-i); %Trying different step sizes (h).
121     [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);
122
123     %*****************Exact Solutions of the 2 Input Signal*****************
124     qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
            * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
            cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
125 %     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
126     %*********************************************************************
127
128     error = max(abs(qcExact - eqout));
129     p3 = plot(log(step), log(error), 'g*'); % doesn't seem to make a big difference
            doing log-log vs plot log.
130     hold on; %10 iterations of plotting
131
132 end
133
134 ylabel("Log (Error)"); % add some labels.
135 xlabel("log (Step-Size)");
136 legend([p1, p2, p3], "Heun's method", "Midpoint method", "Random method", 'Location
        ','northwest');
137 title("Log (Error) against Log (Step-Size)");
138 figure;
139
140
141
142 %%% All three methods for the error vs step-size (NOT LOGLOG! but semilog)
143 %%%     Heun's method Error SemiLog
144 a = 0.5;
145 for i = 16:25
146
147     step = 2^(-i); %Trying different step sizes (h).
148     [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);
149
150     %*****************Exact Solutions of the 2 Input Signal*****************
151     qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
            * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
```

```
        cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
%     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
    %*****************************************************************

    error = max(abs(qcExact - eqout));
    p4 = plot(log(step), error, 'b*');
    hold on;
end
hold on;

%%%     Midpoint method Error SemiLog
a = 0.0;
for i = 16:25

    step = 2^(-i); %Trying different step sizes (h).
    [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);

    %*****************Exact Solutions of the 2 Input Signal*****************
    qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
        * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
        cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
%     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
    %*****************************************************************

    error = max(abs(qcExact - eqout));
    p5 = plot(log(step), error, 'r*');
    hold on;
end
hold on;

%%%     Random method Error SemiLog
a = 0.3;
for i = 16:25

    step = 2^(-i); %Trying different step sizes (h).
    [etout, eqout] = RK2(qcFunc, t0, qc0, step, a, tf);

    %*****************Exact Solutions of the 2 Input Signal*****************
    qcExact = ( exp(-etout/(R*C)) .* ( C^2 * w^2 * R^2 + 2000000 * C^2 * w * R * V
        * exp(etout/(R*C)) .* sin(w*etout) + 2000000 * C * V * exp(etout/(R*C)) .*
        cos(w*etout) - 2000000 * C * V + 1 ) ) ./ (2000000 * (C^2 * w^2 * R^2 + 1));
%     qcExact = V * etout .* exp(-etout/(R*C)) / R + 2 * V * C * exp(-etout/(R*C));
    %*****************************************************************

    error = max(abs(qcExact - eqout));
    p6 = plot(log(step), error, 'g*');
    hold on;
end

ylabel("Error");
xlabel("Step-Size");
legend([p4, p5, p6], "Heun's method", "Midpoint method", "Random method", 'Location
    ','northwest');
title("Error against Log (Step-Size)");
```

RK4.m

```
function [Vout, t, qc, qc_grad] = RK4(Vout, F2, F1, h, t, qc, qc_grad, N)
    for i=1:N
        k1 = F1(t(i), qc(i), qc_grad(i));
        m1 = F2(t(i), qc(i), qc_grad(i));
        k2 = F1(t(i) + 0.5*h, qc(i) + (0.5*h*k1), qc_grad(i)+(0.5*h*m1));
        m2 = F2(t(i) + 0.5*h, qc(i) + (0.5*h*k1), qc_grad(i)+(0.5*h*m1));
        k3 = F1(t(i) + 0.5*h, qc(i) + (0.5*h*k2), qc_grad(i)+(0.5*h*m2));
        m3 = F2(t(i) + 0.5*h, qc(i) + (0.5*h*k2), qc_grad(i)+(0.5*h*m2));
        k4 = F1(t(i) + h, qc(i) + (k3*h), qc_grad(i) + (m3*h));
        m4 = F2(t(i) + h, qc(i) + (k3*h), qc_grad(i) + (m3*h));
        qc(i+1) = qc(i) + ((k1+k4)/6 + (k2+k3)/3)*h;
        qc_grad(i+1) = qc_grad(i) + ((m1+m4)/6 + (m2+m3)/3)*h;
        t(i+1) = t(i) + h;
        Vout(i+1) = qc_grad(i+1)*250;
    end
end
```

RLC_Script.m

```
%System properties
h = 2^(-15);
tf = 0.05;
t0 = 0;
N = tf/h;
qc0 = 500*10^(-9);

%Setting up Variables
qc(1) = qc0;
qc_grad(1) = 0;
Vout(1) = 0;
t(1) = t0;

F1 = @y;
F2 = @z;

%Call to RK4
[Vout, t, qc, qc_grad] = RK4(Vout, F1, F2, h, t, qc, qc_grad, N);

%obtaining plot
plot(t, Vout);
xlabel('Time (s)');
ylabel('Potential Differencev (V)');
title('Sine wave with amplitude = 5V')
hold on;


%System function.
function [yout] = z(t, qc, qc_grad)
    yout = qc_grad;
end

%change number after Vin for different imput signals shown below
function [yout] = y(t, qc, qc_grad)
    R = 10000;
    L = 600*10^(-3);
    C = 3.5*10^(-6);
    yout = (Vin4(t) - (qc/C) - (R*qc_grad))/L;
end

% (1) heaviside Vin
function [volt_out] = Vin1(time)
    volt_out = 5*heaviside(time);
end


% (2) impulsive signal
function [volt_out] = Vin2(time)
    volt_out = 5*exp(-(time^2)/(3*10^(-6)));
end

% (3) square wave
function [volt_out] = Vin3(time)
    freq = 500;
    volt_out = 5*square(2*pi*freq*time);
```

```
56 end
57
58 % (4) sine wave
59 function [volt_out] = Vin4(time)
60     freq = 500;
61     volt_out = 5*sin(2*pi*freq*time);
62 end
```

relaxation.m

```
close all ; clc

% d2u/dx2 + d2u/dy2 = g(x,y)      Poisson Equation
g = @(x,y) 0;

% Initialize grid (all zero)
h = 0.01;n = 1; m = 1;
x = 0:h:n;
y = 0:h:m;
U = zeros(length(x),length(y));

% Set boundary conditions
Ui_min = @(x) x;
Ui_max = @(x) x;
Umin_j = @(y) y;
Umax_j = @(y) y;

for i = 1 : length(x)
    U(i ,1) = Ui_min(x(i));
    U(i,length(y)) = Ui_max(x(i));
end
for j = 1 : length(y)
    U(1, j) = Umin_j(y(j));
    U(length(x), j) = Umax_j(y(j));
end

% Define termination condition    (maximum error)
epsilon = 1e-4;

iterations = -1; % counter
precise = false;

% Apply relexation method on all interior points until the residue r <
while ~precise
    precise = true;

    for i = 2 : length(x)-1
        for j = 2 : length(y)-1

            % Calculate residue r
            r = ( U(i+1,j) + U(i-1,j) + U(i,j+1) + U(i,j-1) - 4*U(i,j)  - (h^2)*g(i
                ,j) ) / (4);
            U(i,j) = U(i,j) + r;

            if abs(r) >= epsilon
                precise = false;
            end

        end
    end

    iterations = iterations+1;
end

disp("Done after " + iterations + " iterations with " + (length(x)-2)^2 + "
```

```
    calculations each.");
55
56 % Plotting
57 [X,Y] = meshgrid(x,y);
58 figure; surf(X,Y,U); % 3D
59 %figure; meshc(X,Y,U) %mesh: contour + surf
60 xlabel('x'); ylabel('y'); zlabel('U(x,y)');
61 title("Resolution h = 0.01");
62 view(-70,20); saveas(gcf,'h001.png');
```

SOR.m

```
% Successive over-relaxation (used by testbench.h)

function [iterations] = SOR(a,h,n,m,Ui_min,Ui_max,Umin_j,Umax_j,g)
% Inputs:
% SOR( Relaxation factor   , grid square size h, 2 grid dimensions
% 4 boundary conditions, 0 )

% Initialize grid (all zero)
x = 0:h:n;
y = 0:h:m;
U = zeros(n+1,m+1);

% Set boundary conditions
for i = 1 : length(x)
    U(i ,1) = Ui_min(x(i));
    U(i,length(y)) = Ui_max(x(i));
end
for j = 1 : length(y)
    U(1, j) = Umin_j(y(j));
    U(length(x), j) = Umax_j(y(j));
end

% Define termination condition    (maximum error)
epsilon = 1e-4;
% Define timeout (maximum iterations)
timeout = 10000;

iterations = -1; % counter
precise = false;

% Apply relexation method on all interior points until the residue r <
while ~precise
    precise = true;

    for i = 2 : length(x)-1
        for j = 2 : length(y)-1

            % Calculate residue r
            r = ( U(i+1,j) + U(i-1,j) + U(i,j+1) + U(i,j-1) - 4*U(i,j)  - (h^2)*g(i
                ,j) ) / (4);
            U(i,j) = U(i,j) + a*r;

            if abs(r) >= epsilon
                precise = false;
            end

        end
    end

    iterations = iterations+1;
    if(iterations >= timeout)
        disp("Time out.");
        break;
    end
end
```

```
55
56 disp("Done after " + iterations + " iterations (    = " + a + ")");
57
58 % Plotting
59 [X,Y] = meshgrid(x,y);
60 figure; surf(X,Y,U); % 3D
61 % figure; contour(X,Y,U); % from above
62 % figure; meshc(X,Y,U) %mesh: contour + surf
63 xlabel('x'); ylabel('y'); zlabel('U(x,y)'); title("    = " + a);
```

testbench.m

```
% SOR testing

% alpha < 1 : under relaxation
% alpha == 1: normal relaxation
% alpha > 1: over relaxation

%close all ; clc
figure;

alpha = horzcat((0.8:0.1:1.4),(1.42:0.02:2));
outputs = zeros(1, length(alpha));
timeout = 10000;

% d2u/dx2 + d2u/dy2 = g(x,y)
g = @(x,y) 0; % !! Poisson eqn other than Laplace not working yet !!

% Define grid
h = 0.01; n = 1; m = 1;

% Set boundary conditions: 1,0; 10,0; x; 0.5*(2*x-1)^2; 0.5*(2*x-1)^4; sin(2*pi*x);
Ui_min = @(x) x;
Ui_max = @(x) e^x;
Umin_j = @(y) e^y;
Umax_j = @(y) e^y;

run = true;

for q = 1:(length(alpha)-1)
    if run

        disp("Testing alpha " + alpha(q));
        output = SOR(alpha(q),h,n,m,Ui_min,Ui_max,Umin_j,Umax_j,g);

        if(output > timeout)
            plotlim = round(max(outputs/100))*100;
            disp("gude morge");
            run = false;
        end
        outputs(q) = output;
    end
end

%simulating alpha = 2
outputs(length(alpha)) = 10000;

hold on;
plot(alpha, outputs); ylim([0,3000]); xlim([0.8,2.01]);
xlabel("Relaxation factor    ");
ylabel("Iterations");
title("SOR with varying     for different BC");


function [yy] = discon(xx)
    if(xx < 0.25)
        yy = 0;
```

```
56      elseif(xx < 0.5)
57          yy = 1;
58      elseif(xx < 0.75)
59          yy = 0;
60      else
61          yy = 1;
62      end
63  end
```