EE 574

Power System Real-Time Monitoring & Control

Fall 2016

Term Project

Final Report

Erencan Duymaz
1814029

11/01/2017

Abstract

This term project report explains the procedure of a Weighted Least Square Estimator written in MATLAB environment. Details are given in the next sections.

Introduction

State estimation is very important for the ongoing system monitoring and control of the Power System. Since the measurements obtained from the field are erroneous most of the time their identification, removal or correction have great importance. State estimation methods give operator the closest results to the actual system results. Weighted Least Square Estimator is one of these methods. In this project, a state estimation algorithm is needed to be constructed in MATLAB environment. The project is based on two main file. One of them is the data related to the bus connectivity. Second file has the measurement data for this network. The code is supposed to read data from these files. Therefore, genericity is the one of the important feature of this project. Since the commercial products is not constructed for one network, one estimator should be compatible to any network whose connectivity data is given with it.

Procedure

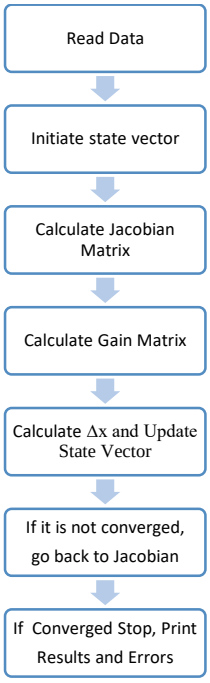


Figure 1:State Estimation Procedure

The simplest procedure of the state estimator is given in the Figure 1. However, all these steps should be investigated in detail.

Reading Data

One of most important features that the state estimator is its being generic. In other words, when the system changes, the new system connectivity should be read by the estimator properly. This can be done with a regular data format. The given data has this feature. For example, the files are given with read_me files which has the information of the connectivity file and the measurement set file.

The system connectivity data is given with 'ieee_cdf.dat' file. This file is composed of two parts. In the first part, bus information is given. Note that this is an IEEE data set, it has various information. Second part of this file has the information of the branches. This data set is read in two different code part.

```
ieee_data = importdata( 'ieee_cdf.dat');
busdata = ieee_data.data;
ieee_data_matrix_size = size(busdata);
busnumber = ieee_data_matrix_size(1,1);
```

By using importdata function, MATLAB takes the first part of the 'ieee_dat.dat' file. However, the taken data is in structure format. Therefore, it is necessary to subtract the data part only. In this way, bus data is read from the file in a matrix whose row number will give us the busnumber.

The branch data is taken with the following code :

```
filename = 'ieee_cdf.dat';
headerlinesIn = busnumber+4;
delimiterIn = ' ';
A = importdata(filename,delimiterIn,headerlinesIn);
Br = A.data;
C = size(Br);
D = C(1,1);
Br(D,:)=[];
fclose('all');
branchdata = Br;
sizebranch = size(branchdata);
branchnumber = sizebranch(1,1);
```

By using the busnumber variable, we exactly know where the branch data starts. By making use of this information, the branch data is also read. Note that branch data ends with '-999' which is a 'message' which says that this is the end of the information. Data can be used in various ways and some of these methods may require such a 'message'. However, in our case it is not necessary. Therefore, it should be disregarded. Then, branch number can be found by using the size of the branchdata matrix.

The measurement file is read in a much easier way since it does not have text file inside. Therefore, dlmread function is used and the taken data is placed in a matrix which is measurement data.

```
measurementdata = dlmread('measure.dat');
```

Note that this file also has a regular format. For example, the first line has the number of voltage magnitude measurements. Therefore, it should be read and directed to n_v variable which is the number of voltage magnitude measurements. Then we should disregard this line. It is

known that after n_v measurements, power injection measurement number is given. Therefore, it should also be read and directed to n_pi variable. In this manner, all the measurement numbers are reached and the measurements are lined up in a way that code can reach any type of measurement with these variables:

n_v: voltage magnitude measurements

n_pi: real power injection measurements

n_qi: reactive power injection measurements

n_pf: active power flow measurements

n_qf: reactive power flow measurements

n_c: current magnitude measurements

```
n_v = measurementdata(1,1); % First line indicates the voltage measurement number.
measurementdata(1,:) = []; % Disregard this line. Voltage measurement number is known anymore.
n_pi = measurementdata(n_v+1,1); % Number of power injection measurement is found. No need to
find n_qi since it is equal to n_pi.
measurementdata(n_v+1,:) = []; % Disregard this line. Power injection measurement number is
known anymore.
measurementdata(n_v+n_pi+1,:) = []; % Disregard this line. Reactive power injection
measurement number is known anymore.
n_pf = measurementdata(n_v+2*n_pi+1,1); % Number of power flow measurement is found. No need
to find n_qf since it is equal to n_pf.
measurementdata(n_v+2*n_pi+1,:) = []; % Disregard this line. Power flow measurement number is
known anymore.
measurementdata(n_v+2*n_pi+n_pf+1,:) = []; % Disregard this line. Reactive power flow
measurement number is known anymore.
n_c = measurementdata(n_v+2*n_pi+2*n_pf+1,1);
measurementdata(n_v+2*n_pi+2*n_pf+1,:) = [];
measurementdata(n_v+2*n_pi+2*n_pf+n_c+1,:) = [];
```

Note that real power measurements are equal to reactive power measurements. Therefore, n_pi=n_qi and n_pf=n_qf.

One last necessary information is the number of the taps. We can reach the number of taps, by using the corresponding column. Following code counts the number of the tapped branches and stores their branch information in order to use them in the following sections.

```
tapnumber = 0;
for count = 1:branchnumber
  if branchdata(count,15) ~=0
      tappedbranches(tapnumber+1,1)=count;
      tapnumber = tapnumber +1;
  end
end
```

Initiating the state vector

After we read the necessary information from the data files, we can initiate our state vector. Our state vector will include voltage magnitudes, phase angles of the buses except for slack bus and the transformer taps whose values will also be updated. Therefore our state vector will have a size of (2*busnumber-1+tapnumber)x1. In the same manner, the size of measurement vector will be measurementnumberx1.

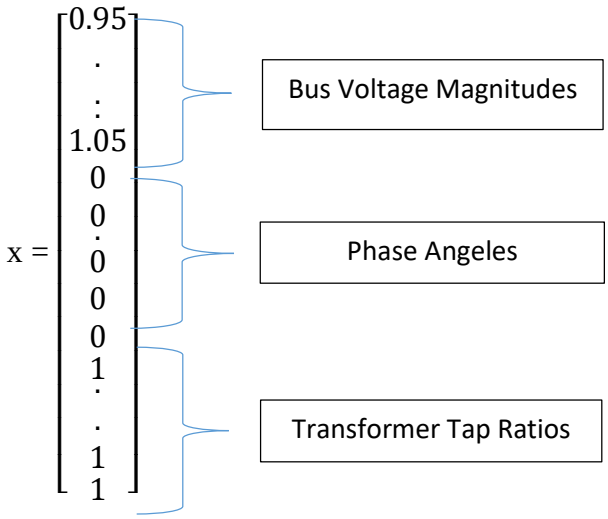
```
x = zeros(2*busnumber-1+tapnumber,1);
for count = 1:busnumber
    x(count,1) = 0.95+(0.1*count)/busnumber;
end
for count = 2*busnumber:2*busnumber+tapnumber-1
    x(count,1) = 1;
end
```

$$x = \begin{bmatrix} |V| \\ \theta \\ a \end{bmatrix}$$

Note that flat start conditions require 1 pu for voltage magnitudes and 0 radians for phase angles but this creates 0 pu line current magnitudes resulting in 0 row in the Jacobian matrix. This creates a problem since Jacobian matrix, H should be positive definite. Therefore, instead of flat start conditions voltages are placed in a range of 0.95 and 1.05. Transformer taps are initialized as 1 pu.

$$z = \begin{bmatrix} |V| \\ Pi \\ Pf \\ Qi \\ Qf \\ |I| \end{bmatrix}$$

Since we also know the measurement number, measurement vector can be created and filled with corresponding measurements. Since we know the measurement numbers, we can reach any measurement by using variables as n_v, n_pi etc. Note that R matrix which consists of diagonal squares of the variances of the corresponding measurement set is filled with the data from measurement data.



Jacobian Matrix

Jacobian matrix, H consists of derivatives of measurements with respect to the state variables. Therefore, H consists of 18 submatrices or parts.

$$H(x) = \begin{bmatrix} \frac{\partial|V1|}{\partial|V1|} & \cdots & \frac{\partial|V1|}{\partial|Vn|} & \frac{\partial|V1|}{\partial\theta 2} & \cdots & \frac{\partial|V1|}{\partial\theta n} & \frac{\partial|V1|}{\partial a1} & \cdots & \frac{\partial|V1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Vnv|}{\partial|V1|} & \cdots & \frac{\partial|Vnv|}{\partial|Vn|} & \frac{\partial|Vnv|}{\partial\theta 2} & \cdots & \frac{\partial|Vnv|}{\partial\theta n} & \frac{\partial|Vn|}{\partial a1} & \cdots & \frac{\partial|Vn|}{\partial ant} \\ \frac{\partial|I1|}{\partial|V1|} & \cdots & \frac{\partial|I1|}{\partial|Vnv|} & \frac{\partial|I1|}{\partial\theta 2} & \cdots & \frac{\partial|I1|}{\partial\theta np} & \frac{\partial|I1|}{\partial a1} & \cdots & \frac{\partial|I1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Inc|}{\partial|V1|} & \cdots & \frac{\partial|Inc|}{\partial|Vnv|} & \frac{\partial|Inc|}{\partial\theta 2} & \cdots & \frac{\partial|Inc|}{\partial\theta np} & \frac{\partial|Inc|}{\partial a1} & \cdots & \frac{\partial|Inc|}{\partial ant} \\ \frac{\partial|Pi1|}{\partial|V1|} & \cdots & \frac{\partial|Pi1|}{\partial|Vnv|} & \frac{\partial|Pi1|}{\partial\theta 2} & \cdots & \frac{\partial|Pi1|}{\partial\theta np} & \frac{\partial|Pi1|}{\partial a1} & \cdots & \frac{\partial|Pi1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Pnpi|}{\partial|V1|} & \cdots & \frac{\partial|Pnpi|}{\partial|Vnv|} & \frac{\partial|Pnpi|}{\partial\theta 2} & \cdots & \frac{\partial|Pnpi|}{\partial\theta np} & \frac{\partial|Pnpi|}{\partial a1} & \cdots & \frac{\partial|Pnpi|}{\partial ant} \\ \frac{\partial|Pf1|}{\partial|V1|} & \cdots & \frac{\partial|Pf1|}{\partial|Vnv|} & \frac{\partial|Pf1|}{\partial\theta 2} & \cdots & \frac{\partial|Pf1|}{\partial\theta np} & \frac{\partial|Pf1|}{\partial a1} & \cdots & \frac{\partial|Pf1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Pnpf|}{\partial|V1|} & \cdots & \frac{\partial|Pnpf|}{\partial|Vnv|} & \frac{\partial|Pnpf|}{\partial\theta 2} & \cdots & \frac{\partial|Pnpf|}{\partial\theta np} & \frac{\partial|Pnpf|}{\partial a1} & \cdots & \frac{\partial|Pnpf|}{\partial ant} \\ \frac{\partial|Qi1|}{\partial|V1|} & \cdots & \frac{\partial|Qi1|}{\partial|Vnv|} & \frac{\partial|Qi1|}{\partial\theta 2} & \cdots & \frac{\partial|Qi1|}{\partial\theta np} & \frac{\partial|Qi1|}{\partial a1} & \cdots & \frac{\partial|Qi1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Qnqi|}{\partial|V1|} & \cdots & \frac{\partial|Qnqi|}{\partial|Vnv|} & \frac{\partial|Qnqi|}{\partial\theta 2} & \cdots & \frac{\partial|Qnqi|}{\partial\theta np} & \frac{\partial|Qnqi|}{\partial a1} & \cdots & \frac{\partial|Qnqi|}{\partial ant} \\ \frac{\partial|Qf1|}{\partial|V1|} & \cdots & \frac{\partial|Qf1|}{\partial|Vnv|} & \frac{\partial|Qf1|}{\partial\theta 2} & \cdots & \frac{\partial|Qf1|}{\partial\theta np} & \frac{\partial|Qf1|}{\partial a1} & \cdots & \frac{\partial|Qf1|}{\partial ant} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial|Qnqf|}{\partial|V1|} & \cdots & \frac{\partial|Qnqf|}{\partial|Vnv|} & \frac{\partial|Qnqf|}{\partial\theta 2} & \cdots & \frac{\partial|Qnqf|}{\partial\theta np} & \frac{\partial|Qnqf|}{\partial a1} & \cdots & \frac{\partial|Qnqf|}{\partial ant} \end{bmatrix}$$

Derivatives of Voltage Magnitudes with respect to the State Variables

In a conventional state estimation, this part is very straightforward. The derivatives of voltage measurement with respect to a voltage is zero unless the measured value is the voltage itself. Therefore:

$$\frac{\partial|Vi|}{\partial|Vi|} = 1 \;\&\; \frac{\partial|Vi|}{\partial|Vj|} = 0$$

The last part of the voltage derivatives are derivatives with respect to taps. If a voltage measurement is incident to a transformer tapped branch, its derivative will be as follows:

$$\frac{\partial|Vi|}{\partial a} = \frac{Vi|}{\partial a}$$

```
for count = 1:n_v
    H(count,measurementdata(count,1))= measurementdata(count,1);
     for count2= 1:tapnumber
        if measurementdata(count,1) == branchdata(tappedbranches(count2,1),1)
        H(count,2*busnumber-1+count2)= x(measuremendata(count,1),1)/x(2*busnumber-1+count2);
        end
        if measurementdata(count,1) == branchdata(tappedbranches(count2,1),2)
        H(count,2*busnumber-1+count2)= x(2*busnumber-1+count2)/x(measuremendata(count,1),1);
```

```
        end
     end
end
```

Derivatives of Active Power Injections with respect to the State Variables

In order to calculate power injection derivatives, bus admittance matrix should be calculated. Note that transformer taps will affect the bus admittance matrix. Therefore, before the iterations, bus admittance matrices will be calculated based on the initial transformer tap values. At the end of each iteration, bus admittance matrices will be updated based on the transformer taps in the state vectors.

```
for count = 1:busnumber

    G(count,count)=busdata(count,15);
    B(count,count)=busdata(count,16);
end
   for counter = 1:branchnumber


G(branchdata2(counter,1),branchdata2(counter,2))=G(branchdata2(counter,1),branchdata2(counter,
2))-branchdata2(counter,10);

B(branchdata2(counter,1),branchdata2(counter,2))=B(branchdata2(counter,1),branchdata2(counter,
2))-branchdata2(counter,11);


G(branchdata2(counter,1),branchdata2(counter,1))=G(branchdata2(counter,1),branchdata2(counter,
1))+branchdata2(counter,10);

B(branchdata2(counter,1),branchdata2(counter,1))=B(branchdata2(counter,1),branchdata2(counter,
1))+branchdata2(counter,11);


G(branchdata2(counter,2),branchdata2(counter,2))=G(branchdata2(counter,2),branchdata2(counter,
2))+branchdata2(counter,10);

B(branchdata2(counter,2),branchdata2(counter,2))=B(branchdata2(counter,2),branchdata2(counter,
2))+branchdata2(counter,11)+branchdata2(counter,12);

   end

   % Upper part of the G and B matrices have been calculated above. Actual matrices
   % are calculated below.

G = G +G'-tril(G,0);
B = B +B'-tril(B,0);
```

Power injection derivatives with respect to voltages and phase angles are independent of the transformer tap changing. They are calculated based on bus admittance matrices which are calculated the current transformer tap values.

$$\frac{\partial P_i}{\partial \theta_i} = \sum_{j=1}^{N} V_i V_j(-G_{ij}\sin\theta_{ij} + B_{ij}\cos\theta_{ij}) - V_i^2 B_{ii}$$

$$\frac{\partial P_i}{\partial \theta_j} = V_i V_j(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij})$$

$$\frac{\partial P_i}{\partial V_i} = \sum_{j=1}^{N} V_j(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) + V_i G_{ii}$$

$$\frac{\partial P_i}{\partial V_j} = V_i(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij})$$

Figure 2: Active Power Injection Derivations [1]

```
for count = n_v+1:n_v+n_pi

    %Vi = x(measurementdata(count,1),1)
    %Vj = x(measurementdata(count,2),1)

    %thetaij value is calculated as follows

    %thetaij = theta(count)-theta(counter)


    for counter = 1:busnumber

     %   delPi/delVi = sum ( Vj(Gij cos thetaij + Bij sin thetaij )+ Vi Gii

           H(count,measurementdata(count,1)) =
sum(x(counter)*(G(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))+...
              B(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))))...

+x(measurementdata(count,1),1)*G(measurementdata(count,1),measurementdata(count,1));

    %   delPi/delVj = Vi(Gij cos thetaij + Bij sin thetaij )

           H(count,counter) = x(measurementdata(count,1),1)*...
        (G(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))+B(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter)));


    %   delPi/delthetai = sum ( ViVj(-Gij sinthetaij+Bij cos thetaij)) -Vi^2 Bii

              if measurementdata(count,1) ~= 1

        H(count,busnumber-1+measurementdata(count,1)) =
sum(x(counter)*x(measurementdata(count,1),1)*...
            (-G(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))+...
            B(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))))-
(x(measurementdata(count,1),1)^2)*B(measurementdata(count,1),measurementdata(count,1));
              end

    %   delPi/delthetaj = ViVj(Gij sin thetaij - Bij cos thetaij )

                  if counter ~= 1
        H(count,busnumber+counter-1) = x(counter)*x(measurementdata(count,1),1)*...
```

```
        (G(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))-B(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter)));
                    end
```

In this part, the expressions are calculated very straightforward. Only challenging part is the derivatives with respect to phase angles. Since our design is containing a slack bus, first bus has been assumed to have zero phase angle. Therefore, we do not have first bus phase angle in the state vector. This is why we should filter our measurement in the derivatives with respect to phase angles. This part is applied with a 'if' function which checks whether the injection is from first bus or not. If so, this value is not calculated.

The active power injection derivative with respect to tap value is equal to power flow derivative with respect to power flow derivative with respect to tap value in the tapped branch. Therefore, this values will be assigned in the power flow derivations part.

$$\frac{\partial P_{tn}}{\partial a} = \frac{\partial P_t}{\partial a}$$

Figure 3: Power Flow and Injection Derivatives Relations [2]

Derivatives of Reactive Power Injections with respect to the State Variables

Reactive power injection derivatives with respect to voltage magnitude, phase angle and tap value are computed in the same manner with the active power injection derivatives part.

$$\begin{aligned}
\frac{\partial Q_{ij}}{\partial \theta_i} &= -V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij}) \\
\frac{\partial Q_{ij}}{\partial \theta_j} &= V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij}) \\
\frac{\partial Q_{ij}}{\partial V_i} &= -V_j (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij}) - 2\, V_i (b_{ij} + b_{si}) \\
\frac{\partial Q_{ij}}{\partial V_j} &= -V_i (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij})
\end{aligned}$$

Figure 4: Reactive Power Injection Derivations [1]

```
for count = n_v+n_pi+1:n_v+2*n_pi

    %Vi = x(measurementdata(count,1),1)
    %Vj = x(measurementdata(count,2),1)

    %thetaij value is calculated as follows
```

```
    %thetaij = theta(count)-theta(counter)


    for counter = 1:busnumber

     %   delQi/delVi = sum ( Vj(Gij sin thetaij - Bij cos thetaij ) - Vi Bii


       H(count,measurementdata(count,1)) =
sum(x(counter)*(G(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))-B(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))))-
x(measurementdata(count,1),1)*B(measurementdata(count,1),measurementdata(count,1));

    %   delQi/delVj = Vi(Gij sin thetaij - Bij cos thetaij )

       H(count,counter) = x(measurementdata(count,1),1)*...
        (G(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))-...
        B(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter)));

    %   delQi/delthetai = sum ( ViVj(Gij costhetaij+Bij sin thetaij)) -Vi^2 Gii

                    if measurementdata(count,1) ~= 1
       H(count,busnumber-1+measurementdata(count,1)) =
sum(x(counter)*x(measurementdata(count,1),1)*(G(measurementdata(count,1),counter)*cos(theta(me
asurementdata(count,1))-
theta(counter))+B(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))))-
(x(measurementdata(count,1),1)^2)*G(measurementdata(count,1),measurementdata(count,1));
                    end


    %   delQi/delthetaj = ViVj(-Gij cos thetaij - Bij sin thetaij )


                    if counter ~= 1
            H(count,busnumber+counter-1) = x(counter)*x(measurementdata(count,1),1)*...
        (-G(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))-B(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter)));
                    End
```

Here, derivative with respect to tap value will also be assigned in the reactive power flow derivation part.

$$\frac{\partial Q_{nt}}{\partial a} = \frac{\partial Q_n}{\partial a}$$

Figure 5: Power injection and Power Flow Derivatives Relations [2]

Derivatives of Active Power Flows with respect to the State Variables

This project's state estimator is more challenging from conventional state estimation is because of the tap inclusion in the state vector. Since tap values also converge to its actual values, our model is updated in each iteration.

While computing the active power flow derivatives with respect to voltage and phase angles are straightforward since the system is fixed unless the derivative is with respect to tap value.

Therefore, for the derivatives with respect to voltage magnitudes and phase angles, we can use following expressions.

$$\frac{\partial P_{ij}}{\partial \theta_i} = V_i V_j (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij})$$
$$\frac{\partial P_{ij}}{\partial \theta_j} = -V_i V_j (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij})$$
$$\frac{\partial P_{ij}}{\partial V_i} = -V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij}) + 2\,(g_{ij} + g_{si}) V_i$$
$$\frac{\partial P_{ij}}{\partial V_j} = -V_i (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij})$$

Figure 6: Active Power Flow Derivations [1]

Since the code updates line conductance, susceptance and shunt admittances, this part can be computed as follows.

```
for count2 = 1:branchnumber
        if measurementdata(count,1) == branchdata(count2,1) && measurementdata(count,2) ==
branchdata(count2,2) || ...
                measurementdata(count,2) == branchdata(count2,1) && measurementdata(count,1)
== branchdata(count2,2)

    %Vi = x(measurementdata(count,1),1)
    %Vj = x(measurementdata(count,2),1)
    %gij = branchdata(count2,10)
    %bij = branchdata(count2,11)

    % delPij/Vi = 2*Vi(gij)-Vj(gij cos theta(ij) + bij sin theta ij )

    H(count,measurementdata(count,1)) = (2*x(measurementdata(count,1),1))*...
        branchdata2(count2,10)-((x(measurementdata(count,2),1))*...
        branchdata2(count2,10)*cos(thetaij)+...
        (branchdata2(count2,11))*sin(thetaij));


    % delPij/Vj = -Vi(gij cos theta(ij) + bij sin theta ij )

    H(count,measurementdata(count,2)) = -(x(measurementdata(count,1),1))*...
        (branchdata2(count2,10)*cos(thetaij)+...
        (branchdata2(count2,11))*sin(thetaij));


          if branchdata2(count2,1) ~= 1

    % delPij/thetai = ViVj(gij sin theta(ij) - bij cos theta ij )

     H(count,busnumber+measurementdata(count,1)-1) =
x(measurementdata(count,1),1)*x(measurementdata(count,2),1)*...
         (branchdata2(count2,10)*sin(thetaij)-branchdata2(count2,11)*cos(thetaij));

    % delPij/thetaj = -ViVj(gij sin theta(ij) - bij cos theta ij )

     H(count,busnumber+measurementdata(count,2)-1) = (-1)*
x(measurementdata(count,1),1)*x(measurementdata(count,2),1)*...
```

```
(branchdata2(count2,10)*sin(thetaij)-branchdata2(count2,11)*cos(thetaij));

   end
```

When derivatives with respect to the taps are considered, below expression is used. Since the expression is derived for the T instead of a, further derivation is performed.

$$\frac{\partial}{\partial T}\begin{bmatrix} P_{tf} \\ Q_{tf} \\ P_{ft} \\ Q_{ft} \end{bmatrix} = \frac{V_t}{T}\frac{\partial}{\partial V_t}\begin{bmatrix} P_{tf} \\ Q_{tf} \\ P_{ft} \\ Q_{ft} \end{bmatrix}$$

Figure 7: Relationship between Power Flow Derivations with respect to taps and voltage magnitude [1]

```
% delPij/dela = -(Vi/a)*(delPij/delVi)

    for counttap = 1:tapnumber
        if count2 == tappedbranches(counttap)


    H(count,2*busnumber-1+counttap) = (H(count,measurementdata(count,1))*
x(measurementdata(count,1))*(-1))/x(2*busnumber-1+counttap);
```

Since the derivatives of injection measurements with respect to taps are equal to the power flow derivatives with respect to tap, this found value is also used in the injection derivatives.

```
%------------------Power Injection Derivatives wrt taps------------
    % How to calculate delPi/dela is as follows:
    % If there exists a delPij/dela, the it is equal to delPi/dela.

        for count3 = n_v+1:n_v+n_pi

            if measurementdata(count,1) == measurementdata(count3,1)
    H(count3,2*busnumber-1+counttap) = H(count,2*busnumber-1+counttap);


            end

            if measurementdata(count,2) == measurementdata(count3,1)

    H(count3,2*busnumber-1+counttap) = -H(count,2*busnumber-1+counttap);

            end
            end
        end
    end
```

Derivatives of Reactive Power Flows with respect to the State Variables

The procedure carried out for active power flow derivatives is performed also in reactive power flow derivatives. Reactive Power Flow derivative with respect to voltage magnitude and phase angles are calculated the expression given below.

$$\frac{\partial Q_{ij}}{\partial \theta_i} = -V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij})$$

$$\frac{\partial Q_{ij}}{\partial \theta_j} = V_i V_j (g_{ij} \cos\theta_{ij} + b_{ij} \sin\theta_{ij})$$

$$\frac{\partial Q_{ij}}{\partial V_i} = -V_j (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij}) - 2\, V_i (b_{ij} + b_{si})$$

$$\frac{\partial Q_{ij}}{\partial V_j} = -V_i (g_{ij} \sin\theta_{ij} - b_{ij} \cos\theta_{ij})$$

Figure 8: Reactive Power Flow Derivations [1]

```
for count = n_v+2*n_pi+n_pf+1:n_v+2*n_pi+2*n_pf

    %thetaij value is calculated as follows

            if measurementdata(count,1) == 1
    thetaij = -x(busnumber+measurementdata(count,2)-1,1);
            else
    thetaij = x(busnumber+measurementdata(count,1),1)-x(busnumber+measurementdata(count,2),1);
            end


    for count2 = 1:branchnumber
        if measurementdata(count,1) == branchdata(count2,1) && measurementdata(count,2) ==
branchdata(count2,2) || ...
                measurementdata(count,2) == branchdata(count2,1) && measurementdata(count,1)
== branchdata(count2,2)

    %Vi = x(measurementdata(count,1),1)
    %Vj = x(measurementdata(count,2),1)
    %gij = branchdata(count2,10)
    %bij = branchdata(count2,11)
    %bsi = branchdata(count2,12)
% delQij/Vi = -Vj(gij sin theta(ij) - bij cos theta ij ) - 2*Vi(bij+bsi)

    H(count,measurementdata(count,1)) = (-1)*x(measurementdata(count,2),1)*...
         (branchdata2(count2,10)*sin(thetaij)-branchdata2(count2,11)*cos(thetaij))...
         -
(2)*(x(measurementdata(count,1),1))*(branchdata2(count2,11)+branchdata2(count2,12));



    % delQij/Vj = -Vi(gij sin theta(ij) - bij cos theta ij )

    H(count,measurementdata(count,2)) = (-1)*x(measurementdata(count,1),1)*...
         (branchdata2(count2,10)*sin(thetaij)-branchdata2(count2,11)*cos(thetaij));



            if branchdata(count2,1) ~= 1

    % delQij/thetai = -ViVj(gij cos theta(ij) - bij sin theta ij )

     H(count,busnumber+measurementdata(count,1)-1) = (-
1)*x(measurementdata(count,1),1)*x(measurementdata(count,2),1)*...
         (branchdata2(count2,10)*cos(thetaij)-branchdata2(count2,11)*sin(thetaij));

    % delQij/thetaj = ViVj(gij cos theta(ij) + bij sin theta ij )
```

```
     H(count,busnumber+measurementdata(count,2)-1) =
x(measurementdata(count,1),1)*x(measurementdata(count,2),1)*...
          (branchdata2(count2,10)*cos(thetaij)+branchdata2(count2,11)*sin(thetaij));
               end
```

Similar to the active power flow case, derivatives with respect to taps will be calculated. This value will also be assigned to reactive injection measurement derivative with respect to taps.

```
       %----------------Reactive Power Flow Derivatives wrt Taps------------
for counttap = 1:tapnumber

                    if count2 == tappedbranches(counttap)

        % delQij/dela = - Vi*delQij/delVi / a
        H(count,2*busnumber-1+counttap) =(H(count,measurementdata(count,1))*
x(measurementdata(count,1))*(-1))/x(2*busnumber-1+counttap);
        %-----------Reactive Power Injection Derivatives wrt Taps---------

        % How to calculate delQi/dela is as follows:
        % If there exists a delQij/dela, the it is equal to delQi/dela.

                        for count3 = n_v+n_pi+1:n_v+2*n_pi

                            if measurementdata(count,1) == measurementdata(count3,1)

                                H(count3,2*busnumber-1+counttap) = H(count,2*busnumber-
1+counttap);


                            end

                            if measurementdata(count,2) == measurementdata(count3,1)

                                 H(count3,2*busnumber-1+counttap) = -H(count,2*busnumber-
1+counttap);

                            end



                        end
                    end
```

Derivatives of Current Magnitude with respect to the State Variables

The expression given below for the current magnitude derivatives are ignoring the shunt branch. Therefore, they give erroneous results.

$$\frac{\partial I_{ij}}{\partial \theta_i} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} V_i V_j \sin\theta_{ij}$$
$$\frac{\partial I_{ij}}{\partial \theta_j} = -\frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} V_i V_j \sin\theta_{ij}$$
$$\frac{\partial I_{ij}}{\partial V_i} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} (V_i - V_j \cos\theta_{ij})$$
$$\frac{\partial I_{ij}}{\partial V_j} = \frac{g_{ij}^2 + b_{ij}^2}{I_{ij}} (V_j - V_i \cos\theta_{ij})$$

Figure 9: Current Magnitude Derivations [1]

Instead, derivative of the current magnitude expression can be utilized.

$$Iij = \frac{\sqrt{Pij^2 + Qij^2}}{Vi}$$
$$\frac{\partial |Iij|}{\partial |Vi|} = -\frac{Iij}{Vi} + \frac{1}{IijVi^2}(Pij\frac{\partial |Pij|}{\partial |Vi|} + Qij\frac{\partial |Qij|}{\partial |Vi|})$$
$$\frac{\partial |Iij|}{\partial |Vj|} = \frac{1}{IijVi^2}(Pij\frac{\partial |Pij|}{\partial |Vj|} + Qij\frac{\partial |Qij|}{\partial |Vj|})$$
$$\frac{\partial |Iij|}{\partial a} = -\frac{Iij}{a} + \frac{1}{IijVi^2}(Pij\frac{\partial |Pij|}{\partial a} + Qij\frac{\partial |Qij|}{\partial a})$$

Measurement Function, h(x)

While computing Jacobian matrix, h(x) is also computed. Since our code will iterate and try to decrease residuals squares, we should have a measurement function.

- Voltage is taken from state vectors.

```
%--------------------h(x)---------------------------------
% Measurement function should be created while Jacobian is computed.


%For Voltage Magnitude take the value in the state vector

for count = 1:n_v
    h(count,1) = x(measurementdata(count,1),1);
```

```
end
%-------------------------------------------------------------
```

- Power Injections are calculated as follows:

```
%--------------------h(x)-------------------------------------------------
    % While calculating activepower injection derivatives, measurement function
    % should also be computed.

    % Pi = Vi*sum(Vj(Gij cos thetaij+Bij sin thetaij))

    h(count,1) = x(measurementdata(count,1),1)*sum(x(counter)*...
        (G(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))+...
        B(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))));

   %-------------------------------------------------------------------------
   %--------------------h(x)-------------------------------------------------
    % While calculating reactive power injection derivatives, measurement function
    % should also be computed.

    % Qi = Vi*sum(Vj(Gij sin thetaij - Bij cos thetaij))
    h(count,1) = x(measurementdata(count,1),1)*sum(x(counter)*...
        (G(measurementdata(count,1),counter)*sin(theta(measurementdata(count,1))-
theta(counter))-...
        B(measurementdata(count,1),counter)*cos(theta(measurementdata(count,1))-
theta(counter))));

   %-------------------------------------------------------------------------
```

- Power Flows are calculated as follows:

```
  %--------------------h(x)----------------------------------
    % While calculating  power flow derivatives, measurement function
    % should also be computed.

    % Pij = Vi^2(gsi+gij)-ViVj(gij cos theta(ij) + bij sin theta ij )
    h(count,1) = (x(measurementdata(count,1),1)^2)*...
        branchdata2(count2,10)-
x(measurementdata(count,1),1)*(x(measurementdata(count,2),1))*...
        (branchdata2(count2,10)*cos(thetaij)+...
        (branchdata2(count2,11))*sin(thetaij));


  %-----------------------------------------------------
 %--------------------h(x)----------------------------------
    % While calculating  reactive power flow derivatives, measurement function
    % should also be computed.

    % Qij = -Vi^2(bsi+bij)-ViVj(bij sin theta(ij) - bij cos theta ij )
    h(count,1) = (-1)*(x(measurementdata(count,1),1)^2)*...
        (branchdata2(count2,11)+branchdata2(count2,12))-
x(measurementdata(count,1),1)*(x(measurementdata(count,2),1))*...
        (branchdata2(count2,10)*sin(thetaij)-...
        (branchdata2(count2,11))*cos(thetaij));
```

- Current Magnitude Flows are calculated as follows:

```
 %-----------------------------------------------------
 %--------------------h(x)----------------------------------
 % While calculating  current flow derivatives, measurement function
 % should also be computed.


    % Pij = Vi^2(gsi+gij)-ViVj(gij cos theta(ij) + bij sin theta ij )
    Pij = (x(measurementdata(count,1),1)^2)*...
        branchdata2(counter,10)-
x(measurementdata(count,1),1)*(x(measurementdata(count,2),1))*...
        (branchdata2(counter,10)*cos(thetaij)+...
```

```
        (branchdata2(counter,11))*sin(thetaij));

    % Qij = -Vi^2(bsi+bij)-ViVj(gij sin theta(ij) - bij cos theta ij )
    Qij = (-1)*(x(measurementdata(count,1),1)^2)*...
        (branchdata2(counter,11)+branchdata2(counter,12))-
x(measurementdata(count,1),1)*(x(measurementdata(count,2),1))*...
        (branchdata2(counter,10)*sin(thetaij)-...
        (branchdata2(counter,11))*cos(thetaij));

    h(count,1) = sqrt(Pij^2+Qij^2)/x(measurementdata(count,1),1);
   %-------------------------------------------------------
```

This measurement vector will be used in computing right hand side equation.

Gain Matrix Calculation, Cholesky Factorization and RHS

As soon as Jacobian H(x) is calculated, gain matrix is calculated. Gain matrix calculation is very straightforward unless a problematic H(x) is encountered.

$$G = H^T R^{-1} H$$

When the Gain matrix is calculated, Cholesky factorization is performed. MATLAB has chol function which creates L and U matrices which are helpful and time saving for inverting huge sized matrices.

Right hand side, RHS equation should be calculated with using measurement function, transposed Jacobian and R^{-1}.

$$T = H^T R^{-1} [z - h(x^k)]$$

Following to Cholesky factorization and RHS computation, forward-backward substitution can be performed.

```
% Then G can be calculated by using H and R

Gain = H'*R*H;

% Now, it is time to decompose G to its lower and upper triangular forms

% For this purpose, Cholesky decomposition can be used.

L = chol(Gain,'lower');
U = L';


%Calculate RHS equation t:

t = H'*R*(z-h);
```

Calculate Δx and Update State Vector

After these calculations, final step of the state estimation is the Δx calculation and the state vector update. Actually, including this step, these algorithm is repeated until state vector difference is below a pre-defined value.

$$L\ U\ \Delta x = t$$

Here, based on the previous state values or initial state vector assumption, the only unknown is the Δx value. Note that this is a linear equation as A x = B where A and B are known. The solution is very trivial such as $x = A^{-1} B$. However, when the size of the gain matrix considered, this inversion takes huge amount of time comparing to the LU factorization when we have thousands of buses. This is why gain matrix should be factorized to its lower and upper triangular matrices. step is forward and backward substitution.

```
% Forward-Backward Substitution

% L L' delX = t

% Define L' delX = W where W is (2*busnumber-1)X1 matrix.

%Then LW=t where W is the only unknown.

%Forward Substituion
for count = 1:size(L,1)
    for counter = 1:size(L,1)

     W(count)=(1/L(count,count))*(t(count)-sum(L(count,counter)*W(counter,1)));

    end

end
% Backward Substituion--> U delX = W
for count = 1:2*busnumber+tapnumber-1
    for counter = 1:2*busnumber+tapnumber-1

     delx(2*busnumber+tapnumber-count)=(1/U(2*busnumber+tapnumber-count,2*busnumber+tapnumber-
count))*(W(2*busnumber+tapnumber-count,1)-sum(U(2*busnumber+tapnumber-
count,2*busnumber+tapnumber-counter)*delx(2*busnumber+tapnumber-counter,1)));

    end

end
```

Note that this is not one step calculation. The solution is performed iteratively. Therefore, the code should be in a while function with pre-defined value. While function should be exited as soon as Δx function has elements lower than pre-defined value. Therefore, the result of forward-backward substitution is finished, our state vector should be updated.

$$x(k+1) = x(k)+\Delta x$$

```
x = x + delx;
k = k+1;
predefinedvalue = max(delx);
```

This is the final part of the code. When code finishes iteration solution and iteration number will be displayed.

```
disp(x)
disp(k)
```

Results

One iteration results are as follows. The code given with this report has a convergence problem possibly due to possible erroneous implementations in the Jacobian Matrix. When the code goes in while loop, it gives an error related to gain matrix. In the trials, at the iterations around 20, gain matrix becomes non-positive definite. Solution time is 1.4 seconds for one iteration and almost 3 seconds for non-converging case.

Voltage	Phase Angle	Tap
0.9673	0	0.9826
0.9443	-0.0253	0.9409
0.9667	0.0085	0.6474
0.8596	-0.2339	0.9983
0.9494	-0.0706	
0.972	0.0176	
0.9573	-0.0008	
0.9722	-0.1504	
0.9837	0.0044	
0.9807	-0.0052	
1.0312	0	
0.9843	-0.0047	
1.059	0.0142	
1.0088	0.0051	
1.002	-0.0047	
1.0074	0.0056	
1.0558	0.0265	
1.0155	0.0089	
1.0097	-0.0041	
1.3464	0.3275	
1.0223	0	
0.9751	0.014	
1.0262	0.0123	
0.9698	-0.0205	
0.9508	0.0014	
1.0114	-0.0086	
1.1725	0.0707	
0.4704	0.0014	
-0.7014	-0.0391	
0.226	-2.0038	

Conclusion

This report basically explains how to create a WLS state estimator in a MATLAB environment. Here, the main idea is presented without detailed information.

It should be again noted that, in this project bus admittance matrix is also updated from iteration to iteration since transformer tap ratios are also included in the state matrix. The first bus admittance matrix will be created based on the first estimate of the transformer taps. Upcoming matrices will be shaped according to the state vector.

In this project, an observability check is not performed. It assumes that the system is observable. MATLAB script is given with this file and its variation with time can be observed from www.github.com/ErencanDuymaz/dersler/.

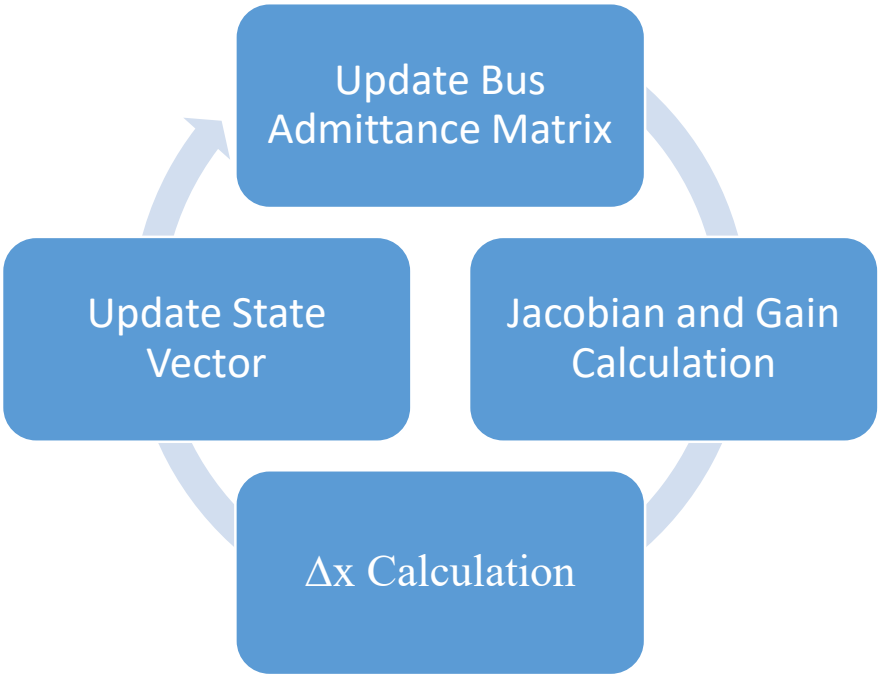


Figure 10: Main Loop of the Project

Bibliography

[1] Ali Abur,Antonio Gomez Exposito, Power System State Estimation Theory and Implementation, New York: Marcel Dekker, Inc., 2004.

[2] P. A. Teixeira, S. R. Brammer, W. L. Rutz, W. C. Merritt and J. L. Salmonsen, “State estimation of voltage and phase-shift transformer tap settings,,” *IEEE Transactions on Power Systems,* vol. vol. 7, no. no. 3, pp. 1386-1393, 1992.