Linh Tang

CEE/MAE M20

UID: 205542275

Oct 31, 2020

HOMEWORK 4

1. LC circuit problem: explicit Euler

1.1 Introduction

In this program, the second order differential equation of LC circuit is solved by using explicit Euler method. After that, the program simulates the current i(t) and calculate the frequency of the system.

1.2 Model and methods

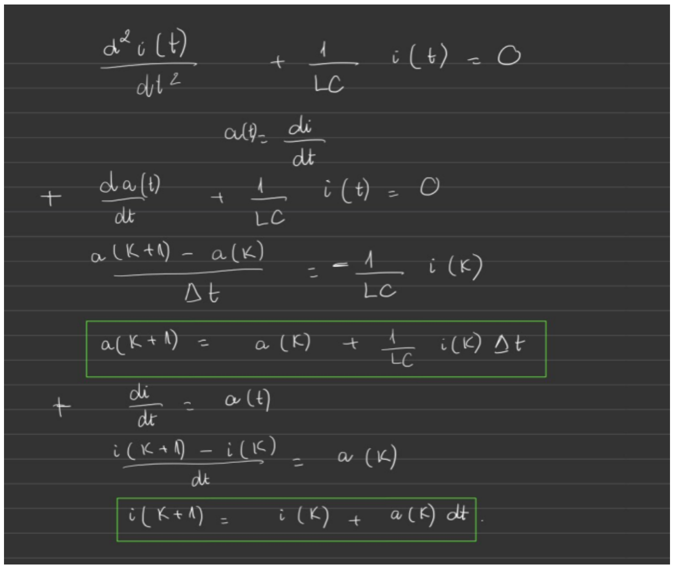
First, given constants and initial conditions are defined at the beginning of the script.

```
L = 1;
C = 0.1;
dt = 0.01;
ti = 0;
tf = 20;
```

Time array is set up by : `t = ti:dt:tf.` Numel () function is called to get number of elements in the time array; it is also total number of time steps that is used in for loop later.

```
N = numel(t);
```

From the governing equation for the LC circuit, di/dt is defined as a new state variable a; two equations for a and I are derived by using Explicit Euler method.



$$\frac{d^2 i(t)}{dt^2} + \frac{1}{LC} i(t) = 0$$

$$a(t) = \frac{di}{dt}$$

$$+ \quad \frac{da(t)}{dt} + \frac{1}{LC} i(t) = 0$$

$$\frac{a(K+1) - a(K)}{\Delta t} = -\frac{1}{LC} i(K)$$

$$\boxed{a(K+1) = a(K) + \frac{1}{LC} i(K) \Delta t}$$

$$+ \quad \frac{di}{dt} = a(t)$$

$$\frac{i(K+1) - i(K)}{dt} = a(K)$$

$$\boxed{i(K+1) = i(K) + a(K)\, dt}.$$

Before doing calculations, two arrays for a and i are set up to contain all the values of N time steps; using zeros() function to create an array with N rows and 1 column for each one.

```
i = zeros(N, 1);
a = zeros(N,1);
```

For loop is used to calculate i and a at the next time step k+1 from the current step k values. Since the equations return values at k+1, so k should go from 1 to N-1, not N.

```
for k = 1:N-1
      % the discretized governing equations
      a(k+1) = a(k) - (1/(L*C))*i(k)*dt;
      i(k+1) = i(k) + a(k)*dt;
end
```

Some functions are used to display the plot with title, axes, position, grid, …

```
figure(1)
p = plot(t, i, 'b');
xlabel ('time (s)');
ylabel ('Current (A)');
title ('LC Circuit: i(t) [Explicit Euler]');
xlim ([ti tf]);
ylim ([1.1*min(i) 1.1*max(i)]);
set(p, 'LineWidth', 2);
set(gcf, 'Position', [100 40 1000 600]);
set(gca, 'LineWidth', 2, 'FontSize', 15);
ax = gca;
ax.XGrid = 'off';
ax.YGrid = 'on';
```

According to the plot, datatip() function is used to show the data for two adjacent peaks based on approximate coordinates.

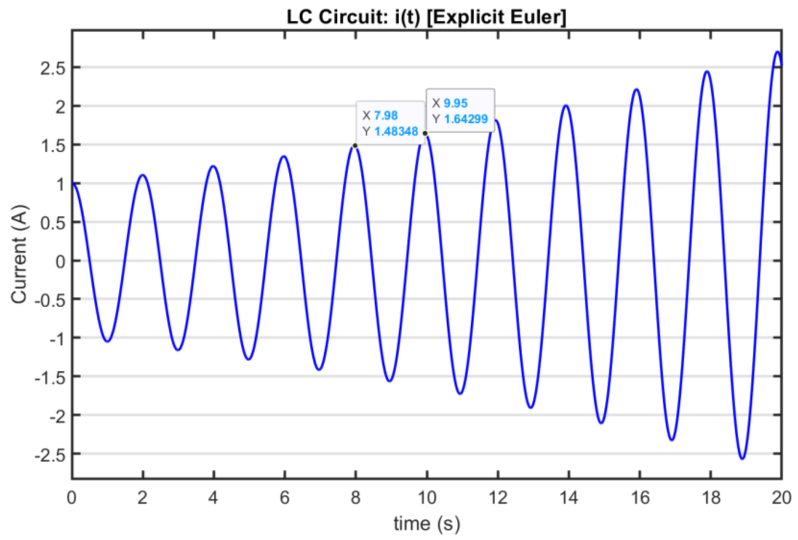
```
datatip(p, 8, 1.5);

datatip(p, 10, 1.7);
```

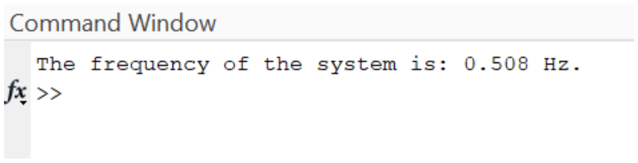
The frequency of the system is calculated based on the period, the time for one oscillation. The period is determined by calculating the difference of time at two adjacent peaks; it is also the difference of x values collected from two datatip functions.

f = 1/T = 1/(x2-x1)

```
freq = 1/abs(9.95-7.98); % f = 1/T

fprintf('The frequency of i(t) is: %.3f Hz.\n',freq);
```

1.3 Results




Command Window

```
The frequency of the system is: 0.508 Hz.
>>
```

1.4 Discussion

Based on the graph, we can see an artificial energy is introduce into the system; therefore, using Explicit Euler method does not give us an accurate simulation for i(t). The problem can be done with other Euler methods, such as implicit and semi implicit to see which one give us the most accurate plot. For the calculation of the frequency, I have not figured out how to extract only x values from datatip() function; by that way, I would not have to hardcode the x values in the calculation.

2. The RLC circuit problem: semi-implicit Euler

2.1 Introduction

The purpose of this script is to simulate the current i(t) and calculate the damping coefficient at three different resistor values in the RLC circuit. The semi- implicit Euler method is used to get the discretized governing equations when solving the second order differential equation.

2.2 Model and methods

Similar to the first problem, constants for R, L, C, time and step size are defined at the beginning of the script. The number of time steps is calculated in the same way as what we did in first problem.

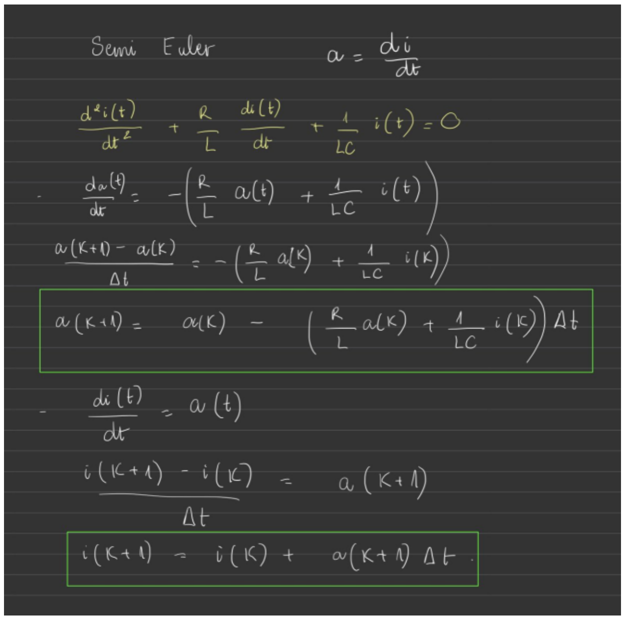
```
L = 1;
C = 0.01;
dt = 0.01;
ti = 0;
tf = 20;
t = ti:dt:tf; % time array
N = numel(t); % number of time steps
```

Since we have three systems corresponding to three different values for R, an array is set up to hold three values of R.

```
R1 = 0.2;
R2 = 2;
```

```
R3 = 20;

R = [R1, R2, R3]; % Creating array for resistance
```

Based on the Kirchhoff equation for the RLC system, semi- implicit Euler method is used to obtain the discretized governing equations when setting up a new state variable for di/dt.



Semi Euler $\quad a = \frac{di}{dt}$

$$\frac{d^2 i(t)}{dt^2} + \frac{R}{L}\frac{di(t)}{dt} + \frac{1}{LC} i(t) = 0$$

$$\frac{da(t)}{dt} = -\left(\frac{R}{L} a(t) + \frac{1}{LC} i(t)\right)$$

$$\frac{a(K+1) - a(K)}{\Delta t} = -\left(\frac{R}{L} a(K) + \frac{1}{LC} i(K)\right)$$

$$\boxed{a(K+1) = a(K) - \left(\frac{R}{L} a(K) + \frac{1}{LC} i(K)\right)\Delta t}$$

$$\frac{di(t)}{dt} = a(t)$$

$$\frac{i(K+1) - i(K)}{\Delta t} = a(K+1)$$

$$\boxed{i(K+1) = i(K) + a(K+1)\,\Delta t}$$

Two arrays for a and i are initialized to hold values calculated in N time steps.

```
i = zeros(N,1); % the current array
a = zeros(N,1); % a = di/dt array
i(1) = i_0;
a(1) = a_0;
```

There are two for loops being used in this problem; the inner for loop calculates values of a and i at the next time step through discretized governing equations.

```
for k = 1:N-1
  % The discretized governing equations
  a(k+1) = a(k) - ((R(n)/L)*a(k)+ (1/(L*C))*i(k))*dt;
  i(k+1) = i(k) + a(k+1)*dt;

end
```

The outer for loop is used to go through values in the R array, then doing the plot and damping coefficient for each value of R.

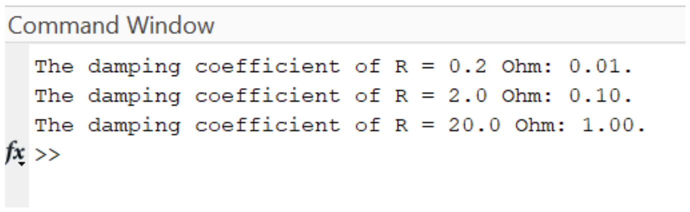
```
for n =1:length(R)
    % Calculate and plot for each R

    for
      …
    end

    % Plotting
    p = plot(t,i);
    hold on

    xlabel ('time (s)');
    ylabel ('Current (A)');
    title ('RLC Circuit: i(t) [Semi-implicit Euler]');
    xlim ([ti tf]);
    set(p, 'LineWidth', 2);
    set(gcf, 'Position', [100 40 1000 600]);
    set(gca, 'LineWidth', 2, 'FontSize', 15);
    ax = gca;
    ax.XGrid = 'off';
    ax.YGrid = 'on';

    % Calculating damping coeff

end
```

Based on the codes shown above, plotting functions are put in the outer for loop with "hold on"; this command asks Matlab to wait to get the next plot instead of closing the figure, so we can get all the plots in the same figure.

For the damping coefficient, an array with the same size with R array is initialized outside the for loop. At the end of the outer for loop, the damping coefficient for each R is calculated and updated to the array.

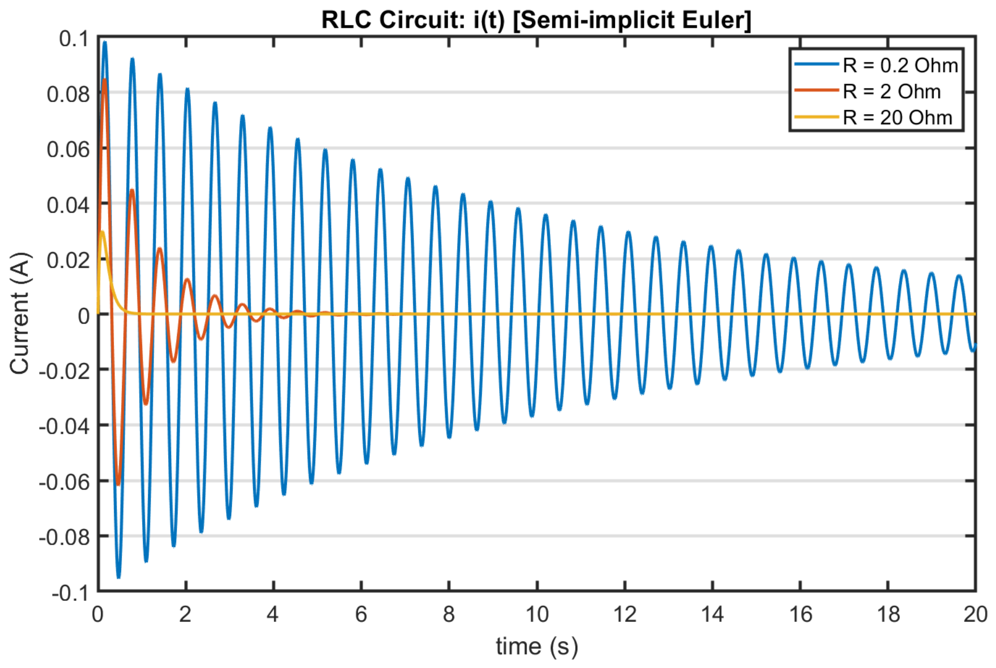
```
coeff = zeros(length(R),1); % array for damping coeff

for
…
% Calculating damping coeff
    coeff(n) = (R(n)/2)*sqrt(C/L);
    fprintf('The damping coefficient of R = %.1f Ohm:
%.2f.\n',R(n), coeff(n));

end
```

2.3 Results



Command Window

```
The damping coefficient of R = 0.2 Ohm: 0.01.
The damping coefficient of R = 2.0 Ohm: 0.10.
The damping coefficient of R = 20.0 Ohm: 1.00.
>>
```



2.4 Discussion

According to the plot, the systems lose energy over time, and there are damping effects in the RLC circuit. When R equal to 20 Ohm, the system is critically damped with its damping coefficient is 1. When R = 0.2 Ohm, the plot is an oscillation with a weak damping effect; the damping coefficient is 0.01. With R = 2 Ohm, the system is more damped with the damping coefficient is 0.1. Comparing to Explicit Euler method in the first problem, there is no artificial energy introduced in the system when using the semi- implicit Euler method. In this problem, a new variable state is set up for di/dt instead of writing the equation between di/dt and v(t); therefore, the program is only able to simulate i(t). If we want to simulate v(t), the discretized governing function will be rewritten to calculate for both i(t) and v(t).

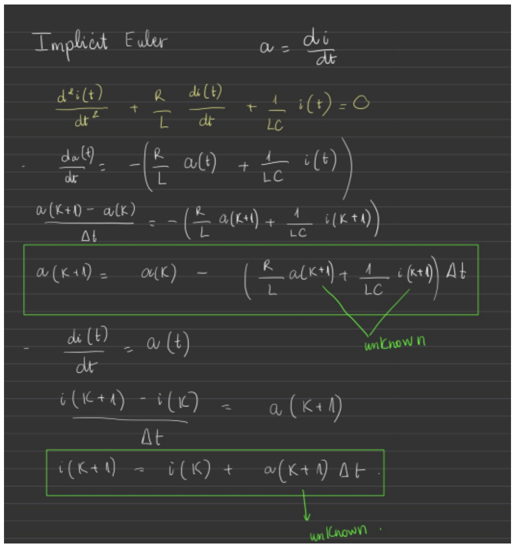
3. The RLC circuit problem: Implicit Euler

3.1 Introduction

The goal of this problem is to solve second order differential equation and simulate the current i(t) of the RLC circuit in the second problem but using Implicit Euler method instead of Semi - implicit Euler method.

3.2 Model and methods

In this problem, variables and initial conditions are set up exactly what we do in the second problem. The difference is how the discretized governing equations are solved in the inner for loop. For the implicit Euler method, the governing equation cannot be solved as what we do with semi-implicit and explicit Euler, since the right-hand side of equations has unknown variables.



Implicit Euler $\quad a = \frac{di}{dt}$

$$\frac{d^2 i(t)}{dt^2} + \frac{R}{L}\frac{di(t)}{dt} + \frac{1}{LC} i(t) = 0$$

$$\frac{da(t)}{dt} = -\left(\frac{R}{L} a(t) + \frac{1}{LC} i(t)\right)$$

$$\frac{a(K+1) - a(K)}{\Delta t} = -\left(\frac{R}{L} a(K+1) + \frac{1}{LC} i(K+1)\right)$$

$$\boxed{a(K+1) = a(K) - \left(\frac{R}{L} \underbrace{a(K+1) + \frac{1}{LC} i(K+1)}_{\text{unknown}}\right)\Delta t}$$

$$\frac{di(t)}{dt} = a(t)$$

$$\frac{i(K+1) - i(K)}{\Delta t} = a(K+1)$$

$$\boxed{i(K+1) = i(K) + \underbrace{a(K+1)}_{\text{unknown}} \Delta t}$$

In that case, the Newton method is used to solve these two equations:

$$f_1 = \frac{a_{k+1} - a_k}{\Delta t} + \frac{R}{L} a_{k+1} + \frac{1}{LC} i_{k+1} = 0,$$
$$f_2 = a_{k+1} - \frac{i_{k+1} - i_k}{\Delta t} = 0.$$

First, we set up a(k) and i(k) as an initial guess, then evaluate two functions with those values. To make it more convenient, guessing values and function values are stored into arrays.

```
x = [i(k); a(k)]; % initial guess for the x = t_k;
f1 = (x(2) - a(k))/dt + (R(n)/L)*x(2) + (1/(L*C))*x(1);
f2 = x(2) - (x(1) - i(k))/dt;

y = [f1;f2];
```

A while loop is used to find the solutions for the function; the boolean condition checks the error of two functions, the Euclidean norm of two functions.

$$\sqrt{(f_1)_{k+1}^2 + (f_2)_{k+1}^2}$$

After that, the Jacobian matrix is defined as following:

$$J = \begin{bmatrix} \frac{df_1}{di_{(k+1)}} & \frac{df_1}{da_{(k+1)}} \\ \frac{df_2}{di_{(k+1)}} & \frac{df_2}{da_{(k+1)}} \end{bmatrix} \Rightarrow J = \begin{bmatrix} \frac{1}{LC} & \frac{1}{dt} + \frac{R}{L} \\ -\frac{1}{dt} & 1 \end{bmatrix}$$

By using Jacobian matrix, a new guessing value for x is determined; two functions are evaluated at that new value. This process repeats until the program gets the approximate solutions.

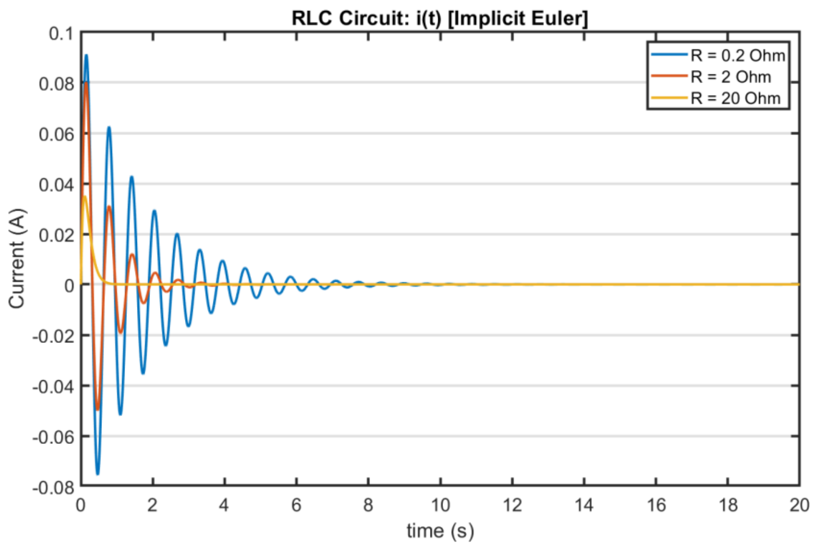
```
while(norm(y) > 1e-6)

    J = [1/(L*C), 1/dt + R(n)/L; -1/dt, 1];
    x = x - J\y;
    f1 = (x(2) - a(k))/dt + (R(n)/L)*x(2) + (1/(L*C))*x(1);
    f2 = x(2) - (x(1) - i(k))/dt;
    y = [f1; f2];
end
```

After getting solutions for that time step, these values are pasted to the arrays for a and I at the index k+1.

```
i(k+1) = x(1);
a(k+1) = x(2);
```

The outer for loop and the plot functions are the same with what we do in the second problem.

3.3 Results



3.4 Discussion

Comparing to the plots obtained by using semi- implicit, the damping of all three system are more extreme when using the Implicit Euler method. With R = 0.2 Ohm and the damping coefficient is 0.01, the plot is strongly damped which is different from what is shown in semi-implicit Euler method. Based on the difference of the plots for two methods, we can say that the semi-implicit method gives us more accurate results. The semi-implicit method is considered as the combination of implicit and explicit method to limit the weakness of each method. In the future script, the Runge Kutta method might be used to solve the second order differential equation of RLC circuit besides Euler methods.