Transient Recovery Voltage (TRV) in Circuit Breakers with the CP-Line Model

Assignment 3

Due: 2021/02/26

1 Introduction

This assignment investigates the transient recovery voltage in circuit breakers of a shorted transmission line using a the constant parameter transmission line model. Specifically, the assignment implements Dommel/Bergeron transmission line model, a version of the constant parameter transmission line model which accounts for line resistances. This is to compare with the previous assignment's lumped circuit transmission line model. Just as in the previous assignment, this scenario has a short circuit occur 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 which are produced from both the custom computer program as well as 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:

$$\begin{array}{ccc} R_1 = 3\,\Omega & L_1 = 350\,mH & C_1 = 10\,nF \\ R_{line} = 0.5\,\Omega/km & L_{line} = 1\,mH/km & C_{line} = 12\,nF/km \\ & linelength = 100km & \\ & \Delta t = 10\mu s & \end{array}$$

The same Δt value is used from the previous assignment to allow for meaningful comparisons between the two. These values are substituted into the hand-implemented simulation into the solution equations and the PSCAD schematic in their respective locations, as the following sections will detail.

2.1 Hand-Implemented Simulation

As with the previous assignment, the nodes were numbered V_1 through V_5. These are translated back into the assignment's original node names to more clearly answer the assignment questions. Then, the L

and C components 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. The transmission line was then represented using the Dommel/Bergeron transmission model equivalent. Figure 1 shows the breakdown of these steps into the final model consisting of resistances and current sources.

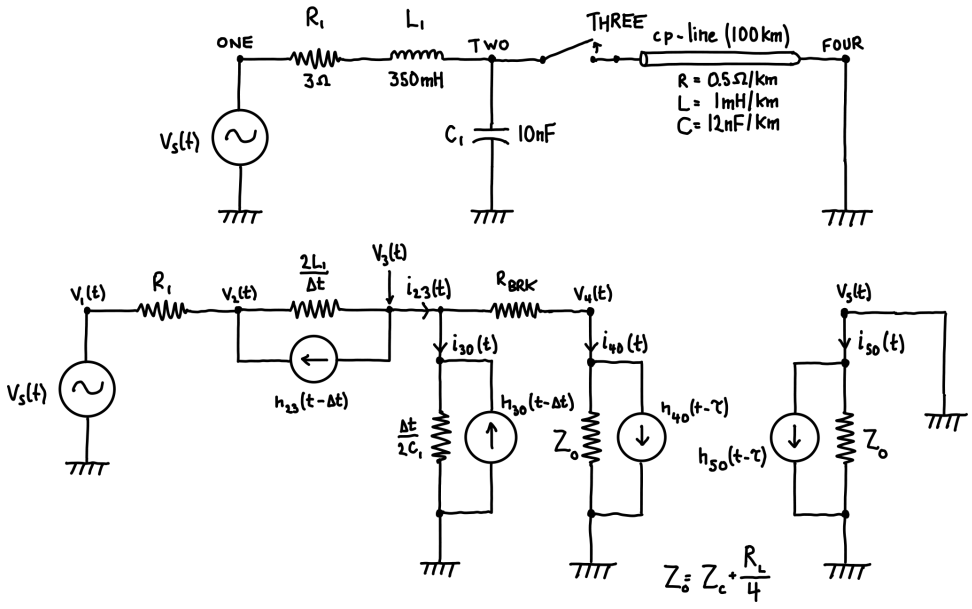

Figure 1: The original circuit and the model used to simulate it using the Dommel/Bergeron transmission line model.

The Dommel/Bergeron transmission model's resistances Z_o are composed of the characteristic impedance Z_c and a quarter of the total line resistance as follows:

$$Z_o = Z_c \cdot \frac{R_{line} \cdot linelength}{4} \tag{1}$$

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

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

$$h_{30}(t) = \frac{2\,C_1\,v_3(t-\Delta t) + \Delta t\,i_{30}(t-\Delta t)}{\Delta t} \tag{3}$$

$$\tag{4}$$

However, the transmission line history current sources are slightly different, yielding the following equations. These are unique the the Dommel/Bergeron model, which augment the regular CP-Line equations by taking into account the line resistance. The equations were found in PSCAD's documentation, which outline how they implement the Dommel/Bergeron transmission line model in its program.

$$h_{40}(t) = \frac{1+H}{2}\left[-\frac{1}{Z_o}V_5(t-\tau) - H \cdot i_{50}(t-\tau)\right] + \frac{1-H}{2}\left[-\frac{1}{Z_o}V_4(t-\tau) - H \cdot i_{40}(t-\tau)\right] \quad (5)$$

$$h_{50}(t) = \frac{1+H}{2}\left[-\frac{1}{Z_o}V_4(t-\tau) - H \cdot i_{40}(t-\tau)\right] + \frac{1-H}{2}\left[-\frac{1}{Z_o}V_5(t-\tau) - H \cdot i_{50}(t-\tau)\right] \quad (6)$$

where:

$$H = \frac{Z_o + \frac{R_{line}}{4}}{Z_o - \frac{R_{line}}{4}} \quad (7)$$

What is important to note here is the fact that these equations rely on a time τ in the past instead of Δt. τ is the time it takes for the signal to propagate the length of the line and is calculated using the transmission line's velocity of propagation, a, as follows:

$$a = \frac{1}{\sqrt{L_{line} \cdot C_{line}}}\,(m/s) \quad (8)$$

$$\tau = \frac{linelength}{a}\,(s) \quad (9)$$

Since τ isn't a multiple of Δt, it was necessary to interpolate the voltage and current values in the hand-implemented solution. In order to do this, a simple linear interpolation was performed at every step of the program. If history wasn't available, such as the very beginning of the simulation, then a value of 0 is returned from the interpolation function. The linear interpolation was performed as follows:

$$interpolated = y_0 + (t - t_0)\frac{y_1 - y_0}{t_1 - t_0} \quad (10)$$

Here t is the time-stamp we want to interpolate at ($t - \tau$ at each time slice in this case) and t_0, t_1 are two times that are above and below $t - \tau$ that we currently have in our history with y_0, y_1 containing the voltage or current value at those times in our history.

One thing to keep in mind with this hand-implemented simulation is the fact that the final nodal voltage is always 0 due to the transmission line being shorted. Therefore, the matrix does not need to include this node in the solution. By performing a source transformation on the right hand side of the transmission line, as shown in Figure 2, we get a clearer picture of what the short circuit current should be, the opposite of the i_{50} current we will be calculating at each time step.

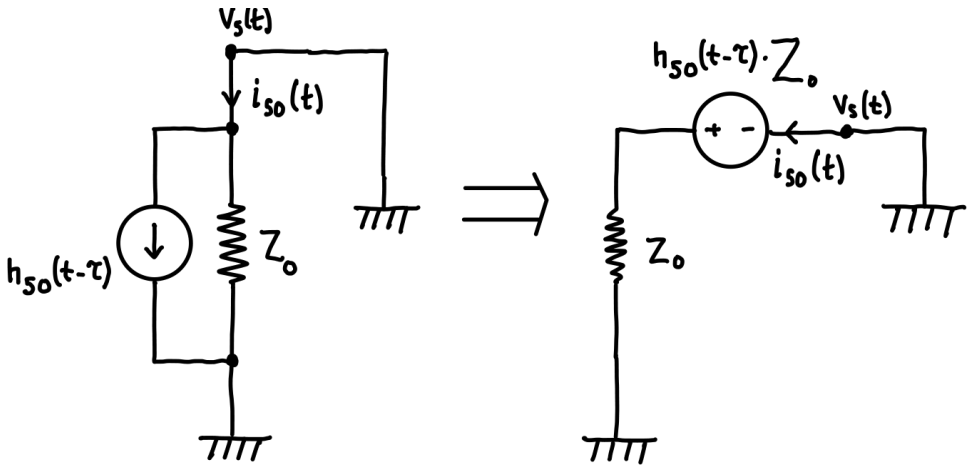

Figure 2: The source transformation performed on the right-hand side of the transmission line model.

With the relationships outlined above, we can now form equations for the branch currents, which are required values for calculating the next nodal voltage values:

$$i_{23}(t) = \frac{v_{23}(t)}{R_{23}} - h_{23}(t) \tag{11}$$

$$i_{30}(t) = \frac{v_{30}(t)}{R_{30}} - h_{30}(t) \tag{12}$$

$$i_{40}(t) = \frac{v_{40}(t)}{Z_o} - h_{40}(t) \tag{13}$$

$$i_{50}(t) = -h_{50}(t) \tag{14}$$

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. The last branch only has a short circuited current source, so there is no contribution from a resistance.

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 (note that in the previous assignment, $\mathbf{G_{AB}}$ in this equation was incorrectly labeled as $\mathbf{G_{AA}}$! This was only incorrect in the report and not the code).

$$[\mathbf{V_A(t)}] = [\mathbf{G_{AA}}]^{-1}[\mathbf{h_A(t)}] - [\mathbf{G_{AA}}]^{-1}[\mathbf{G_{AB}}][\mathbf{V_B(t)}] \tag{15}$$

Instead of creating two matrices like in the previous assignment, it was decided to use a breaker resistance instead, R_{brk}. This was done to simplify the hand-implemented simulation and to more closely emulate how PSCAD modeled the breaker. When the breaker is closed, the resistance R_{brk} is $1\,n\Omega$ (very low, almost shorted), and when the breaker is opened, the resistance R_{brk} is $1\,M\Omega$ (very high, almost open). This simplifies things by only requiring single conductance and current matrices for the entire analysis with a variable value for R_{brk}.

$$[\mathbf{G}] = \left[\begin{array}{c:c} \mathbf{G_{AA}} & \mathbf{G_{AB}} \\ \hdashline \mathbf{G_{BA}} & \mathbf{G_{BB}} \end{array}\right] = \left[\begin{array}{ccc:c} \frac{\Delta_t}{2\,L_1} + \frac{1}{R_1} & -\frac{\Delta_t}{2\,L_1} & 0 & -\frac{1}{R_1} \\ -\frac{\Delta_t}{2\,L_1} & \frac{2\,C_1}{\Delta_t} + \frac{\Delta_t}{2\,L_1} + \frac{1}{R_{brk}} & -\frac{1}{R_{brk}} & 0 \\ 0 & -\frac{1}{R_{brk}} & \frac{1}{\frac{R_{line} \cdot linelength}{4} + Z_c} + \frac{1}{R_{brk}} & 0 \\ \hdashline -\frac{1}{R_1} & 0 & 0 & \frac{1}{R_1} \end{array}\right] \tag{16}$$

$$[\mathbf{H}] = \left[\begin{array}{c} \mathbf{H_A} \\ \hdashline \mathbf{H_B} \end{array}\right] = \left[\begin{array}{c} h_{23}(t) \\ h_{30}(t) - h_{23}(t) \\ -h_{40}(t) \\ \hdashline 0 \end{array}\right] \tag{17}$$

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. For each time step, first, an interpolation is performed to obtain values that are not in the history table. Specifically, values for V_4, i_{40}, and i_{50} at $t - \tau$. The Python script then calculates the next nodal voltage value using all of the up to date values for the current time step. 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 breaker resistance is made very large until the end of the simulation. This is repeated for every step $t = t + \Delta t$ until the $200ms$ simulation time is reached. These results are plotted alongside the equivalent PSCAD solution for comparison using the Python matplotlib plotting library.

2.2 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 3. 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 Dommel/Bergeron model was placed using PSCAD's component wizard, with the values shown in Figure 4. The breaker's resistances were set to $1\,n\Omega$ and $1\,M\Omega$ for closed and open states to match the hand implementation and configured to not chop current.

We tell the transmission model to interpolate travel time (as we have done in the hand-simulated portion of this assignment). We also specify the frequency of the line to be $60Hz$ and its length to be $100km$ with a single conductor. The line resistance is input in units of Ω/m, the travel time is the inverse of the velocity of propagation in units of s/m, and the surge impedance is another word for the characteristic impedance of the line, Z_c. 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.

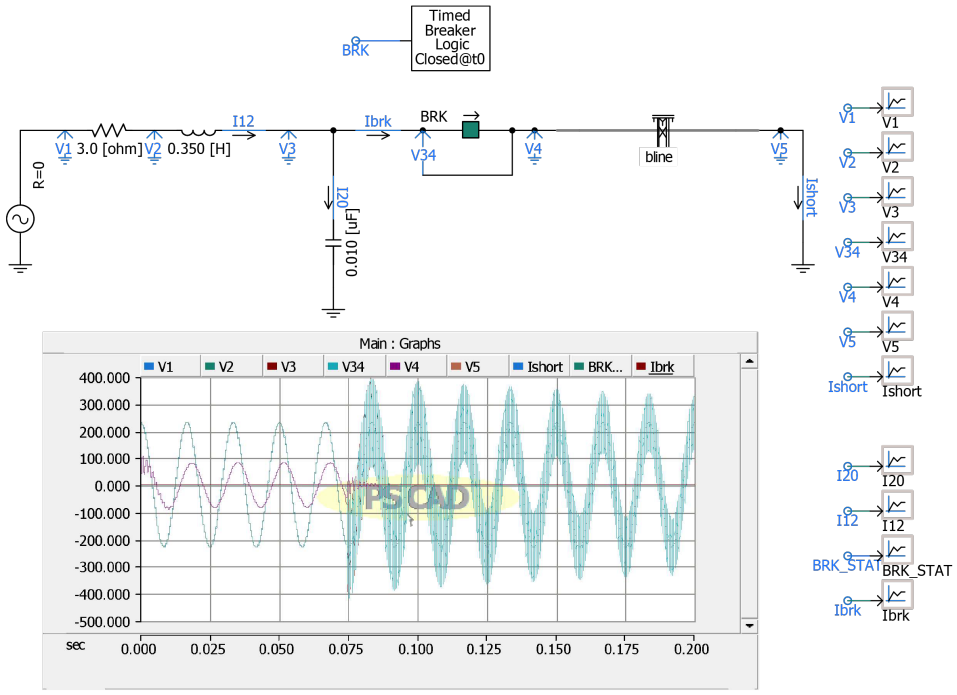

Figure 3: PSCAD schematic of the transmission line.

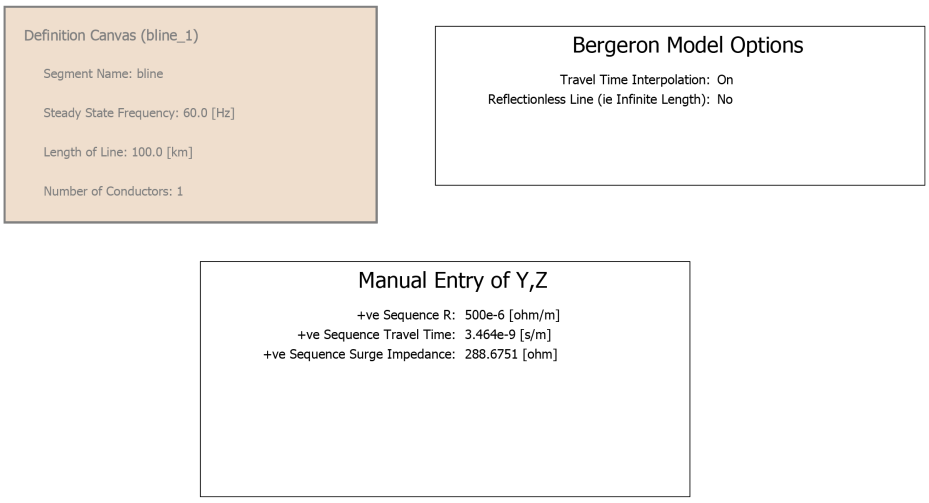

Figure 4: PSCAD parameters for Dommel/Bergeron transmission model.

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. Plots named simulation refer to the hand-modeled circuit and those named PSCAD refer to the PSCAD modeled circuit.

a) Plots of V_{one}, V_{two}, V_{three} for the simulation and PSCAD (Figures 5, 6, and 7).

b) Plots of $I_{four,ground}$ which represents the short-circuit current at the end of the transmission line (Figures 10, 11, and 12).

c) Plots of $V_{two,three}$ which represents the voltage across the breaker (Figures 8 and 9).

Additionally, the assignment also requests that the PSCAD results from the second assignment and the third assignment (this one) be compared as well. To make the comparison, the following PSCAD plots are used from assignments 2 and 3.

a) Plots of V_{one}, V_{two}, V_{three} for the simulation and PSCAD (Figure 13).

b) Plots of $I_{four,ground}$ which represents the short-circuit current at the end of the transmission line (Figures 14, 15, and 16).

c) Plots of $V_{two,three}$ which represents the voltage across the breaker (Figures 17 and 18).

A full discussion of the results can be found in the Conclusion section.

3.1 Hand-Implemented Simulation vs. PSCAD

In this section, we compare the hand-implemented simulation ($Simulation$) with the PSCAD results ($PSCAD$).

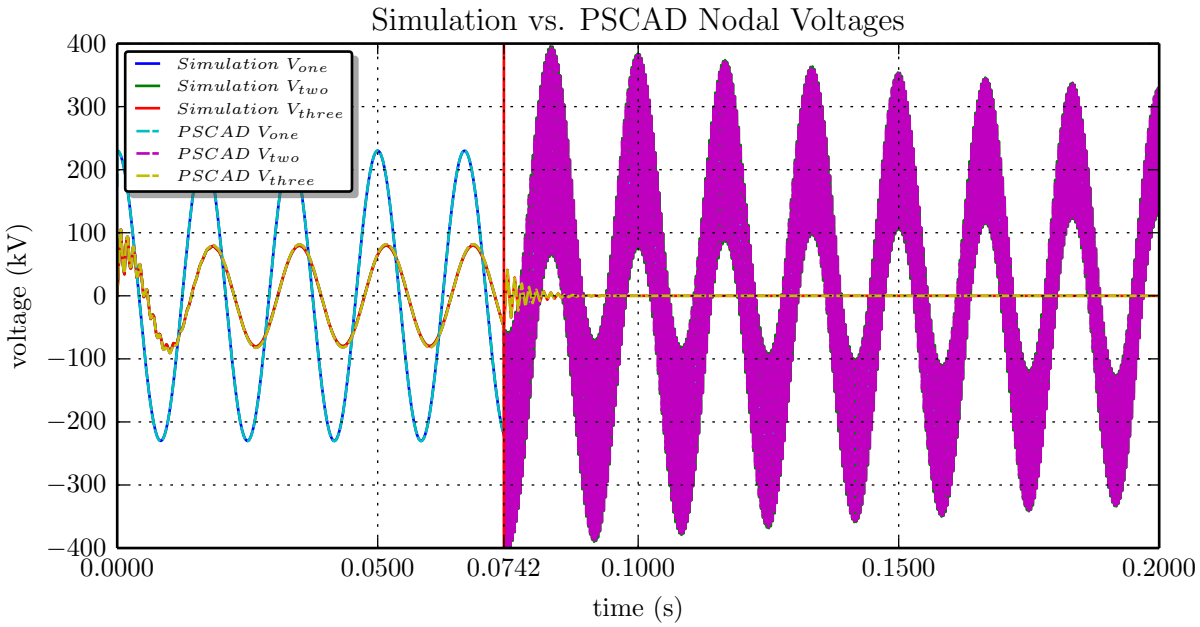

Figure 5: Plots of V_{one}, V_{two}, and V_{three} nodal voltages

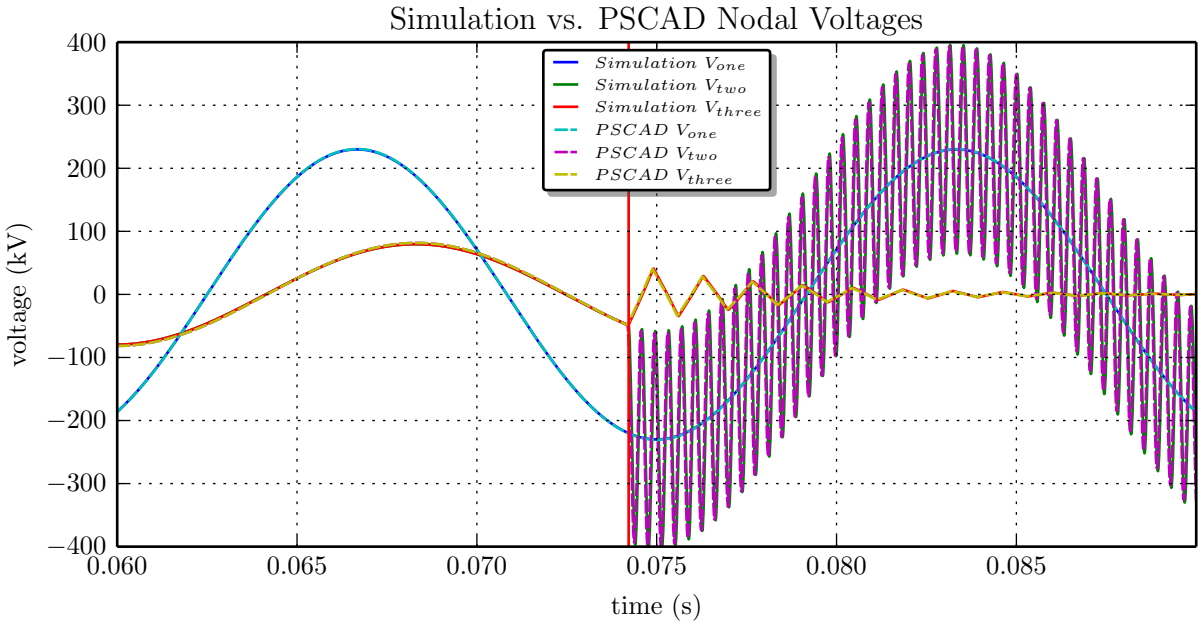

Figure 6: Plots of V_{one}, V_{two}, and V_{three} nodal voltages (zoomed at breaker opening)

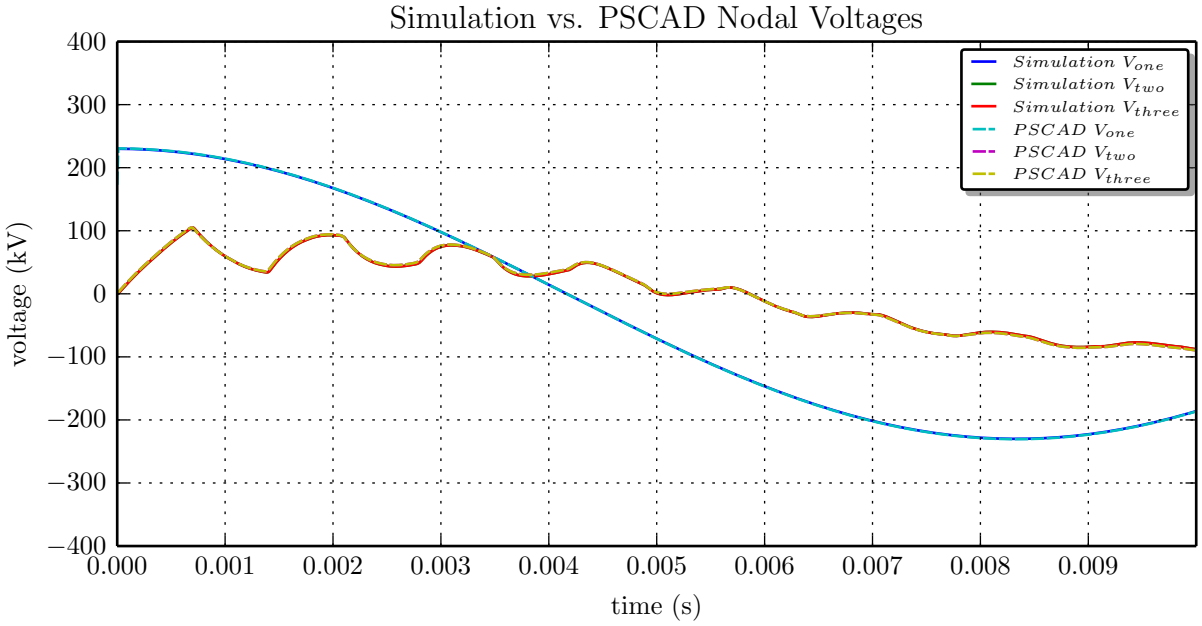

Figure 7: Plots of V_{one}, V_{two}, and V_{three} nodal voltages (zoomed at simulation start)

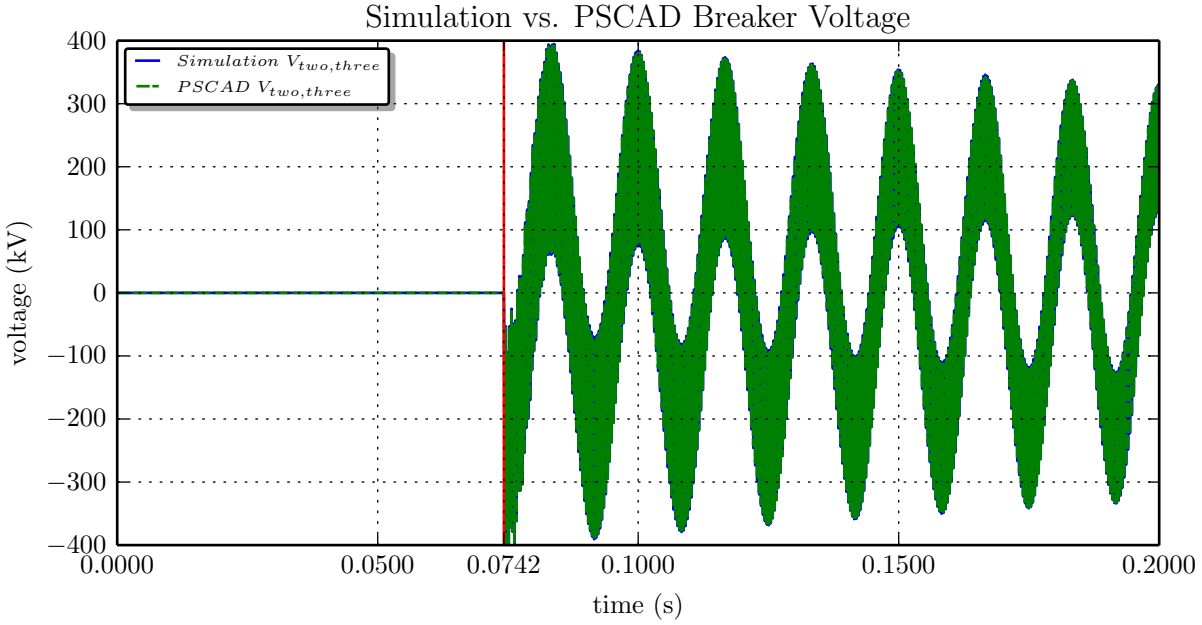

Figure 8: Plots of $V_{two,three}$ breaker voltage

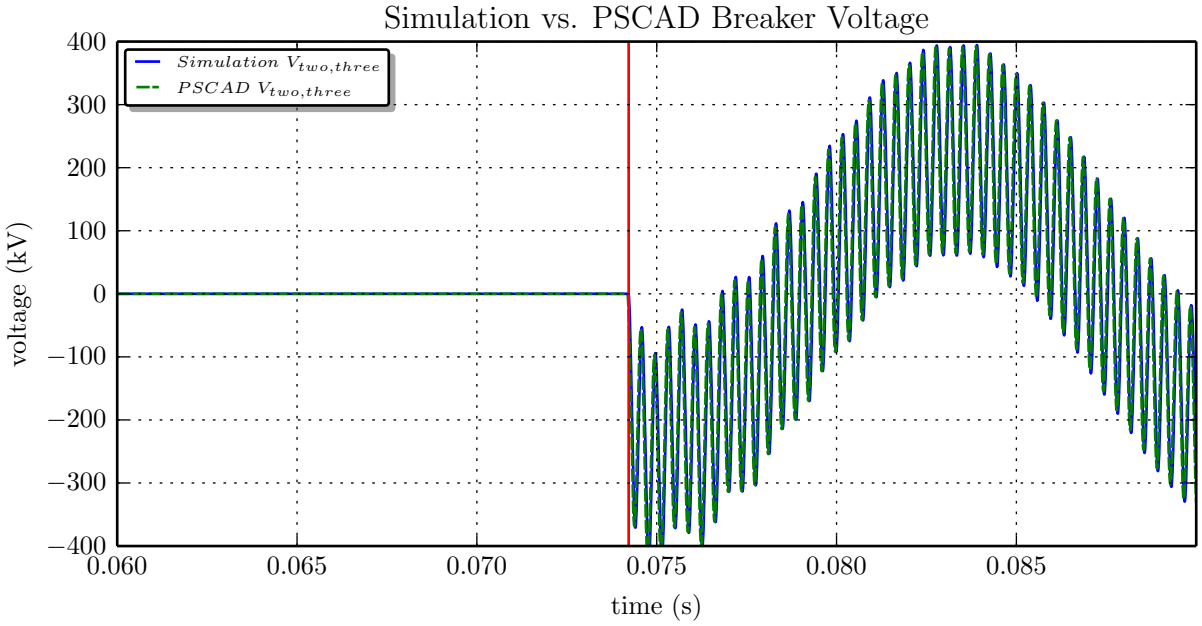

Figure 9: Zoomed plots of $V_{two,three}$ breaker voltage (zoomed at breaker opening)

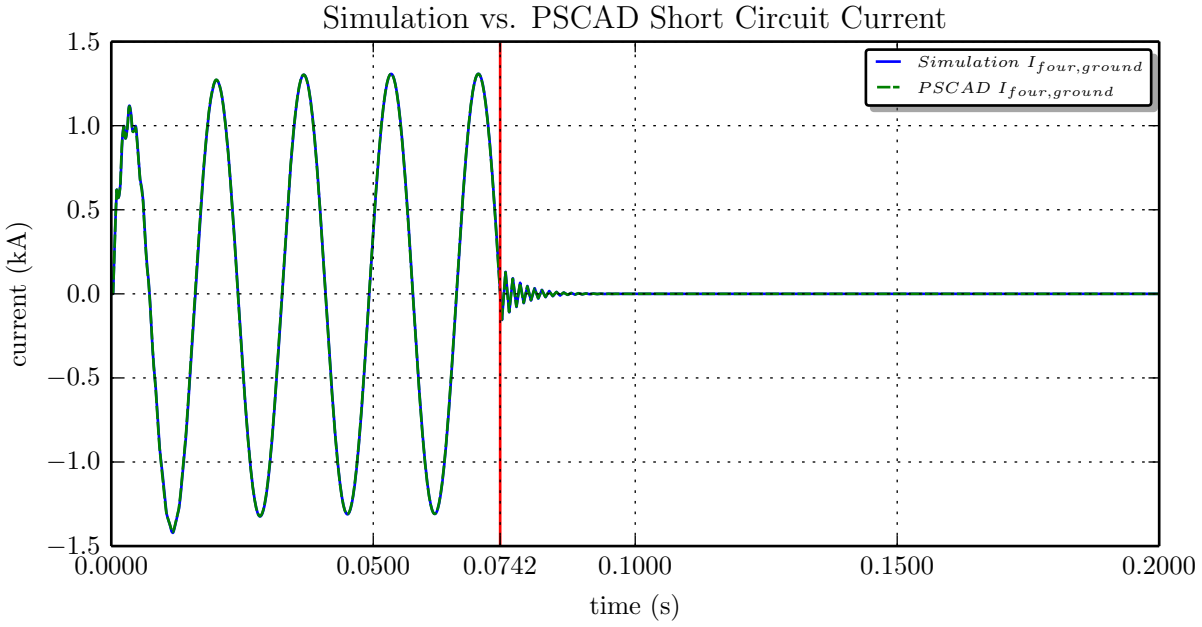

Figure 10: Plots of $I_{four,ground}$ short circuit current

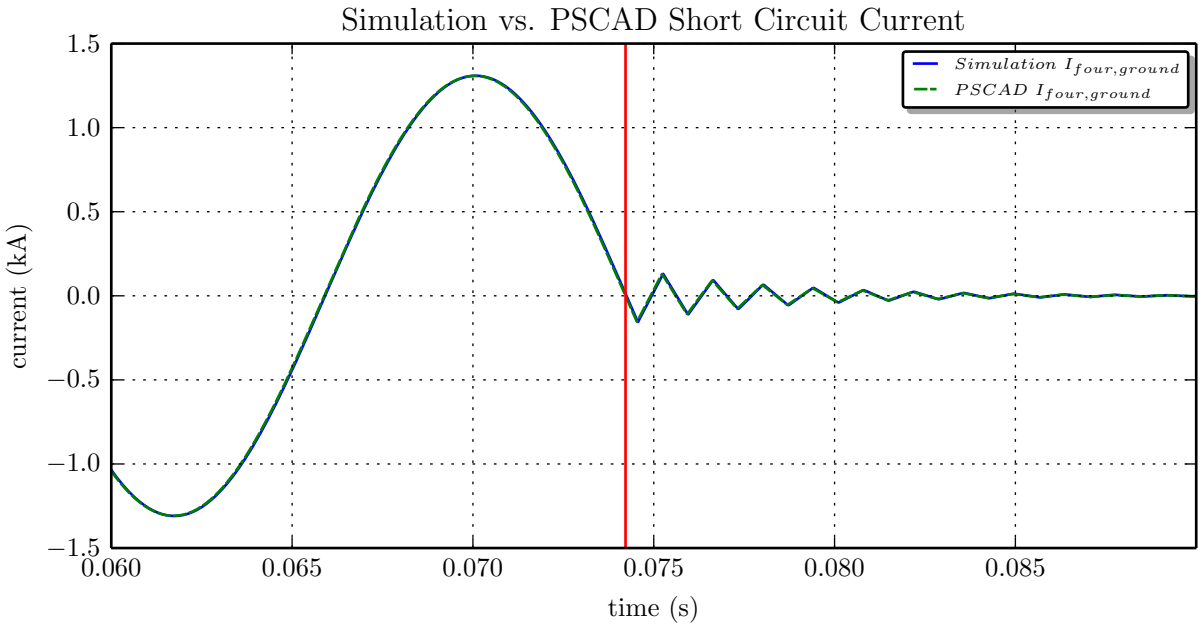

Figure 11: Zoomed plots of $I_{four,ground}$ short circuit current (zoomed at breaker opening)

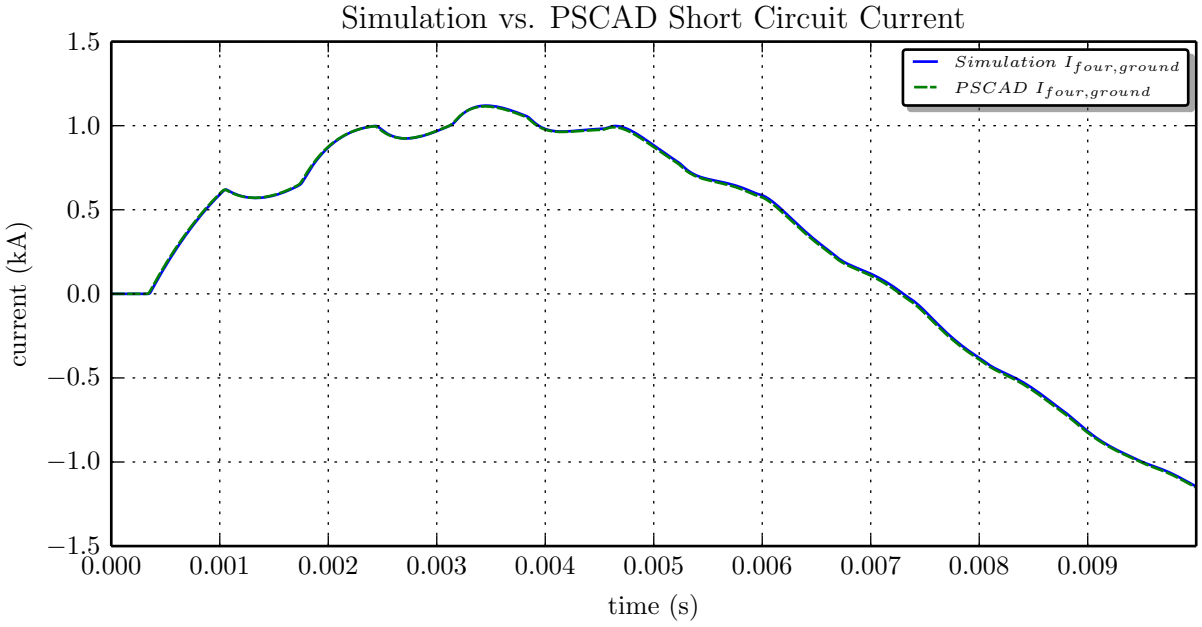

Figure 12: Zoomed plots of $I_{four,ground}$ short circuit current (zoomed at simulation start)

3.2 Assignment 2 PSCAD vs. Assignment 3 PSCAD

In this section, we compare the second assignment's PSCAD results ($PSCAD_2$) with the third assignment's PSCAD results ($PSCAD_3$).

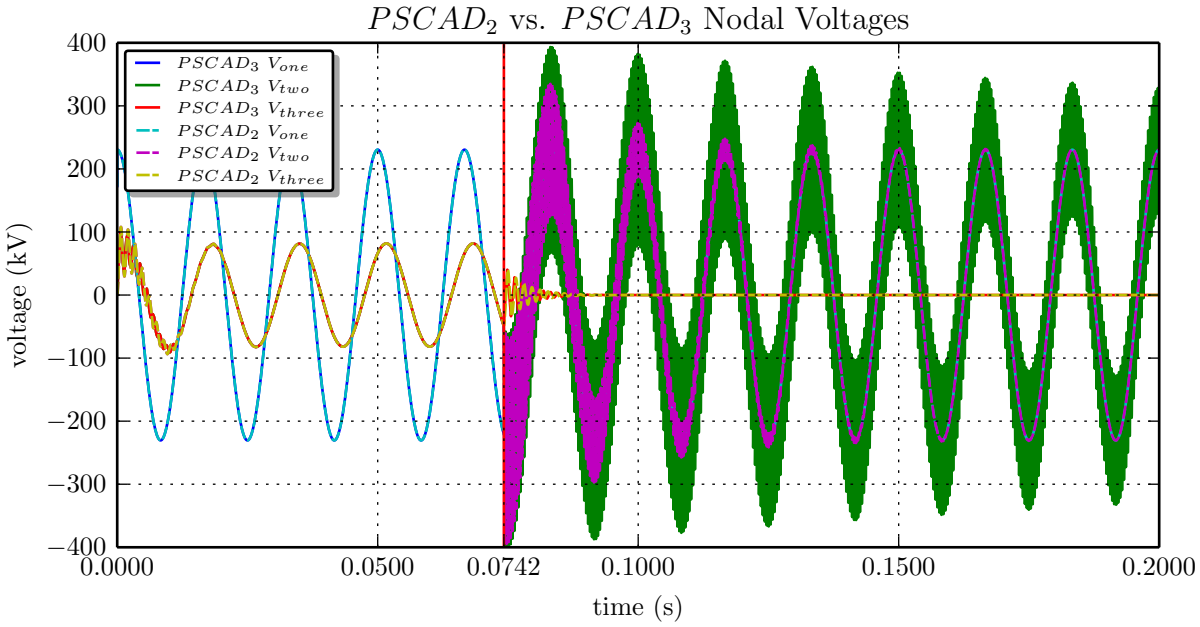

Figure 13: Plots of V_{one}, V_{two}, and V_{three} nodal voltages

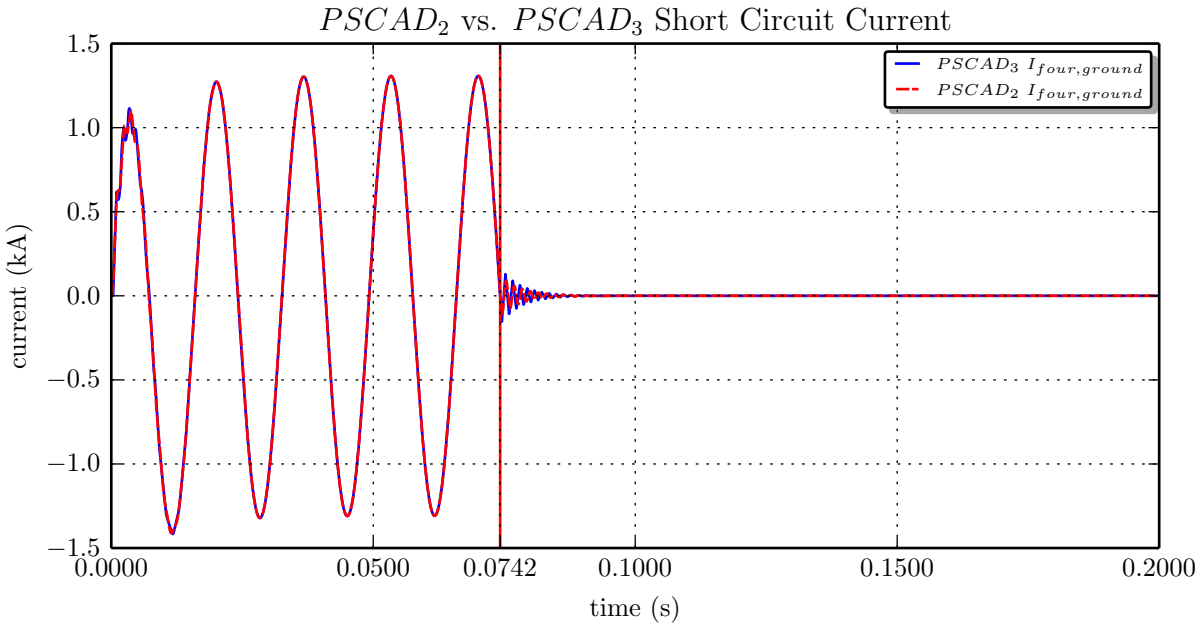

Figure 14: Plots of $I_{four,ground}$ short circuit current

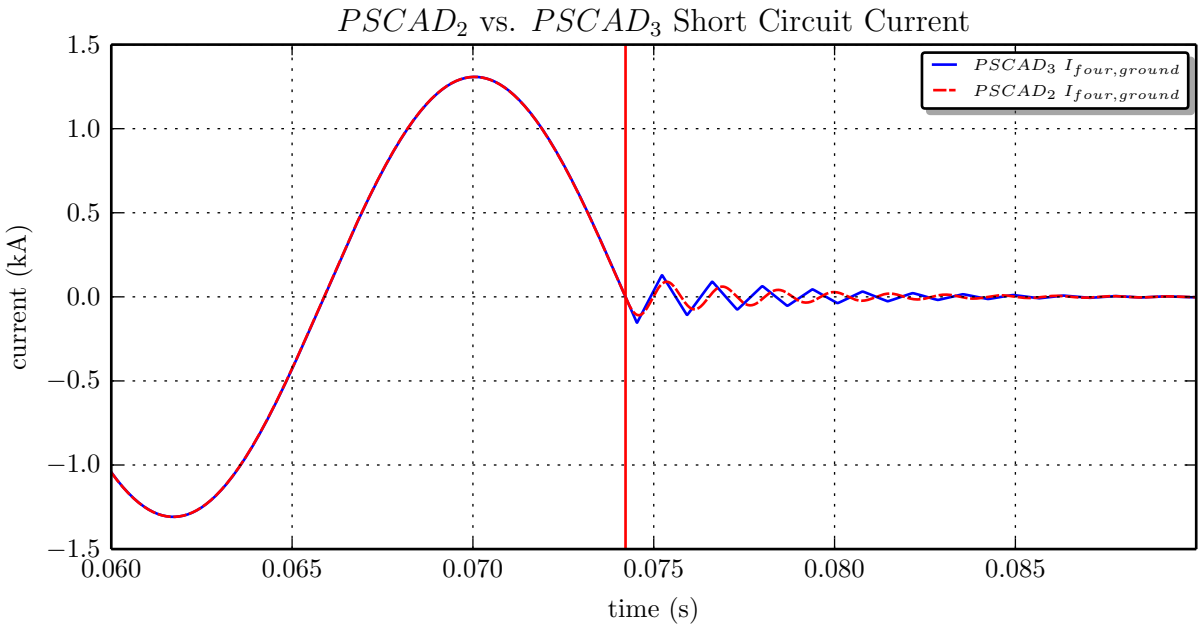

Figure 15: Zoomed plots of $I_{four,ground}$ short circuit current (zoomed at breaker opening)

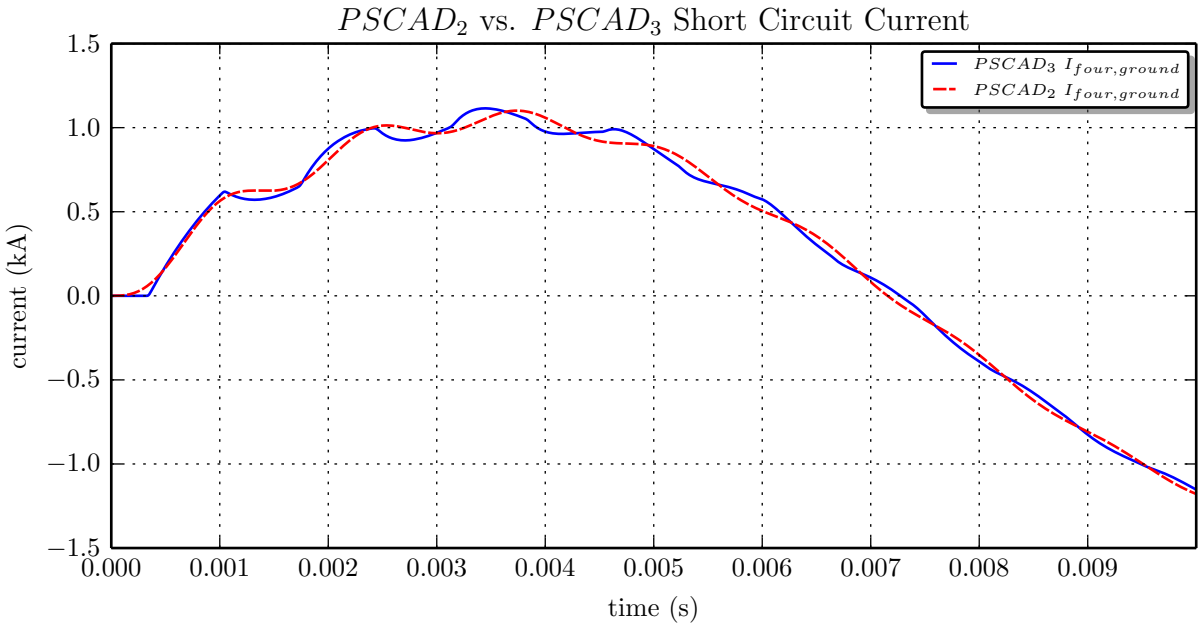

Figure 16: Zoomed plots of $I_{four,ground}$ short circuit current (zoomed at simulation start)

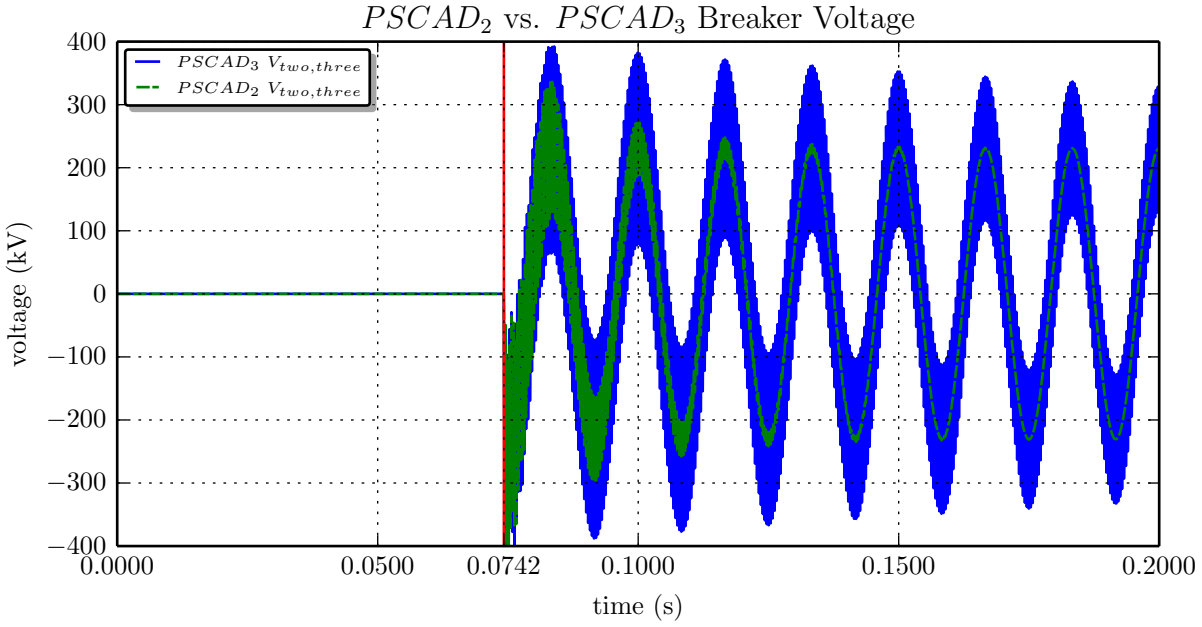

Figure 17: Plots of $V_{two,three}$ breaker voltage

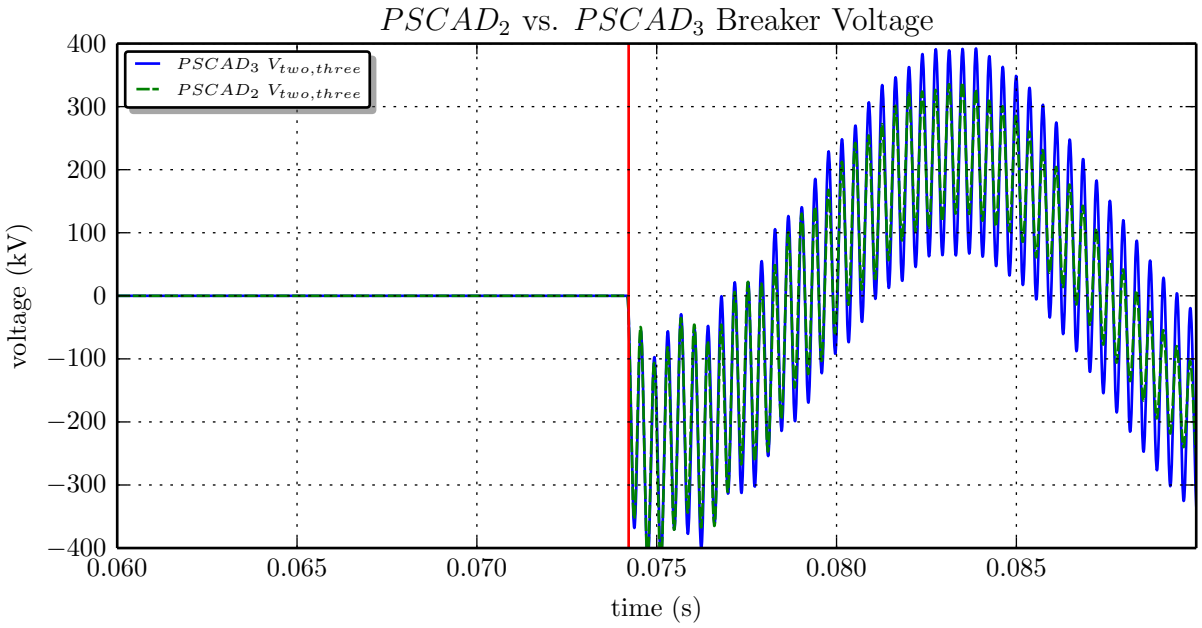

Figure 18: Zoomed plots of $V_{two,three}$ breaker voltage (zoomed at breaker opening)

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.

- After changing the hand-implemented simulation solution to use a breaker resistance instead of two separate matrices, the results match perfectly with the ones generated in PSCAD. The resistances were made identical in each implementation, causing the ringing on the waveform to attenuate at the same rate. Examining the zoomed in sections also reveal that there are no discernible differences between the two. This is evidence that the hand-simulation is performing the same calculations as PSCAD.
- Preprocessing the matrices in MATLAB and using the resulting equations directly in Python has enabled very fast executions of the program. This has allowed iterations to be performed very quickly when implementing and debugging the programmed portions. This of course relies on the MATLAB equations and matrices to be correct.
- What becomes more interesting is when we compare the PSCAD results from the second assignment to this third assignment. These assignments are modeling the same scenario but with two different transmission line models.
  - The first obvious difference between the previous assignment and this assignment is the ringing that occurs across the breaker voltage. In the previous assignment, different values for the breaker's open and closed resistances were chosen compared to this assignment This caused the ringing to attenuate much quicker in the previous assignment compared to this one.
  - There are subtle differences between the previous assignment and this assignment. They can be observed at the very beginning of the simulation and when the breaker is opened. For example, when looking at the short circuit solution zoomed in at the beginning in Figure 16, we see that assignment 3's solutions starts with a flat line - likely due to the interpolator returning 0 for values beyond our history table at the very beginning. We don't see this in assignment 2 because we only assume values for t = 0.
  - The other difference we can see between assignments 2 and 3 is how much sharper the transitions are in assignment 3 when the solution is unstable at the beginning and during breaker opening. Assignment 2's solution reacts with much more smoothness, almost as if it was filtered. This is evident in both Figures 15 and 16, where we zoom into these active moments in the circuit's simulation.
- Overall, it would appear that the two models settle into nearly identical solutions. An interesting experiment would be to see how much differently the two circuit behave with varying Δts. Since $t - \tau$ was much larger than Δt for this implementation of the assignment, a quick experiment was performed to see if rounding up to the nearest table value would make much of a difference and the waveforms were nearly identical. This is likely not possible if the Δt is larger, however.
- Given more time, further comparisons could have been made to see how differently a pure CP-Line model with no line resistance would have behaved. This would have simplified the equation quite a bit. Chances are that its result would be very similar. There is a trade-off to be decided by the engineer if the less accurate model is sufficient for the analysis at hand.

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
})

# Interpolation function
# t0 < t < t1
def interpolate(t, t0, y0, t1, y1):
    # Linear interpolation between points (t0, y0) and (t1, y1)
    return y0 + (t - t0)*(y1 - y0)/(t1 - t0)
    # return y1

# Main function
def main(args):
    # Lists for results
    pscad_assignment2_t_vals = []
    pscad_assignment2_v1_vals = []
    pscad_assignment2_v2_vals = []
    pscad_assignment2_v3_vals = []
    pscad_assignment2_v34_vals = []
    pscad_assignment2_v4_vals = []
    pscad_assignment2_v5_vals = []
    pscad_assignment2_ishort_vals = []

    pscad_assignment3_t_vals = []
    pscad_assignment3_v1_vals = []
    pscad_assignment3_v2_vals = []
    pscad_assignment3_v3_vals = []
    pscad_assignment3_v34_vals = []
    pscad_assignment3_v4_vals = []
    pscad_assignment3_v5_vals = []
    pscad_assignment3_ishort_vals = []

    # tuple indices
    V1_INDEX = 0
    V2_INDEX = 1
    V3_INDEX = 2
    V34_INDEX = 3
    V4_INDEX = 4
    V5_INDEX = 5
    I23_INDEX = 6
    I30_INDEX = 7
    I40_INDEX = 8
    I50_INDEX = 9
    ISHORT_INDEX = 10
```

```
# (v1, v2, v3, v34_vals, v4, v5, i23, i30, i40, i50, ishort)
vals = dict()

# Read in PSCAD .CSV data
print('*** Opening Assignment 2 PSCAD CSV file...')
with open('pscad_assignment2_dat.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    pscad_assignment2_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_assignment2_t_vals.append(row[0])
            pscad_assignment2_v1_vals.append(row[1])
            pscad_assignment2_v2_vals.append(row[2])
            pscad_assignment2_v3_vals.append(row[3])
            pscad_assignment2_v34_vals.append(row[4])
            pscad_assignment2_v4_vals.append(row[5])
            pscad_assignment2_v5_vals.append(row[6])
            pscad_assignment2_ishort_vals.append(row[7])
            # Check if break opned
            if float(row[8]) > 0 and pscad_assignment2_breaker_open_time == 0:
                pscad_assignment2_breaker_open_time = float(row[0])
                print('PSCAD breaker opened at %g' % float(
                    pscad_assignment2_breaker_open_time))
            line_count += 1
    # Figure out when break switched
    print('Processed ' + str(line_count) + ' lines.')
print('*** Opening Assignment 3 PSCAD CSV file...')
with open('pscad_assignment3_dat.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    pscad_assignment3_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_assignment3_t_vals.append(row[0])
            pscad_assignment3_v1_vals.append(row[1])
            pscad_assignment3_v2_vals.append(row[2])
            pscad_assignment3_v3_vals.append(row[3])
            pscad_assignment3_v34_vals.append(row[4])
            pscad_assignment3_v4_vals.append(row[5])
            pscad_assignment3_ishort_vals.append(row[6])
            # Check if break opned
            if float(row[7]) > 0 and pscad_assignment3_breaker_open_time == 0:
                pscad_assignment3_breaker_open_time = float(row[0])
                print('PSCAD breaker opened at %g' % float(
                    pscad_assignment3_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

# Propagation constants
Lline = 1e-3 # mH/km
Cline = 12e-9 # nF/km
line_length = 100 # km
# Velocity of propagation
a = math.sqrt(1/(Lline*Cline)) # km/s
# Transmission line history time delta
tau = line_length/a # s
print("a = " + str(a) + " km/s")
print("tau = " + str(tau) + " s")

# Zero initial conditions
v1 = 0.0
v2 = 0.0
v3 = 0.0
v34 = 0.0
v4 = 0.0
v4_tmintau = 0.0
v5 = 0.0
v5_tmintau = 0.0
i23 = 0.0
i30 = 0.0
i40 = 0.0
i40_tmintau = 0.0
i50 = 0.0
i50_tmintau = 0.0
ishort = 0.0
t_vals = []
vals = []
t_vals.append(0)
vals.append((230, v2, v3, v3-v4, v4, v5, i23, i30, i40, i50, ishort))

# Start at t = delta_t
t = delta_t

# Run simulation
while t < simulation_time:

    # Iterpolate required values at t - tau
    # We are guaranteed a t entry for every delta_t
    # Make sure we have appropriate data to extrapolate from
    if len(t_vals) > 1 and (t - tau) > 0:
        # Find the smallest value that t - tau is above
        t0 = int(round((t - tau)/delta_t)) * delta_t
        # Next value to interpolate with is at t0 + delta_t
        t1 = t0 + delta_t
        t0_index = int(round(t0/delta_t))
        t1_index = int(round(t1/delta_t))
        v4_tmintau = interpolate(t - tau, t0, vals[t0_index][V4_INDEX], t1, vals[
            t1_index][V4_INDEX])
        v5_tmintau = 0.0 # always 0
        i40_tmintau = interpolate(t - tau, t0, vals[t0_index][I40_INDEX], t1, vals[
            t1_index][I40_INDEX])
        i50_tmintau = interpolate(t - tau, t0, vals[t0_index][I50_INDEX], t1, vals[
            t1_index][I50_INDEX])
    else:
        # Can't extrapolate, use 0
        t0 = 0.0
        t1 = 0.0
        t0_index = int(round(t0/delta_t))
        t1_index = int(round(t1/delta_t))
        v4_tmintau = 0.0
        v5_tmintau = 0.0
        i40_tmintau = 0.0
        i50_tmintau = 0.0
```

```
        # Check if breaker is opened or closed and set breaker resistance accordingly
        if breaker_open_flag == False:
            # Breaker has very small resistance (closed)
            Rbrk = 1e-9
        else:
            # Break has very large resistance (open)
            Rbrk = 1e9

        # Voltage calculations
        # These voltage calculations are generated in MATLAB
        # by v_next[] = inv(GAA)*hA - inv(GAA)*GAB*vs
        v2_next = (1.135814937804493543741046072016 9e
            +37*(0.036674077968279601044949287851997*i40_tmintau +
            0.883625629458980926081173436587 49*i50_tmintau +
            0.000132315525781974948032402046265 91*v4_tmintau +
            0.003188011702909576512471947744615*v5_tmintau))
            /(1.772575739976319342526327845379e+36*Rbrk + 1.413858144947116222377107354388 4e
            +39) - (59109745109237760000.0*(89959801610818431.0*Rbrk +
            71754549774979273936.0)*(i23 + 0.000014285714285714285714285714285714*v2 -
            0.000014285714285714285714285714285714*v3))/(1.772575739976319342526327845379e
            +36*Rbrk + 1.413858144947116222377107354388 4e+39) +
            (19703248369745920000.0*(1914038332145073.0*Rbrk + 576460752303423488.0)*(i23 +
            i30 + 0.000014285714285714285714285714285714*v2 +
            0.001985714285714285714285714285143*v3))/(1.772575739976319342526327845379e+36*
            Rbrk + 1.413858144947116222377107354388 4e+39) + (45317471250415616000 00.0*math.
            cos(377.0*t)*(89959801610818431.0*Rbrk + 71754549774979273936.0))
            /(1.772575739976319342526327845379e+36*Rbrk + 1.413858144947116222377107354388 4e
            +39)
        v3_next = (2.650348436370931702638844437332e
            +41*(0.036674077968279601044949287851997*i40_tmintau +
            0.883625629458980926081173436587 49*i50_tmintau +
            0.000132315525781974948032402046265 91*v4_tmintau +
            0.003188011702909576512471947744615*v5_tmintau))
            /(1.772575739976319342526327845379e+36*Rbrk + 1.413858144947116222377107354388 4e
            +39) - (19703248369745920000.0*(1914038332145073.0*Rbrk + 576460752303423488.0)
            *(i23 + 0.000014285714285714285714285714285714*v2 -
            0.000014285714285714285714285714285714*v3))/(1.772575739976319342526327845379e
            +36*Rbrk + 1.413858144947116222377107354388 4e+39) +
            (151058237501385386666 6.6666666667*math.cos(377.0*t)*(1914038332145073.0*Rbrk +
            576460752303423488.0))/(1.772575739976319342526327845379e+36*Rbrk +
            1.413858144947116222377107354388 4e+39) +
            (459762165209107818750000.0*(1914038332145073.0*Rbrk + 576460752303423488.0)*(
            i23 + i30 + 0.000014285714285714285714285714285714*v2 +
            0.001985714285714285714285714285143*v3))/(1.772575739976319342526327845379e+36*
            Rbrk + 1.413858144947116222377107354388 4e+39)
        v4_next = (8.707914523167783835348019885 4629e+38*math.cos(377.0*t))
            /(1.772575739976319342526327845379e+36*Rbrk + 1.413858144947116222377107354388 4e
            +39) - (1.135814937804493543741046072016 9e+37*(i23 +
            0.000014285714285714285714285714285714*v2 -
            0.000014285714285714285714285714285714*v3))/(1.772575739976319342526327845379e
            +36*Rbrk + 1.413858144947116222377107354388 4e+39) +
            (2.650348436370931702638844437332e+41*(i23 + i30 +
            0.000014285714285714285714285714285714*v2 +
            0.001985714285714285714285714285143*v3))/(1.772575739976319342526327845379e+36*
            Rbrk + 1.413858144947116222377107354388 4e+39) +
            (576460752303423488.0*(926092079874797609969.0*Rbrk +
            459762165209107818750000.0)*(0.036674077968279601044949287851997*i40_tmintau +
            0.883625629458980926081173436587 49*i50_tmintau +
            0.000132315525781974948032402046265 91*v4_tmintau +
            0.003188011702909576512471947744615*v5_tmintau))
            /(1.772575739976319342526327845379e+36*Rbrk + 1.413858144947116222377107354388 4e
            +39)
        v5_next = 0.0 # v5 always zero!

        # Next currents
        i23_next = i23 + 0.000014285714285714285714285714285714*v2 -
            0.000014285714285714285714285714285714*v3 +
```

```
            0.000014285714285714285714285714285714*v2_next -
            0.000014285714285714285714285714285714*v3_next
        i30_next = 0.002*v3_next - 0.002*v3 - 1.0*i30
        i40_next = 0.0033203272286915516454473282287533*v4_next -
            0.88362562945898092608117343658749*i50_tmintau -
            0.036674077968279601044949287851997*i40_tmintau -
            0.00013231552578197494803240204626591*v4_tmintau -
            0.0031880117029095765124719477446615*v5_tmintau
        i50_next = - 0.88362562945898092608117343658749*i40_tmintau -
            0.036674077968279601044949287851997*i50_tmintau -
            0.0031880117029095765124719477446615*v4_tmintau -
            0.00013231552578197494803240204626591*v5_tmintau

        # Check if breaker has opened
        if (t >= breaker_time) and (not breaker_open_flag):
            # Check for zero crossing
            i_break_prev = (v4 - v3)/Rbrk
            i_break_next = (v4_next - v3_next)/Rbrk
            if ((i_break_next * i_break_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
        ishort = -i50_next # ishort's direction is opposite of i50
        v2 = v2_next
        v3 = v3_next
        v4 = v4_next
        v5 = v5_next

        # Append the values for the current iteration
        # (v1, v2, v3, v34_vals, v4, v5, i23, i30, i40, i50, ishort)
        t_vals.append(t)
        vals.append((230*math.cos(377*t), v2, v3, v3-v4, v4, v5, i23, i30, i40, i50, ishort)
            )

        # Go to next time step
        t = t + delta_t

    # Transpose data
    transposed_vals = zip(*vals)

    # Plots for publication
    legend_font_size = 6

    # Superimposed
    # Nodal Voltage Comparison Plots (Zoomed)
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(t_vals, transposed_vals[V1_INDEX], label='$Simulation\\ V_{one}$')
    ax.plot(t_vals, transposed_vals[V3_INDEX], label='$Simulation\\ V_{two}$')
    ax.plot(t_vals, transposed_vals[V4_INDEX], label='$Simulation\\ V_{three}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v1_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{one}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v3_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{two}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v4_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{three}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', 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')

# Superimposed
# Nodal Voltage Comparison Plots (Zoomed)
# PSCAD & PSCAD
fig, ax = plt.subplots(1)
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v1_vals, label='$PSCAD_3\\ V_{one}$'
    )
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v3_vals, label='$PSCAD_3\\ V_{two}$'
    )
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v4_vals, label='$PSCAD_3\\ V_{three}
    $')
ax.plot(pscad_assignment2_t_vals, pscad_assignment2_v1_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD_2\\ V_{one}$')
ax.plot(pscad_assignment2_t_vals, pscad_assignment2_v3_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD_2\\ V_{two}$')
ax.plot(pscad_assignment2_t_vals, pscad_assignment2_v4_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD_2\\ V_{three}$')
ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='$PSCAD_2$ vs. $PSCAD_3$ 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('pscad_nodal_voltage_comparison_plots.pgf')
fig.savefig('pscad_nodal_voltage_comparison_plots.png')

# Nodal Voltage Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, transposed_vals[V1_INDEX], label='$Simulation\\ V_{one}$')
ax.plot(t_vals, transposed_vals[V3_INDEX], label='$Simulation\\ V_{two}$')
ax.plot(t_vals, transposed_vals[V4_INDEX], label='$Simulation\\ V_{three}$')
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v1_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD\\ V_{one}$')
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v3_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD\\ V_{two}$')
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v4_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD\\ V_{three}$')
ax.axvline(x=breaker_open_time, linestyle='solid', 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_1.pgf')
fig.savefig('nodal_voltage_comparison_plots_zoom_1.png')

# Nodal Voltage Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
```

```
    ax.plot(t_vals, transposed_vals[V1_INDEX], label='$Simulation\\ V_{one}$')
    ax.plot(t_vals, transposed_vals[V3_INDEX], label='$Simulation\\ V_{two}$')
    ax.plot(t_vals, transposed_vals[V4_INDEX], label='$Simulation\\ V_{three}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v1_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{one}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v3_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{two}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v4_vals, linestyle='--', dashes=(5,
        1), label='$PSCAD\\ V_{three}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', 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(0.0, 0.01, step=0.001).tolist())
    ax.set_ylim(-400,400)
    ax.set_xlim(0.0, 0.01)
    fig.set_size_inches(6.5,3.5)
    fig.tight_layout()
    fig.savefig('nodal_voltage_comparison_plots_zoom_2.pgf')
    fig.savefig('nodal_voltage_comparison_plots_zoom_2.png')

    # Short Circuit Current Comparison Plots
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(t_vals, transposed_vals[ISHORT_INDEX], label='$Simulation\\ I_{four, ground}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='--', dashes
        =(5, 1), label='$PSCAD\\ I_{four, ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', 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')

    # Short Circuit Current Comparison Plots
    # PSCAD & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='solid',
        label='$PSCAD_3\\ I_{four, ground}$')
    ax.plot(pscad_assignment2_t_vals, pscad_assignment2_ishort_vals, c='r', linestyle='--',
        dashes=(5, 1), label='$PSCAD_2\\ I_{four, ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
    ax.set(xlabel='time (s)', ylabel='current (kA)', title='$PSCAD_2$ vs. $PSCAD_3$ 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('pscad_short_circuit_current_comparison_plots.pgf')
    fig.savefig('pscad_short_circuit_current_comparison_plots.png')

    # Short Circuit Current Comparison Plots (Zoomed)
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(t_vals, transposed_vals[ISHORT_INDEX], label='$Simulation\\ I_{four, ground}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='--', dashes
        =(5, 1), label='$PSCAD\\ I_{four, ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', 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_1.pgf')
    fig.savefig('short_circuit_current_comparison_plots_zoom_1.png')

    # Short Circuit Current Comparison Plots (Zoomed)
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='solid',
        label='$PSCAD_3\\␣I_{four␣ground}$')
    ax.plot(pscad_assignment2_t_vals, pscad_assignment2_ishort_vals, c='r', linestyle='--',
        dashes=(5, 1), label='$PSCAD_2\\␣I_{four␣ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
    ax.set(xlabel='time␣(s)', ylabel='current␣(kA)', title='$PSCAD_2$␣vs.␣$PSCAD_3$␣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('pscad_short_circuit_current_comparison_plots_zoom_1.pgf')
    fig.savefig('pscad_short_circuit_current_comparison_plots_zoom_1.png')

    # Short Circuit Current Comparison Plots (Zoomed)
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='solid',
        label='$PSCAD_3\\␣I_{four␣ground}$')
    ax.plot(pscad_assignment2_t_vals, pscad_assignment2_ishort_vals, c='r', linestyle='--',
        dashes=(5, 1), label='$PSCAD_2\\␣I_{four␣ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
    ax.set(xlabel='time␣(s)', ylabel='current␣(kA)', title='$PSCAD_2$␣vs.␣$PSCAD_3$␣Short␣
        Circuit␣Current')
    ax.legend(loc='best', prop={'size':legend_font_size}, fancybox=True, shadow=True)
    ax.grid()
    ax.set_xticks(np.arange(0.0, 0.01, step=0.001).tolist())
    ax.set_ylim(-1.5,1.5)
    ax.set_xlim(0.0, 0.01)
    fig.set_size_inches(6.5,3.5)
    fig.tight_layout()
    fig.savefig('pscad_short_circuit_current_comparison_plots_zoom_2.pgf')
    fig.savefig('pscad_short_circuit_current_comparison_plots_zoom_2.png')

    # Short Circuit Current Comparison Plots (Zoomed)
    # Simulation & PSCAD
    fig, ax = plt.subplots(1)
    ax.plot(t_vals, transposed_vals[ISHORT_INDEX], label='$Simulation\\␣I_{four␣ground}$')
    ax.plot(pscad_assignment3_t_vals, pscad_assignment3_ishort_vals, linestyle='--', dashes
        =(5, 1), label='$PSCAD\\␣I_{four␣ground}$')
    ax.axvline(x=breaker_open_time, linestyle='solid', 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(0.0, 0.01, step=0.001).tolist())
    ax.set_ylim(-1.5,1.5)
    ax.set_xlim(0.0, 0.01)
    fig.set_size_inches(6.5,3.5)
    fig.tight_layout()
```

```
fig.savefig('short_circuit_current_comparison_plots_zoom_2.pgf')
fig.savefig('short_circuit_current_comparison_plots_zoom_2.png')

# Breaker Voltage Comparison Plots
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, transposed_vals[V34_INDEX], label='$Simulation\\ V_{two, three}$')
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v34_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD\\ V_{two, three}$')
ax.axvline(x=breaker_open_time, linestyle='solid', 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')

# Breaker Voltage Comparison Plots (Zoomed)
# Simulation & PSCAD
fig, ax = plt.subplots(1)
ax.plot(t_vals, transposed_vals[V34_INDEX], label='$Simulation\\ V_{two, three}$')
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v34_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD\\ V_{two, three}$')
ax.axvline(x=breaker_open_time, linestyle='solid', 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')

# Breaker Voltage Comparison Plots
# PSCAD & PSCAD
fig, ax = plt.subplots(1)
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v34_vals, linestyle='solid', label='
    $PSCAD_3\\ V_{two, three}$')
ax.plot(pscad_assignment2_t_vals, pscad_assignment2_v34_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD_2\\ V_{two, three}$')
ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='$PSCAD_2$ vs. $PSCAD_3$ 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('pscad_breaker_voltage_comparison_plots.pgf')
fig.savefig('pscad_breaker_voltage_comparison_plots.png')

# Breaker Voltage Comparison Plots (Zoomed)
# PSCAD & PSCAD
fig, ax = plt.subplots(1)
ax.plot(pscad_assignment3_t_vals, pscad_assignment3_v34_vals, linestyle='solid', label='
    $PSCAD_3\\ V_{two, three}$')
ax.plot(pscad_assignment2_t_vals, pscad_assignment2_v34_vals, linestyle='--', dashes=(5,
    1), label='$PSCAD_2\\ V_{two, three}$')
```

```
    ax.axvline(x=breaker_open_time, linestyle='solid', c='r')
    ax.set(xlabel='time (s)', ylabel='voltage (kV)', title='$PSCAD_2$ vs. $PSCAD_3$ 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('pscad_breaker_voltage_comparison_plots_zoom.pgf')
    fig.savefig('pscad_breaker_voltage_comparison_plots_zoom.png')

if __name__ == '__main__':
    # the following sets up the argument parser for the program
    parser = argparse.ArgumentParser(description='Assignment 3 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 R_line L1 L_line C1 C_line line_length Zc Zo a Rbrk frequency

R1 = 3
L1 = 350e-3
C1 = 10e-9

R_line = 0.5 % Ohm/km
L_line = 1e-3 % mH/km
C_line = 12e-9 % nF/km
line_length = 100 % km

frequency = 60 % Hz

Xl_line = 2*pi*frequency*L_line % Ohm/km
Xc_line = 1/(2*pi*frequency*C_line) % Ohm/km

% delta_t = 10e-6

% Velocity of propagation
a = sqrt(1/(L_line*C_line)) % km/s

% Transmission line history time delta
tau = line_length/a % s

% Characteristic line impedance
Zc = sqrt(L_line/C_line)

% Bergeron equivalent impedance
Zo = Zc + (R_line*line_length/4)
```

```
Hb = (Zo - (R_line*line_length/4))/(Zo + (R_line*line_length/4))

% Discretized Resistances:
R23 = (2*L1)/delta_t
R30 = delta_t/(2*C1)

% History Equations:
syms eh23 eh30
syms h23 h30 h40 h50
syms v2 v3 v4 v5
syms v4_tmintau v5_tmintau
syms v2_next v3_next v4_next v5_next
syms i23 i50 i30 i40
syms i40_tmintau i50_tmintau
syms i23_next i50_next i30_next i40_next

% History voltage source for L1 and C1
eh23 = -(v2 - v3) - ((2*L1)/delta_t)*i23
eh30 = v3 + (delta_t/(2*C1))*i30

% History current sources
h23 = eh23/R23
h30 = eh30/R30
h40 = ((1 + Hb)/2)*((-1/Zo)*v5_tmintau - Hb*i50_tmintau) + ((1 - Hb)/2)*((-1/Zo)*v4_tmintau
    - Hb*i40_tmintau)
h50 = ((1 + Hb)/2)*((-1/Zo)*v4_tmintau - Hb*i40_tmintau) + ((1 - Hb)/2)*((-1/Zo)*v5_tmintau
    - Hb*i50_tmintau)

i23_next = (v2_next - v3_next)/R23 - h23
i30_next = v3_next/R30 - h30
i40_next = v4_next/Zo + h40
i50_next = h50

syms g11 g12 g13 g14
syms g21 g22 g23 g24
syms g31 g32 g33 g34
syms g41 g42 g43 g44
syms h1 h2 h3 h4

h1 = 0
h2 = h23
h3 = h30 - h23
h4 = -h40

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

g11 = 1/R1
g12 = -1/R1
g13 = 0
g14 = 0

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

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

g41 = 0
g42 = 0
g43 = -1/Rbrk
g44 = 1/Rbrk + 1/Zo
```

```
GAA = [g22 g23 g24; g32 g33 g34; g42 g43 g44]
GAB = [g21 ; g31 ; g41]
GBA = [g12 g13 g14]
GBB = [g11]
G = [GAA GAB ; GBA GBB]

% Nodal voltage solution vector:
v_next = inv(GAA)*hA - inv(GAA)*GAB*vs
vpa(subs(v_next))
```

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_STAT, BRK
      REAL     Ibrk, V5, V4, V34, I12, I20
      REAL     Ishort, V3, V2, V1
```

```
! 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 + 10
      IPGB      = NPGB
      NPGB      = NPGB + 11
      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 7
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

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

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


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

```
! 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+2),1.0e-09,1000000000.0,0,NINT(1.0-RE&
     &AL(BRK)))
      IVD1_1 = 2*E_BtoI(OPENBR( (IBRCH+2),SS))
      IF (FIRSTSTEP .OR. (STORI(IVD1_2) .NE. IVD1_1)) THEN
         CALL PSCAD_AGI2(ICALL_NO,1421318096,IVD1_1,"BOpen")
      ENDIF
      STORI(IVD1_2) = 2*E_BtoI(OPENBR( (IBRCH+2),SS))
      BRK_STAT = IVD1_1

! 220:[pgb] Output Channel 'BRK_STAT'

      PGB(IPGB+10) = 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+5),RVD1_4,1,RVD1_1,RVD1_2,RVD1_3)

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

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

```
      STOF(ISTOF + 10) = V1


!---------------------------------------
! 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_3, NT_4, NT_7, NT_8

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

! 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 + 11
      INODE     = NNODE + 2
      NNODE     = NNODE + 8
      IBRCH     = NBRCH(SS)
      NBRCH(SS) = NBRCH(SS) + 7
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

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

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


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

      NT_1  = NODE(INODE + 1)
      NT_2  = NODE(INODE + 2)
      NT_3  = NODE(INODE + 3)
      NT_4  = NODE(INODE + 4)
      NT_7  = NODE(INODE + 5)
      NT_8  = 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_8, 0)

! 30:[voltmetergnd] Voltmeter (Line - Ground) 'V2'
      V2 = EMTDC_VVDC(SS, NT_7, 0)

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

! 50:[voltmetergnd] Voltmeter (Line - Ground) 'V3'
      V3 = EMTDC_VVDC(SS, NT_2, 0)

! 60:[ammeter] Current Meter 'Ibrk'
      Ibrk = ( CBR((IBRCH+1), SS))

! 70:[voltmeter] Voltmeter (Line - Line) 'V34'
      V34 = EMTDC_VVDC(SS, NT_1, NT_3)

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

! 90:[voltmetergnd] Voltmeter (Line - Ground) 'V4'
      V4 = EMTDC_VVDC(SS, NT_3, 0)

! 100:[pgb] Output Channel 'V1'

      PGB(IPGB+1) = V1

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

! 120:[voltmetergnd] Voltmeter (Line - Ground) 'V5'
      V5 = EMTDC_VVDC(SS, NT_4, 0)

! 130:[pgb] Output Channel 'V2'

      PGB(IPGB+2) = V2

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

! 150:[pgb] Output Channel 'V3'

      PGB(IPGB+3) = V3

! 160:[pgb] Output Channel 'V34'

      PGB(IPGB+4) = V34

! 170:[pgb] Output Channel 'V4'

      PGB(IPGB+5) = V4

! 180:[pgb] Output Channel 'V5'

      PGB(IPGB+6) = V5

! 190:[pgb] Output Channel 'Ishort'

      PGB(IPGB+7) = Ishort

! 200:[pgb] Output Channel 'I20'
```

```
      PGB(IPGB+8) = I20

! 210:[pgb] Output Channel 'I12'

      PGB(IPGB+9) = I12

! 230:[pgb] Output Channel 'Ibrk'

      PGB(IPGB+11) = Ibrk

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

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


!---------------------------------------
! 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) + 7
      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,1421318096)
      CALL E1PBRKR1_CFG(1.0e-09,1000000000.0,0.0)

! 220:[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_3, NT_4, NT_7, NT_8

! 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) + 7
      NCSCS     = NCSCS + 0
      NCSCR     = NCSCR + 0

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

      NT_1  = NODE(INODE + 1)
      NT_2  = NODE(INODE + 2)
      NT_3  = NODE(INODE + 3)
      NT_4  = NODE(INODE + 4)
      NT_7  = NODE(INODE + 5)
      NT_8  = NODE(INODE + 6)

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


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

```
! 130:[pgb] Output Channel 'V2'

! 150:[pgb] Output Channel 'V3'

! 160:[pgb] Output Channel 'V34'

! 170:[pgb] Output Channel 'V4'

! 180:[pgb] Output Channel 'V5'

! 190:[pgb] Output Channel 'Ishort'

! 200:[pgb] Output Channel 'I20'

! 210:[pgb] Output Channel 'I12'

! 230:[pgb] Output Channel 'Ibrk'

      RETURN
      END
```