# Program Overview

Our program operates by calling a main PowerFlowAnalysis() function for each of the three cases. This function uses the information location to solve for a desired power flow case. Fig. 1 show’s a basic overview of how our power flow analysis program operates. The line data is parsed to create the admittance matrix for the system, and the bus data is turned into a 2D array containing each buses power and voltage information. This 2D matrix is then sent through the power flow process to give each bus’s voltage magnitude and angle, and the real and reactive power.

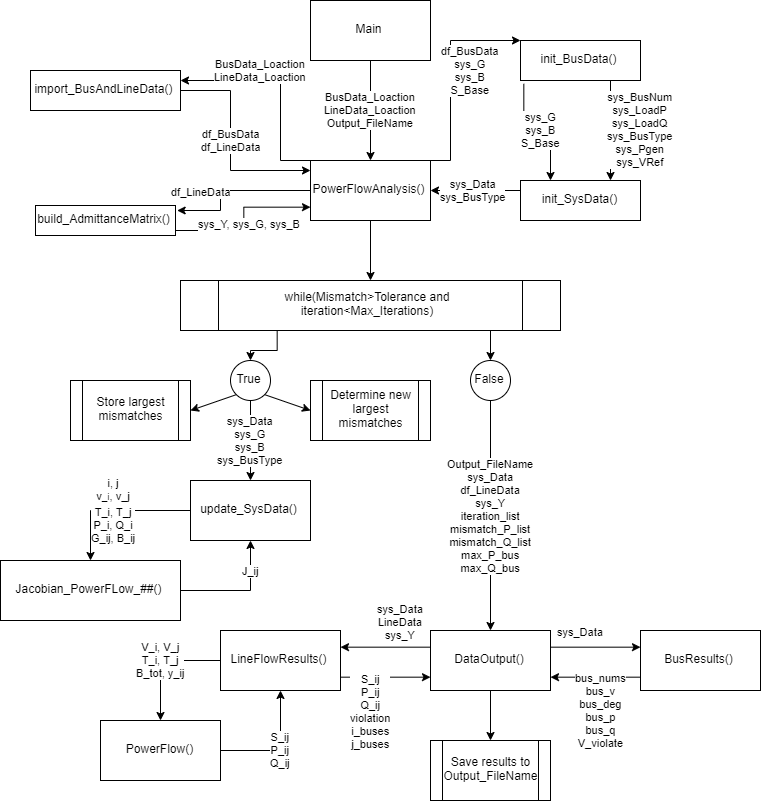


Figure 1: Flow chart of Power Flow Analysis program

The parameters passed to and returned from each function is placed near its corresponding arrow. Important internal processes are shown by process boxes that have stripped left and right borders. A more complete description of each function and variable can be obtained from the code comments in the Appendix.

# Program Validation

Verification of our program was broken up into two main steps: individual function verification, and overall functionality. Individual functions were first tested by having them attempt to return specific values. For instance, the Jacobian support functions were initially assigned to return a unique value for each of the 8 possibilities to ensure it was placing the desired values in the desired locations. After ensuring that the intent was being fulfilled, specific input and output pairs were used to verify that internal processing of the functions were working correctly. This was accomplished through the use of previous homework problems to ensure that things like the admittance matrix and Jacobian were being correctly built.

Finally, overall functionality was verified by testing it on a simple three-bus network. The expected values for this network were calculated by hand so that we would have a reference expectation for each variable in our program. We then systematically checked each variable to ensure that the program was creating the expected values. Both at the output, and internally from each function.

# Results

## Base Case



Table 1: Table showing the voltage in per unit, angle in degrees, active power injected in MW, reactive power injected in MVar, and whether the voltage limit was violated at each bus.



Table 2: Table showing the active, reactive, and apparent power flowing from one bus to another and whether the line MVA limit is violated.



Table 3: Table showing the iterations of convergence with the max active and reactive mismatches and what bus they belong to.

## Contingency 1



Table 4: Table showing the voltage in per unit, angle in degrees, active power injected in MW, reactive power injected in MVar, and whether the voltage limit was violated at each bus.



Table 5: Table showing the active, reactive, and apparent power flowing from one bus to another and whether the line MVA limit is violated.



Table 6: Table showing the iterations of convergence with the max active and reactive mismatches and what bus they belong to.

## Contingency 2



Table 7: Table showing the voltage in per unit, angle in degrees, active power injected in MW, reactive power injected in MVar, and whether the voltage limit was violated at each bus.



Table 8: Table showing the active, reactive, and apparent power flowing from one bus to another and whether the line MVA limit is violated.



Table 9: Table showing the iterations of convergence with the max active and reactive mismatches and what bus they belong to.

## Discussion of Results

The results of contingency case 1 appear to make sense. In contingency case 1, one of the two lines connecting bus 1 and 2 is taken offline. This results in an increase in the impedance between the two lines. As such the increase in power injection from bus 1 would make sense as it would be compensating the additional power loss. Additionally, the power flow information shows that bus 2 is exporting less power to its connected busses. This also makes sense as the generator from bus 1 would no longer be as effective at providing power to bus 2, and so the generator at bus 2 would need to cover that load.

The results from contingency case 2 also make sense. In contingency case 2, the line connecting bus 10 and 11 is taken offline. The effects of this is most evident in the line flow analysis. It can be seen that bus 6 starts drawing more power from its connecting busses and exports more power to bus 11. It can also be seen how the loss of that line most impacts the region around bus 11, where the lower region maintains fairly similar results for bus and line results.

# Team Contribution

Both members contributed to writing the power flow program. In addition to writing specific functions, both members reviewed the others work to ensure understanding of process. Main focus of each member was as follows:

|  |  |
| --- | --- |
| Brandon Beaty | Admittance Matrix, initializing bus and system data, solving the system |
| Eliot Nichols | Data import, line flow analysis, violation check, data export |

Additionally, both members contributed to writing report and discussing results.

# Appendix

## Main

1. """
2. Power Flow Analysis: Main Function
3. Created By:
4. Brandon Beaty
5. Eliot Nichols
6. """
7. **from** PowerSysAnalysis\_HeaderFile **import** \*
9. """
10. ########################
11. Main Section of Code
12. ########################
13. """
15. """
16. Loction for Bus and line data
17. First variable is for file location, Second variable is for sheet name
18. """
19. BusData\_Location = ['system\_basecase.xlsx', 'BusData'] #BaseCase Data
20. LineData\_Location\_BaseCase = ['system\_basecase.xlsx', 'LineData'] #BaseCase Data
21. LineData\_Location\_Contingency1 = ['system\_basecase.xlsx', 'ContingencyCase1'] #BaseCase Data
22. LineData\_Location\_Contingency2 = ['system\_basecase.xlsx', 'ContingencyCase2'] #BaseCase Data
24. tolerance = [.001, .001] #P.U.
25. S\_Base = 100 #MW
27. PowerFlowAnalysis(BusData\_Location, LineData\_Location\_BaseCase, "Base Case Solution", tolerance, S\_Base)
28. PowerFlowAnalysis(BusData\_Location, LineData\_Location\_Contingency1, "Contingency Case 1 Solution", tolerance, S\_Base)
29. PowerFlowAnalysis(BusData\_Location, LineData\_Location\_Contingency2, "Contingency Case 2 Solution", tolerance, S\_Base)

## Support Files

1. """
2. Power Flow Analysis: Support Functions
3. Created By:
4. Brandon Beaty
5. Eliot Nichols
6. """
8. **import** numpy as np
9. **from** numpy.linalg **import** inv
10. **import** pandas as pd

13. """
14. Imports Bus and line data from excel sheets
15. Takes in an array containing ['File Location', 'Sheet Name']
16. Returns two panda data frames for the bus and line data
17. """
18. **def** import\_BusAndLineData(BusData\_Location, LineData\_Location):
19. BusData = pd.read\_excel(BusData\_Location[0], sheet\_name=BusData\_Location[1])
20. LineData = pd.read\_excel(LineData\_Location[0], sheet\_name=LineData\_Location[1])
21. **return** BusData, LineData

24. """
25. Builds G and B matrices to be used in Power Flow calculations
26. Takes in data frame containing all line information, and number of busses in system
27. Returns G and B arrays
28. """
29. **def** build\_AdmittanceMatrix(LineData, sys\_Size):
30. col = np.array(LineData.columns)
31. line\_From = np.array(LineData[col[0]])
32. line\_To = np.array(LineData[col[1]])
33. line\_R = np.array(LineData[col[2]])
34. line\_X = np.array(LineData[col[3]])
35. line\_Z = np.array(LineData[col[2]]) + 1j\*np.array(LineData[col[3]])
36. line\_Y = 1/line\_Z
37. line\_B = np.array(LineData[col[4]])
38. line\_Fmax = np.array(LineData[col[5]])
39. sys\_Y = np.array([[0 **for** j **in** range(sys\_Size)] **for** i **in** range(sys\_Size)], dtype = complex)
40. sys\_G = np.zeros((sys\_Size, sys\_Size))
41. sys\_B = np.zeros((sys\_Size, sys\_Size))
43. #X\_ij
44. **for** i **in** range(sys\_Size): #Row
45. **for** j **in** range(sys\_Size): #Column
46. **if** i==j: # Diagonal, sum of Y(From==i || To==i) + .5B(From==i || To ==i)
47. sys\_Y[i][j] = np.sum(line\_Y[np.array(line\_From==i+1) + np.array(line\_To==i+1)]) \
48. +.5j\*np.sum(line\_B[np.array(line\_From==i+1) + np.array(line\_To==i+1)])
49. **elif** i<j: #Non Diagonal, -Y(From==i && To==j)
50. sys\_Y[i][j] = -np.sum(line\_Y[np.multiply(np.array(line\_From==i+1), np.array(line\_To==j+1))])
51. **else**: #i>j =[j][i]
52. sys\_Y[i][j] = sys\_Y[j][i]
53. sys\_G = sys\_Y.real
54. sys\_B = sys\_Y.imag
55. **return** sys\_Y, sys\_G, sys\_B
57. """
58. Parses intial bus information from data
59. Takes in Bus Data data frame
60. Returns sys\_:
61. LoadP - active power consumed at node
62. LoadQ - reactive power consumed at node
63. BusType - type of bus<(S)lack, (G)enerator, (D)rain>
64. PGen - Active Power produced by each generator node
65. VRef - Reference voltages at PV busses
66. """
67. **def** init\_BusData(BusData):
68. col = np.array(BusData.columns)
69. sys\_BusNum = np.array(BusData[col[0]])
70. sys\_LoadP = np.array(BusData[col[1]])
71. sys\_LoadQ = np.array(BusData[col[2]])
72. sys\_BusType = np.array(BusData[col[3]])
73. sys\_PGen = np.array(BusData[col[4]])
74. sys\_VRef = np.array(BusData[col[5]])
75. **return** sys\_BusNum, sys\_LoadP, sys\_LoadQ, sys\_BusType, sys\_PGen, sys\_VRef
77. """
78. Initializes System Data for processing
79. Takes in sys\_:
80. LoadP - active power consumed at node
81. LoadQ - reactive power consumed at node
82. BusType - type of bus<(S)lack, (G)enerator, (D)rain>
83. PGen - Active Power produced by each generator node
84. VRef - Reference voltages at PV busses
85. Returns a 2D array containing each buses's current information
86. [i,:] - Bus i's information
87. [:,0] - Bus #
88. [:,1] - Voltage (V)
89. [:,2] - Angle (T)
90. [:,3] - Active Power (P\_inj)
91. [:,4] - P(T,V)-P\_inj (mismatch)
92. [:,5] - Reactive Power (Q\_inj)
93. [:,6] - Q(T,V)-Q\_inj (mismatch)
94. """
95. **def** init\_SysData(sys\_BusNum, sys\_LoadP, sys\_LoadQ, sys\_BusType, sys\_PGen, sys\_VRef, sys\_G, sys\_B, S\_Base):
96. n= sys\_LoadP.size
97. sys\_Data = np.zeros((n,7))
98. sys\_Data[:,0] = sys\_BusNum
99. sys\_Data[:,1] = sys\_VRef #Sets initial voltages to provided reference
100. sys\_Data[:,2] = np.zeros(n) #Sets initial angles to zero
101. sys\_Data[:,3] = (sys\_PGen-sys\_LoadP)/S\_Base #Sets initial power inject to Bus generation minus load in per unit
102. sys\_Data[sys\_BusType=='S',3] = (np.sum(sys\_LoadP)-np.sum(sys\_PGen))/S\_Base #Sets initial guess for active power required from slack bus
103. sys\_Data[:,5] = (-sys\_LoadQ)/S\_Base #Sets initial power inject to Bus generation minus load in per unit
104. sys\_Data[sys\_BusType=='S',5] = (-np.sum(sys\_LoadQ))/S\_Base #Sets initial guess for reactive power required from slack bus
105. **for** i **in** range(n): #Sets initial mismatch to calculated power from (V,T) minus expected inject
106. sys\_Data[i,4] = -sys\_Data[i,3]
107. sys\_Data[i,6] = -sys\_Data[i,5]
108. **for** j **in** range(n):
109. sys\_Data[i,4] += sys\_Data[i,1]\*sys\_Data[j,1]\*\
110. (sys\_G[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2])+\
111. sys\_B[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2]))
112. sys\_Data[i,6] += sys\_Data[i,1]\*sys\_Data[j,1]\*\
113. (sys\_G[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2])-\
114. sys\_B[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2]))
116. **return** sys\_Data
118. """
119. Determines Jacobian value for a given J\_11 cell (dP/dT)
120. Takes in: i, j, n, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij
121. Returns Jacobian cell value
122. """
123. **def** Jacobian\_PowerFlow\_11(i, j, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij):
124. **if**(i==j):
125. J\_ij = -Q\_i - B\_ij\*(V\_i\*\*2)
126. **else**:
127. J\_ij =  V\_i\*V\_j\*(G\_ij\*np.sin(T\_i-T\_j)-B\_ij\*np.cos(T\_i-T\_j))
128. **return** J\_ij
130. """
131. Determines Jacobian value for a given J\_12 cell (dP/dV)
132. Takes in: i, j, n, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij
133. Returns Jacobian cell value
134. """
135. **def** Jacobian\_PowerFlow\_12(i, j, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij):
136. **if**(i==j):
137. J\_ij = (P\_i/V\_i) + G\_ij\*V\_i
138. **else**:
139. J\_ij = V\_i\*(G\_ij\*np.cos(T\_i-T\_j)+B\_ij\*np.sin(T\_i-T\_j))
140. **return** J\_ij
142. """
143. Determines Jacobian value for a given J\_21 cell (dQ/dT)
144. Takes in: i, j, n, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij
145. Returns Jacobian cell value
146. """
147. **def** Jacobian\_PowerFlow\_21(i, j, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij):
148. **if**(i==j):
149. J\_ij = P\_i-G\_ij\*(V\_i\*\*2)
150. **else**:
151. J\_ij = -V\_i\*V\_j\*(G\_ij\*np.cos(T\_i-T\_j)+B\_ij\*np.sin(T\_i-T\_j))
152. **return** J\_ij
154. """
155. Determines Jacobian value for a given J\_22 cell (dQ/dV)
156. Takes in: i, j, n, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij
157. Returns Jacobian cell value
158. """
159. **def** Jacobian\_PowerFlow\_22(i, j, V\_i, V\_j, T\_i, T\_j, P\_i, Q\_i, G\_ij, B\_ij):
160. **if**(i==j):
161. J\_ij = (Q\_i/V\_i)-B\_ij\*V\_i
162. **else**:
163. J\_ij = V\_i\*(G\_ij\*np.sin(T\_i-T\_j)-B\_ij\*np.cos(T\_i-T\_j))
164. **return** J\_ij

167. """
168. Processes 1 iteration of current system data
169. Takes in sys\_Data, a 2D array containing each node's current information
170. [0] - Bus #
171. [1] - Voltage (V)
172. [2] - Angle (T)
173. [3] - Active Power (P\_inj)
174. [4] - P(T,V)-P\_inj (mismatch)
175. [5] - Reactive Power (Q\_inj)
176. [6] - Q(T,V)-Q\_inj (mismatch)
177. As well as, the systems G and B matrices, and node types
178. Returns the updated array
179. """
180. **def** update\_SysData(sys\_Data, sys\_G, sys\_B, sys\_BusType):
181. n = sys\_BusType.size
182. D\_index = sys\_BusType=='D'
183. G\_index = sys\_BusType=='G'
184. S\_index = sys\_BusType=='S'
186. """Determine Jacobian"""
187. J = np.zeros((2\*n,2\*n))
188. **for** i **in** range(n):
189. **for** j **in** range(n):                  #(i, j, V\_i,           V\_j,           T\_i,           T\_j,           P\_i(T,V),                    Q\_i(T,V),                    G\_ij,       B\_ij)
190. J[i,j] =    Jacobian\_PowerFlow\_11(i, j, sys\_Data[i,1], sys\_Data[j,1], sys\_Data[i,2], sys\_Data[j,2], sys\_Data[i,4]+sys\_Data[i,3], sys\_Data[i,6]+sys\_Data[i,5], sys\_G[i,j], sys\_B[i,j])
191. J[i,j+n] =  Jacobian\_PowerFlow\_12(i, j, sys\_Data[i,1], sys\_Data[j,1], sys\_Data[i,2], sys\_Data[j,2], sys\_Data[i,4]+sys\_Data[i,3], sys\_Data[i,6]+sys\_Data[i,5], sys\_G[i,j], sys\_B[i,j])
192. J[i+n,j] =  Jacobian\_PowerFlow\_21(i, j, sys\_Data[i,1], sys\_Data[j,1], sys\_Data[i,2], sys\_Data[j,2], sys\_Data[i,4]+sys\_Data[i,3], sys\_Data[i,6]+sys\_Data[i,5], sys\_G[i,j], sys\_B[i,j])
193. J[i+n,j+n] =Jacobian\_PowerFlow\_22(i, j, sys\_Data[i,1], sys\_Data[j,1], sys\_Data[i,2], sys\_Data[j,2], sys\_Data[i,4]+sys\_Data[i,3], sys\_Data[i,6]+sys\_Data[i,5], sys\_G[i,j], sys\_B[i,j])
195. """Remove non-implicit values"""
196. **for** i **in** range(n-1,-1,-1):
197. **if** S\_index[i]:
198. J=np.delete(J, i+n, 0)
199. J=np.delete(J, i+n, 1)
200. J=np.delete(J, i, 0)
201. J=np.delete(J, i, 1)
202. **elif** G\_index[i]:
203. J=np.delete(J, i+n, 0)
204. J=np.delete(J, i+n, 1)
206. """Determine Inverse"""
207. J\_inv = inv(J)
209. """Determine Delta T,V"""
210. PQ = np.concatenate((sys\_Data[np.invert(S\_index), 4], sys\_Data[D\_index, 6]))
211. Delta = -J\_inv @ PQ
212. Delta\_T = Delta[0:sum(np.invert(S\_index))]
213. Delta\_V = Delta[sum(np.invert(S\_index)):sum(np.invert(S\_index))+sum(D\_index)]
214. """Update T for non-slack buses, and V for PQ buses"""
215. Delta\_T\_index = 0
216. Delta\_V\_index = 0
217. **for** i **in** range(n):
218. **if** G\_index[i]:
219. sys\_Data[i,2] += Delta\_T[Delta\_T\_index]
220. Delta\_T\_index += 1
221. **elif** D\_index[i]:
222. sys\_Data[i,1] += Delta\_V[Delta\_V\_index]
223. Delta\_V\_index += 1
224. sys\_Data[i,2] += Delta\_T[Delta\_T\_index]
225. Delta\_T\_index += 1
227. """Update P\_inj for slack bus, and Q\_inj for non PQ buses"""
228. **for** i **in** range(n):
229. **if** S\_index[i]:#Update Slack P\_inj
230. sys\_Data[i,3] = 0
231. **if** (S\_index[i] **or** G\_index[i]):#Update non PQ Q\_inj
232. sys\_Data[i,5] = 0
233. **for** j **in** range(n):
234. **if** S\_index[i]:#Update Slack
235. sys\_Data[i,3] += sys\_Data[i,1]\*sys\_Data[j,1]\*((sys\_G[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2]))+(sys\_B[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2])))
236. **if** (S\_index[i] **or** G\_index[i]):#Update non PQ
237. sys\_Data[i,5] += sys\_Data[i,1]\*sys\_Data[j,1]\*((sys\_G[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2]))-(sys\_B[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2])))
239. """Update mismatch columns"""
240. **for** i **in** range(n):
241. sys\_Data[i,4] = -sys\_Data[i,3]
242. sys\_Data[i,6] = -sys\_Data[i,5]
243. **for** j **in** range(n):
244. sys\_Data[i,4] += sys\_Data[i,1]\*sys\_Data[j,1]\*((sys\_G[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2]))+(sys\_B[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2])))
245. sys\_Data[i,6] += sys\_Data[i,1]\*sys\_Data[j,1]\*((sys\_G[i,j]\*np.sin(sys\_Data[i,2]-sys\_Data[j,2]))-(sys\_B[i,j]\*np.cos(sys\_Data[i,2]-sys\_Data[j,2])))
247. **return** sys\_Data


251. """
252. Takes in voltage and theta values, shunt capacitance, and the admittance matrix
253. Returns Power Values:
254. S\_ij - Apparent Power
255. P\_ij - Real Power
256. Q\_ij - Reactive Power
257. """
258. **def** PowerFlow (V\_i,T\_i,V\_j,T\_j,B\_tot,y\_ij):
259. I\_ij = y\_ij \* (V\_i \* np.cos(T\_i) + 1j \* V\_i \* np.sin(T\_i) - V\_j \* np.cos(T\_j) - 1j \* V\_j
260. \* np.sin(T\_j)) + (1j\*B\_tot / 2) \* (V\_i \* np.cos(T\_i) + 1j \* V\_i \* np.sin(T\_i))
261. S\_ij = (V\_i\*np.cos(T\_i)+1j\*V\_i\*np.sin(T\_i)) \* (I\_ij.conjugate())
262. **return** abs(S\_ij), S\_ij.real, S\_ij.imag
264. """
265. Takes in matrices sys\_Data, LineData, and sys\_Y
266. Returns lists:
267. i Bus # (i\_buses)
268. j Bus # (j\_buses)
269. Apparent Power (S\_ij)
270. Active Power (P\_ij)
271. Reactive Power (Q\_ij)
272. Violation Occurrence (violation)
273. """
274. **def** LineFlowResults (sys\_Data, LineData, sys\_Y):
275. LD\_val = LineData.values
276. S\_ij = []
277. P\_ij = []
278. Q\_ij = []
279. i\_buses = LD\_val[0:,0]
280. j\_buses = LD\_val[0:,1]
281. violation = []
282. **for** i **in** range (0, len(LineData)):
283. B\_tot = (LD\_val[i:i+1,4])
284. V\_i = sys\_Data[(int(LD\_val[i:i+1,0]))-1:(int(LD\_val[i:i+1,0])),1]
285. T\_i = sys\_Data[(int(LD\_val[i:i + 1, 0])) - 1:(int(LD\_val[i:i + 1, 0])), 2]
286. V\_j = sys\_Data[(int(LD\_val[i:i + 1, 1])) - 1:(int(LD\_val[i:i + 1, 1])), 1]
287. T\_j = sys\_Data[(int(LD\_val[i:i + 1, 1])) - 1:(int(LD\_val[i:i + 1, 1])), 2]
288. y\_ij = -1 \* ( sys\_Y[(int(LD\_val[i:i + 1, 0]) - 1):(int(LD\_val[i:i + 1, 0])),
289. (int(LD\_val[i:i + 1, 1]) - 1)])
291. PowerFlow(V\_i, T\_i, V\_j, T\_j, B\_tot, y\_ij)
292. s\_ij, p\_ij, q\_ij = PowerFlow(V\_i,T\_i,V\_j,T\_j,B\_tot,y\_ij)
294. **if** s\_ij\*100 < (LD\_val[i:i+1,5]):
295. violation.append('FALSE')
296. **else**:
297. violation.append('TRUE')
298. S\_ij.append(100 \* float(s\_ij))
299. P\_ij.append(100 \* float(p\_ij))
300. Q\_ij.append(100 \* float(q\_ij))
301. **return** S\_ij, P\_ij, Q\_ij, violation, i\_buses, j\_buses
303. """
304. Collects needed bus data from sys\_Data
305. Returns lists:
306. Bus Number (bus\_nums)
307. Bus Voltages (bus\_v)
308. Bus Thetas (bus\_deg)
309. Bus Active Power (bus\_p)
310. Bus Reactive Power (bus\_q)
311. Reactive Power (Q\_ij)
312. Voltage Violation Occurrence (V\_violate)
313. """
314. **def** BusResults(sys\_Data):
315. V\_violate = []
316. **for** i **in** range (0,len(sys\_Data)):
317. **if**  (sys\_Data[i:i+1,1] <= 1.05 **and** sys\_Data[i:i+1,1] >= 0.95):
318. V\_violate.append('FALSE')
319. **else**:
320. V\_violate.append('TRUE')
321. bus\_nums = sys\_Data[0:,0]
322. bus\_v = sys\_Data[0:,1]
323. bus\_deg = 180\* sys\_Data[0:,2] / np.pi
324. bus\_p = 100 \* sys\_Data[0:,3]
325. bus\_q = 100 \* sys\_Data[0:,5]
326. **return** bus\_nums.astype(int), bus\_v, bus\_deg, bus\_p, bus\_q, V\_violate
328. """
329. Collects the filename, sys\_Data, LineData, sys\_Y, list of iterations, lists of max P mismatches and it's bus number,
330. lists of max q mismatches and it's bus number
331. Creates excel file with the given file name and given sheetnames:
332. [0] - BusData - Data of each bus
333. [1] - LineData - Data of the line between buses
334. [2] - ConvergenceHistory - History of Convergence
335. [3] - Y\_matrix(Admittance) - Admittance matrix
336. [4] - G\_matrix - Real part of the Admittance matrix
337. [5] - B\_matrix - Imag. part of the Admittance matrix
338. """
339. **def** DataOutput(FileName, sys\_Data, LineData, sys\_Y, iteration\_list, mismatch\_P\_list, mismatch\_Q\_list, max\_P\_bus, max\_Q\_bus):
341. y\_matrix = pd.DataFrame(data=sys\_Y)
342. g\_matrix = pd.DataFrame(data=sys\_Y.real)
343. b\_matrix = pd.DataFrame(data=sys\_Y.imag)
345. bus\_nums, bus\_v, bus\_deg, bus\_p, bus\_q, V\_violate = BusResults(sys\_Data)
346. S\_ij, P\_ij, Q\_ij, S\_violation, i\_buses, j\_buses = LineFlowResults(sys\_Data, LineData, sys\_Y)
348. df\_Convergence = pd.DataFrame({'Iteration': iteration\_list, 'Max P Misr': mismatch\_P\_list, 'P Bus': max\_P\_bus
349. , 'Max Q Misr': mismatch\_Q\_list, 'Q Bus': max\_Q\_bus})
351. df\_BusOutput = pd.DataFrame({'Bus Number':bus\_nums,'V (pu)':bus\_v,'Angle (deg)':bus\_deg
352. ,'P inj. (MW)':bus\_p,'Q inj. (MVar)':bus\_q,
353. 'Voltage Violation':V\_violate})
355. df\_LineOutput = pd.DataFrame({'From Bus': i\_buses, 'To Bus': j\_buses, 'P Flow (MW)': P\_ij, 'Q Flow (MVar)': Q\_ij, 'S Flow (MVA)': S\_ij,
356. 'Line MVA Violation': S\_violation})
358. writer = pd.ExcelWriter(FileName+".xlsx", engine='xlsxwriter')
360. df\_BusOutput.to\_excel(writer, sheet\_name='BusData',startrow=1,header=False,index=False)
361. df\_LineOutput.to\_excel(writer, sheet\_name='LineData', startrow=1, header=False, index=False)
362. df\_Convergence.to\_excel(writer, sheet\_name='ConvergenceHistory', startrow=1, header=False, index=False)
363. y\_matrix.to\_excel(writer, sheet\_name='Y\_matrix(Admittance)', startrow=0,header=False,index=False)
364. g\_matrix.to\_excel(writer, sheet\_name='G\_matrix', startrow=0,header=False,index=False)
365. b\_matrix.to\_excel(writer, sheet\_name='B\_matrix', startrow=0,header=False,index=False)
367. workbook = writer.book
368. busworksheet = writer.sheets['BusData']
369. lineworksheet = writer.sheets['LineData']
370. convergencesheet = writer.sheets['ConvergenceHistory']
372. header\_format = workbook.add\_format({'bold': True,
373. 'text\_wrap': True,
374. 'valign': 'top',
375. 'fg\_color': '#D7E4BC',
376. 'border': 1})
378. **for** col\_num, value **in** enumerate(df\_BusOutput.columns.values):
379. busworksheet.write(0, col\_num,value,header\_format)
380. **for** col\_num, value **in** enumerate(df\_LineOutput.columns.values):
381. lineworksheet.write(0, col\_num, value, header\_format)
382. **for** col\_num, value **in** enumerate(df\_Convergence.columns.values):
383. convergencesheet.write(0, col\_num, value, header\_format)
385. writer.save()
387. **def** PowerFlowAnalysis(BusData\_Location, LineData\_Location, Output\_FileName, tolerance, S\_Base):
388. """Data Frame creation for initial Bus and Line Data"""
389. df\_BusData, df\_LineData = import\_BusAndLineData(BusData\_Location, LineData\_Location)
390. n = df\_BusData.shape[0]
391. """Create Admittance Matrix in forms of Y and seperated into G and B"""
392. sys\_Y, sys\_G, sys\_B = build\_AdmittanceMatrix(df\_LineData, n)
393. """Creation of sys\_Data"""
394. sys\_BusNum, sys\_LoadP, sys\_LoadQ, sys\_BusType, sys\_PGen, sys\_VRef = init\_BusData(df\_BusData)
395. sys\_Data = init\_SysData(sys\_BusNum, sys\_LoadP, sys\_LoadQ, sys\_BusType, sys\_PGen, sys\_VRef, sys\_G, sys\_B, S\_Base)
396. """Initial Prime for mismatch detetction and storage"""
397. mismatch\_P = sys\_Data[1:n,4]
398. mismatch\_Q = sys\_Data[1:n,6]
399. mismatch\_max = [max(abs(mismatch\_P)), max(abs(mismatch\_Q))]
400. iteration = 0
401. iteration\_list = []
402. mismatch\_P\_list = []
403. mismatch\_Q\_list = []
404. max\_P\_bus = []
405. max\_Q\_bus = []
407. """Loop until solution is reached or max iteration is exceeded"""
408. **while**(iteration<15 **and** mismatch\_max>tolerance):
409. iteration\_list.append(iteration)
411. bus\_P, = np.where(mismatch\_P == max(abs(mismatch\_P)))
412. **if** len(bus\_P) == 0:
413. bus\_P, = np.where(mismatch\_P == -1\*max(abs(mismatch\_P)))
414. max\_P\_bus.append(int(bus\_P+2))
415. bus\_Q, = np.where(mismatch\_Q == max(abs(mismatch\_Q)))
416. **if** len(bus\_Q) == 0:
417. bus\_Q, = np.where(mismatch\_Q == -1\*max(abs(mismatch\_Q)))
418. max\_Q\_bus.append(int(bus\_Q+2))
419. mismatch\_P\_list.append(max(abs(mismatch\_P)))
420. mismatch\_Q\_list.append(max(abs(mismatch\_Q)))
422. sys\_Data = update\_SysData(sys\_Data, sys\_G, sys\_B, sys\_BusType)
423. mismatch\_P = sys\_Data[1:n,4]
424. mismatch\_Q = sys\_Data[1:n,6]
425. mismatch\_max = [max(abs(mismatch\_P)), max(abs(mismatch\_Q))]
426. iteration += 1
428. """Final add to convergency history"""
429. iteration\_list.append(iteration)
430. bus\_P, = np.where(mismatch\_P == max(abs(mismatch\_P)))
431. **if** len(bus\_P) == 0:
432. bus\_P, = np.where(mismatch\_P == -1\*max(abs(mismatch\_P)))
433. max\_P\_bus.append(int(bus\_P+2))
434. bus\_Q, = np.where(mismatch\_Q == max(abs(mismatch\_Q)))
435. **if** len(bus\_Q) == 0:
436. bus\_Q, = np.where(mismatch\_Q == -1\*max(abs(mismatch\_Q)))
437. max\_Q\_bus.append(int(bus\_Q+2))
438. mismatch\_P\_list.append(max(abs(mismatch\_P)))
439. mismatch\_Q\_list.append(max(abs(mismatch\_Q)))
441. """Export final solution to excel file"""
442. DataOutput(Output\_FileName, sys\_Data, df\_LineData, sys\_Y,iteration\_list,mismatch\_P\_list,mismatch\_Q\_list,max\_P\_bus,max\_Q\_bus)