Basic Circuit Discretization

Assignment 1a

Due: 2021/01/22

1 Introduction

In this first assignment, we are analyzing the transient behaviour of a series RL circuit as we close its main switch. Since this circuit and its constituent components can be modeled as an easy-to-solve first order differential equation, this allows us to simultaneously solve for its continuous time solution as well as apply some discretization techniques to enable solving the circuit in a step-by-step manner. Comparing the two allows us to analyze how well each discretization technique is able to approximate the true solution.

The following sections will discuss the derivation of the continuous time function as well as the derivation of the discretized functions. The trapezoidal and backward Euler approximations of a differential equation will be employed for this assignment. Following the derivations, we will plot different the approximations and compare their performance with respect to different time steps as well to each other. The continuous solutions will be overlayed to understand how well the approximations come to solving the circuit's solution.

Finally, a discussion of the results as well as improvements will be proposed. A full code listing of the code used in this project is provided as well for review.

2 Setup

The following sections go through the derivation of the different solutions used to produce the final plots for this assignment.

2.1 Homogeneous Plus Steady-State Solutions

In this section, we will derive the homogeneous plus steady-state solutions for $i(t)$ and $v_L(t)$. These solutions will plot the continuous time value for each of the quantities, and will be used as a baseline comparison to see how well the approximations match the ground truth. Using KVL, the circuit can be represented by the following equation:

$$10 - Ri(t) = L\frac{di(t)}{d(t)} \tag{1}$$

Solving this first order differential equation will provide the continuous time system solution for $i(t)$ and $v_L(t)$. First, we will derive the continuous time system solution for $i(t)$in terms of homogeneous plus steady-state equations:

$$\frac{10 - Ri(t)}{L} = \frac{di(t)}{dt} \tag{2}$$

$$\frac{dt}{L} = \frac{di(t)}{10 - Ri(t)} \tag{3}$$

$$\frac{1}{L}\int dt = \int \frac{1}{10 - R\,i(t)} di(t) \tag{4}$$

$$\frac{1}{L}t + K = -\frac{1}{R}\ln|10 - R\,i(t)| \tag{5}$$

$$e^{-\frac{R}{L} - RK} = 10 - R\,i(t) \tag{6}$$

$$i(t) = \frac{10}{R} - A\,e^{-\frac{R}{L}t} \qquad A = e^{-RK} \tag{7}$$

$$\boxed{i(t) = \frac{10}{R} - A\,e^{-\frac{R}{L}t}} \tag{8}$$

We know from our initial conditions that $i(0) = 0$; this allows us to solve for the steady-state solution by solving for the constant A.

$$0 = \frac{10}{R} - A\,e^{-\frac{R}{L}(0)} \tag{9}$$

$$A = \frac{10}{R} \tag{10}$$

This gives us our final, continuous time solution for $i(t)$ as follows:

$$i(t) = \frac{10}{R} - \frac{10}{R}\,e^{-\frac{R}{L}t} \qquad i(t) = 0 \tag{11}$$

$$\boxed{i(t) = \frac{10}{R} - \frac{10}{R}\,e^{-\frac{R}{L}t}} \tag{12}$$

Obtaining our continuous time $v_L(t)$ relies on the formula for the voltage across an inductor. Since we have obtained our $i(t)$, we can take its derivative with respect to time to obtain the equation's $\frac{di(t)}{dt}$ term.

$$\frac{di(t)}{dt} = A\,\frac{R}{L}\,e^{-\frac{R}{L}t} \tag{13}$$

This derivative can then be substituted into the equation for the voltage across our inductor:

$$v_L(t) = L\,\frac{di(t)}{dt} \tag{14}$$

$$\boxed{v_L(t) = A\,R\,e^{-\frac{R}{L}t}} \tag{15}$$

Knowing our initial conditions, we obtain the following steady-state equation for $v_L(t)$

$$\boxed{v_L(t) = 10\,e^{-\frac{R}{L}t}} \tag{16}$$

2.2 Isolating Components for Discretization

In order to apply the trapezoidal and backward Euler discretizations used by the step-by-step solution, we must isolate the integration of $i(t)dt$. This is the component that will be approximated by the following discretization techniques:

Trapezoidal:

$$\int_{t-\Delta t}^{t} i(t)dt \simeq \frac{i(t)+i(t-\Delta t)}{2}\Delta t \tag{17}$$

Backward Euler:

$$\int_{t-\Delta t}^{t} i(t)dt \simeq i(t)\Delta t \tag{18}$$

The following steps will manipulate our original KVL relationship into a differential equation that can be discretized, isolating the integration of $i(t)dt$.

$$10 - Ri(t) = L\frac{di(t)}{d(t)} \tag{19}$$

$$\frac{10}{L} - \frac{R}{L}i(t) = \frac{di(t)}{dt} \tag{20}$$

$$\frac{10}{L}dt - \frac{R}{L}i(t)dt = di(t) \tag{21}$$

$$\frac{10}{L}\int_{t-\Delta t}^{t} dt - \frac{R}{L}\int_{t-\Delta t}^{t} i(t)dt = i(t) - i(t-\Delta t) \tag{22}$$

$$\frac{10}{L}[t-(t-\Delta t)] - \frac{R}{L}\int_{t-\Delta t}^{t} i(t)dt = i(t) - i(t-\Delta t) \tag{23}$$

$$-\frac{R}{L}\int_{t-\Delta t}^{t} i(t)dt = i(t) - i(t-\Delta t) - \frac{10}{L}\Delta t \tag{24}$$

$$\boxed{\int_{t-\Delta t}^{t} i(t)dt = -\frac{L}{R}i(t) + \frac{L}{R}i(t-\Delta t) + \frac{10}{R}\Delta t} \tag{25}$$

With $\int_{t-\Delta t}^{t} i(t)dt$ isolated, we can now apply our discretization techniques to obtain our step-by-step solution for the current. Similarly, discretizing $v_L(t)$ can be performed by manipulating the fundamental equation for the voltage across an inductor:

$$v_L(t) = L\frac{di(t)}{dt} \tag{26}$$

$$\int_{t-\Delta t}^{t} v_L(t) = Ldi(t) \tag{27}$$

$$\boxed{\int_{t-\Delta t}^{t} v_L(t) = L[i(t) - i(t-\Delta t)]} \tag{28}$$

2.3 Trapezoidal Discretization

The following is the derivation of the step-by-step solution for $i(t)$ and $v_L(t)$ using the trapezoidal discretization technique.

Current $i(t)$:

$$\frac{i(t)+i(t-\Delta t)}{2}\Delta t \simeq -\frac{L}{R}i(t)+\frac{L}{R}i(t-\Delta t)+\frac{10}{R}\Delta t \tag{29}$$

$$i(t)\Delta t + i(t-\Delta t)\Delta t \simeq \frac{2L}{R}i(t-\Delta t)-\frac{2L}{R}i(t)+\frac{20}{R}\Delta t \tag{30}$$

$$i(t)\Delta t + \frac{2L}{R}i(t) \simeq \frac{2L}{R}i(t-\Delta t)-i(t-\Delta t)\Delta t+20\Delta t \tag{31}$$

$$\boxed{i(t) \simeq \frac{i(t-\Delta t)[2L-R\Delta t]+20\Delta t}{R\Delta t+2L}} \tag{32}$$

Voltage $v_L(t)$:

$$\frac{v_L(t)+v_L(t-\Delta t)}{2}\Delta t \simeq L[i(t)-i(t-\Delta t)] \tag{33}$$

$$\boxed{v_L(t) \simeq \frac{2L}{\Delta t}[i(t)-i(t-\Delta t)]-v_L(t-\Delta t)} \tag{34}$$

2.4 Backward Euler Discretization

The following is the derivation of the step-by-step solution for $i(t)$ and $v_L(t)$ using the backward Euler discretization technique.

Current $i(t)$:

$$i(t)\Delta t \simeq -\frac{L}{R}i(t)+\frac{L}{R}i(t-\Delta t)+\frac{10}{R}\Delta t \tag{35}$$

$$i(t)R\Delta t + Li(t) \simeq Li(t-\Delta t)+10\Delta t \tag{36}$$

$$i(t)[R\Delta t+L] \simeq Li(t-\Delta t)+10\Delta t \tag{37}$$

$$\boxed{i(t) \simeq \frac{Li(t-\Delta t)+10\Delta t}{R\Delta t+L}} \tag{38}$$

Voltage $v_L(t)$:

$$v_L(t)\Delta t \simeq L[i(t)-i(t-\Delta t)] \tag{39}$$

$$\boxed{v_L(t) \simeq \frac{L[i(t)-i(t-\Delta t)]}{\Delta t}} \tag{40}$$

3 Simulation

With the equations derived, we can now plot our results to see how they perform with different parameters. The assignment requests the following solutions to be generated using a computer program:

a) Using the trapezoidal rule with $\Delta t_1 = 0.1ms$

b) Using the backward Euler rule with $\Delta t_1 = 0.1ms$

c) Using the trapezoidal rule with $\Delta t_2 = 0.8ms$

d) Using the backward Euler rule with $\Delta t_2 = 0.8ms$

In all plots, the continuous solution is plotted as a dotted line to show what the exact solution should be. This provides a baseline to compare the performance of the different approximations. Figure 1. shows the performance of the Trapezoidal approximation using time steps of $0.1ms$ and $0.8ms$. Figure 2. shows the performance of the backward Euler approximation using time steps of $\Delta t_1 = 0.1ms$ and $\Delta t_2 = 0.8ms$. Finally, Figure 3. compares both approximation techniques using the time step of $\Delta t_2 = 0.8ms$.

As expected, the smaller the time step, the closer the step-by-step solutions approximate the real solution. This makes sense because as the Δt becomes smaller, the closer it becomes to approximating the integration. What was unexpected, however, is the backward Euler solution more closely approximating the solution numerically versus the trapezoidal one. Intuitively, the trapezoidal solution should approximate the solution better due to its inclusion of a triangle area above the square below. The trapezoidal approximation does seem to follow the shape better than the backward Euler one, however. The backward Euler approximation can be seen to cross the true solution towards the end of the simulation in Figure 3. whereas the trapezoidal solution asymptotically approaches the true solution. At small time steps the trapezoidal solution is likely the better one.

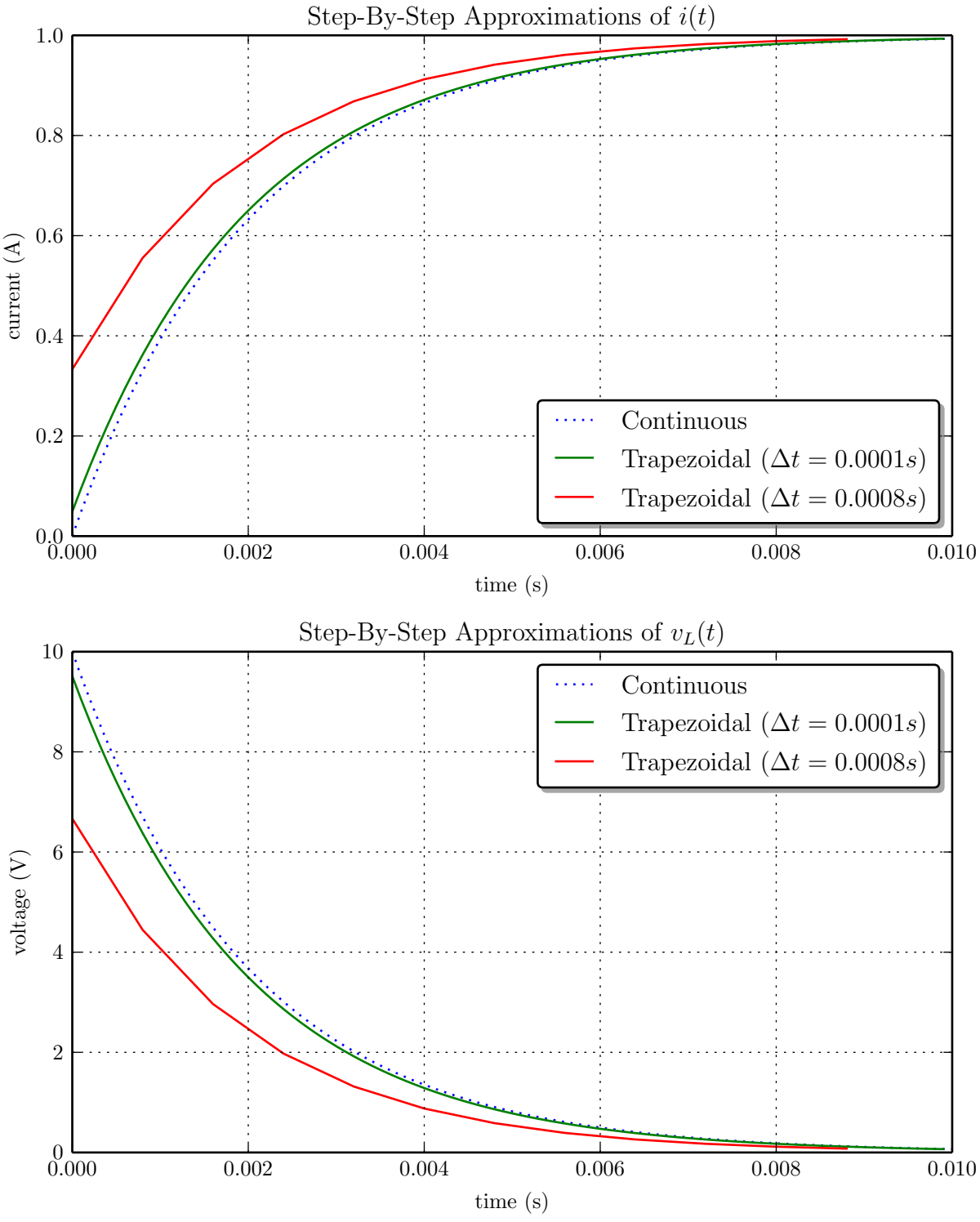

Figure 1: Trapezoidal Approximations at $\Delta t_1 = 0.1ms, \Delta t_2 = 0.8ms$

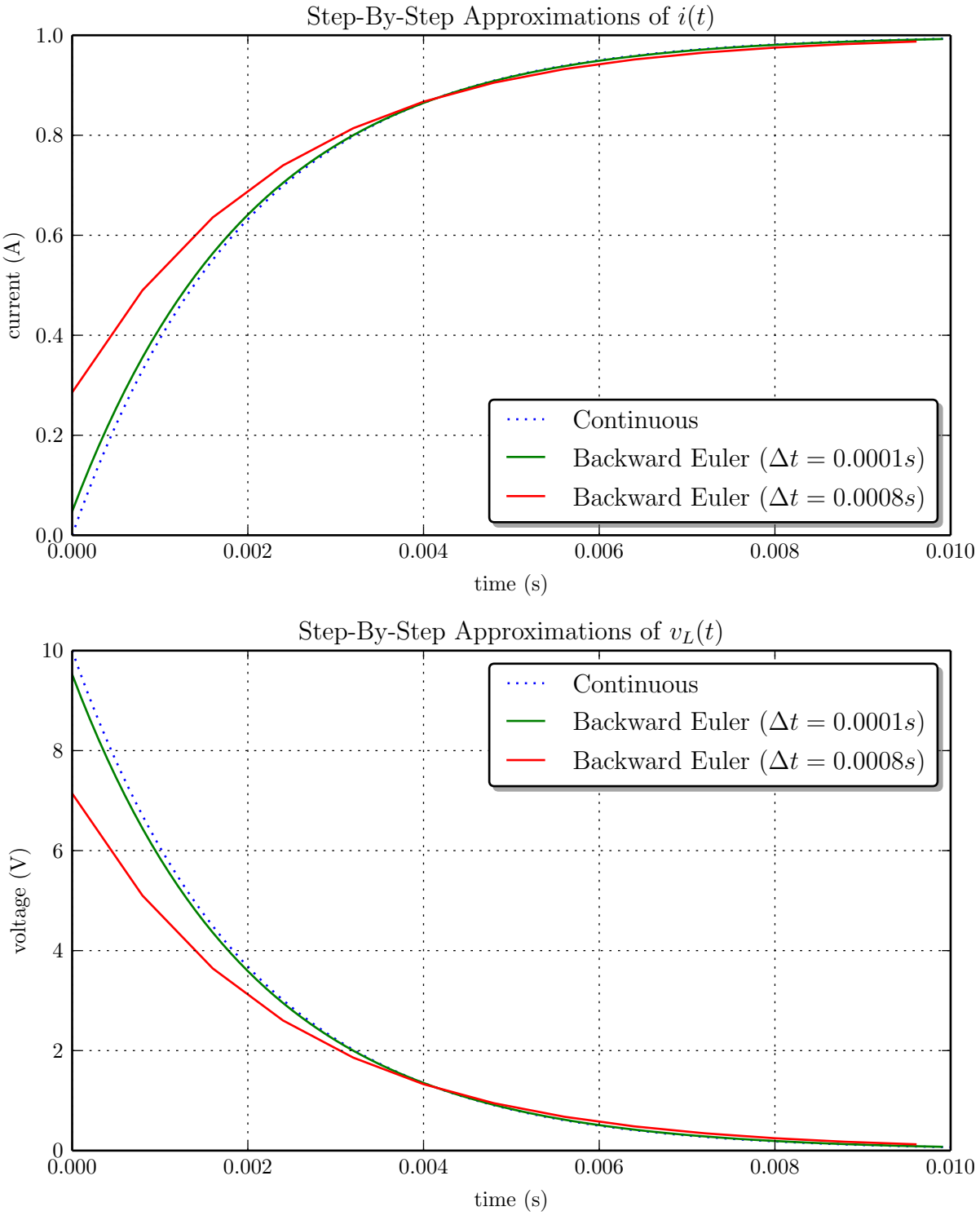

Figure 2: Backward Euler Approximations at $\Delta t_1 = 0.1ms, \Delta t_2 = 0.8ms$

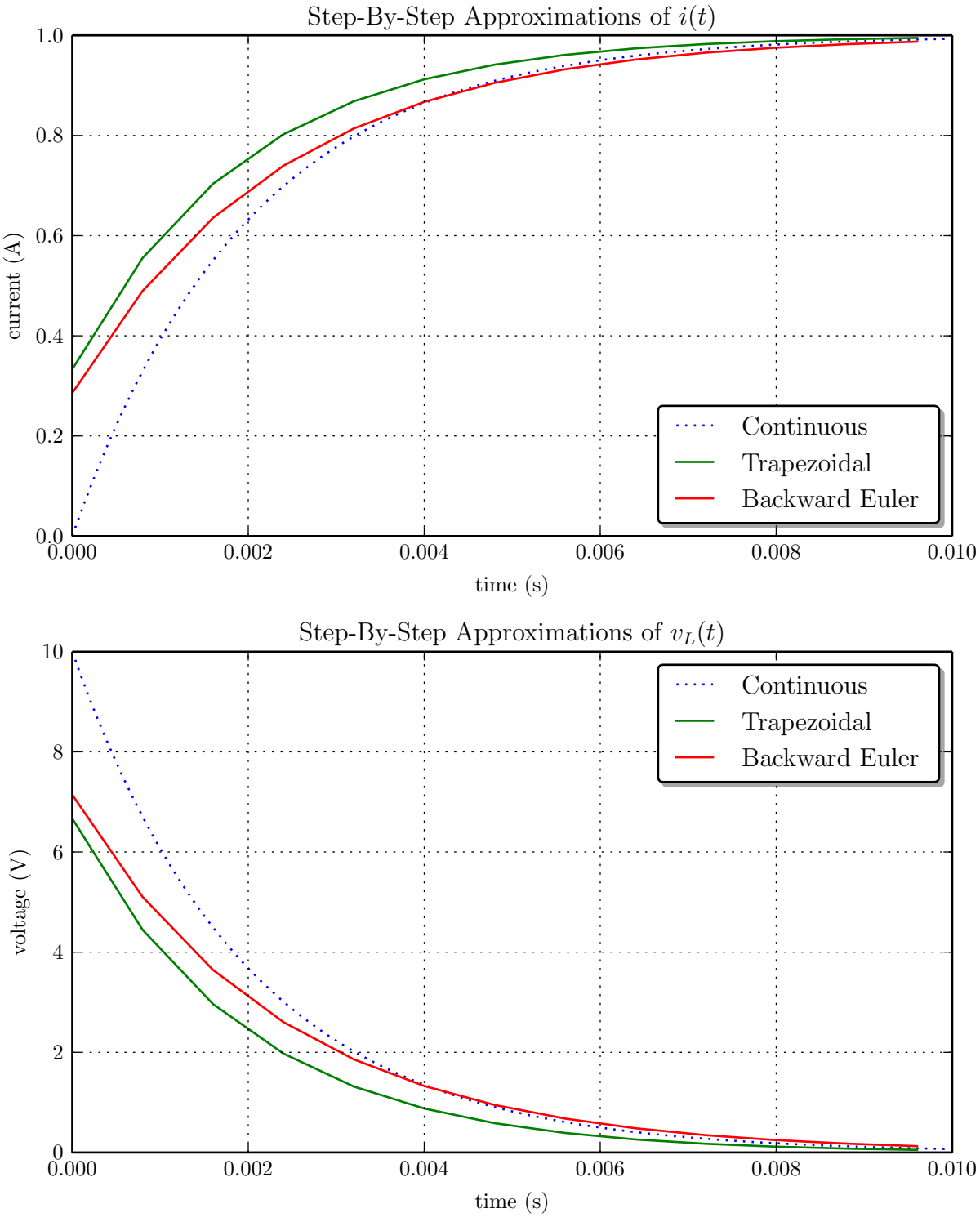

Figure 3: Comparison of Approximations at $\Delta t_2 = 0.8ms$

4 Conclusion

Overall this was a good refresher on transient circuit analysis as well as a productive introduction to discretization techniques. Further optimizations and improvements are proposed below:

- In order to perform more efficient calculation steps, the program could decide to dynamically modify its solution's step size if it detects the output to be stable. This would allow it to more effectively make use of its compute cycles. A threshold could be implemented to determine the range of change for which it should attempt to modify its step size.
- In a similar vein, the program could also attempt changing its discretization rules on the fly to better approximate the transient solution. Perhaps in some scenarios, a circuit exhibits several different stages where certain approximations work better than others.
- Using a programming language that supports special compiler hints, a program could be written to make use of special processing units that accelerate an approximation's iteration. Perhaps a program could also run multiple discretization rules in parallel, and through some kind of analysis, dynamically switch between the more accurate solution.

5 Code Listing

The following is the code written in Python to perform the calculations derived for this homework assignment as well as generate the plots used in this report.

```
import argparse
import math
import matplotlib
import matplotlib.pyplot as plt

# Matplotlib export settings
matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    "font.size": 10 ,
    "font.family": "serif",  # use serif/main font for text elements
    "text.usetex": True,     # use inline math for ticks
    "pgf.rcfonts": False     # don't setup fonts from rc parameters
})

# Main function
def main(args):
    # Circuit parameters
    inductance_L = 0.02 # 20 mH
    resistance_R = 10 # 10 Ohms
    voltage_E = 10 # 10 Volts

    print("Simulation time = %g" % args.simulation_time)
    print("Will run for the following time deltas:")
    for delta in args.time_delta:
        print("Time delta = %g" % delta)

    cont_results = [[],[],[]]
    trap_results = [[[],[],[]]]
    back_results = [[[],[],[]]]
    # indices for the results tuple
    TIME_INDEX = 0
    CURRENT_INDEX = 1
    VOLTAGE_INDEX = 2

    # Calculate results using continuous solution
    print("Simulating using continuous solution")
    # Calculate 3000 points for the continuous solution
    for step in range(3000):
        time = 0.0 + (step * args.simulation_time/3000)
        # calculate continuous solution
        # current
        i_continuous_next = (10/resistance_R) - ((10/resistance_R)*math.exp(-(resistance_R/
            inductance_L)*time))
        # inductor voltage
        v_continuous_next = 10 * math.exp((-(resistance_R)/(inductance_L))*time)
        # append the result
        cont_results[TIME_INDEX].append(0+(step*args.simulation_time/3000))
        cont_results[CURRENT_INDEX].append(i_continuous_next)
        cont_results[VOLTAGE_INDEX].append(v_continuous_next)

    # Calculate results using trapezoidal discretization
    print("Simulating using trapezoidal discretization")
    for delta_index, delta in enumerate(args.time_delta):
        sim_steps = int(args.simulation_time/delta)
        for step in range(sim_steps):
            if step == 0:
                # Use initial conditions
                i_prev = args.i_0
                v_prev = args.v_0
            else:
                # Use previous value
```

```
            i_prev = trap_results[delta_index][CURRENT_INDEX][step - 1]
            v_prev = trap_results[delta_index][VOLTAGE_INDEX][step - 1]
        # perform trapezoidal discretization step
        # current
        i_approx_next = ((i_prev*((2*inductance_L) - (resistance_R*delta))) + (20 *
            delta))/((resistance_R*delta)+(2*inductance_L))
        # voltage
        v_approx_next = (((2*inductance_L)/(delta))*(i_approx_next - i_prev)) - v_prev
        # append the result
        trap_results[delta_index][TIME_INDEX].append(0+(step*delta))
        trap_results[delta_index][CURRENT_INDEX].append(i_approx_next)
        trap_results[delta_index][VOLTAGE_INDEX].append(v_approx_next)
    # append a new set of results for the next delta
    trap_results.append([[],[],[]])

# Calculate results using backwards euler discretization
print("Simulating␣using␣backward␣euler␣discretization")
for delta_index, delta in enumerate(args.time_delta):
    sim_steps = int(args.simulation_time/delta)
    for step in range(sim_steps):
        if step == 0:
            # Use initial conditions
            i_prev = args.i_0
        else:
            # Use previous value
            i_prev = back_results[delta_index][CURRENT_INDEX][step - 1]
        # perform backward eurler discretization step
        # current
        i_approx_next = ((inductance_L*i_prev)+(10*delta))/((resistance_R*delta) +
            inductance_L)
        # voltage
        v_approx_next = ((inductance_L)/(delta))*(i_approx_next - i_prev)
        # append the result
        back_results[delta_index][TIME_INDEX].append(0+(step*delta))
        back_results[delta_index][CURRENT_INDEX].append(i_approx_next)
        back_results[delta_index][VOLTAGE_INDEX].append(v_approx_next)
    # append a new set of results for the next delta
    back_results.append([[],[],[]])

# Plots for publication
# Comparisons between all techniques
for delta_index, delta in enumerate(args.time_delta):
    # Current plot
    fig, ax = plt.subplots(2)
    ax[0].plot(cont_results[TIME_INDEX], cont_results[CURRENT_INDEX], linestyle='dotted'
        , label='Continuous')
    ax[0].plot(trap_results[delta_index][TIME_INDEX], trap_results[delta_index][
        CURRENT_INDEX], label='Trapezoidal')
    ax[0].plot(back_results[delta_index][TIME_INDEX], back_results[delta_index][
        CURRENT_INDEX], label='Backward␣Euler␣␣␣␣')
    ax[0].set(xlabel='time␣(s)', ylabel='current␣(A)', title='Step-By-Step␣
        Approximations␣of␣$i(t)$')
    ax[0].legend(loc='best', fancybox=True, shadow=True)
    ax[0].grid()
    # Voltage plot
    ax[1].plot(cont_results[TIME_INDEX], cont_results[VOLTAGE_INDEX], linestyle='dotted'
        , label='Continuous')
    ax[1].plot(trap_results[delta_index][TIME_INDEX], trap_results[delta_index][
        VOLTAGE_INDEX], label='Trapezoidal')
    ax[1].plot(back_results[delta_index][TIME_INDEX], back_results[delta_index][
        VOLTAGE_INDEX], label='Backward␣Euler␣␣␣␣')
    ax[1].set(xlabel='time␣(s)', ylabel='voltage␣(V)', title='Step-By-Step␣
        Approximations␣of␣$v_L(t)$')
    ax[1].legend(loc='best', fancybox=True, shadow=True)
    ax[1].grid()
    fig.set_size_inches(6.5,8)
    fig.tight_layout()
    # plt.show()
```

```
        fig.savefig('compare_plot_' + str(delta).replace('.','p') + '.pgf')
        fig.savefig('compare_plot_' + str(delta).replace('.','p') + '.png')
    # Comparisons between individual techniques
    # Trapezoidal
    # Current plot
    fig, ax = plt.subplots(2)
    ax[0].plot(cont_results[TIME_INDEX], cont_results[CURRENT_INDEX], linestyle='dotted',
        label='Continuous')
    for delta_index, delta in enumerate(args.time_delta):
        ax[0].plot(trap_results[delta_index][TIME_INDEX], trap_results[delta_index][
            CURRENT_INDEX], label='Trapezoidal␣($\Delta{}t␣=␣%g␣s$)'%delta)
        ax[0].set(xlabel='time␣(s)', ylabel='current␣(A)', title='Step-By-Step␣
            Approximations␣of␣$i(t)$')
        ax[0].legend(loc='best', fancybox=True, shadow=True)
    ax[0].grid()
    # Voltage plot
    ax[1].plot(cont_results[TIME_INDEX], cont_results[VOLTAGE_INDEX], linestyle='dotted',
        label='Continuous')
    for delta_index, delta in enumerate(args.time_delta):
        ax[1].plot(trap_results[delta_index][TIME_INDEX], trap_results[delta_index][
            VOLTAGE_INDEX], label='Trapezoidal␣($\Delta{}t␣=␣%g␣s$)'%delta)
        ax[1].set(xlabel='time␣(s)', ylabel='voltage␣(V)', title='Step-By-Step␣
            Approximations␣of␣$v_L(t)$')
        ax[1].legend(loc='best', fancybox=True, shadow=True)
    ax[1].grid()
    # Backward Euler
    # Current plot
    fig, ax = plt.subplots(2)
    ax[0].plot(cont_results[TIME_INDEX], cont_results[CURRENT_INDEX], linestyle='dotted',
        label='Continuous')
    for delta_index, delta in enumerate(args.time_delta):
        ax[0].plot(back_results[delta_index][TIME_INDEX], back_results[delta_index][
            CURRENT_INDEX], label='Backward␣Euler␣($\Delta{}t␣=␣%g␣s$)'%delta)
        ax[0].set(xlabel='time␣(s)', ylabel='current␣(A)', title='Step-By-Step␣
            Approximations␣of␣$i(t)$')
        ax[0].legend(loc='best', fancybox=True, shadow=True)
    ax[0].grid()
    # Voltage plot
    ax[1].plot(cont_results[TIME_INDEX], cont_results[VOLTAGE_INDEX], linestyle='dotted',
        label='Continuous')
    for delta_index, delta in enumerate(args.time_delta):
        ax[1].plot(back_results[delta_index][TIME_INDEX], back_results[delta_index][
            VOLTAGE_INDEX], label='Backward␣Euler␣($\Delta{}t␣=␣%g␣s$)'%delta)
        ax[1].set(xlabel='time␣(s)', ylabel='voltage␣(V)', title='Step-By-Step␣
            Approximations␣of␣$v_L(t)$')
        ax[1].legend(loc='best', fancybox=True, shadow=True)
    ax[1].grid()
    fig.set_size_inches(6.5,8)
    fig.tight_layout()
    # plt.show()
    fig.savefig('backeuler_plots.pgf')
    fig.savefig('backeuler_plots.png')

if __name__ == '__main__':
    # the following sets up the argument parser for the program
    parser = argparse.ArgumentParser(description='Assignment␣1a␣solution␣generator')
    # allow the user to input multiple time steps if they desire (e.g. 0.1 ms and 0.8 ms
        simultaneously)
    parser.add_argument('-d', type=float, dest='time_delta', nargs='+', action='store',
        help='time␣delta␣to␣use␣in␣discretization')
    parser.add_argument('-l', type=float, dest='simulation_time', action='store',
        help='time␣to␣run␣the␣simulation␣for')
    parser.add_argument('-i', type=float, dest='i_0', action='store',
        help='i␣at␣i(0)')
    parser.add_argument('-v', type=float, dest='v_0', action='store',
        help='v_L␣at␣v_L(0)')
    parser.add_argument('-o', type=str, default='output.csv', dest='output_file', action='
        store',
```

```
        help='path␣and␣name␣of␣the␣file␣containing␣the␣results')

    args = parser.parse_args()

    main(args)
```