6 Dec 2019
EE 454
Brandon Beaty
Eliot Nichols
Project 2 Report

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.
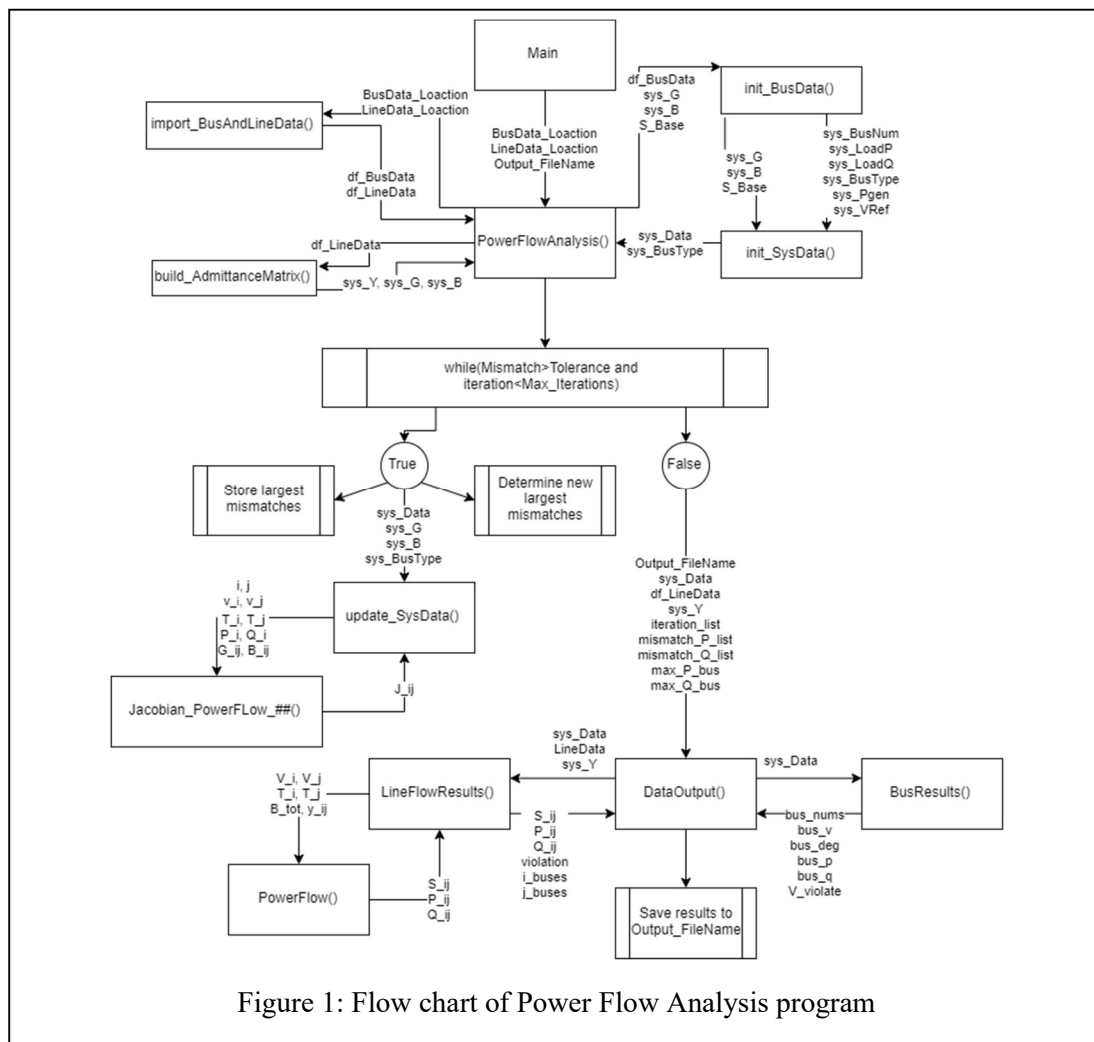

Figure 1: Flow chart of Power Flow Analysis program

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

Bus Number	V (pu)	Angle (deg)	P inj. (MW)	Q inj. (MVar)	Voltage Violation
1	1.05	0	145.6856	-26.0757	FALSE
2	1.045	-3.28899	18.3	26.91573	FALSE
3	1.01	-9.40654	-61.2	2.797827	FALSE
4	1.026686	-6.44693	-57.8	3.9	FALSE
5	1.030416	-5.44317	-7.6	-1.6	FALSE
6	1.023128	-7.93228	-13.5	-8.5	FALSE
7	0.9899	-10.6247	-29.5	-13.6	FALSE
8	0.98714	-10.4719	-9	-5.8	FALSE
9	0.999982	-9.37929	-4.3	-2.1	FALSE
10	1.06	-6.81345	27.8	11.53806	TRUE
11	1.024521	-8.13704	-13.5	-5.8	FALSE
12	1.04	-9.24019	12.1	18.43624	FALSE

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.

From Bus	To Bus	P Flow (MW)	Q Flow (MVar)	S Flow (MVA)	Line MVA Violation
1	2	99.61155	-23.6093	102.3712	TRUE
1	5	46.07401	-2.46643	46.13998	FALSE
2	3	58.61897	5.205768	58.84967	FALSE
2	4	33.73802	-1.19732	33.75926	FALSE
2	5	23.7444	-0.46364	23.74893	FALSE
3	4	-4.08192	6.295154	7.502737	FALSE
4	5	-42.4995	4.764792	42.76578	FALSE
4	7	13.31228	7.276089	15.17097	FALSE
5	6	18.16744	3.374567	18.47819	FALSE
6	9	15.27092	4.777734	16.00087	FALSE
6	10	-12.4507	-8.6842	15.18007	FALSE
6	11	1.846623	-2.02636	2.741556	FALSE
7	8	-1.63333	3.852514	4.184453	FALSE
7	12	-14.5541	-11.389	18.48059	FALSE
8	9	-10.6386	-1.96216	10.81799	FALSE
10	11	15.08206	2.290741	15.25503	FALSE
11	12	2.968071	-5.95796	6.656329	FALSE

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

Iteration	Max P Misr	P Bus	Max Q Misr	Q Bus
0	0.586028	3	0.642762	2
1	0.034927	2	0.045026	5
2	0.000156	5	0.00016	5

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

Contingency 1

Bus Number	V (pu)	Angle (deg)	P inj. (MW)	Q inj. (MVar)	Voltage Violation
1	1.05	0	147.2691	-25.2576	FALSE
2	1.045	-5.77589	18.3	33.12146	FALSE
3	1.01	-11.8486	-61.2	4.033702	FALSE
4	1.027447	-8.34706	-57.8	3.9	FALSE
5	1.030785	-7.18776	-7.6	-1.6	FALSE
6	1.023313	-9.70917	-13.5	-8.5	FALSE
7	0.990003	-12.4463	-29.5	-13.6	FALSE
8	0.987246	-12.2855	-9	-5.8	FALSE
9	1.000113	-11.1745	-4.3	-2.1	FALSE
10	1.06	-8.58742	27.8	11.41097	TRUE
11	1.024605	-9.91867	-13.5	-5.8	FALSE
12	1.04	-11.0418	12.1	18.37847	FALSE

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.

From Bus	To Bus	P Flow (MW)	Q Flow (MVar)	S Flow (MVA)	Line MVA Violation
1	2	86.97579	-20.7989	89.42809	TRUE
1	5	60.29327	-4.45868	60.4579	FALSE
2	3	58.21651	5.257021	58.45339	FALSE
2	4	27.91578	-0.04022	27.91581	FALSE
2	5	16.36755	1.48101	16.43441	FALSE
3	4	-4.46377	7.666111	8.87099	FALSE
4	5	-48.4361	7.724335	49.04811	FALSE
4	7	13.07348	7.384894	15.01507	FALSE
5	6	18.41296	3.461424	18.73549	FALSE
6	9	15.42169	4.739988	16.13369	FALSE
6	10	-12.4414	-8.61682	15.13399	FALSE
6	11	1.931911	-1.99082	2.774102	FALSE
7	8	-1.77747	3.904416	4.289971	FALSE
7	12	-14.6488	-11.3054	18.50403	FALSE
8	9	-10.783	-1.91098	10.95098	FALSE
10	11	15.09363	2.234642	15.25815	FALSE
11	12	3.064908	-5.97884	6.718643	FALSE

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

Iteration	Max P Misr	P Bus	Max Q Misr	Q Bus
0	0.586028	3	0.697052	2
1	0.03508	2	0.058125	5
2	0.000211	5	0.000262	5

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

Contingency 2

Bus Number	V (pu)	Angle (deg)	P inj. (MW)	Q inj. (MVar)	Voltage Violation
1	1.05	0	146.1084	-25.4523	FALSE
2	1.045	-3.30292	18.3	28.7638	FALSE
3	1.01	-9.44286	-61.2	2.965406	FALSE
4	1.025269	-6.46012	-57.8	3.9	FALSE
5	1.028827	-5.43151	-7.6	-1.6	FALSE
6	1.015803	-7.83903	-13.5	-8.5	FALSE
7	0.988043	-11.0063	-29.5	-13.6	FALSE
8	0.984175	-10.7768	-9	-5.8	FALSE
9	0.994724	-9.4945	-4.3	-2.1	FALSE
10	1.06	-4.42512	27.8	5.701583	TRUE
11	1.010304	-9.01272	-13.5	-5.8	FALSE
12	1.04	-10.0433	12.1	23.28966	FALSE

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.

From Bus	To Bus	P Flow (MW)	Q Flow (MVar)	S Flow (MVA)	Line MVA Violation
1	2	100.0258	-23.719	102.7995	TRUE
1	5	46.0826	-1.73325	46.11519	FALSE
2	3	58.81998	5.18029	59.04766	FALSE
2	4	33.93844	-0.42501	33.9411	FALSE
2	5	23.742	0.481022	23.74687	FALSE
3	4	-3.89086	6.395161	7.485777	FALSE
4	5	-43.2405	5.447717	43.58236	FALSE
4	7	14.43659	7.435188	16.23876	FALSE
5	6	17.41949	5.683032	18.32309	FALSE
6	9	16.21961	3.232004	16.53849	FALSE
6	10	-26.9234	-3.86746	27.19973	FALSE
6	11	14.62643	-2.97445	14.92581	FALSE
7	8	-2.54199	5.488936	6.048976	FALSE
7	12	-12.5206	-13.0466	18.08257	FALSE
8	9	-11.5529	-0.34215	11.55798	FALSE
11	12	0.98303	-9.05463	9.107835	FALSE

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

Iteration	Max P Misr	P Bus	Max Q Misr	Q Bus
0	0.586028	3	0.642762	2
1	0.034186	2	0.043841	5
2	0.000147	5	0.000166	5

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 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

```python
"""
Power Flow Analysis: Main Function
Created By:
    Brandon Beaty
    Eliot Nichols
"""
from PowerSysAnalysis_HeaderFile import *

"""
########################
    Main Section of Code
########################
"""

"""
Loction for Bus and line data
First variable is for file location, Second variable is for sheet name
"""
BusData_Location = ['system_basecase.xlsx', 'BusData'] #BaseCase Data
LineData_Location_BaseCase = ['system_basecase.xlsx', 'LineData'] #BaseCase Data
LineData_Location_Contingency1 = ['system_basecase.xlsx', 'ContingencyCase1'] #BaseCase
    Data
LineData_Location_Contingency2 = ['system_basecase.xlsx', 'ContingencyCase2'] #BaseCase
    Data

tolerance = [.001, .001] #P.U.
S_Base = 100 #MW

PowerFlowAnalysis(BusData_Location, LineData_Location_BaseCase, "Base Case Solution", t
    olerance, S_Base)
PowerFlowAnalysis(BusData_Location, LineData_Location_Contingency1, "Contingency Case 1
     Solution", tolerance, S_Base)
PowerFlowAnalysis(BusData_Location, LineData_Location_Contingency2, "Contingency Case 2
     Solution", tolerance, S_Base)
```

Support Files

```python
"""
Power Flow Analysis: Support Functions
Created By:
    Brandon Beaty
    Eliot Nichols
"""

import numpy as np
from numpy.linalg import inv
import pandas as pd


"""
Imports Bus and line data from excel sheets
Takes in an array containing ['File Location', 'Sheet Name']
Returns two panda data frames for the bus and line data
"""
def import_BusAndLineData(BusData_Location, LineData_Location):
```

```python
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
22.
23.
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))
42.
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
56.
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
```

```python
76.
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 bu
       s
105.       for i in range(n): #Sets initial mismatch to calculated power from (V,T) minus expe
       cted 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]))
115.
116.       return sys_Data
117.
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
```

```python
129.
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
141.
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
153.
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
165.
166.
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'
185.
186.        """Determine Jacobian"""
187.        J = np.zeros((2*n,2*n))
188.        for i in range(n):
```

```python
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_D
        ata[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_D
        ata[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_D
        ata[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_D
        ata[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])
194.
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)
205.
206.        """Determine Inverse"""
207.        J_inv = inv(J)
208.
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
226.
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_Da
        ta[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_Da
        ta[i,2]-sys_Data[j,2]))-(sys_B[i,j]*np.cos(sys_Data[i,2]-sys_Data[j,2])))
```

```python
238.
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])))
246.
247.        return sys_Data
248.
249.
250.
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
263.
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)])
290.
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)
293.
294.            if s_ij*100 < (LD_val[i:i+1,5]):
```

```python
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
302.
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
327.
328.     """
329.     Collects the filename, sys_Data, LineData, sys_Y, list of iterations, lists of max P mi
        smatches 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, mi
        smatch_Q_list, max_P_bus, max_Q_bus):
340.
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)
344.
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, LineDat
        a, sys_Y)
347.
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
        })
350.
```

```python
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})
354.
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})
357.
358.      writer = pd.ExcelWriter(FileName+".xlsx", engine='xlsxwriter')
359.
360.        df_BusOutput.to_excel(writer, sheet_name='BusData',startrow=1,header=False,index=Fa
      lse)
361.        df_LineOutput.to_excel(writer, sheet_name='LineData', startrow=1, header=False, ind
      ex=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=Fals
      e,index=False)
364.        g_matrix.to_excel(writer, sheet_name='G_matrix', startrow=0,header=False,index=Fals
      e)
365.        b_matrix.to_excel(writer, sheet_name='B_matrix', startrow=0,header=False,index=Fals
      e)
366.
367.      workbook = writer.book
368.      busworksheet = writer.sheets['BusData']
369.      lineworksheet = writer.sheets['LineData']
370.      convergencesheet = writer.sheets['ConvergenceHistory']
371.
372.      header_format = workbook.add_format({'bold': True,
373.      'text_wrap': True,
374.      'valign': 'top',
375.      'fg_color': '#D7E4BC',
376.      'border': 1})
377.
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)
384.
385.      writer.save()
386.
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, sy
      s_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))]
```

```python
400.        iteration = 0
401.        iteration_list = []
402.        mismatch_P_list = []
403.        mismatch_Q_list = []
404.        max_P_bus = []
405.        max_Q_bus = []
406.
407.        """Loop until solution is reached or max iteration is exceeded"""
408.        while(iteration<15 and mismatch_max>tolerance):
409.            iteration_list.append(iteration)
410.
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)))
421.
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
427.
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)))
440.
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)
```