Project Report
EEE 598 - Mini Project 2 Phase 1

Objective:

The project aims to develop a static IR Drop solver using Python. The given spice netlists contain details on the resistors, voltage sources, current sources, and their location in the power grid.

Methodology:

For each line in the Spice netlist, the component type is identified and appended to lists to form the G matrix, a Voltage vector, and a Current vector, as shown in Fig. 1.

```
for line in data:
    if not line.startswith(".") and len(line.split()) == 4:
        components = line.split()
        # Process Resistances:
        if(components[0].startswith("R")):
            self.g_data.append((components[1], components[2], 1/float(components[3])))
        # Process Currents:
        elif(components[0].startswith("I")):
            self.i_data.append((components[1], components[2], float(components[3])))
        # Process Voltages:
        elif(components[0].startswith("V")):
            self.v_data.append((components[1], components[2], float(components[3])))
        # Save node information:
        components[1] = components[2] if components[1] == '0' else components[1]
        components[2] = components[1] if components[2] == '0' else components[2]
        self.nodes.update([components[1], components[2]])
```

Fig.1. Code Snippet to parse the Spice Netlist

To populate values to the G matrix based on the resistor location and values, for each resistor in the Spice Netlist, its inverse value is added in the (i, i) and (j, j) locations and is subtracted from the (i, j) and (j, i) locations as shown in Fig.2.

```
# Form G-matrix:
G_mat_dict = {} # Sparse G-matrix as dictionary
for net1, net2, G in self.g_data:
    i,j = self.node_index[net1], self.node_index[net2]
    G_mat_dict[(i,i)] = (G_mat_dict[(i,i)] + G) if (i,i) in G_mat_dict else G
    G_mat_dict[(i,j)] = (G_mat_dict[(i,j)] - G) if (i,j) in G_mat_dict else -G
    G_mat_dict[(j,i)] = (G_mat_dict[(j,i)] - G) if (j,i) in G_mat_dict else -G
    G_mat_dict[(j,j)] = (G_mat_dict[(j,j)] + G) if (j,j) in G_mat_dict else G
```

Fig.2. Code Snippet to Populate value of the G matrix

The G matrix is modified by adding 1 for each Voltage source at (M, i) and (i, M) locations. Simultaneously, the I matrix is updated with the Voltage values, as shown in Fig.3 and Fig.4, respectively.

```
# Modified G-matrix:
if len(self.v_data) > 0:
    for net1, net2, V in self.v_data:
        i_vector = np.append(i_vector, V)
        if net2 == '0':
            i = self.node_index[net1]
            G_mat_dict[(i,max_n)] = 1
            G_mat_dict[(max_n,i)] = 1
        elif net1 == '0':
            i = self.node_index[net2]
            G_mat_dict[(i,max_n)] = 1
            G_mat_dict[(max_n,i)] = 1
        max_n += 1
```

Fig.3. Code Snippet to Update G matrix

```
# Form Current matrix:
i_vector = np.zeros(max_n)
for net1, net2, I in self.i_data:
    if net2 == '0':
        i = self.node_index[net1]
        i_vector[i] = -I
    elif net1 == '0':
        i = self.node_index[net2]
        i_vector[i] = I
```

Fig.4. Code Snippet to form Current Vector

Fig.5. To form the coordinate matrix, the row and column values from the G matrix dictionary (G_mat_dict) were separated and given as input to the scipy.coo_matrix() function. The Coordinate matrix was converted to a Compressed sparse (CSR) matrix and is solved using scipy.spsolve() function to form the Voltage vector.

```
sparse_g_matrix = coo_matrix((self.val_mat, (self.row_mat, self.col_mat))).tocsr()
```

```
# Compute the Voltage vector:
def get_voltage_vector(self):
    self.v_vector = spsolve(self.sparse_g_matrix, self.i_vector)
```

Fig.5. Code Snippet to form CSR matrix and Solve for Voltage

Challenges faced:

1. Sorting the node names according to the Metal layers was not possible with just the sort function since the names are made of alphanumeric characters. So we implemented a custom function to strip off the alphabets from the node name and sort the resulting numbers to give them unique index values to form the G matrix. The code snippet for the custom sort function is shown in Fig. 6.

```
# Sort inorder:
def sort_key(self, s):
    parts = re.split(r'(\d+)', s)  # Split by numbers while keeping them
    return [int(p) if p.isdigit() else p for p in parts]
```

Fig.6. Code Snippet to sort nodes

2. Writing the result to a file took a lot of time. Python's zip() function was used which drastically reduced the runtime. The code snippet is as shown in Fig.7.

```
# Write the node voltages:
def write_voltage_vector(self, outfilename):
    with open(f'{outfilename}', 'w') as file:
        for node, voltage in zip(self.nodes, self.v_vector):
            file.write(f"{node}\t{voltage}\n")
```

Fig.7. Code Snippet to write to a file

Scatter-plots:

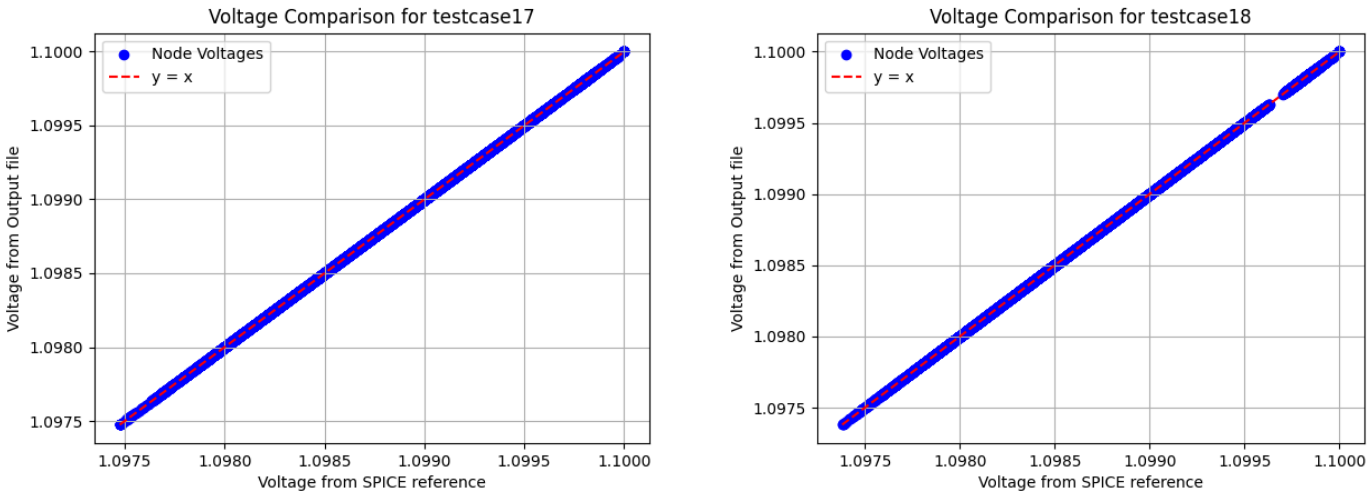

Fig.8. Voltage comparison for testcase17 & testcase18