EE-472 Power System Analysis II

Project II: Solving Power Flow Problem with Newton - Rapson Method

May 3, 2019

Instructor: Asst. Prof. Murat GÖL
Assistant : M. Erdem SEZGİN

Student: Akif ARSLAN - 1949080

Contents

List of Figures

1 INTRODUCTION

This project is a continuation of Project 1. At Project 1 we studied how to utilize an IEEE common data format (CDF) text file to acquire information and use it to build Y_{BUS} by writing a computer program for a given power system. At Project 2, this data and Y_{BUS} are used to solve power flow problem with Newton-Rapson iteration to find out the bus voltage magnitudes and angles. As computation tool MATLAB is used for both projects.

2 FLOW CHART

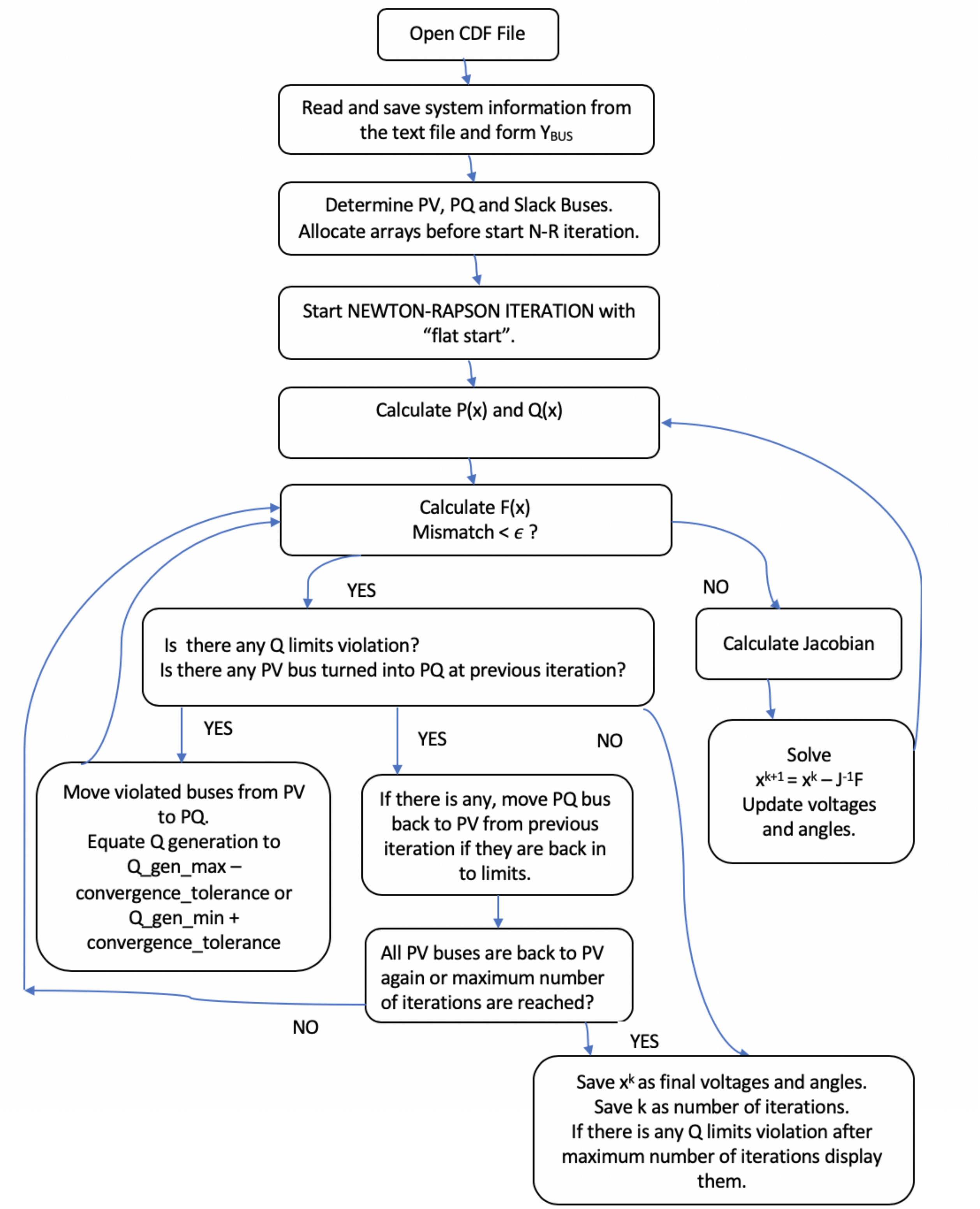

Figure 1: Flow Chart

3 SOLUTION PROCESS

3.1 Y_{BUS}:

To calculated Y_{BUS} we used the MATLAB function created for Project 1 with minor changes. At Project one, verification of $Y_{BUS}$is done by using PET software for 14 BUS case, using the prepared test cases by course assistant, and also Y_{BUS} example which we solved in class. Solution time of Y_{BUS} is measured for 300 BUS system as 0.09 seconds by using *tic-toc* function.

3.2 NEWTON – RAPSON ITERATION:

To solve the power flow problem, we used NEWTON – RAPSON iteration method. Before start iterating, we allocated necessary arrays and matrices for faster iteration. To solve the problem, we used the methods explained in the text book POWER SYSTEM ANALYSIS (Bergen, Vittal) 2nd edition, especially the sections 10.4, 10.5 and 10.6. At Section 10.6 in the textbook, Example 10.6 is a 3 Bus system example solved step by step by N-R iteration, while solving the problem we benefited from this example checking our solution steps by comparing the example.

We first solved the problem by ignoring Q limits. After we checked and verified our solution is correct by comparing the given examples in the text book, and the final voltage magnitudes and phase angles in systems' CDF files we modified our solution to also check Q limits and reach the solution in the limits. Again, the solutions are checked and verified by using PET software, and comparing with the results in the CDF files.

We wrote Example 10.6 in the text book as CDF format, put arbitrary Q limits as shown in Figure 2 and Figure 3, and checked the results.

Overall solution time, including constructing Y_{BUS} for IEEE 300 BUS system is measured as 0.52 second in average again by using *tic – toc* function of MATLAB. And solution is reached in 7 iteration with $\epsilon = 0.1$MVA. The methods which are used to increase the speed of solution are explained at the next section.

Example 10.6

Find $\theta_2, |V_3|, \theta_3, S_{G1}$, and Q_{G2} for the system shown in Figure E10.6. In the transmission system all the shunt elements are capacitors with an admittance $y_C = j0.01$, while all the series elements are inductors with an impedance of $z_L = j0.1$.

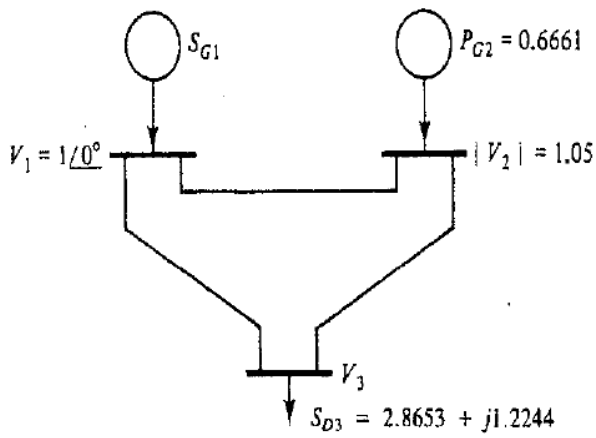

Figure E10.6

Solution The $\mathbf{Y}_{\text{bus}}$ for the system shown in Figure E10.6 is given by

$$\mathbf{Y}_{\text{bus}} = \begin{bmatrix} -j19.98 & j10 & j10 \\ j10 & -j19.98 & j10 \\ j10 & j10 & -j19.98 \end{bmatrix}$$

Figure 2: Example 10.6

```
TAPE
 13/05/91 CYME INTERNATIONAL    100.0 1991 S IEEE 300-BUS TEST SYSTEM
BUS DATA FOLLOWS                            300 ITEMS
   1  1            1  1  3 1.0000   0.00    90.00    49.00     0.00     0.00  115.00 1.0000    0.00     0.00  0.0000  0.0000    0    1
   2  1            1  1  2 1.0500   7.74   000.00   000.00    66.61     0.00  115.00 1.0500  300.00 -300.00  0.0000  0.0000    0    2
   3  1            1  1  0 0.9971   6.64   286.53   122.44     0.00     0.00  230.00 0.0000    0.00     0.00  0.0000  0.0000    0    3

-999 1
BRANCH DATA FOLLOWS                         411 ITEMS
   1    2  1  9 1 0  0.000000  0.100000  0.02000     0     0    75    0 0  1.0082     0.00 0.90431.10435 .00400     0.0   15.0     1
   1    3  1  9 1 0  0.000000  0.100000  0.02000     0     0     0    0 0  0.0000     0.00 0.0000 0.0000 .00000  0.0000 0.0000     2
   2    3  1  9 1 0  0.000000  0.100000  0.02000     0     0     0 9006 0  0.9668     0.00 0.9391 1.1478 .00417  0.9900 1.0100     3

-999 1
LOSS ZONES FOLLOWS                            0 ITEMS
-99
END OF DATA
```

Figure 3: Example 10.6 CDF

4 METHODS TO INCREASE SOLUTION SPEED, COMMENTS AND REASONING

4.1 MEMORY ALLOCATION:

In general, for every iteration we allocated matrices before the iteration and write the code by avoiding changing sizes of the matrices inside any loop. Since MATLAB move all information from one place to another when size of a matrix is changed it causes significant speed decrease.

4.2 Y_{BUS}: :

As we know Y_{BUS} is a sparse matrix, and sparsity pattern of Y_{BUS} increases as the bus numbers increase. Hence, while constructing Y_{BUS} we only interested in non-zero elements and their indices. Once we obtained non-zero elements and their indices, we constructed Y_{BUS} by using MATLAB's sparse function. We made a minor change to get a modest speed increase and accuracy at Y_{BUS}. At Project 1 we used str2double(str) to get data from text file at this project we used sscanf(str, format). sscanf is faster than srt2double but also, we can specify the format as stated in CDF file as integer, float, ASCI character.

4.3 $P(x)$ and $Q(x)$:

To calculate $P(x)$ and $Q(x)$ we used the Formula 10.32 in text book as shown in Figure 4.

$$P_i(\mathbf{x}) \triangleq \sum_{k=1}^{n} |V_i|\,|V_k|\,[G_{ik}\cos(\theta_i - \theta_k) + B_{ik}\sin(\theta_i - \theta_k)] \qquad i = 1, 2, 3, \ldots, n$$

$$Q_i(\mathbf{x}) \triangleq \sum_{k=1}^{n} |V_i|\,|V_k|\,[G_{ik}\sin(\theta_i - \theta_k) - B_{ik}\cos(\theta_i - \theta_k)] \qquad i = 1, 2, 3, \ldots, n$$

Figure 4: Formula 10.32 in the text book

If we write this formula in a double for loop it will look like as shown in Figure 5. For a 300 Bus system this for loops means $2 \times 300 \times 300 = 180.000$ iteration. But we know Y_{BUS} is a sparse matrix and therefore we can simply assign zeros to these entries where Y_{BUS} is zero before the iteration. When we check IEEE 300 BUS system's Y_{BUS} we see that only 1118 entries of it are non-zero. By this way we get about $2 \times 300 \times 4 = 2400$ iterations. $180.000/2400 = 75$ times faster iteration. At Figure 6, we can see the modified for loop.

```
for i = N
    for k = 1:N
        P(i) = P(i) + v(i)*v(k)*(G(i,k)*cos(phi(i)-phi(k)) + B(i,k)*sin(phi(i)-phi(k)));
    end
end

for i = N
    for k = 1:N
        Q(i) = Q(i) + v(i)*v(k)*(G(i,k)*sin(phi(i)-phi(k)) - B(i,k)*cos(phi(i)-phi(k)));
    end
end
```

Figure 5: for loop for $P(x), Q(x)$ without modification

```
for i = P_idx
    for k = find(Y_Bus(i,:) ~= 0) % ==> Iterate only for non-zero Y(i,k)
        P(i) = P(i) + v(i)*v(k)*(G(i,k)*cos(phi(i)-phi(k)) + B(i,k)*sin(phi(i)-phi(k)));
    end

    for k = find(Y_Bus(i,:) ~= 0) % ==> Iterate only for non-zero Y(i,k)
        Q(i) = Q(i) + v(i)*v(k)*(G(i,k)*sin(phi(i)-phi(k)) - B(i,k)*cos(phi(i)-phi(k)));
    end
end
```

Figure 6: for loop for $P(x), Q(x)$ with modification

At Figure 7 we can see the exponential increase of sparsity of Y_{BUS} as the number of buses increase. As the system grows, although number of iterations increase at the same time sparsity pattern of Y_{BUS} increases and we benefit from it more.

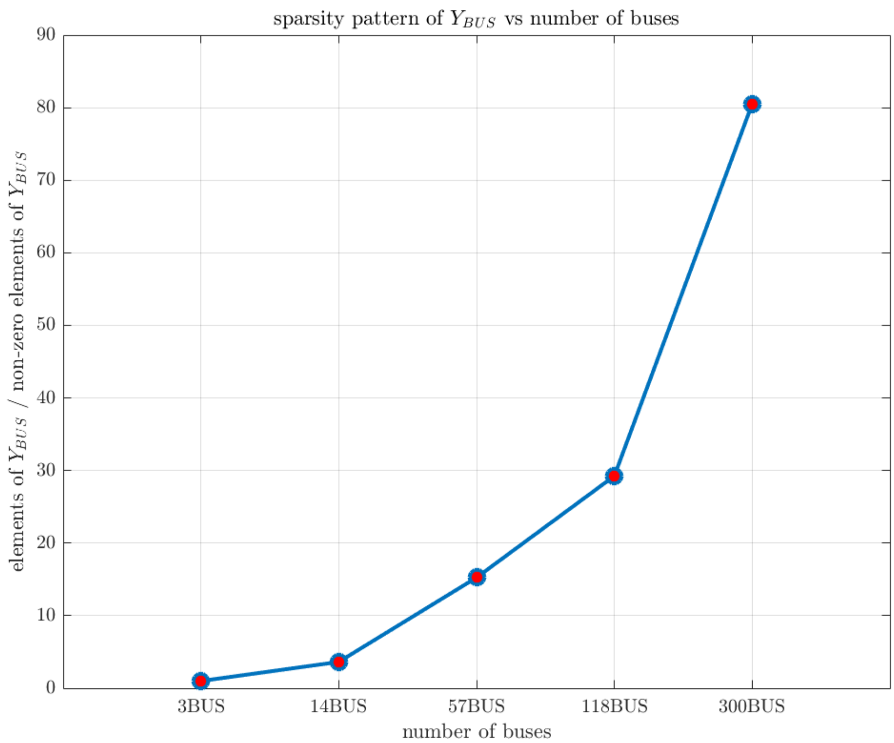

Figure 7: Spasity Increase

4.4 JACOBIAN ($J(x)$):

To calculate $J(x)$ we used Formulas 10.40 and 10.41 in the text book as shown in Figure 8 and Figure 9. Again, we benefited from the sparsity pattern of the Y_{BUS} and iterate only for non-zero entries of Y_{BUS}. This causes a significant decrease of iteration as in the case of calculation of $P(x)$ and $Q(x)$. Also, for Q_p and P_p we did nott calculated again for Jacobian, instead we used already calculated values when we found $Q(x)$ and $P(x)$.

For indices $p \neq q$

$$
\begin{aligned}
J^{11}_{pq} &= \frac{\partial P_p(\mathbf{x})}{\partial \theta_q} = |V_p||V_q|(G_{pq}\sin\theta_{pq} - B_{pq}\cos\theta_{pq}) \\
J^{21}_{pq} &= \frac{\partial Q_p(\mathbf{x})}{\partial \theta_q} = -|V_p||V_q|(G_{pq}\cos\theta_{pq} + B_{pq}\sin\theta_{pq}) \\
J^{12}_{pq} &= \frac{\partial P_p(\mathbf{x})}{\partial |V_q|} = |V_p|(G_{pq}\cos\theta_{pq} + B_{pq}\sin\theta_{pq}) \\
J^{22}_{pq} &= \frac{\partial Q_p(\mathbf{x})}{\partial |V_q|} = |V_p|(G_{pq}\sin\theta_{pq} - B_{pq}\cos\theta_{pq})
\end{aligned}
\tag{10.40}
$$

Figure 8: for loop for $P(x), Q(x)$ with modification

For indices $p = q$

$$
\begin{aligned}
J^{11}_{pp} &= \frac{\partial P_p(\mathbf{x})}{\partial \theta_p} = -Q_p - B_{pp}|V_p|^2 \\
J^{21}_{pp} &= \frac{\partial Q_p(\mathbf{x})}{\partial \theta_p} = P_p - G_{pp}|V_p|^2 \\
J^{12}_{pp} &= \frac{\partial P_p(\mathbf{x})}{\partial |V_p|} = \frac{P_p}{|V_p|} + G_{pp}|V_p| \\
J^{22}_{pp} &= \frac{\partial Q_p(\mathbf{x})}{\partial |V_p|} = \frac{Q_p}{|V_p|} - B_{pp}|V_p|
\end{aligned}
\tag{10.41}
$$

Figure 9: for loop for $P(x), Q(x)$ with modification

4.5 CALCULATING $x^{k+1} = x^k - J(x^k)^{-1}F(x^k)$:

As it has been discussed in the class rather than taking inverse of a large $J(x)$ matrix using LU factorization preferred. When we search for how to utilize LU factorization in MATLAB, we found out that we can simply use "\"[1] operator to utilize LU factorization as it recommended in Project 1 definition. Hence, we modified our solution equation as $x^{k+1} = x^k - J(x^k) \setminus F(x^k)$.

4.6 INTRODUCING Q_{max} Q_{min} TOLERANCE:

While checking Q limits, we subtracted a small amount (0.001MVAR) from maximum Q generation for over limits and added same amount for under limits. This small amount tolerance decreased iteration number significantly therefore solution time. For IEEE 300 BUS system, without this toleration solution is reached after 39 iteration and about 4 seconds. After we introduced tolerance it reached solution in 7 iteration and about 0.52 second. In this way we directed convergence not around the bound but inside the bounds.

5 TEST RESULTS and COMMENTS

5.1 Y_{BUS}

We tested our Y_{BUS} with the cases which prepared by course assistant M. Erdem SEZGIN. We calculate total error and maximum error between given result and calculated results by the method given below. Results are shown below.

As we can see from the results, our solutions are consistent with the results. There are some minor differences which we thought they could be rounding or simulation method errors between our solution method and PET software.

$$MaximumError = \max(\max(\text{abs}(Y_{BusCalculated} - Y_{BusGiven}))) \quad (1)$$
$$TotalError = \text{sum}(\text{sum}(\text{abs}(Y_{BusCalculated} - Y_{BusGiven}))) \quad (2)$$

- Test Case 1:

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 2;

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 3;

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 4;

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 5;

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 6;

  There was a typing error in the original file of test case 6 and test case 7. Phase shifter was was written as 1 means fixed tap changer. When we corrected this error, results are verified.

$$MaximumError = 0$$
$$TotalError = 0$$

- Test Case 7;

$$MaximumError = 0$$
$$TotalError = 0$$

5.2 VOLTAGE MAGNITUDE ERROR in p.u.

With same method we calculated maximum and total error in voltage magnitudes.

- IEEE 300 BUS SYSTEM:

$$MaximumError = 0.0039$$
$$TotalError = 0.0401$$

- IEEE 118 BUS SYSTEM;

$$MaximumError = 0.0175$$
$$TotalError = 0.0369$$

- IEEE 14 BUS SYSTEM;

$$MaximumError = 0.0013$$
$$TotalError = 0.0035$$

5.3 PHASE ANGLE ERROR in DEGREES

With same method we calculated maximum and total error in voltage magnitudes.

- IEEE 300 BUS SYSTEM:

$$MaximumError = 0.0511^{\circ}$$
$$TotalError = 4.4001^{\circ}$$

- IEEE 118 BUS SYSTEM;

$$MaximumError = 0.3123^{\circ}$$
$$TotalError = 16.2303^{\circ}$$

- IEEE 14 BUS SYSTEM;

$$MaximumError = 0.0171^{\circ}$$
$$TotalError = 0.0631^{\circ}$$

5.4 NUMBER of PV BUSES VIOLATES Q LIMITS BEFORE and AFTER CHECKING Q LIMITS

- IEEE 300 BUS SYSTEM:

$$Before = 11$$
$$After = 0$$

- IEEE 118 BUS SYSTEM;

$$Before = 6$$
$$After = 0$$

- IEEE 14 BUS SYSTEM;

$$Before = 0$$
$$After = 0$$

6 CONVERGENCE TOLERANCE ϵ, COMMENTS AND REASONING

$$\epsilon = 0.1\text{MVA}$$

We made some internet search to determine convergence tolerance ϵ. We found out that, conventionally ϵ is taken as 0.1 MVA[2]. Since we used p.u. in our solution we defined convergence tolerance as $\epsilon = 0.1/S_{Base}$.

7 Q LIMIT TOLERANCE, COMMENTS AND REASONING

$$Q \text{ LIMIT TOLERANCE} = 0.001\text{MVAR}$$

While checking Q limits we introduced a 0.001 MVAR tolerance. By the help of this tolerance iteration convergence directed into the limits. Without this tolerance solution converges around the limits but not necessarily into the limits. Since it is chosen very small does not affect the accuracy of the solution.

8 CONCLUSION

Our algorithm solve power flow problem for IEEE 300 BUS SYSTEM in;

$$\text{solution time} = 0.52 \text{ second}$$
$$\text{iteration number} = 7$$

The accuracy of the solution is verified by the tests listed at previous sections. We concluded that knowing the systems structure and benefiting from it, like we did iterating only over non-zero terms may enhance solution speed significantly. Also for memory management sparsity pattern of Y_{BUS} and Jacobian matrix could be utilized. At Figure 10 and Figure 11 we can see sparsity patterns of Y_{BUS} and Jacobian respectively for 300 bus system.

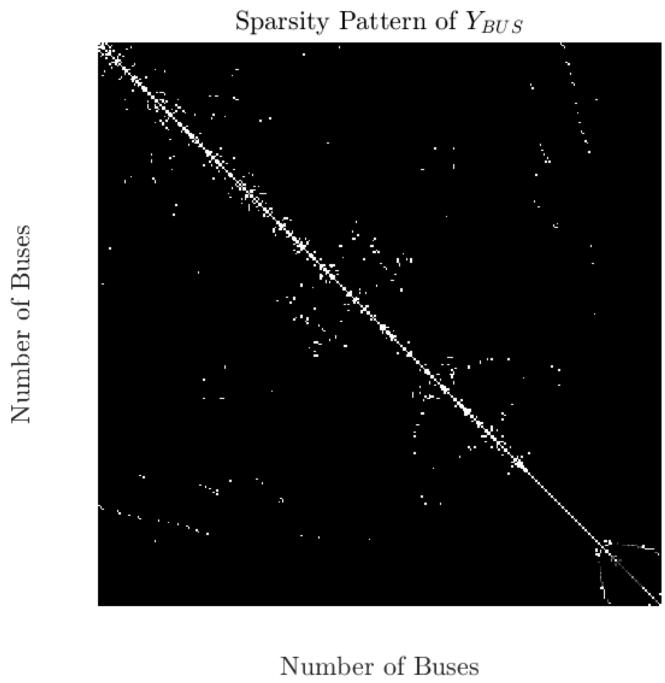

Figure 10: Sparsity Pattern of Y_{BUS}

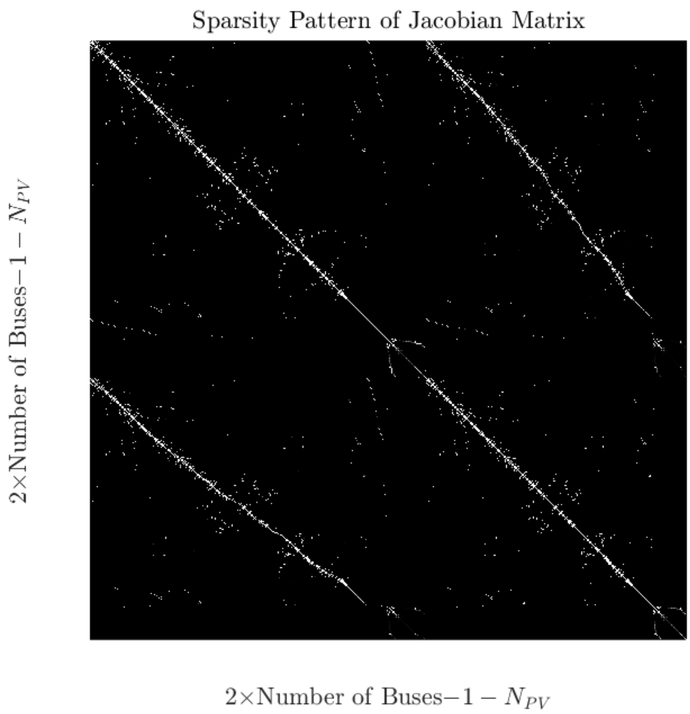

Figure 11: Sparsity Pattern of Jacobian

9 References

[1] https://www.mathworks.com/help/matlab/ref/lu.html

[2] https://www.powerworld.com/WebHelp/Content/MainDocumentation_HTML/Power_Flow_Solution_Common_Options.htm

Appendices

A MATLAB CODE

```
function [v,theta] = e194908_arslan_PF(inputArg1)
% [voltage_magnitude,phase_angle] = e194908_arslan_PF(inputArg1) reads data
% from a text file in IEEE Common Data Format (CDF), returns voltage
% magnitudes in p.u. and voltage phase angles in degrees.
%
%
%  input   : text file in cdf format
%  outputs : voltage magnitutes in p.u.
%            voltage phase angle in degrees
%
%
%  outputs' size is Nx1, where N is number of buses in given system.
%
%
%
%============== EE 472 SYSTEM ANALYSIS II =========================
%                     SPRING 2019
%                      Project-2
%
%
% Instructor: Asst. Prof Murat GOL
% Assistant : M. Erdem SEZGIN
%
% Student   : Akif ARSLAN - 1949080


%% SECTION 1: OBTAIN DATA FROM TEXT FILE AND CONSTRUCT Y_BUS.

% Open the text file.
fileID = fopen(inputArg1,'r');

% Get S_Base
while true
    line_data = fgetl(fileID);
    if length(line_data) ≥ 37
        S_Base = sscanf(line_data(32:37),'%f');
        break
    end
     if length(line_data) ≥ 36
        S_Base = sscanf(line_data(32:36),'%f');
        break
    end
end
```

```

% Find where bus data starts.
while true
    line_data = fgetl(fileID);
    if length(line_data) ≥ 3
        if strcmpi(line_data(1:3),'BUS')
            break
        end
    end
end

% Allocate arrays for faster iteration.
indexing_matrix = uint16(zeros(1000,1)); % Indexing matrix to follow ...
    the buses order.
shunt_G_and_B    = zeros(1000,1);         % Shunt conductance and ...
    susceptance.
Bus_Type         = zeros(1000,1);         % Bus type.
Load_P           = zeros(1000,1);         % Load MW.
Load_Q           = zeros(1000,1);         % Load MVAR.
Gen_P            = zeros(1000,1);         % Generated MW.
Gen_Q            = zeros(1000,1);         % Generated MVAR.
theta_final      = zeros(1000,1);         % Final voltage angle in degrees.
v_desired        = zeros(1000,1);         % Desired voltage in p.u
max_Q_or_v       = zeros(1000,1);         % Maximum MVAR or voltage limit.
min_Q_or_v       = zeros(1000,1);         % Mininum MVAR or voltage limit.

% Start iteration for BUS DATA.
i = uint16(1);
while true
    line_data = fgetl(fileID);
    % break the loop when BUS DATA finishes
    if length(line_data) < 50
        % cut unused parts
        indexing_matrix = indexing_matrix(1:i-1);
        shunt_G_and_B   = shunt_G_and_B(1:i-1);
        Bus_Type        = Bus_Type(1:i-1);
        Load_P          = Load_P(1:i-1);
        Load_Q          = Load_Q(1:i-1);
        Gen_P           = Gen_P(1:i-1);
        Gen_Q           = Gen_Q(1:i-1);
        theta_final     = theta_final(1:i-1);
        v_desired       = v_desired(1:i-1);
        max_Q_or_v      = max_Q_or_v(1:i-1);
        min_Q_or_v      = min_Q_or_v(1:i-1);
        number_of_buses = i-1;
        break
    end
    indexing_matrix(i)  = sscanf(line_data(1:4),    '%i');
    shunt_G_and_B(i)    = sscanf(line_data(107:114),'%f')...
                        + sscanf(line_data(115:122),'%f')*1i;
    Bus_Type(i)         = sscanf(line_data(25:26),  '%i');
    Load_P(i)           = sscanf(line_data(41:49),  '%f');
    Load_Q(i)           = sscanf(line_data(50:59),  '%f');
    Gen_P(i)            = sscanf(line_data(60:67),  '%f');
    Gen_Q(i)            = sscanf(line_data(68:75),  '%f');
    theta_final(i)      = sscanf(line_data(34:40),  '%f');
    v_desired(i)        = sscanf(line_data(85:90),  '%f');
    max_Q_or_v(i)       = sscanf(line_data(91:98),  '%f');
    min_Q_or_v(i)       = sscanf(line_data(99:106), '%f');
```

```
    i = i + 1;
end

% Go where BRANCH DATA STARTS
while true
    line_data = fgetl(fileID);
    if length(line_data) ≥ 6
        if strcmpi(line_data(1:6),'BRANCH')
            break
        end
    end
end

% allocate memory for Y_Bus matrix for faster iteration.
Y_Bus = zeros(number_of_buses*4,4);

% Create a 2x2 temporary pair admittance matrix.
Y_Bus_temp = zeros(2,2);

% Start iteration for BRANCH DATA and construct Y_Bus.
k=uint16(1);
while true
    line_data = fgetl(fileID);
    % break the loop when BRANCH DATA finishes
    if length(line_data) < 50
        if sum(Y_Bus(:,1) == 0,1)
            % Cut unused parts of the Y_Bus
            zero_cut = find(Y_Bus(:,1)==0, 1, 'first');
            Y_Bus = Y_Bus(1:zero_cut-1,:);
        end
        break
    end
    %Use indexing as given in BUS data order.
    Yi = find(indexing_matrix == sscanf(line_data(1:4),'%i')); % "from" bus
    Yj = find(indexing_matrix == sscanf(line_data(6:9),'%i')); % "to" bus

    %Get Resistance and Reactance data and turn into line admittance.
    %R = sscanf(line_data(20:29),'%f');
    %X = sscanf(line_data(30:40),'%f');
    line_admittance = 1/(sscanf(line_data(20:29),'%f') + ...
        sscanf(line_data(30:40),'%f')*1i);

    %Get line charing B data and divide by 2.
    line_charging = (sscanf(line_data(41:50),'%f')/2)*1i;

    % Construct temporary 2x2 pair admittance matrix.
    % If there is any tap or phase shifter include their effects.
    switch sscanf(line_data(19),'%i')
        case 0 % 0 ==> A line.
            Y_Bus_temp(1,1) = line_admittance + line_charging + ...
                shunt_G_and_B(Yi);% Yii
            Y_Bus_temp(1,2) = -line_admittance; % Yij
            Y_Bus_temp(2,1) = -line_admittance; %Yji
            Y_Bus_temp(2,2) = line_admittance + line_charging + ...
                shunt_G_and_B(Yj); % Yjj

        case {1,2,3} % 1,2,3 ==> There is a tap changer.
            tap = sscanf(line_data(77:82),'%f');
```

```
            Y_Bus_temp(1,1) = (line_admittance/(tap^2)) + ...
                shunt_G_and_B(Yi);% Yii
            Y_Bus_temp(1,2) = -line_admittance/tap; % Yij
            Y_Bus_temp(2,1) = -line_admittance/tap; % Yji
            Y_Bus_temp(2,2) = line_admittance + shunt_G_and_B(Yj); % Yjj

        case 4 % 4 ==> There is a phase shifter.
            tap = sscanf(line_data(77:82),'%f');
            p_shift = sscanf(line_data(84:90),'%f');
            p_shift = p_shift*pi/180;
            Y_Bus_temp(1,1) = line_admittance/tap^2 + ...
                shunt_G_and_B(Yi);% Yii
            Y_Bus_temp(1,2) = -line_admittance/(cos(p_shift) - ...
                sin(p_shift)*1i); % Conjugate
            Y_Bus_temp(2,1) = -line_admittance/(cos(p_shift) + ...
                sin(p_shift)*1i);
            Y_Bus_temp(2,2) = line_admittance + shunt_G_and_B(Yj);
        otherwise
            disp('line information has not found')
    end
    % Once shunt values are used remove them to avoid adding again at next
    % iterations.
    shunt_G_and_B(Yi) = 0;
    shunt_G_and_B(Yj) = 0;

    % Yi or/and Yj is used at previous iteration find where Yii and Yjj.
    Yi_idx = find(Y_Bus(:,1) == Yi & Y_Bus(:,2) == Yi);
    Yj_idx = find(Y_Bus(:,1) == Yj & Y_Bus(:,2) == Yj);

    is_Yi_used = sum(Yi_idx); % If Yi is used before, make a logical ...
        true for 'if' decision.
    is_Yj_used = sum(Yj_idx); % If Yj is used before, make a logical ...
        true for 'if' decision.

    % If both Yi and Yj busses are used at previous iteration, don't create
    % new Yii and Yjj, add new Yii/Yjj values to them. Only create Yij ...
        and Yji.
    if is_Yi_used && is_Yj_used % ==> Both used before.
        Y_Bus(Yi_idx ,3) = Y_Bus(Yi_idx ,3) + real(Y_Bus_temp(1,1));
        Y_Bus(Yi_idx ,4) = Y_Bus(Yi_idx ,4) + imag(Y_Bus_temp(1,1));

        Y_Bus(Yj_idx,3) = Y_Bus(Yj_idx,3) + real(Y_Bus_temp(2,2));
        Y_Bus(Yj_idx,4) = Y_Bus(Yj_idx,4) + imag(Y_Bus_temp(2,2));
    else
        if is_Yi_used || is_Yj_used % ==> One of them used before.
            % Check, which one of Yi and Yj are used before
            if is_Yi_used
                Y_Bus(Yi_idx,3) = Y_Bus(Yi_idx,3) + real(Y_Bus_temp(1,1));
                Y_Bus(Yi_idx,4) = Y_Bus(Yi_idx,4) + imag(Y_Bus_temp(1,1));

                Y_Bus(k,1) = Yj;
                Y_Bus(k,2) = Yj;
                Y_Bus(k,3) = real(Y_Bus_temp(2,2));
                Y_Bus(k,4) = imag(Y_Bus_temp(2,2));
                k = k + 1;

            else % if is_Yj_used
                Y_Bus(k,1) = Yi;
                Y_Bus(k,2) = Yi;
```

```
                Y_Bus(k,3) = real(Y_Bus_temp(1,1));
                Y_Bus(k,4) = imag(Y_Bus_temp(1,1));
                k = k + 1;

                Y_Bus(Yj_idx,3) = Y_Bus(Yj_idx,3) + real(Y_Bus_temp(2,2));
                Y_Bus(Yj_idx,4) = Y_Bus(Yj_idx,4) + imag(Y_Bus_temp(2,2));
            end
            % If neighter Yi or Yj bus number is used previously,
            %construct new Yii and Yjj.
        else
            Y_Bus(k,1) = Yi;
            Y_Bus(k,2) = Yi;
            Y_Bus(k,3) = real(Y_Bus_temp(1,1));
            Y_Bus(k,4) = imag(Y_Bus_temp(1,1));
            k = k + 1;

            Y_Bus(k,1) = Yj;
            Y_Bus(k,2) = Yj;
            Y_Bus(k,3) = real(Y_Bus_temp(2,2));
            Y_Bus(k,4) = imag(Y_Bus_temp(2,2));
            k = k + 1;
        end
    end
    % Cunstruct Yij, Yji element for any of the cases.
    Y_Bus(k,1) = Yi;
    Y_Bus(k,2) = Yj;
    Y_Bus(k,3) = real(Y_Bus_temp(1,2));
    Y_Bus(k,4) = imag(Y_Bus_temp(1,2));
    k = k + 1;

    Y_Bus(k,1) = Yj;
    Y_Bus(k,2) = Yi;
    Y_Bus(k,3) = real(Y_Bus_temp(2,1));
    Y_Bus(k,4) = imag(Y_Bus_temp(2,1));
    k = k + 1;
end
% Close the text file after iteration.
fclose(fileID);

% Use "sparse" fuction to obtain Y_Bus as a sparse matrix using
% 'sparse(i,j,v)': sparse(Yi_idx,Yj_idx, Y(Yi_idx,Yj_idx)).
Y_Bus = sparse(Y_Bus(:,1),Y_Bus(:,2), Y_Bus(:,3) + Y_Bus(:,4)*1i);

%% SECTION 2: NEWTON - RAPSON ITERATION.

%============= Create variables before iteration =========================

N           = number_of_buses;          % Number of buses.

G           = real(Y_Bus);              % Y_Bus conductance in p.u.
B           = imag(Y_Bus);              % Y_Bus susceptance in p.u.

Gen_P       = Gen_P/S_Base;             % Generated MW in p.u.
Gen_Q       = Gen_Q/S_Base;             % Generated MVAR in p.u.

Load_P      = Load_P/S_Base;            % Load MW in p.u.
Load_Q      = Load_Q/S_Base;            % Load MVAR in p.u.

P_hat       = (Gen_P - Load_P);         % Given Bus MW in p.u.
```

```
Q_hat      = (Gen_Q - Load_Q);         % Given Bus MVAR in p.u.

max_Q_or_v = max_Q_or_v/S_Base;        % Maximum MVAR or voltage limit ... in p.u.
min_Q_or_v = min_Q_or_v/S_Base;        % Minumum MVAR or voltage limit ... in p.u.

% Create an array to hold indices of N - Slack (N_PV + N_PQ) unknow buses.
% Since we need N - Slack = N -1 times P(x) equation we named it as P_idx.
P_idx = 1:N;

% Create an array to hold indices of N-N_PV-Slack (N_PQ) unknow buses.
% Since we need N - N_PV - Slack times Q(x) equation we named it as
% Q_idx.
Q_idx            = 1:N;
slack_idx        = find(Bus_Type == 3);    % slack Bus index.
PV_idx           = find(Bus_Type == 2);    % PV Buses' indices.
P_idx(slack_idx) = [];                     % Remove slack bus from P(x) ... equation indices.
len_P_idx        = length(P_idx);          % Length of P(x) equations


% Since during Q limits checks the length of Q(x) equations may change,
% instead removing PV and slack indices, we replace voids with zeros, so
% MATLAB doesnt have to change the size of the Q_idx array inside the ... for loop.
% Make slack bus and PV bus indices 0 to obtain N-N_PV-Slack  non-zero
% bus indices.
Q_idx([PV_idx; slack_idx]) = 0;                  % Non-zero indices of ... Q(x) unknown equation.
len_Q_idx                  = nnz(Q_idx);         % The lenght of Q_idx.
nzQ_idx                    = Q_idx(Q_idx ≠ 0); % Do iteration only for ... non zero indices.

% Create a voltage vector with ones for "flat start" in p.u.
v_flat = ones(N,1);

% Create an angle vector with zeros for "flat start" in radians.
theta_flat = zeros(N,1);

% Change Slack and PV buses voltages in the flat start array to the set ... voltages.
v_flat([slack_idx;PV_idx]) = v_desired([slack_idx;PV_idx]);

% change Slack bus angle in the flat start array to the set angle.
theta_flat(slack_idx) = theta_final(slack_idx)*pi/180;

v             = v_flat;       % initial voltages to start iteration in p.u.
theta         = theta_flat;   % initial voltage angles to start ... iteration in radians.

eps           = 0.1/S_Base;   % mistatch tolerance epsilon = 0.1 MVA.
Qlim_cnv_tol  = 0.001/S_Base; % Q limits tolerance 0.001MVAR. Explained ... at Q limits parts.

under_idx     = zeros(N,1);   % PV Bus indices under Q limits.
over_idx      = zeros(N,1);   % PV Bus indices over Q limits.

flag          = false;        % Flag for starting to check Q limits.
```

```
P              = zeros(N,1);    % P(x)
Q              = zeros(N,1);    % Q(x)

max_iteration = 100;           % Maximum number of iteration in case of ...
    no convergence.
iteration_num = 0;             % Number of iterations

%Allocate memory for Jacobian.
J_11 = zeros(len_P_idx);                    % dP(x)/dtheta(x)
J_12 = zeros(len_P_idx,len_Q_idx);          % dP(x)/dv(x)
J_21 = zeros(len_Q_idx,len_P_idx);          % dQ(x)/dtheta(x)
J_22 = zeros(len_Q_idx);                    % dQ(x)/dv(x)


%======================= Start N-R Iteration ===============================
while true

    %============= P(x) and Q(x) =======================================
    P(:) = 0;
    Q(:) = 0;
    for i = P_idx
        for k = find(Y_Bus(i,:) ≠ 0) % ==> Iterate only for non-zero Y(i,k)
            P(i) = P(i) + v(i)*v(k)*(G(i,k)*cos(theta(i)-theta(k)) + ...
                B(i,k)*sin(theta(i)-theta(k)));
        end

        for k = find(Y_Bus(i,:) ≠ 0) % ==> Iterate only for non-zero Y(i,k)
            Q(i) = Q(i) + v(i)*v(k)*(G(i,k)*sin(theta(i)-theta(k)) - ...
                B(i,k)*cos(theta(i)-theta(k)));
        end
    end


    %===================== F(x) and Mismatch ...
        ================================
    if ¬flag
        F = -([P_hat(P_idx);Q_hat(nzQ_idx)] - [P(P_idx);Q(nzQ_idx)]);
        mismatch = max(abs(F));
    end

    %================== Check Q_limits ============================

    % Once mismatch < epsilon continue iteration by checking Q limits.
    if (mismatch < eps) || flag
        flag = true; % Make flag true to continue iteration in this ...
            'if' logic.

        % PV buses violates limits.
        over_limits  = find( Q > (max_Q_or_v - Load_Q) & (Bus_Type == 2));
        under_limits = find( Q < (min_Q_or_v - Load_Q) & (Bus_Type == 2));

        %--------------- PV <-- PQ ---------------------

        % If there are any PV bus which turned into PQ bus at previous
        % iterations, check if any of them are back in Q limits. If there
        % is any, put them back into PV bus indices.
        if ¬isempty(under_idx(under_idx ≠ 0))
            for i = under_idx(under_idx ≠ 0)'
```

```
            if Q(i) ≥ (min_Q_or_v(i) − Load_Q(i))
                Q_idx(i) = 0;
                under_idx(i) = 0;
            end
        end
    end

    if ¬isempty(over_idx(over_idx ≠ 0))
        for i = over_idx(over_idx ≠ 0)'
            if (Q(i) ≤ (max_Q_or_v(i) − Load_Q(i)))
                Q_idx(i) = 0;
                over_idx(i) = 0;
            end
        end
    end

    %—————————————————— PV ——> PQ ——————————————————————————————————

    % If there are any PV bus violates Q limits put it in PQ buses.
    % Assign Q_hat(i) to maximum/minimum limits − Q load. Also
    % add/subtract a very small convergence tolerance to put Q_hat(i)
    %in limits so iteration converges much faster.
    if ¬isempty(over_limits)
        for i = over_limits'
            Q_hat(i) = max_Q_or_v(i) − Load_Q(i) − Qlim_cnv_tol;
            if over_idx(i) == 0
                Q_idx(i) = i;
                over_idx(i) = i;
            end
        end
    end
    if ¬isempty(under_limits)
        for i = under_limits'
            Q_hat(i) = min_Q_or_v(i) − Load_Q(i) + Qlim_cnv_tol;
            if under_idx(i) == 0
                Q_idx(i) = i;
                under_idx(i) = i;
            end
        end
    end

    %================= Update Q_idx ===================================
    nzQ_idx = Q_idx(Q_idx ≠ 0);
    len_Q_idx = nnz(Q_idx);

    %============= F(x) and mismatch ==================================
    F = −([P_hat(P_idx);Q_hat(nzQ_idx)] − [P(P_idx);Q(nzQ_idx)]);
    mismatch = max(abs(F));

    %============== Stop Iteration ...
       =====================================

    % If there is no Q limit violation and mismatch < epsilon, or if
    % maximum number of iterations are reached stop iteration.
    if (¬nnz(under_idx) && ¬nnz(over_idx)  && (mismatch < eps))...
            ||  (iteration_num ≥ max_iteration)
        theta = theta*180/pi; % Give output angle in degrees.
        break
    end
```

```
        %=============== Update size of Jacobian ==========================
        J_12 = zeros(len_P_idx,len_Q_idx);  % dP(x)/dv(x)
        J_21 = zeros(len_Q_idx,len_P_idx);  % dQ(x)/dtheta(x)
        J_22 = zeros(len_Q_idx);            % dQ(x)/dv(x)
    end



    %===================== Jacobian, J(x)  ...
        ========================================

    for p = 1:len_P_idx
        i = P_idx(p);
        % J_11 ---------------------------------------
        for q = find(Y_Bus(i, P_idx) ≠ 0) % ==> iterate only for ...
            non-zero Y(i,j)
            j = P_idx(q);
            if i ≠ j
                J_11(p,q) = v(i)*v(j)*(G(i,j)*sin(theta(i) - theta(j)) ...
                    - B(i,j)*cos(theta(i) -theta(j)));
            else
                J_11(p,q) =-Q(i) - B(i,j)*v(i)^2;
            end
        end
        % J_12 ---------------------------------------
        for q = find(Y_Bus(i, nzQ_idx) ≠ 0) % ==> iterate only for ...
            non-zero Y(i,j)
            j = nzQ_idx(q);
            if i ≠ j
                J_12(p,q) = v(i)*(G(i,j)*cos(theta(i)-theta(j)) + ...
                    B(i,j)*sin(theta(i) -theta(j)));
            else
                J_12(p,q) = P(i)/v(j) + G(i,i)*v(i);
            end
        end
    end

    for p = 1:len_Q_idx
        i = nzQ_idx(p);
        % J_21 ---------------------------------------
        for q = find(Y_Bus(i, P_idx) ≠ 0) % ==> iterate only for ...
            non-zero Y(i,j)
            j = P_idx(q);
            if i ≠ j
                J_21(p,q) = -v(i)*v(j)*(G(i,j)*cos(theta(i) - theta(j)) ...
                    + B(i,j)*sin(theta(i) -theta(j)));
            else
                J_21(p,q) = P(i) - G(i,i)*v(i)^2;
            end
        end
        % J_22 ---------------------------------------
        for q = find(Y_Bus(i, nzQ_idx) ≠ 0) % ==> iterate only for ...
            non-zero Y(i,j)
            j = nzQ_idx(q);
            if i ≠ j
                J_22(p,q) = v(i)*(G(i,j)*sin(theta(i) - theta(j)) - ...
                    B(i,j)*cos(theta(i) -theta(j)));
            else
```

```
                J_22(p,q) = Q(i)/v(i) - B(i,i)*v(i);
            end
        end
    end

    % J ----------------------------------------
    J = [J_11, J_12; J_21, J_22];

    %==================== x(i+1) =================================

    % Calculate next values of unknow vector x by LU factorization ("\").
    x_next = [theta(P_idx);v(nzQ_idx)] - J\F;

    % Update voltage and angle vectors with new values.
    theta(P_idx)   = x_next(1:len_P_idx);      % New voltage angles in ...
       radians.
    v(nzQ_idx)     = x_next(len_P_idx+1:end);  % New voltages magnitudes ...
       in p.u.
    iteration_num = iteration_num + 1;

end

end
```