Transient Recovery Voltage (TRV) in Circuit Breakers using a Lumped-Circuit Line Model

Assignment 2

Due: 2021/02/12

1 Introduction

This assignment investigates the transient recovery voltage in circuit breakers of a shorted transmission line using a lumped-circuit line model. For this scenario, the short circuit happens at t = 0 s at the end of the transmission line and a breaker exists between the transmission line and the power source. By design, the breaker cannot chop current, so after it has been signaled to open (66.67 ms for this assignment), it waits for current to cross zero before opening. In practice, this would help minimize arcing. This behaviour is modeled in the program for the assignment as well as in PSCAD.

This report will outline how the circuit was modeled and simulated and go over the resulting waveforms using both the custom computer program as well as through PSCAD.

2 Setup

In order to provide some background to the results, the following sections outline the derivation of the solutions. Both the hand-implemented as well as the PSCAD solution have their own section. As provided by the assignment directives, the values used for the components are as follows:

$$\begin{array}{lll} R_1 = 3\,\Omega & L_1 = 350\,mH & C_1 = 10\,nF \\ R_2 = 50\,\Omega & L_2 = 100\,mH & C_2 = 600\,nF \end{array}$$

These values are substituted into the hand-implemented simulation into the solution equations and the PSCAD schematic.

2.1 Hand-Implemented Simulation

When hand modeling the circuit, a new set of nodes were chosen to better represent the lumped-circuit model. First, the L and C component were discretized into their respective resistance and voltage sources. Current source transformations were then performed on the L and C resistor/voltage source pairs to simplify nodal analysis of the circuit. Figure 1 shows the breakdown of these steps into the final model consisting of resistances and current sources. In total, five nodal voltages are generated, which will later be renamed to the original numbers to keep in line with the original naming convention.

Using trapezoidal discretization and a subsequent current source transformation, the following history current source equations were generated for each of the four current sources:

$$h_{23}(t) = -\frac{2\,L_1\,i_{23}(t-\Delta t) + \Delta t\,v_2(t-\Delta t) - \Delta t\,v_3(t-\Delta t)}{2\,L_1} \tag{1}$$

$$h_{30}(t) = \frac{2\,C_1\,v_3(t-\Delta t) + \Delta t\,i_{30}(t-\Delta t)}{\Delta t} \tag{2}$$

$$h_{40}(t) = \frac{2\,C_2\,v_4(t-\Delta t) + \Delta t\,i_{40}(t-\Delta t)}{\Delta t} \tag{3}$$

$$h_{50}(t) = -\frac{2\,L_2\,i_{50}(t-\Delta t) + \Delta t\,v_5(t-\Delta t)}{2\,L_2} \tag{4}$$

Having the equations retain variables for the component values as well as the time delta allows the resulting solutions to easily simulate different component values and time deltas for experimentation. The Python program substitutes the components values at runtime for the simulation.

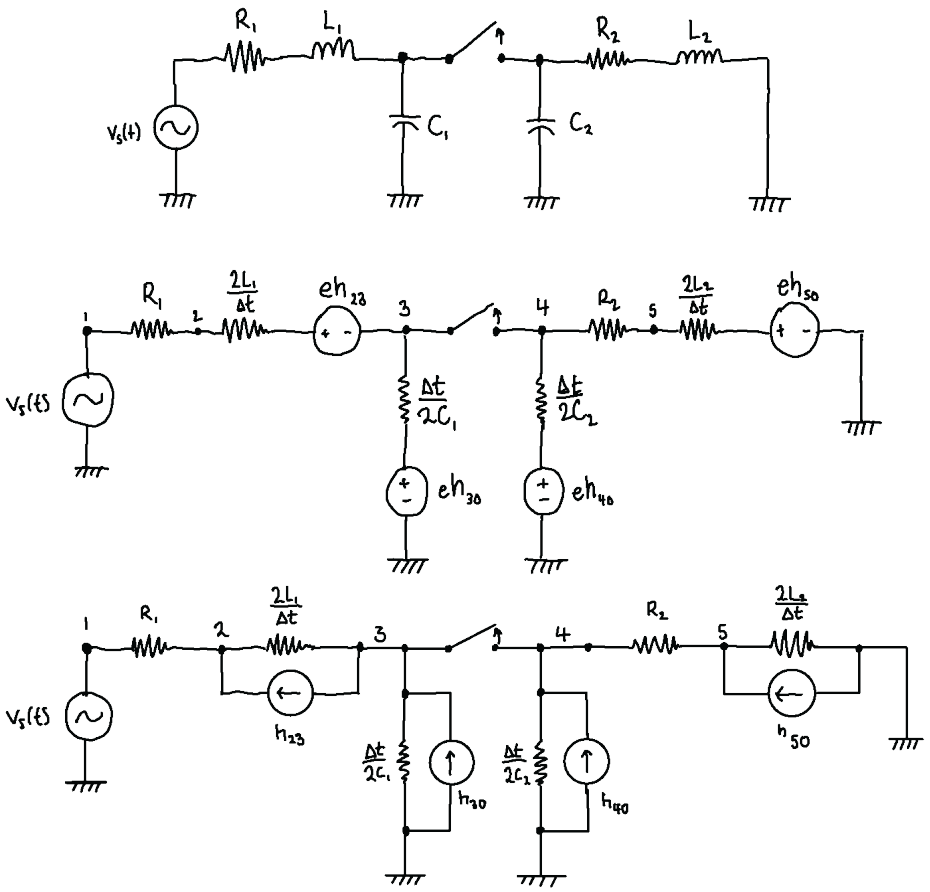

Figure 1: Transmission line's equivalent circuit and derivation of discretized components.

Once the current source histories are determined, the branch currents, which are required values for calculating the next nodal voltage values, can be calculated using the following relationships:

$$i_{23}(t) = v_{23}(t)/R_{23} - h_{23}(t) \tag{5}$$
$$i_{30}(t) = v_{30}(t)/R_{30} - h_{30}(t) \tag{6}$$
$$i_{40}(t) = v_{40}(t)/R_{40} - h_{40}(t) \tag{7}$$
$$i_{50}(t) = v_{50}(t)/R_{50} - h_{50}(t) \tag{8}$$

For this exercise, the convention used for currents leaving the node marks them as positive and currents entering the node marks them as negative. Therefore the resulting branch currents are the nodal voltage divided by the branch resistance, minus the discretized component current sources for all branch currents in this circuit.

Finally, we can now calculate the nodal voltage equations using some clever matrix manipulation taught in class. This technique subdivides the usual nodal analysis conductance matrix into four sections, with nodes connected to known voltage sources occupying the outside matrices $\mathbf{G_{AB}}$, $\mathbf{G_{BA}}$, and $\mathbf{G_{BB}}$. This allows us to perform the following operation to obtain our nodal voltage equation vector:

$$[\mathbf{V_A(t)}] = [\mathbf{G_{AA}}]^{-1}[\mathbf{h_A(t)}] - [\mathbf{G_{AA}}]^{-1}[\mathbf{G_{AA}}][\mathbf{V_B(t)}] \tag{9}$$

For this implementation, the breaker switch closing and opening has the effect of either connecting or breaking nodes 3 and 4 of the circuit (see Figure 1 for node numbering). This has the result of generating two different conductance and history matrices. Shown below are both conductance and history matrices used for this circuit's solution:

$$[\mathbf{G_{closed}}] = \left[\begin{array}{c:c} \mathbf{G_{AA}} & \mathbf{G_{AB}} \\ \hdashline \mathbf{G_{BA}} & \mathbf{G_{BB}} \end{array}\right] = \left[\begin{array}{ccc:c} \frac{\Delta t}{2L_1} + \frac{1}{R_1} & -\frac{\Delta t}{2L_1} & 0 & -\frac{1}{R_1} \\ -\frac{\Delta t}{2L_1} & \frac{2C_1}{\Delta t} + \frac{2C_2}{\Delta t} + \frac{\Delta t}{2L_1} + \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ 0 & -\frac{1}{R_2} & \frac{\Delta t}{2L_2} + \frac{1}{R_2} & 0 \\ \hdashline -\frac{1}{R_1} & 0 & 0 & \frac{1}{R_1} \end{array}\right] \tag{10}$$

$$[\mathbf{G_{open}}] = \left[\begin{array}{c:c} \mathbf{G_{AA}} & \mathbf{G_{AB}} \\ \hdashline \mathbf{G_{BA}} & \mathbf{G_{BB}} \end{array}\right] = \left[\begin{array}{cccc:c} \frac{\Delta t}{2L_1} + \frac{1}{R_1} & -\frac{\Delta t}{2L_1} & 0 & 0 & -\frac{1}{R_1} \\ -\frac{\Delta t}{2L_1} & \frac{2C_1}{\Delta t} + \frac{\Delta t}{2L_1} & 0 & 0 & 0 \\ 0 & 0 & \frac{2C_2}{\Delta t} + \frac{1}{R_2} & -\frac{1}{R_2} & 0 \\ 0 & 0 & -\frac{1}{R_2} & \frac{\Delta t}{2L_2} + \frac{1}{R_2} & 0 \\ \hdashline -\frac{1}{R_1} & 0 & 0 & 0 & \frac{1}{R_1} \end{array}\right] \tag{11}$$

$$[\mathbf{H_{closed}}] = \left[\begin{array}{c} \mathbf{H_A} \\ \hdashline \mathbf{H_B} \end{array}\right] = \left[\begin{array}{c} h_{23}(t) \\ -h_{23}(t) + h_{30}(t) + h_{40}(t) \\ h_{50}(t) \\ \hdashline 0 \end{array}\right] \tag{12}$$

$$[\mathbf{H_{open}}] = \left[\begin{array}{c} \mathbf{H_A} \\ \hdashline \mathbf{H_B} \end{array}\right] = \left[\begin{array}{c} h_{23}(t) \\ -h_{23}(t) + h_{30}(t) \\ h_{40}(t) \\ h_{50}(t) \\ \hdashline 0 \end{array}\right] \tag{13}$$

The resulting nodal voltage equations were generated using MATLAB to minimize algebraic errors; the equations used to generate the nodal voltage equations were saved into a MATLAB file which can be viewed in Listing 5.2. The resulting equations were then ported to a Python script which is provided in Listing 5.1). There are two sets of nodal voltage solutions, one for the closed breaker and another for the open breaker. The closed breaker solution is used to begin with. Initial conditions are assumed to be zero for all variables. For each time step, the Python script then calculates the next nodal voltage value. These values are then used to calculate the next branch currents. After calculating the branch currents, the Python script checks if the breaker should have activated and if so, checks if the breaker current has changed sign since the last iteration. If both of these conditions are true, the breaker opens, effectively avoiding chopping current, and the open breaker solutions are used to calculate the nodal voltages until the end of the simulation. These results are plotted alongside the equivalent PSCAD solution for comparison using the Python matplotlib plotting library.

2.2 Choosing a Simulation Time Step

In order to choose an appropriate Δt to simulate the circuit with, both sides of the circuit were analyzed for their resonant frequencies. This was performed by the following calculations:

$$f_{left} = \frac{1}{2\pi\sqrt{L_1 C_1}} = 2.69\,kHz \tag{14}$$

$$f_{right} = \frac{1}{2\pi\sqrt{L_2 C_2}} = 650\,Hz \tag{15}$$

The higher of the two frequencies, $2.69\,kHz$, corresponds to a period of $372\,\mu s$, so in order to get a good simulation result, an order of magnitude higher was chosen, at $\Delta t = 10\,\mu s$. 10 was chosen because it was a nice even number.

2.3 PSCAD Simulation

PSCAD provides a master library of components as well as primitives to help model the circuit. The resulting PSCAD circuit is shown in Figure 2. The "Single Phase Voltage Source Model" component was used as the source and it was made ideal by setting the source impedance to 0Ω with a ramp-up time of $0s$ (instantaneous voltage). The frequency was set to 60 Hz with a magnitude of $\frac{230}{\sqrt{2}}\,kV_{rms}$ and an offset of 90 degrees to match the assignment's voltage source equation of $230cos(377t)\,kV$. The ideal resistor, inductor, and capacitor were used to model the passive components and a series of voltmeters and ammeters were placed around the circuit to match the hand implemented solution. The voltage and current readings were piped to a plot on the same page as the schematic. In order to determine when PSCAD's breaker opened, the breaker's state variable was observed by plotting its value, and the resulting data is observed for a change that corresponds to an open state. The Python program plots both breaker opening times for the hand-implemented and PSCAD solutions (which turned out to be identical).

Once all of the components were hooked together and instrumented, the build button was clicked. This generated a Fortran program that models the entire circuit. A listing of this program can be found in Section 5.3.

Clicking the run button ran the simulation and outputted the results on the plots. The data was extracted by right-clicking each plot to save it to the clipboard. This was then pasted into a CSV file for the Python program to read in as raw values.

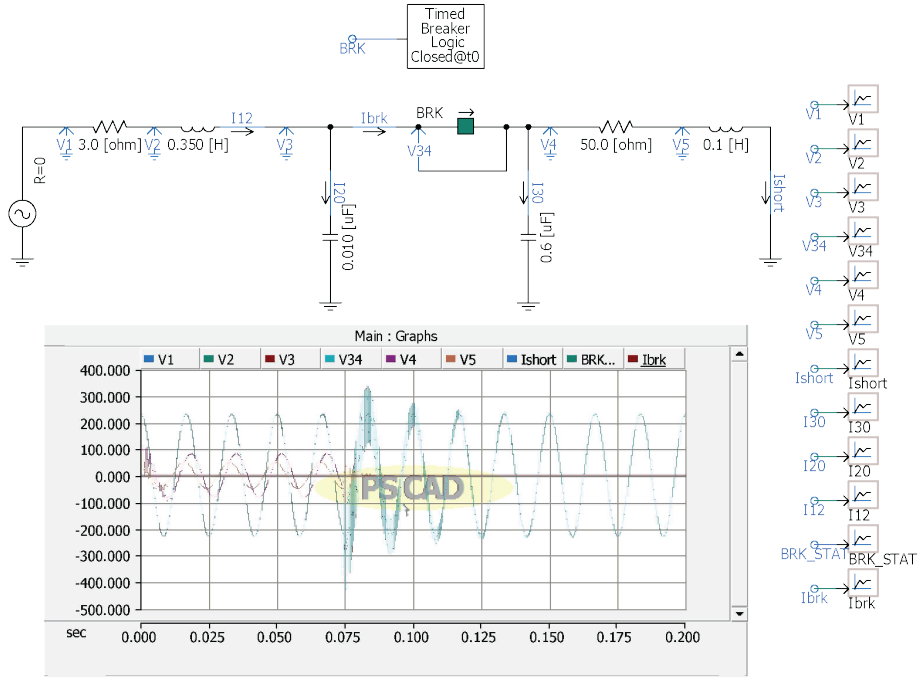

Figure 2: PSCAD schematic of transmission line.

3 Simulation

With the hand-implemented and PSCAD solutions complete, the results can then be simulated and compared. This assignment requests the following comparison plots along with a zoomed-in portion for analysis. The figures following contain all of the plots required for the assignment.

a) Plots of V_1, V_2, V_3 for the simulation and PSCAD (Figures 3 and 4).

b) Plots of I_{40} which represents the short-circuit current at the end of the transmission line (Figures 7 and 8).

c) Plots of V_{23} which represents the voltage across the breaker (Figures 5 and 6).

Plots named simulation refer to the hand-modeled circuit and those named PSCAD refer to the PSCAD modeled circuit. In each plot, the red dashed line represents where the breaker has opened in the solution, which was identical between the hand-simulated and PSCAD solutions.

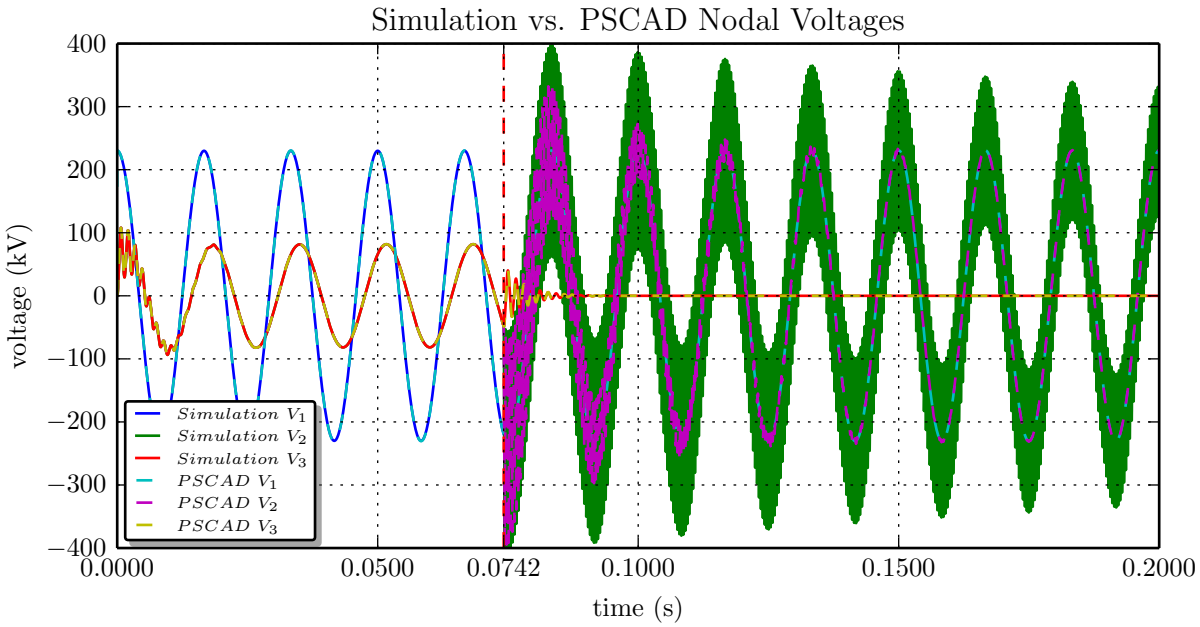

Figure 3: Plots of V_1, V_2, and V_3 nodal voltages

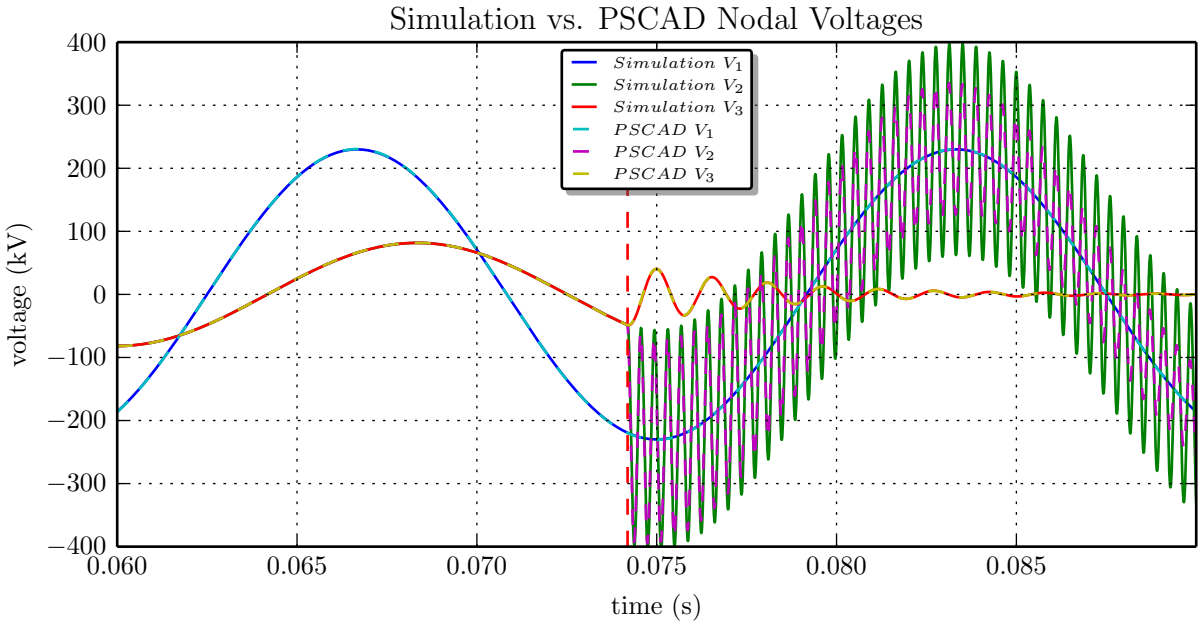

Figure 4: Plots of V_1, V_2, and V_3 nodal voltages (zoomed)

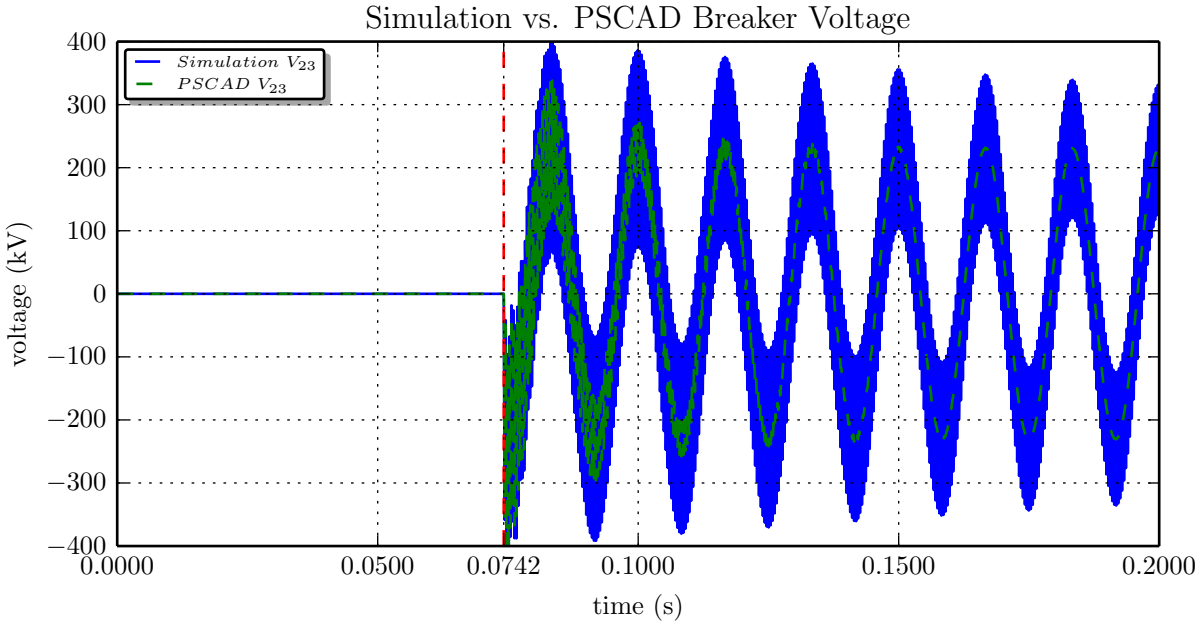

Figure 5: Plots of V_{23} breaker voltage

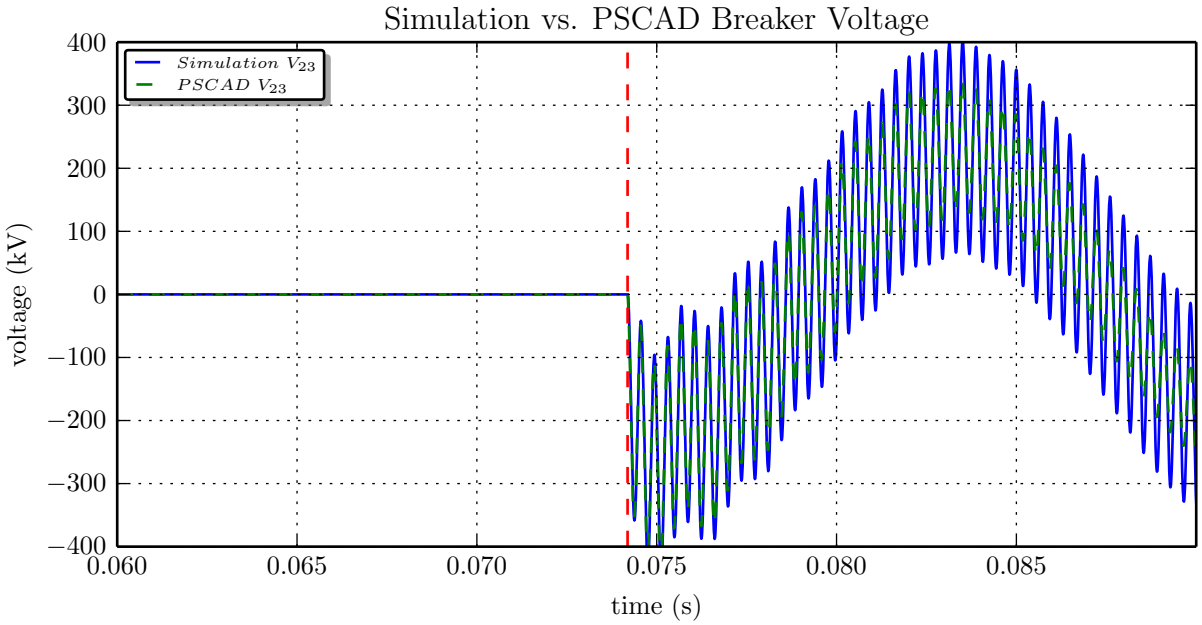

Figure 6: Zoomed plots of V_{23} breaker voltage (zoomed)

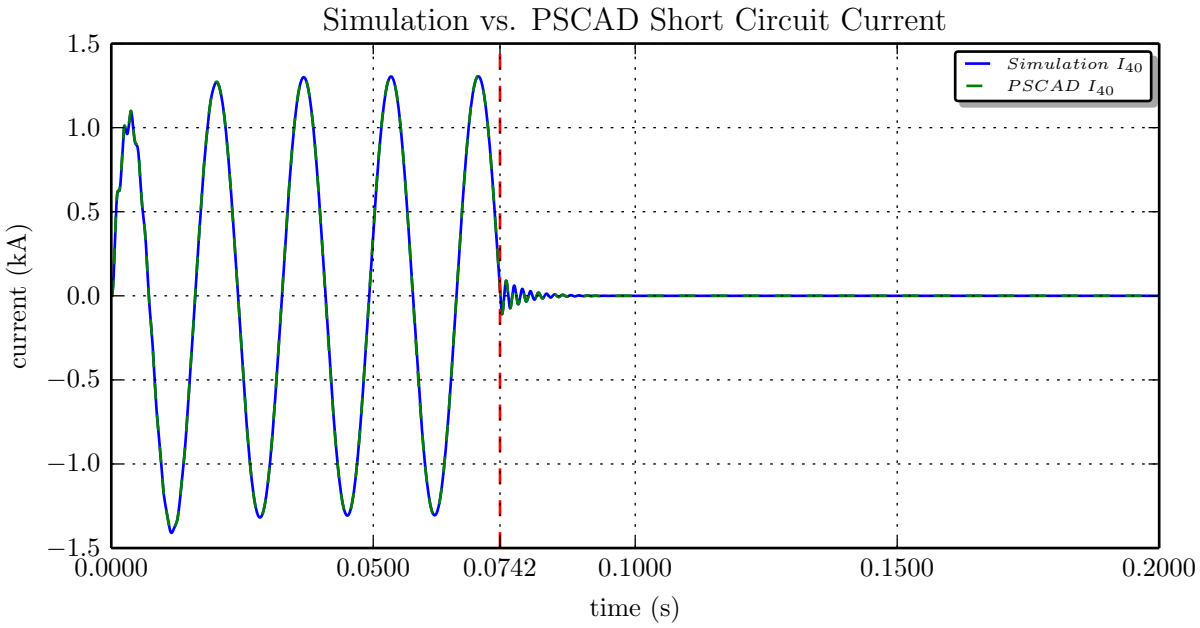

Figure 7: Plots of I_{40} short circuit current

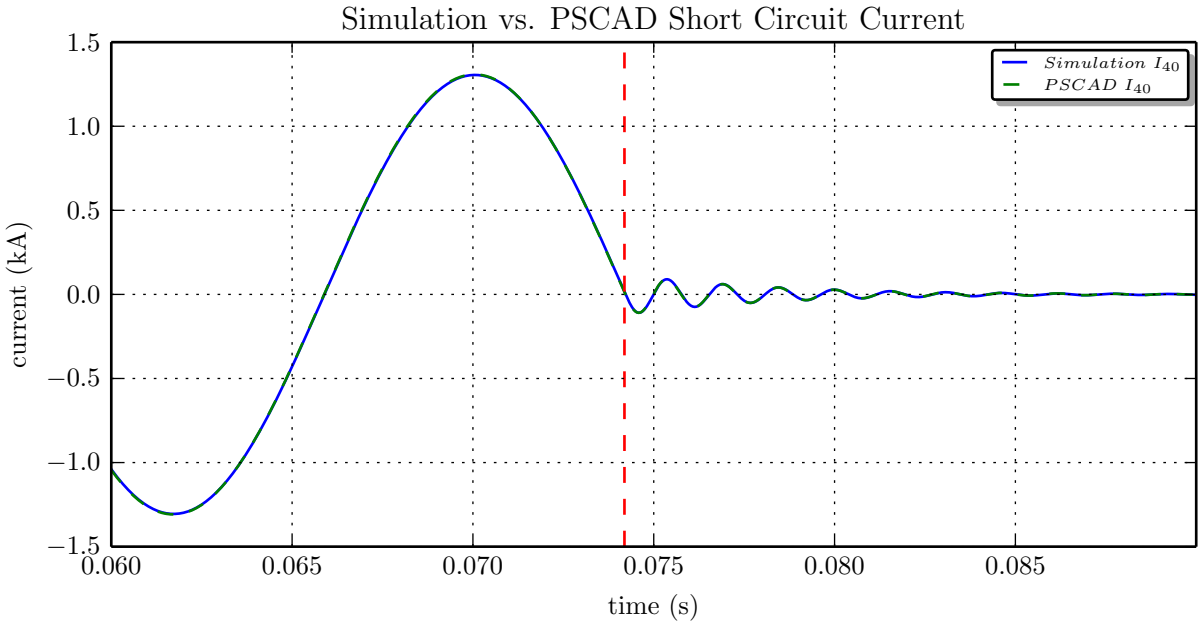

Figure 8: Zoomed plots of I_{40} short circuit current (zoomed)

4 Conclusion

Overall, the hand-implemented solution matched the results generated by PSCAD, with some exceptions. The following are observations and comments from the results of this assignment.

- Both simulations exhibited instability at the beginning, very likely due to the initial conditions being 0 for all cases. As the simulation steps forward, however, the solutions stabilize fairly quickly.
- Both simulation and PSCAD breakers opened at the same time. This means that the zero crossing detection implemented by the Python simulation was valid and that the simulations followed each other very closely.
- The voltage across the breaker exhibited a ringing voltage wave riding on top of the larger 60 Hz wave. Interestingly enough, in PSCAD's simulation, the ringing attenuated much quicker than the simulation's did. The ringing had a frequency of approximately $2.7\,kHz$, which is very close to the left-hand side's resonant frequency. The abrupt opening of the breaker must have introduced enough noise to excite the left hand side at its resonant frequency.
- PSCAD implemented the breaker differently than the hand-implemented solution. This was discovered after configuring the breaker in PSCAD, which requests the user to specify closed and open resistances. This allows the simulation to retain the same solution matrix, unlike the hand-implemented solution which must switch from one to the other when the breaker state changes. It is possible that the resistance allowed the oscillations a path to dissipate its energy in the PSCAD solution, something that was not possible as quickly through the hand-implemented solution.
- The left-hand side of the circuit exhibited a resonant frequency of $2.69\,kHz$ compared to the right-hand side's resonant frequency of $650\,Hz$. When the breaker is closed, the circuit as a whole will have a different resonant frequency. This change in resonant frequencies when the breaker opens means that each side will absorb different frequencies. Since breakers will never be perfect, many different harmonic components are introduced when it opens, which will be introduced on either side of the breaker. The new resonant frequencies of either side will therefore absorb/dissipate the most energy at these frequencies. Perhaps this is a consideration for devices connected to either network which may be sensitive to certain frequencies.
- It would be interesting to implement the breaker as PSCAD did (change of breaker branch resistance vs. change of G matrix) to see if the V_{23} voltages can more closely match between the two solutions.

5 Code Listings and Data

5.1 Python 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. The PSCAD results were saved into a CSV file which the Python script reads in at the start of the program.

```
import argparse
import csv
import math
import matplotlib
import numpy as np


 # Matplotlib export settings
matplotlib.use("pgf")
import matplotlib.pyplot as plt
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):
    # Lists for results
    pscad_t_vals = []
    pscad_v1_vals = []
    pscad_v2_vals = []
    pscad_v3_vals = []
    pscad_v34_vals = []
    pscad_v4_vals = []
    pscad_v5_vals = []
    pscad_ishort_vals = []

    t_vals = []
    v1_vals = []
    v2_vals = []
    v3_vals = []
    v34_vals = []
    v4_vals = []
    v5_vals = []

    i23_vals = []
    i30_vals = []
    i40_vals = []
    i50_vals = []

    # Read in PSCAD .CSV data
    print('Opening␣PSCAD␣CSV␣file...')
    with open('PSCAD_data.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        pscad_breaker_open_time = 0
        # Read in row data
        for row in csv_reader:
            if line_count == 0:
                # print('Column names are: ' + ', '.join(row))
                line_count += 1
            else:
                pscad_t_vals.append(row[0])
                pscad_v1_vals.append(row[1])
```

```
                pscad_v2_vals.append(row[2])
                pscad_v3_vals.append(row[3])
                pscad_v34_vals.append(row[4])
                pscad_v4_vals.append(row[5])
                pscad_v5_vals.append(row[6])
                pscad_ishort_vals.append(row[7])
                # Check if break opned
                if float(row[8]) > 0 and pscad_breaker_open_time == 0:
                    pscad_breaker_open_time = float(row[0])
                    print('PSCAD breaker opened at %g' % float(pscad_breaker_open_time))
                line_count += 1
        # Figure out when break switched
        print('Processed ' + str(line_count) + ' lines.')

    # Time delta used for calculations
    delta_t = 10e-6
    # Time for breaker switch to open
    breaker_time = 66.67e-3
    # Zero crossing flag
    zero_crossing_flag = False
    # Breaker open flag
    breaker_open_flag = False
    # Time breaker opened
    breaker_open_time = 0
    # Simulation time
    simulation_time = 200e-3

    # Component values
    R1 = 3
    L1 = 0.35
    C1 = 10e-9
    R2 = 50
    L2 = 0.1
    C2 = 600e-9

    # Start at t = delta_t
    t = delta_t
    # Zero initial conditions
    v2 = 0.0
    v3 = 0.0
    v4 = 0.0
    v5 = 0.0
    i23 = 0.0
    i30 = 0.0
    i40 = 0.0
    i50 = 0.0

    # Run simulation
    while t < simulation_time:

        # Append the values for the current iteration
        t_vals.append(t)
        v1_vals.append(230*math.cos(377*t))
        v2_vals.append(v2)
        v3_vals.append(v3)
        v34_vals.append(v3-v4)
        v4_vals.append(v4)
        v5_vals.append(v5)
        i23_vals.append(i23)
        i30_vals.append(i30)
        i40_vals.append(i40)
        i50_vals.append(i50)

        # Voltage calculations
        # These voltage calculations are generated in MATLAB
        # by v_next[] = inv(GAA)*hA - inv(GAA)*GAB*vs
        if breaker_open_flag == False:
            # Closed switch solution (breaker closed)
```

```
            v2_next = (230.0*math.cos(377.0*t)*(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 4.0*C1*L1*R2*delta_t + 4.0*C2*L1*
                R2*delta_t))/(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t) - (1.0*R1*delta_t**3*(v5 + (2.0*L2*i50)/
                delta_t))/(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t) + (R1*delta_t**2*(2.0*L2 + R2*delta_t)
                *((2.0*C1*(v3 + (0.5*delta_t*i30)/C1))/delta_t + (2.0*C2*(v4 + (0.5*delta_t*
                i40)/C2))/delta_t + (0.5*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/L1))
                /(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*delta_t**3 +
                8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t
                **2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t +
                4.0*C2*L2*R1*delta_t) - (0.5*R1*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t)
                *(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R2*delta_t**3 + 8.0*C1*L1*L2 +
                8.0*C2*L1*L2 + 4.0*C1*L1*R2*delta_t + 4.0*C2*L1*R2*delta_t))/(L1*(2.0*L1*
                delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*delta_t**3 + 8.0*C1*L1*
                L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t**2 + 4.0*
                C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t + 4.0*C2*L2*
                R1*delta_t))
            v3_next = (230.0*delta_t**2*math.cos(377.0*t)*(2.0*L2 + R2*delta_t))/(2.0*L1*
                delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*delta_t**3 + 8.0*C1*L1*
                L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t**2 + 4.0*
                C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t + 4.0*C2*L2*
                R1*delta_t) - (1.0*delta_t**2*(2.0*L1 + R1*delta_t)*(v5 + (2.0*L2*i50)/
                delta_t))/(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t) + (delta_t*(2.0*L1 + R1*delta_t)*(2.0*L2
                + R2*delta_t)*((2.0*C1*(v3 + (0.5*delta_t*i30)/C1))/delta_t + (2.0*C2*(v4 +
                (0.5*delta_t*i40)/C2))/delta_t + (0.5*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/
                delta_t))/L1))/(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t) - (0.5*R1*delta_t**3*(2.0*L2 + R2*delta_t
                )*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/(L1*(2.0*L1*delta_t**2 + 2.0*L2*
                delta_t**2 + R1*delta_t**3 + R2*delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 +
                2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t +
                4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t + 4.0*C2*L2*R1*delta_t))
            v4_next = v3_next
            v5_next = (460.0*L2*delta_t**2*math.cos(377.0*t))/(2.0*L1*delta_t**2 + 2.0*L2*
                delta_t**2 + R1*delta_t**3 + R2*delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 +
                2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t +
                4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t + 4.0*C2*L2*R1*delta_t) - (1.0*
                delta_t*(v5 + (2.0*L2*i50)/delta_t)*(2.0*L1*delta_t + R1*delta_t**2 + R2*
                delta_t**2 + 4.0*C1*L1*R2 + 4.0*C2*L1*R2 + 2.0*C1*R1*R2*delta_t + 2.0*C2*R1*
                R2*delta_t))/(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t) + (2.0*L2*delta_t*(2.0*L1 + R1*delta_t)
                *((2.0*C1*(v3 + (0.5*delta_t*i30)/C1))/delta_t + (2.0*C2*(v4 + (0.5*delta_t*
                i40)/C2))/delta_t + (0.5*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/L1))
                /(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*delta_t**3 +
                8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*R1*R2*delta_t
                **2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*R2*delta_t +
                4.0*C2*L2*R1*delta_t) - (1.0*L2*R1*delta_t**3*(v2 - 1.0*v3 + (2.0*L1*i23)/
                delta_t))/(L1*(2.0*L1*delta_t**2 + 2.0*L2*delta_t**2 + R1*delta_t**3 + R2*
                delta_t**3 + 8.0*C1*L1*L2 + 8.0*C2*L1*L2 + 2.0*C1*R1*R2*delta_t**2 + 2.0*C2*
                R1*R2*delta_t**2 + 4.0*C1*L1*R2*delta_t + 4.0*C1*L2*R1*delta_t + 4.0*C2*L1*
                R2*delta_t + 4.0*C2*L2*R1*delta_t))
    else:
        # Open switch solution (breaker opened)
        v2_next = (230.0*math.cos(377.0*t)*(delta_t**2 + 4.0*C1*L1))/(delta_t**2 + 2.0*
            C1*R1*delta_t + 4.0*C1*L1) + (R1*delta_t**2*((2.0*C1*(v3 + (0.5*delta_t*i30)
            /C1))/delta_t + (0.5*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/L1))/(
            delta_t**2 + 2.0*C1*R1*delta_t + 4.0*C1*L1) - (0.5*R1*delta_t*(delta_t**2 +
```

```
            4.0*C1*L1)*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/(L1*(delta_t**2 + 2.0*C1*R1
            *delta_t + 4.0*C1*L1))
        v3_next = (230.0*delta_t**2*math.cos(377.0*t))/(delta_t**2 + 2.0*C1*R1*delta_t +
             4.0*C1*L1) + (delta_t*(2.0*L1 + R1*delta_t)*((2.0*C1*(v3 + (0.5*delta_t*i30
            )/C1))/delta_t + (0.5*delta_t*(v2 - 1.0*v3 + (2.0*L1*i23)/delta_t))/L1))/(
            delta_t**2 + 2.0*C1*R1*delta_t + 4.0*C1*L1) - (0.5*R1*delta_t**3*(v2 - 1.0*
            v3 + (2.0*L1*i23)/delta_t))/(L1*(delta_t**2 + 2.0*C1*R1*delta_t + 4.0*C1*L1)
            )
        v4_next = (2.0*C2*(2.0*L2 + R2*delta_t)*(v4 + (0.5*delta_t*i40)/C2))/(delta_t**2
             + 2.0*C2*R2*delta_t + 4.0*C2*L2) - (1.0*delta_t**2*(v5 + (2.0*L2*i50)/
            delta_t))/(delta_t**2 + 2.0*C2*R2*delta_t + 4.0*C2*L2)
        v5_next = (4.0*C2*L2*(v4 + (0.5*delta_t*i40)/C2))/(delta_t**2 + 2.0*C2*R2*
            delta_t + 4.0*C2*L2) - (1.0*delta_t*(v5 + (2.0*L2*i50)/delta_t)*(delta_t +
            2.0*C2*R2))/(delta_t**2 + 2.0*C2*R2*delta_t + 4.0*C2*L2)

    # Next currents
    i23_next = (delta_t*(v2_next - v3_next))/(2*L1) + (delta_t*(v2 - v3 + (2*L1*i23)/
        delta_t))/(2*L1)
    i30_next = (2*C1*v3_next)/delta_t - (2*C1*(v3 + (delta_t*i30)/(2*C1)))/delta_t
    i40_next = (2*C2*v4_next)/delta_t - (2*C2*(v4 + (delta_t*i40)/(2*C2)))/delta_t
    i50_next = (delta_t*(v5 + (2*L2*i50)/delta_t))/(2*L2) + (delta_t*v5_next)/(2*L2)

    # Check if breaker has opened
    if (t >= breaker_time) and (not breaker_open_flag):
        # Check for zero crossing
        i_left_prev = i23 - i30
        i_left_next = i23_next - i30_next
        if ((i_left_next * i_left_prev) < 0):
            # Sign has changed, set our flag to true
            zero_crossing_flag = True
            # Breaker has now officially opened
            breaker_open_flag = True
            # Record time breaker has opened
            breaker_open_time = t
            print("Snap!␣Breaker␣opened␣@␣t␣=␣%f" % breaker_open_time)

    # Next iteration
    i23 = i23_next
    i30 = i30_next
    i40 = i40_next
    i50 = i50_next
    v2 = v2_next
    v3 = v3_next
    v4 = v4_next
    v5 = v5_next

    # Go to next time step
    t = t + delta_t

# Plots for publication

legend_font_size = 6
# Superimposed
fig, ax = plt.subplots(1)
ax.plot(t_vals, v1_vals, label='$Simulation\\␣V_1$')
ax.plot(t_vals, v3_vals, label='$Simulation\\␣V_2$')
ax.plot(t_vals, v4_vals, label='$Simulation\\␣V_3$')
ax.plot(pscad_t_vals, pscad_v1_vals, linestyle='dashed', label='$PSCAD\\␣V_1$')
ax.plot(pscad_t_vals, pscad_v3_vals, linestyle='dashed', label='$PSCAD\\␣V_2$')
ax.plot(pscad_t_vals, pscad_v4_vals, linestyle='dashed', label='$PSCAD\\␣V_3$')
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time␣(s)', ylabel='voltage␣(kV)', title='Simulation␣vs.␣PSCAD␣Nodal␣
    Voltages')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(list(ax.get_xticks()) + [breaker_open_time])
ax.set_ylim(-400,400)
ax.set_xlim(0, 0.2)
```

```
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('nodal_voltage_comparison_plots.pgf')
fig.savefig('nodal_voltage_comparison_plots.png')

# 1. Nodal Voltage Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, v1_vals, label='$Simulation\\ V_1$')
ax.plot(t_vals, v3_vals, label='$Simulation\\ V_2$')
ax.plot(t_vals, v4_vals, label='$Simulation\\ V_3$')
ax.plot(pscad_t_vals, pscad_v1_vals, linestyle='dashed', label='$PSCAD\\ V_1$')
ax.plot(pscad_t_vals, pscad_v3_vals, linestyle='dashed', label='$PSCAD\\ V_2$')
ax.plot(pscad_t_vals, pscad_v4_vals, linestyle='dashed', label='$PSCAD\\ V_3$')
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation vs. PSCAD Nodal
    Voltages')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
ax.set_ylim(-400,400)
ax.set_xlim(0.06, 0.09)
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('nodal_voltage_comparison_plots_zoom.pgf')
fig.savefig('nodal_voltage_comparison_plots_zoom.png')

# 2. Short Circuit Current Comparison Plots
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, i50_vals, label='$Simulation\\ I_{40}$')
ax.plot(pscad_t_vals, pscad_ishort_vals, linestyle='dashed', label='$PSCAD\\ I_{40}$')
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time (s)', ylabel='current (kA)', title='Simulation vs. PSCAD Short
    Circuit Current')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(list(ax.get_xticks()) + [breaker_open_time])
ax.set_ylim(-1.5,1.5)
ax.set_xlim(0, 0.2)
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('short_circuit_current_comparison_plots.pgf')
fig.savefig('short_circuit_current_comparison_plots.png')

# 2. Short Circuit Current Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, i50_vals, label='$Simulation\\ I_{40}$')
ax.plot(pscad_t_vals, pscad_ishort_vals, linestyle='dashed', label='$PSCAD\\ I_{40}$')
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time (s)', ylabel='current (kA)', title='Simulation vs. PSCAD Short
    Circuit Current')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
ax.set_ylim(-1.5,1.5)
ax.set_xlim(0.06, 0.09)
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('short_circuit_current_comparison_plots_zoom.pgf')
fig.savefig('short_circuit_current_comparison_plots_zoom.png')

# 3. Breaker Voltage Comparison Plots
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, v34_vals, label='$Simulation\\ V_{23}$')
ax.plot(pscad_t_vals, pscad_v34_vals, linestyle='dashed', label='$PSCAD\\ V_{23}$')
```

```
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation vs. PSCAD Breaker
    Voltage')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(list(ax.get_xticks()) + [breaker_open_time])
ax.set_ylim(-400,400)
ax.set_xlim(0, 0.2)
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('breaker_voltage_comparison_plots.pgf')
fig.savefig('breaker_voltage_comparison_plots.png')

# 3. Breaker Voltage Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, v34_vals, label='$Simulation\\ V_{23}$')
ax.plot(pscad_t_vals, pscad_v34_vals, linestyle='dashed', label='$PSCAD\\ V_{23}$')
ax.axvline(x=breaker_open_time, linestyle='dashed', c='r')
ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation vs. PSCAD Breaker
    Voltage')
ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
ax.grid()
ax.set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
ax.set_ylim(-400,400)
ax.set_xlim(0.06, 0.09)
fig.set_size_inches(6.5,3.5)
fig.tight_layout()
fig.savefig('breaker_voltage_comparison_plots_zoom.pgf')
fig.savefig('breaker_voltage_comparison_plots_zoom.png')

# Side-by-side

# # 1. Nodal Voltage Comparison Plots
# # Simulation & PSCAD
# fig, ax = plt.subplots(2)
# ax[0].plot(t_vals, v1_vals, label='$Simulation\\ V_1$')
# ax[0].plot(t_vals, v3_vals, label='$Simulation\\ V_2$')
# ax[0].plot(t_vals, v4_vals, label='$Simulation\\ V_3$')
# ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
# ax[0].set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation Nodal Voltages')
# ax[0].legend(loc='best', fancybox=True, shadow=True)
# ax[0].grid()
# ax[0].set_xticks(list(ax[0].get_xticks()) + [breaker_open_time])
# ax[0].set_ylim(-400,400)
# ax[0].set_xlim(0, 0.2)
# # PSCAD
# ax[1].plot(pscad_t_vals, pscad_v1_vals, label='$PSCAD\\ V_1$')
# ax[1].plot(pscad_t_vals, pscad_v3_vals, label='$PSCAD\\ V_2$')
# ax[1].plot(pscad_t_vals, pscad_v4_vals, label='$PSCAD\\ V_3$')
# ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
# ax[1].set(xlabel='time (s)', ylabel='voltage (kV)', title='PSCAD Nodal Voltages')
# ax[1].legend(loc='best', fancybox=True, shadow=True)
# ax[1].grid()
# ax[1].set_xticks(list(ax[1].get_xticks()) + [pscad_breaker_open_time])
# ax[1].set_ylim(-400,400)
# ax[1].set_xlim(0, 0.2)
# fig.set_size_inches(6.5,8)
# fig.tight_layout()
# plt.show()
# fig.savefig('nodal_voltage_comparison_plots.pgf')
# fig.savefig('nodal_voltage_comparison_plots.png')

# # 1. Nodal Voltage Comparison Plots (Zoomed)
# # Simulation & PSCAD
# fig, ax = plt.subplots(2)
# ax[0].plot(t_vals, v1_vals, label='$Simulation\\ V_1$')
# ax[0].plot(t_vals, v3_vals, label='$Simulation\\ V_2$')
```

```
    # ax[0].plot(t_vals, v4_vals, label='$Simulation\\ V_3$')
    # ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
    # ax[0].set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation Nodal Voltages')
    # ax[0].legend(loc='best', fancybox=True, shadow=True)
    # ax[0].grid()
    # ax[0].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[0].set_ylim(-400,400)
    # ax[0].set_xlim(0.06, 0.09)
    # # PSCAD
    # ax[1].plot(pscad_t_vals, pscad_v1_vals, label='$PSCAD\\ V_1$')
    # ax[1].plot(pscad_t_vals, pscad_v3_vals, label='$PSCAD\\ V_2$')
    # ax[1].plot(pscad_t_vals, pscad_v4_vals, label='$PSCAD\\ V_3$')
    # ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
    # ax[1].set(xlabel='time (s)', ylabel='voltage (kV)', title='PSCAD Nodal Voltages')
    # ax[1].legend(loc='best', fancybox=True, shadow=True)
    # ax[1].grid()
    # ax[1].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[1].set_ylim(-400,400)
    # ax[1].set_xlim(0.06, 0.09)
    # fig.set_size_inches(6.5,8)
    # fig.tight_layout()
    # plt.show()
    # fig.savefig('nodal_voltage_comparison_plots_zoom.pgf')
    # fig.savefig('nodal_voltage_comparison_plots_zoom.png')

    # # 2. Short Circuit Current Comparison Plots
    # # Simulation & PSCAD
    # fig, ax = plt.subplots(2)
    # ax[0].plot(t_vals, i50_vals, label='$Simulation\\ I_{40}$')
    # ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
    # ax[0].set(xlabel='time (s)', ylabel='current (kA)', title='Simulation Short Circuit
        Current')
    # ax[0].legend(loc='best', fancybox=True, shadow=True)
    # ax[0].grid()
    # ax[0].set_xticks(list(ax[0].get_xticks()) + [breaker_open_time])
    # ax[0].set_ylim(-1.5,1.5)
    # ax[0].set_xlim(0, 0.2)
    # # PSCAD
    # ax[1].plot(pscad_t_vals, pscad_ishort_vals, label='$PSCAD\\ I_{40}$')
    # ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
    # ax[1].set(xlabel='time (s)', ylabel='current (kA)', title='PSCAD Short Circuit Current
        ')
    # ax[1].legend(loc='best', fancybox=True, shadow=True)
    # ax[1].grid()
    # ax[1].set_xticks(list(ax[1].get_xticks()) + [pscad_breaker_open_time])
    # ax[1].set_ylim(-1.5,1.5)
    # ax[1].set_xlim(0, 0.2)
    # fig.set_size_inches(6.5,8)
    # fig.tight_layout()
    # plt.show()
    # fig.savefig('short_circuit_current_comparison_plots.pgf')
    # fig.savefig('short_circuit_current_comparison_plots.png')

    # # 2. Short Circuit Current Comparison Plots (Zoomed)
    # # Simulation & PSCAD
    # fig, ax = plt.subplots(2)
    # ax[0].plot(t_vals, i50_vals, label='$Simulation\\ I_{40}$')
    # ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
    # ax[0].set(xlabel='time (s)', ylabel='current (kA)', title='Simulation Short Circuit
        Current')
    # ax[0].legend(loc='best', fancybox=True, shadow=True)
    # ax[0].grid()
    # ax[0].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[0].set_ylim(-1.5,1.5)
    # ax[0].set_xlim(0.06, 0.09)
    # # PSCAD
    # ax[1].plot(pscad_t_vals, pscad_ishort_vals, label='$PSCAD\\ I_{40}$')
    # ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
```

```
    # ax[1].set(xlabel='time (s)', ylabel='current (kA)', title='PSCAD Short Circuit Current
      ')
    # ax[1].legend(loc='best', fancybox=True, shadow=True)
    # ax[1].grid()
    # ax[1].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[1].set_ylim(-1.5,1.5)
    # ax[1].set_xlim(0.06, 0.09)
    # fig.set_size_inches(6.5,8)
    # fig.tight_layout()
    # plt.show()
    # fig.savefig('short_circuit_current_comparison_plots_zoom.pgf')
    # fig.savefig('short_circuit_current_comparison_plots_zoom.png')

    # # 3. Breaker Voltage Comparison Plots
    # # Simulation & PSCAD
    # fig, ax = plt.subplots(2)
    # ax[0].plot(t_vals, v34_vals, label='$Simulation\\ V_{23}$')
    # ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
    # ax[0].set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation Breaker Voltage
      ')
    # ax[0].legend(loc='best', fancybox=True, shadow=True)
    # ax[0].grid()
    # ax[0].set_xticks(list(ax[0].get_xticks()) + [breaker_open_time])
    # ax[0].set_ylim(-400,400)
    # ax[0].set_xlim(0, 0.2)
    # # PSCAD
    # ax[1].plot(pscad_t_vals, pscad_v34_vals, label='$PSCAD\\ V_{23}$')
    # ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
    # ax[1].set(xlabel='time (s)', ylabel='voltage (kV)', title='PSCAD Breaker Voltage')
    # ax[1].legend(loc='best', fancybox=True, shadow=True)
    # ax[1].grid()
    # ax[1].set_xticks(list(ax[1].get_xticks()) + [pscad_breaker_open_time])
    # ax[1].set_ylim(-400,400)
    # ax[1].set_xlim(0, 0.2)
    # fig.set_size_inches(6.5,8)
    # fig.tight_layout()
    # plt.show()
    # fig.savefig('breaker_voltage_comparison_plots.pgf')
    # fig.savefig('breaker_voltage_comparison_plots.png')

    # # 3. Breaker Voltage Comparison Plots (Zoomed)
    # # Simulation & PSCAD
    # fig, ax = plt.subplots(2)
    # ax[0].plot(t_vals, v34_vals, label='$Simulation\\ V_{23}$')
    # ax[0].axvline(x=breaker_open_time, linestyle='dashed', c='r')
    # ax[0].set(xlabel='time (s)', ylabel='voltage (kV)', title='Simulation Breaker Voltage
      ')
    # ax[0].legend(loc='best', fancybox=True, shadow=True)
    # ax[0].grid()
    # ax[0].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[0].set_ylim(-400,400)
    # ax[0].set_xlim(0.06, 0.09)
    # # PSCAD
    # ax[1].plot(pscad_t_vals, pscad_v34_vals, label='$PSCAD\\ V_{23}$')
    # ax[1].axvline(x=pscad_breaker_open_time, linestyle='dashed', c='r')
    # ax[1].set(xlabel='time (s)', ylabel='voltage (kV)', title='PSCAD Breaker Voltage')
    # ax[1].legend(loc='best', fancybox=True, shadow=True)
    # ax[1].grid()
    # ax[1].set_xticks(np.arange(0.06, 0.09, step=0.005).tolist())
    # ax[1].set_ylim(-400,400)
    # ax[1].set_xlim(0.06, 0.09)
    # fig.set_size_inches(6.5,8)
    # fig.tight_layout()
    # plt.show()
    # fig.savefig('breaker_voltage_comparison_plots_zoom.pgf')
    # fig.savefig('breaker_voltage_comparison_plots_zoom.png')

if __name__ == '__main__':
```

```
    # the following sets up the argument parser for the program
    parser = argparse.ArgumentParser(description='Assignment 2 solution generator')

    args = parser.parse_args()

    main(args)
```

5.2 MATLAB Code Listing

The following are the equations that were typed into MATLAB to generate the nodal voltage vector solution equations. MATLAB was instrumental in generating the equations the Python script would later use to calculate the iterative solution.

```
% Source voltage

syms vs delta_t t

vs = 230*cos(377*t)

% Circuit values

syms R1 R2 L1 L2 C1 C2

R1 = 3
L1 = 0.35
C1 = 10e-9
R2 = 50
L2 = 0.1
C2 = 600e-9
delta_t = 10e-6

% Discretized Resistances:
R23 = (2*L1)/delta_t
R50 = (2*L2)/delta_t
R30 = delta_t/(2*C1)
R40 = delta_t/(2*C2)

% History Equations:

syms eh23 eh50 eh30 eh40
syms h23 h50 h30 h40
syms v2 v3 v4 v5
syms v2_next v3_next v4_next v5_next
syms i23 i50 i30 i40
syms i23_next i50_next i30_next i40_next

eh23 = -(v2 - v3) - ((2*L1)/delta_t)*i23
eh50 = -v5 - ((2*L2)/delta_t)*i50
eh30 = v3 + (delta_t/(2*C1))*i30
eh40 = v4 + (delta_t/(2*C2))*i40

h23 = eh23/R23
h50 = eh50/R50
h30 = eh30/R30
h40 = eh40/R40

i23_next = (v2_next - v3_next)/R23 - h23
i30_next = v3_next/R30 - h30
i40_next = v4_next/R40 - h40
i50_next = v5_next/R50 - h50

% Switch Closed (v3 = v4):

syms g11 g12 g13 g14 g15
```

```
syms g21 g22 g23 g24 g25
syms g31 g32 g33 g34 g55
syms g41 g42 g43 g44 g45
syms g51 g52 g53 g54 g55
syms h1 h2 h3 h4 h5

h1 = 0
h2 = h23
h3 = -h23 + h30 + h40
h5 = h50

hA = [h2 ; h3 ; h5]
hB = h1
H = [hA ; hB]

g11 = 1/R1
g12 = -1/R1
g13 = 0
g15 = 0

g21 = -1/R1
g22 = 1/R1 + 1/R23
g23 = -1/R23
g25 = 0

g31 = 0
g32 = -1/R23
g33 = 1/R23 + 1/R30 + 1/R40 + 1/R2
g35 = -1/R2

g51 = 0
g52 = 0
g53 = -1/R2
g55 = 1/R50 + 1/R2

GAA = [g22 g23 g25 ; g32 g33 g35 ; g52 g53 g55]
GAB = [g21 ; g31 ; g51]
GBA = [g12 g13 g15]
GBB = [g11]
G = [GAA GAB ; GBA GBB]

% Nodal voltage solution vector:

v_next = inv(GAA)*hA - inv(GAA)*GAB*vs
vpa(subs(v_next))

% Switch Open (v3 != v4):

syms g11 g12 g13 g14 g15
syms g21 g22 g23 g24 g25
syms g31 g32 g33 g34 g55
syms g41 g42 g43 g44 g45
syms g51 g52 g53 g54 g55
syms h1 h2 h3 h4 h5

h1 = 0
h2 = h23
h3 = -h23 + h30
h4 = h40
h5 = h50

hA = [h2 ; h3 ; h4 ; h5]
hB = h1
H = [hA ; hB]

g11 = 1/R1
g12 = -1/R1
g13 = 0
```

```
g14 = 0
g15 = 0

g21 = -1/R1
g22 = 1/R1 + 1/R23
g23 = -1/R23
g24 = 0
g25 = 0

g31 = 0
g32 = -1/R23
g33 = 1/R23 + 1/R30
g34 = 0
g35 = 0

g41 = 0
g42 = 0
g43 = 0
g44 = 1/R40 + 1/R2
g45 = -1/R2

g51 = 0
g52 = 0
g53 = 0
g54 = -1/R2
g55 = 1/R50 + 1/R2

GAA = [g22 g23 g24 g25 ; g32 g33 g34 g35 ; g42 g43 g44 g45 ; g52 g53 g54 g55]
GAB = [g21 ; g31 ; g41 ; g51]
GBA = [g12 g13 g14 g15]
GBB = [g11]
G = [GAA GAB ; GBA GBB]

% Nodal voltage solution vector:

v_next = inv(GAA)*hA - inv(GAA)*GAB*vs
vpa(subs(v_next))

% Resonant frequency approximation

% f1 = 1/(2*pi*sqrt(L1*C1)) = 2.69 kHz => t1 = 372 us
% f2 = 1/(2*pi*sqrt(L2*C2)) = 650 Hz
% f3 = 1/(2*pi*sqrt((L1+L2)*(C1+C2))) = 303 Hz
% choose 10 us time step
```

5.3 Fortran Code Listing

The following is the code generated by PSCAD to simulate the circuit for this assignment.

```
!=======================================================================
! Generated by  : PSCAD v4.6.3.0
!
! Warning:  The content of this file is automatically generated.
!           Do not modify, as any changes made here will be lost!
!-----------------------------------------------------------------------
! Component     : Main
! Description   :
!-----------------------------------------------------------------------


!=======================================================================

      SUBROUTINE MainDyn()

!---------------------------------------
```

```
! Standard includes
!---------------------------------------

      INCLUDE 'nd.h'
      INCLUDE 'emtconst.h'
      INCLUDE 'emtstor.h'
      INCLUDE 's0.h'
      INCLUDE 's1.h'
      INCLUDE 's2.h'
      INCLUDE 's4.h'
      INCLUDE 'branches.h'
      INCLUDE 'pscadv3.h'
      INCLUDE 'fnames.h'
      INCLUDE 'radiolinks.h'
      INCLUDE 'matlab.h'
      INCLUDE 'rtconfig.h'

!---------------------------------------
! Function/Subroutine Declarations
!---------------------------------------


!---------------------------------------
! Variable Declarations
!---------------------------------------


! Subroutine Arguments

! Electrical Node Indices

! Control Signals
      INTEGER  BRK, BRK_STAT
      REAL     V1, V2, V3, Ishort, I30, I20, I12
      REAL     V34, V4, V5, Ibrk

! Internal Variables
      INTEGER  IVD1_1, IVD1_2
      REAL     RVD1_1, RVD1_2, RVD1_3, RVD1_4

! Indexing variables
      INTEGER ICALL_NO                          ! Module call num
      INTEGER ISTOI, ISTOF, IT_0                ! Storage Indices
      INTEGER IPGB                              ! Control/Monitoring
      INTEGER SS, INODE, IBRCH                  ! SS/Node/Branch/Xfmr


!---------------------------------------
! Local Indices
!---------------------------------------

! Dsdyn <-> Dsout transfer index storage

      NTXFR = NTXFR + 1

      TXFR(NTXFR,1) = NSTOL
      TXFR(NTXFR,2) = NSTOI
      TXFR(NTXFR,3) = NSTOF
      TXFR(NTXFR,4) = NSTOC

! Define electric network subsystem number

      SS        = NODE(NNODE+1)

! Increment and assign runtime configuration call indices

      ICALL_NO  = NCALL_NO
      NCALL_NO  = NCALL_NO + 1
```

```
! Increment global storage indices

      ISTOI     = NSTOI
      NSTOI     = NSTOI + 2
      ISTOF     = NSTOF
      NSTOF     = NSTOF + 11
      IPGB      = NPGB
      NPGB      = NPGB + 12
      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 9
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

!---------------------------------------
! Transfers from storage arrays
!---------------------------------------

      V1        = STOF(ISTOF + 1)
      V2        = STOF(ISTOF + 2)
      V3        = STOF(ISTOF + 3)
      Ishort    = STOF(ISTOF + 4)
      I30       = STOF(ISTOF + 5)
      I20       = STOF(ISTOF + 6)
      I12       = STOF(ISTOF + 7)
      V34       = STOF(ISTOF + 8)
      V4        = STOF(ISTOF + 9)
      V5        = STOF(ISTOF + 10)
      BRK       = STOI(ISTOI + 1)
      BRK_STAT  = STOI(ISTOI + 2)
      Ibrk      = STOF(ISTOF + 11)


!---------------------------------------
! Electrical Node Lookup
!---------------------------------------


!---------------------------------------
! Configuration of Models
!---------------------------------------

      IF ( TIMEZERO ) THEN
         FILENAME = 'Main.dta'
         CALL EMTDC_OPENFILE
         SECTION = 'DATADSD:'
         CALL EMTDC_GOTOSECTION
      ENDIF
!---------------------------------------
! Generated code from module definition
!---------------------------------------


! 10:[tbreakn] Timed Breaker Logic
! Timed breaker logic
      IF ( TIMEZERO ) THEN
         BRK = 0
      ELSE
         BRK = 0
         IF ( TIME .GE. 0.06667 ) BRK = (1-0)
      ENDIF

! 80:[breaker1] Single Phase Breaker 'BRK'
      IVD1_2 = NSTORI
      NSTORI = NSTORI + 1
      CALL E1PBRKR1_EXE(SS, (IBRCH+6),1.0e-06,1000000.0,0,NINT(1.0-REAL(&
```

```
     &BRK)))
      IVD1_1 = 2*E_BtoI(OPENBR( (IBRCH+6),SS))
      IF (FIRSTSTEP .OR. (STORI(IVD1_2) .NE. IVD1_1)) THEN
         CALL PSCAD_AGI2(ICALL_NO,189360772,IVD1_1,"BOpen")
      ENDIF
      STORI(IVD1_2) = 2*E_BtoI(OPENBR( (IBRCH+6),SS))
      BRK_STAT = IVD1_1

! 240:[pgb] Output Channel 'BRK_STAT'

      PGB(IPGB+11) = REAL(BRK_STAT)

! 1:[source_1] Single Phase Voltage Source Model 2 'vs'
!  Single Phase AC source: Type: Ideal
      RVD1_1 = RTCF(NRTCF)
      RVD1_2 = RTCF(NRTCF+1)
      RVD1_3 = RTCF(NRTCF+2)
      RVD1_4 = RTCF(NRTCF+3)
      NRTCF = NRTCF + 4
      CALL EMTDC_1PVSRC(SS, (IBRCH+1),RVD1_4,1,RVD1_1,RVD1_2,RVD1_3)

!---------------------------------------
! Feedbacks and transfers to storage
!---------------------------------------

      STOF(ISTOF + 1) = V1
      STOF(ISTOF + 2) = V2
      STOF(ISTOF + 3) = V3
      STOF(ISTOF + 4) = Ishort
      STOF(ISTOF + 5) = I30
      STOF(ISTOF + 6) = I20
      STOF(ISTOF + 7) = I12
      STOF(ISTOF + 8) = V34
      STOF(ISTOF + 9) = V4
      STOF(ISTOF + 10) = V5
      STOI(ISTOI + 1) = BRK
      STOI(ISTOI + 2) = BRK_STAT
      STOF(ISTOF + 11) = Ibrk


!---------------------------------------
! Transfer to Exports
!---------------------------------------

!---------------------------------------
! Close Model Data read
!---------------------------------------

      IF ( TIMEZERO ) CALL EMTDC_CLOSEFILE
      RETURN
      END

!=======================================================================

      SUBROUTINE MainOut()

!---------------------------------------
! Standard includes
!---------------------------------------

      INCLUDE 'nd.h'
      INCLUDE 'emtconst.h'
      INCLUDE 'emtstor.h'
      INCLUDE 's0.h'
      INCLUDE 's1.h'
      INCLUDE 's2.h'
      INCLUDE 's4.h'
      INCLUDE 'branches.h'
```

```
      INCLUDE 'pscadv3.h'
      INCLUDE 'fnames.h'
      INCLUDE 'radiolinks.h'
      INCLUDE 'matlab.h'
      INCLUDE 'rtconfig.h'

!---------------------------------------
! Function/Subroutine Declarations
!---------------------------------------

      REAL    EMTDC_VVDC    !
      REAL    VBRANCH       !

!---------------------------------------
! Variable Declarations
!---------------------------------------


! Electrical Node Indices
      INTEGER  NT_1, NT_2, NT_6, NT_7, NT_9
      INTEGER  NT_10

! Control Signals
      REAL     V1, V2, V3, Ishort, I30, I20, I12
      REAL     V34, V4, V5, Ibrk

! Internal Variables

! Indexing variables
      INTEGER ICALL_NO                          ! Module call num
      INTEGER ISTOL, ISTOI, ISTOF, ISTOC, IT_0  ! Storage Indices
      INTEGER IPGB                              ! Control/Monitoring
      INTEGER SS, INODE, IBRCH                  ! SS/Node/Branch/Xfmr


!---------------------------------------
! Local Indices
!---------------------------------------

! Dsdyn <-> Dsout transfer index storage

      NTXFR = NTXFR + 1

      ISTOL = TXFR(NTXFR,1)
      ISTOI = TXFR(NTXFR,2)
      ISTOF = TXFR(NTXFR,3)
      ISTOC = TXFR(NTXFR,4)

! Define electric network subsystem number

      SS        = NODE(NNODE+1)

! Increment and assign runtime configuration call indices

      ICALL_NO  = NCALL_NO
      NCALL_NO  = NCALL_NO + 1

! Increment global storage indices

      IPGB      = NPGB
      NPGB      = NPGB + 12
      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 9
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0
```

```
!---------------------------------------
! Transfers from storage arrays
!---------------------------------------

      V1        = STOF(ISTOF + 1)
      V2        = STOF(ISTOF + 2)
      V3        = STOF(ISTOF + 3)
      Ishort    = STOF(ISTOF + 4)
      I30       = STOF(ISTOF + 5)
      I20       = STOF(ISTOF + 6)
      I12       = STOF(ISTOF + 7)
      V34       = STOF(ISTOF + 8)
      V4        = STOF(ISTOF + 9)
      V5        = STOF(ISTOF + 10)
      Ibrk      = STOF(ISTOF + 11)


!---------------------------------------
! Electrical Node Lookup
!---------------------------------------

      NT_1  = NODE(INODE + 1)
      NT_2  = NODE(INODE + 2)
      NT_6  = NODE(INODE + 3)
      NT_7  = NODE(INODE + 4)
      NT_9  = NODE(INODE + 5)
      NT_10 = NODE(INODE + 6)

!---------------------------------------
! Configuration of Models
!---------------------------------------

      IF ( TIMEZERO ) THEN
         FILENAME = 'Main.dta'
         CALL EMTDC_OPENFILE
         SECTION = 'DATADSO:'
         CALL EMTDC_GOTOSECTION
      ENDIF
!---------------------------------------
! Generated code from module definition
!---------------------------------------


! 20:[voltmetergnd] Voltmeter (Line - Ground) 'V1'
      V1 = EMTDC_VVDC(SS, NT_1, 0)

! 30:[voltmetergnd] Voltmeter (Line - Ground) 'V2'
      V2 = EMTDC_VVDC(SS, NT_2, 0)

! 40:[ammeter] Current Meter 'I12'
      I12 = ( CBR((IBRCH+3), SS))

! 50:[voltmetergnd] Voltmeter (Line - Ground) 'V3'
      V3 = EMTDC_VVDC(SS, NT_9, 0)

! 60:[ammeter] Current Meter 'Ibrk'
      Ibrk = ( CBR((IBRCH+2), SS))

! 70:[voltmeter] Voltmeter (Line - Line) 'V34'
      V34 = EMTDC_VVDC(SS, NT_10, NT_6)

! 80:[breaker1] Single Phase Breaker 'BRK'
! Single phase breaker current
!

! 90:[voltmetergnd] Voltmeter (Line - Ground) 'V4'
      V4 = EMTDC_VVDC(SS, NT_6, 0)
```

```
! 100:[pgb] Output Channel 'V1'

      PGB(IPGB+1) = V1

! 110:[voltmetergnd] Voltmeter (Line - Ground) 'V5'
      V5 = EMTDC_VVDC(SS, NT_7, 0)

! 120:[ammeter] Current Meter 'I20'
      I20 = (-CBR((IBRCH+4), SS))

! 130:[ammeter] Current Meter 'I30'
      I30 = (-CBR((IBRCH+5), SS))

! 140:[pgb] Output Channel 'V2'

      PGB(IPGB+2) = V2

! 150:[ammeter] Current Meter 'Ishort'
      Ishort = ( CBR((IBRCH+7), SS))

! 160:[pgb] Output Channel 'V3'

      PGB(IPGB+3) = V3

! 170:[pgb] Output Channel 'V34'

      PGB(IPGB+4) = V34

! 180:[pgb] Output Channel 'V4'

      PGB(IPGB+5) = V4

! 190:[pgb] Output Channel 'V5'

      PGB(IPGB+6) = V5

! 200:[pgb] Output Channel 'Ishort'

      PGB(IPGB+7) = Ishort

! 210:[pgb] Output Channel 'I30'

      PGB(IPGB+8) = I30

! 220:[pgb] Output Channel 'I20'

      PGB(IPGB+9) = I20

! 230:[pgb] Output Channel 'I12'

      PGB(IPGB+10) = I12

! 250:[pgb] Output Channel 'Ibrk'

      PGB(IPGB+12) = Ibrk

!---------------------------------------
! Feedbacks and transfers to storage
!---------------------------------------

      STOF(ISTOF + 1) = V1
      STOF(ISTOF + 2) = V2
      STOF(ISTOF + 3) = V3
      STOF(ISTOF + 4) = Ishort
      STOF(ISTOF + 5) = I30
      STOF(ISTOF + 6) = I20
      STOF(ISTOF + 7) = I12
      STOF(ISTOF + 8) = V34
```

```
      STOF(ISTOF + 9) = V4
      STOF(ISTOF + 10) = V5
      STOF(ISTOF + 11) = Ibrk


!---------------------------------------
! Close Model Data read
!---------------------------------------

      IF ( TIMEZERO ) CALL EMTDC_CLOSEFILE
      RETURN
      END

!=======================================================================

      SUBROUTINE MainDyn_Begin()

!---------------------------------------
! Standard includes
!---------------------------------------

      INCLUDE 'nd.h'
      INCLUDE 'emtconst.h'
      INCLUDE 's0.h'
      INCLUDE 's1.h'
      INCLUDE 's4.h'
      INCLUDE 'branches.h'
      INCLUDE 'pscadv3.h'
      INCLUDE 'radiolinks.h'
      INCLUDE 'rtconfig.h'

!---------------------------------------
! Function/Subroutine Declarations
!---------------------------------------


!---------------------------------------
! Variable Declarations
!---------------------------------------


! Subroutine Arguments

! Electrical Node Indices

! Control Signals

! Internal Variables

! Indexing variables
      INTEGER ICALL_NO                          ! Module call num
      INTEGER IT_0                              ! Storage Indices
      INTEGER SS, INODE, IBRCH                  ! SS/Node/Branch/Xfmr


!---------------------------------------
! Local Indices
!---------------------------------------


! Define electric network subsystem number

      SS        = NODE(NNODE+1)

! Increment and assign runtime configuration call indices

      ICALL_NO  = NCALL_NO
      NCALL_NO  = NCALL_NO + 1
```

```
! Increment global storage indices

      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 9
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

!---------------------------------------
! Electrical Node Lookup
!---------------------------------------


!---------------------------------------
! Generated code from module definition
!---------------------------------------


! 80:[breaker1] Single Phase Breaker 'BRK'
      CALL COMPONENT_ID(ICALL_NO,189360772)
      CALL E1PBRKR1_CFG(1.0e-06,1000000.0,0.0)

! 240:[pgb] Output Channel 'BRK_STAT'

! 1:[source_1] Single Phase Voltage Source Model 2 'vs'
      CALL E_1PVSRC_CFG(1,0,6,163.0,60.0,90.0,0.0,0.0,0.0,0.0,0.0,0.0)

      RETURN
      END

!=======================================================================

      SUBROUTINE MainOut_Begin()

!---------------------------------------
! Standard includes
!---------------------------------------

      INCLUDE 'nd.h'
      INCLUDE 'emtconst.h'
      INCLUDE 's0.h'
      INCLUDE 's1.h'
      INCLUDE 's4.h'
      INCLUDE 'branches.h'
      INCLUDE 'pscadv3.h'
      INCLUDE 'radiolinks.h'
      INCLUDE 'rtconfig.h'

!---------------------------------------
! Function/Subroutine Declarations
!---------------------------------------


!---------------------------------------
! Variable Declarations
!---------------------------------------


! Subroutine Arguments

! Electrical Node Indices
      INTEGER  NT_1, NT_2, NT_6, NT_7, NT_9
      INTEGER  NT_10

! Control Signals
```

```
! Internal Variables

! Indexing variables
      INTEGER ICALL_NO                          ! Module call num
      INTEGER IT_0                              ! Storage Indices
      INTEGER SS, INODE, IBRCH                  ! SS/Node/Branch/Xfmr


!---------------------------------------
! Local Indices
!---------------------------------------


! Define electric network subsystem number

      SS        = NODE(NNODE+1)

! Increment and assign runtime configuration call indices

      ICALL_NO  = NCALL_NO
      NCALL_NO  = NCALL_NO + 1

! Increment global storage indices

      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 9
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

!---------------------------------------
! Electrical Node Lookup
!---------------------------------------

      NT_1  = NODE(INODE + 1)
      NT_2  = NODE(INODE + 2)
      NT_6  = NODE(INODE + 3)
      NT_7  = NODE(INODE + 4)
      NT_9  = NODE(INODE + 5)
      NT_10 = NODE(INODE + 6)

!---------------------------------------
! Generated code from module definition
!---------------------------------------


! 100:[pgb] Output Channel 'V1'

! 140:[pgb] Output Channel 'V2'

! 160:[pgb] Output Channel 'V3'

! 170:[pgb] Output Channel 'V34'

! 180:[pgb] Output Channel 'V4'

! 190:[pgb] Output Channel 'V5'

! 200:[pgb] Output Channel 'Ishort'

! 210:[pgb] Output Channel 'I30'

! 220:[pgb] Output Channel 'I20'

! 230:[pgb] Output Channel 'I12'

! 250:[pgb] Output Channel 'Ibrk'
```

```
    RETURN
    END
```