LCR Circuits: Jessica Murphy

27/02/2023

Homework 17

This python programme involves an Initial Value Problem where it uses a technique to model the oscillatory behavior of a LCR circuit obeying Kirchhoff's loop rule - it is called the **Euler-Cromer** technique.

Kirchhoff's loop rule, applied to a series LCR circuit, gives the following differential equation.

$$L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{1}{C}Q = V_s(t)$$

This can be re-written as

$$\frac{d^2Q}{dt^2} = -\frac{R}{L}\frac{dQ}{dt} - \frac{1}{LC}Q + \frac{1}{L}V_s(t)$$

Using the following component values: Inductance L = 2.0 mH, Capacitance C = 3.0 μF and Resistance R = 5.0 Ω, these were converted to SI units, namely Henry (H), Farads (F), Ohms (Ω). The SI units for time (seconds) and charge (Coulomb) were also converted.

A Voltage source consisting of a DC power supply is assumed which initially is outputting a constant 10 V, but is turned off at t = 0 s. This will have charged the capacitor up to a charge of Q = CV = 30 μC at t = 0 s. Vs(t>0) = 0. When t > 0, V = 0 as the contribution from V when t = 0 is so small, it is neglible throughout.

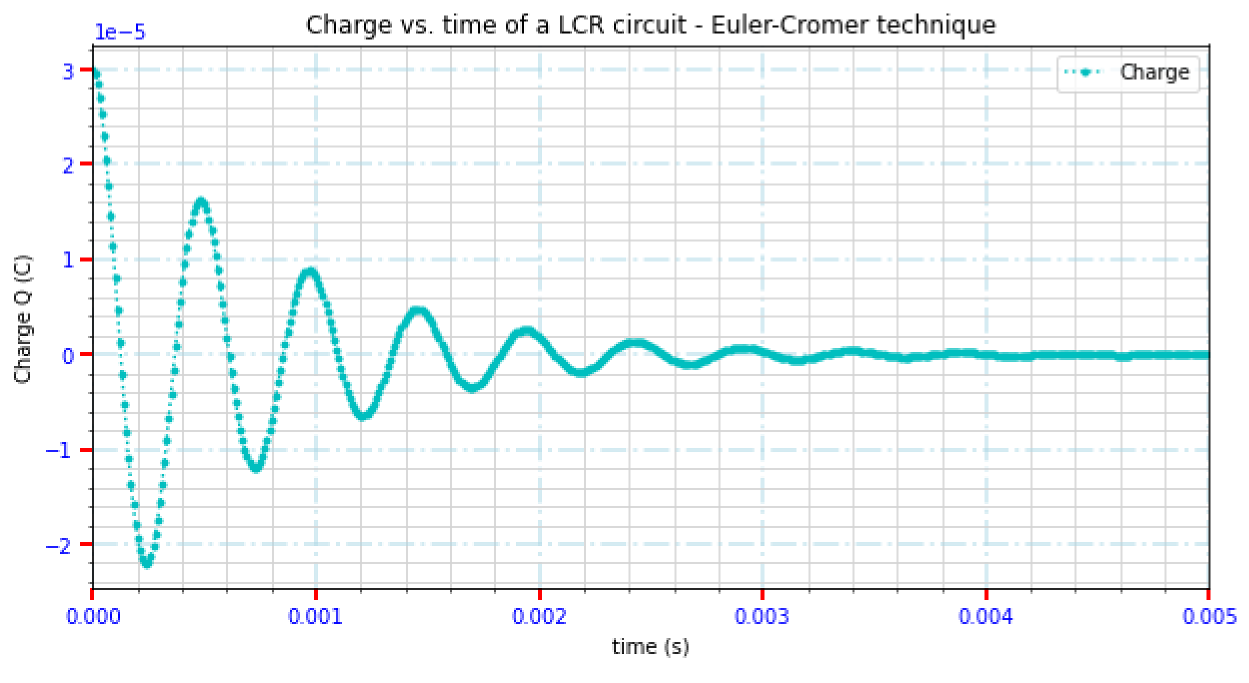
The initial values od the variables are stated, a time step is fixed, arrays to hold the changing variables of Q, I, t are created, and a for loop is determined to iterate along the t-axis to model the inner-workings of the sytem. $\frac{dQ}{dt}$ is equal to the current

A graph is then depicted of the LCR circuits charge over a time of 5 ms. It is displayed as oscillatory in nature with its amplitude gradually declining.

1. Import the python libraries NumPy and Matplotlib.
2. Set the initial condiitons.
3. Set the number of steps required.
4. Set up NumPy arrays to hold the charge (***Q***), current(***I***) and time (***t***) values and initialise.
5. Use a for loop to step along the t-axis. Use the Euler-cromer technique to obtain a generalised formula to simulate more accurate physical conditions by using the current at the end of the time interval, when estimating the charge.
6. Plot the numerical solution on a graph with charge on the y-axis and time on the x-axis

In [1]:

```
# Model of the behavior of a LCR cicuit using the Euler-Cromer technique
# The 2nd order ODE is: d2Q/dt2 = (-R/L)dQ/dt - (1/LC)Q + (1/L)V(t)

import numpy as np # import python libraries as needed
import matplotlib.pyplot as plt

L = 0.002 # the amount of henry, the SI unit of inductance (1 H = 1 kg m2 s-2 A-2)
C = 3e-6 # the initial amount of capacitance in the capacitor in farads, the SI unit of conductance
R = 5.0 # the amount of resistance in the system in ohms, the SI unit of resistance
Q_0 = 3e-5 # initial charge on the capacitor in coulombs, the SI unit of charge

t_max = 0.005 # iterate up to a maximum of t_max in seconds
delta_t = 1e-5 # set the time step (s)
N = int(t_max/delta_t) # calculate the number of time jumps

# set up numpy arrays to hold the charge Q, current I, and time t values and initialise
# need one more element in each array than the number of jumps, to include the zeroth element
Q = np.zeros(N + 1)
I = np.zeros(N + 1)
t = np.zeros(N + 1)

# initialise the zeroth values, explicitly stating variables I and t start at 0
Q[0] = Q_0
I[0] = 0
t[0] = 0

# use a for loop to step along the t-axis N times and apply the Euler-Cromer technique
# use brackets as appropriate
for i in range(N):
    D = -(Q[i]/ (C*L)) - ((R*I[i])/ L) # second order ODE with damping term R
    I[i+1] = I[i] + D * delta_t # estimate I at the end of the interval
    Q[i+1] = Q[i] + I[i+1] * delta_t # Euler-Cromer technique applied to the index of I, estimates Q at the end of the time interval using I at the end of the time interval
    t[i+1] = t[i] + delta_t # calculate t at the end of the interval


plt.figure(figsize= (10,5)) # increase the size of the figure

# plot the numerical solution as points on a graph of charge as a function of time - model oscillatory motion
plt.plot(t, Q, 'c.:', label='Charge')
plt.xlim(0, t_max) # maximise area of graph
plt.title('Charge vs. time of a LCR circuit - Euler-Cromer technique')
plt.xlabel('time (s)')
plt.ylabel('Charge Q (C)')
plt.tick_params(axis='both', direction='out', length=6, width=2, labelcolor='b', colors='r', grid_color='gray', grid_alpha=0.5) # make axes ticks red and writing blue
plt.grid(visible=True, color='lightblue',linestyle='-.', linewidth=2) # major grid lines
plt.minorticks_on()
plt.grid(visible=True, which='minor', color='lightgrey', alpha=0.8, ls='-', lw=1) # minor grid lines
plt.legend()
plt.show()
```



In the above graph we see a lightly damped system, shown by a falling exponential envelope. The function can be described as being bounded, or modulated by this exponential envelope.