```
% EE454_Project_Dosch_Htut_Olivas_Shafawi.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

% script to run PowerFlow program
% NOTE: Do not modify 'EE454_Project_InputData.xlsx' file which contains
%       the given data (from the manual) formatted in a particular way.
%       The zip folder already ocntains 'EE454_Project_OutputData.xlsx'.
%       To run the program from a fresh start, simply rename the outputFile
%       variable, and the program will create a new file with the same
%       output values.

clear all;

% inputFile contains data for Line_Data for base case, con1, con2,
% Load_Data, and PV_Data
inputFile = 'EE454_Project_InputData.xlsx';
baseCaseInput = 'Line_Data';
case1Input = 'Line_Data_con1';
case2Input = 'Line_Data_con2';

% outputFile has four sheets (generalOutput, lineDataOutput,
% iterRecord, genPowerInfo) for each case for a total of 12 sheets
outputFile = 'EE454_Project_OutputData.xlsx';

% generalOutput has P, Q, V, theta, VlimitCheck for each bus
baseCaseGeneralOutput = 'BaseCase';
case1GeneralOutput = 'Con1Case';
case2GeneralOutput = 'Con2Case';

% lineDataOutput has |S|, P, Q, FlimitCheck for each line
baseCaseLineDataOutput = 'LineDataBaseCase';
case1LineDataOutput = 'LineDataCon1Case';
case2LineDataOutput = 'LineDataCon2Case';

% iterRecord has a log of max real and reactive mismatches of
% each NR iteration
baseCaseIterRecord = 'iterRecordBaseCase';
case1IterRecord = 'iterRecordCase1';
case2IterRecord = 'iterRecordCase2';

% genPowerInfo has P, Q values for each generator
baseCaseGenPowerInfo = 'genPowerInfoBaseCase';
case1GenPowerInfo = 'genPowerInfoCase1';
case2GenPowerInfo = 'genPowerInfoCase2';

% call main function for three different cases
main(inputFile, outputFile, baseCaseInput, baseCaseGeneralOutput,...
    baseCaseLineDataOutput, baseCaseIterRecord, baseCaseGenPowerInfo);
main(inputFile, outputFile, case1Input, case1GeneralOutput,...
    case1LineDataOutput, case1IterRecord, case1GenPowerInfo);
main(inputFile, outputFile, case2Input, case2GeneralOutput,...
    case2LineDataOutput, case2IterRecord, case2GenPowerInfo);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% main.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function main(inputFile, outputFile, inputLineData, generalOutput,...
    lineDataOutput, iterRecordOutput, powerInfoOutput)
%{
runs the powerflow program by importing in the given data excel file and
exporting four output data sheets (generalOutput, lineDataOutput,
iterRecord, genPowerInfo)
inputs:
```

```
    inputFile: excel file filled with given data (from the manual)
    outputFile: targeted excel file to export four output sheets
    generalOutput: info on P, Q, V, theta, VlimitCheck for each bus
    lineDataOutput: info on |S|, P, Q, FlimitCheck for each line
    iterRecordOutput: log of max real and reactive mismatches of
                      each NR iteration
    powerInfoOutput: info on P, Q values for each generator
outputs:
    N/A: void function
%}
    S_BASE = 100; %MVA
    EPS = 0.1/S_BASE;
    thetaSwing = 0;

    % read in data of the transmission lines. separate for bus renumbering
    Ydata = xlsread(inputFile, inputLineData);
    sendingBuses = Ydata(:, 1);
    receivingBuses = Ydata(:, 2);
    RXBvalues = Ydata(:, [3, 4, 5]);
    % assumption: the Nth bus will have a connection in the system
    N = max([sendingBuses; receivingBuses]);

    % read in PV data and manipulate as necessary
    PVdata = xlsread(inputFile, 'PV_Data');
    PVdata(:,2) = PVdata(:,2)./S_BASE;
    m = length(PVdata);
    Vswing = PVdata(1, 3);
    % list of buses in the system which are PV
    PV_buses = PVdata((2:end), 1);
    PV = [PVdata((2:end), 2); PVdata((2:end), 3)];

    % create dictionary for bus renumbering
    dictionary = createDictionary(PV_buses, N);

    % renumber buses and prepare data for creating the Y matrix
    sendingBusesRenumbered =   renumberBuses(sendingBuses, dictionary);
    receivingBusesRenumbered = renumberBuses(receivingBuses, dictionary);
    YdataRenumbered = [sendingBusesRenumbered, ...
                       receivingBusesRenumbered, RXBvalues];

    % create Y matrix
    Y = createY(YdataRenumbered, N);

    % read in the PQ data of the loads
    % renumber to line up with new convention
    PQ = xlsread(inputFile, 'Load_Data');
    PQ_original = [PQ(:, 2); PQ(:, 3)];
    PQ_original = PQ_original./S_BASE;
    PQ_renumbered = renumberPQ(PQ_original, dictionary, N);

    % initial guess with all theta = 0 and all V = 1 pu
    x = [zeros(N - 1, 1); ones(N - m, 1)];

    % form initial mismatch equations
    [f_x, f_comp] = createMismatch(x, Y, N, m, PV, PQ_renumbered, ...
                                   Vswing, thetaSwing);

    %prepare iteration record for looping.
    %we have no way of knowing the size, so it will grow each time
    iterationRecord = [];

    % perform newton raphson until convergence is satisfied
    count = 1;
    while max(f_x) > EPS
        iterationRecord = [iterationRecord; recordIteration(f_x, ...
```

```
                             S_BASE, count, dictionary, N, m)];
        jacobian = createJacobian(x, Y, N, m, PV, f_comp, Vswing, thetaSwing);
        x = newtonRaphson(jacobian, f_x, x);
        f_x = createMismatch(x, Y, N, m, PV, PQ_renumbered, Vswing, thetaSwing);
        count = count + 1;
    end

    % extract all the renumbered data
    [theta_renumbered, V_renumbered, P_renumbered, Q_renumbered] = ...
    solveExplicitEquations(x, Y, N, m, PV, PQ_renumbered, PV_buses, Vswing, ...
    thetaSwing, S_BASE, outputFile, powerInfoOutput);

    % recover the original numbering
    theta_original = recover(theta_renumbered, [1; dictionary], N);
    theta_deg = (180/pi).*theta_original;
    V_original = recover(V_renumbered, [1; dictionary], N);
    P_original = S_BASE.*(recover(P_renumbered, [1; dictionary], N));
    Q_original = S_BASE.*(recover(Q_renumbered, [1; dictionary], N));

    % determine whether the calculated V values exceed the range:
    % 0.95 < V < 1.05
    % genearlData includes P, Q, V, theta values, V limit check at each bus
    VLimit = checkVLimit(V_original);
    busNumber = createBusNumber(N);
    generalData = [busNumber, theta_deg, V_original, P_original, ...
                   Q_original, VLimit];
    generalLabel = ["Bus Number", "Angle (degrees)", "V (p.u.)", ...
              "P (MW)", "Q (MVAr)", "Exceeds Vlimit?"];
    xlswrite(outputFile, [generalLabel; generalData], generalOutput);

    % lineData has |S|, P, Q, Fmax check for each transmission line
    lineData = createLineData(S_BASE, V_original, theta_original, Ydata);
    lineLabel = ["sendingBus", "receivingBus", "|S| (MVA)", ...
                 "P (MW)", "Q (MVAr)", "Exceeds Fmax?"];
    xlswrite(outputFile, [lineLabel; lineData], lineDataOutput);

    iterLabel = ["Iteration Number",...
                 "Max. P" + newline + "Mismatch Magnitude (MW)",...
                 "Bus Number",...
                 "Max. Q" + newline + "Mismatch Magnitude (MVAr)",...
                 "Bus Number"];
    xlswrite(outputFile,[iterLabel; iterationRecord],iterRecordOutput);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createDictionary.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function dictionary = createDictionary(PV_buses, N)
%{
createDictionary
  creates a dictionary for renumbering other data in the system
  input:
      PV_buses: column list of numbers in the system which are PV buses
      N: number of buses in the sytem
  output:
      dictionary: look up table to be used for renumbering data
%}

    ref = zeros(N -1, 1);
    for j = 1:(N - 1)
        ref(j) = j + 1;
    end
    dictionary = ref;
```

```
    for k = 1:length(PV_buses)
        if PV_buses(k) ~= (k + 1)
            dictionary(k) = PV_buses(k);
            dictionary(PV_buses(k) - 1) = ref(k);
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% renumberBuses.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function newBusData = renumberBuses(buses, dictionary)
%{
renumberBuses renumbers the buses for the Y matrix
inputs:
    buses: column list of buses to be renumbered
    dictionary: look up table used for renumbering
outputs:
    newBusData: renumbered buses to match up with mismatch equations
 %}

    dictionary = [1; dictionary];
    newBusData = zeros(length(buses), 1);

    for k = 1:length(buses)
        newBusData(k) = dictionary(buses(k));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createY.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function Y = createY(input, N)
%{
outputs admittance matrix Y
inputs:
    input: matrix form of Line_Data table, which has format:
    sendingBus receivingBus  R    X    B    Fmax
    ---------------------------------------------
        1             2       #    #    #    #
    and is (# of Lines x 6) big
    N: number of buses in the system
outputs:
    Y: admittance matrix
%}

    % Y is NxN
    Y = zeros(N);

    % modify input to get Y
    for i = 1:length(input(:, 1))
        % adding R+jX
        currentRow = input(i,:);
        line1 = currentRow(1);
        line2 = currentRow(2);
        Y_total = 1/(currentRow(3) + j * currentRow(4));
        Y(line1, line1) = Y(line1, line1) + Y_total;
        Y(line2, line2) = Y(line2, line2) + Y_total;
        Y(line1, line2) = Y(line1, line2) - Y_total;
        Y(line2, line1) = Y(line2, line1) - Y_total;
```

```matlab

        % adding shunt reactances
        B_total = currentRow(5);
        if B_total ~= 0
            % Y = G + jB
            % since G is zero, Y = jB
            % in pi model, each bus 'gets' half of Y
            halfY = 0.5 * (j * B_total);
            Y(line1, line1) = Y(line1, line1) + halfY;
            Y(line2, line2) = Y(line2, line2) + halfY;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% renumberPQ.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function PQ_renumbered = renumberPQ(PQ_original, dictionary, N)
%{
renumberPQ: renumber the PQ data of the loads in the system
inputs:
    PQ_original: original data for the PQs of the loads
    dictionary: look up table for renumbering
    N: number of buses in the system
output:
    PQ_renumbered: renumbered PQ data of the loads
%}
    P_original = PQ_original(1: N - 1);
    Q_original = PQ_original(N: length(PQ_original));
    P_renumbered = zeros(length(P_original), 1);
    Q_renumbered = zeros(length(Q_original), 1);

    for k = 1:(N - 1)
        P_renumbered(k) = P_original(dictionary(k) - 1);
        Q_renumbered(k) = Q_original(dictionary(k) - 1);
    end

    PQ_renumbered = [P_renumbered; Q_renumbered];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createMismatch.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function [f_x_new, f_comp] = createMismatch(x, Y, N, m, PV, PQ, Vswing, thetaSwing)
%{
createMismatch Creates the mismatch equations
inputs:
    x: column vector of unknowns theta(2:N) and V(m+1:N) (
    Y: admittance matrix (N x N)
    N: number of buses in the system
    m: m - 1 = number of PV buses in the system
    PV: values of PV from the generators in the system (2*m - 2)
    PQ: values of PQ from the loads in the system (2*N - 2)
    Vswing: voltage at the swing bus (bus 1)
    thetaSwing: angle at the swing bus (bus 1)
outputs:
    f_x_new: new mismatch equations
%}

    %voltage values are taken from PV buses if known. if not known, they are
    %taken from the values of x[]
```

```
    V      = [Vswing; PV(m : length(PV)); x(N : length(x))];
    %theta values are the first 2:N entries in x
    theta  =  [thetaSwing; x(1 : (N - 1))];
    %P values from PV will be positive injections into the system (ie a gen.)
    Pgen   = [PV(1 : (m - 1)); zeros(N-m,1)];
    %P values from PQ will be negative injections into the system (ie a load)
    Pload  = PQ(1 : (N - 1));
    %Q values taken from the "bottom" of PQ will be negative injections
    %positive Q injections for the generators will be explicit equations
    Qload  = PQ(N : length(PQ));
    %pKnown = pGen - pLoad
    Pknown = Pgen - Pload;

    %initiate f_x_new (new mismatch equations)
    f_x_new = zeros(2*N - m - 1, 1);
    f_comp  = zeros(N - 1, 1);
    %loop through the P mismatches first. k and i represent the same indices as
    %the P and Q equations in the lecture 10 notes
    for k = 2:N
        Pcomp = 0;
        for i = 1:N
            sum = V(k)*V(i) * ((real(Y(k,i))*cos(theta(k) - theta(i))) +
            (imag(Y(k,i))*sin(theta(k) - theta(i))));
            Pcomp = Pcomp + sum;
        end
        f_x_new(k - 1) = Pcomp - Pknown(k - 1);
        f_comp (k - 1) = Pcomp;
    end

    %next loop through Q mismatches
    for k = (m+1):N
        Qcomp = 0;
        for i = 1:N
            sum = V(k)*V(i) * ((real(Y(k,i))*sin(theta(k) - theta(i))) -
            (imag(Y(k,i))*cos(theta(k) - theta(i))));
            Qcomp = Qcomp + sum;
        end
        f_x_new(k + N - m -1) = Qcomp + Qload(k - 1);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% recordIteration.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function iterationRecord = recordIteration(f_x, S_BASE, count, dictionary, N, m)
%{
iterationRecord: keeps a log of the maxiumum real and reactive mismatches
of each Newton Raphson iteration
inputs:
    f_x: the current mismatch equations
    S_BASE: apparent power base for per unit scaling
    count: the iteration number
    dictionary: look up table used for renumbering
    N: number of buses in the system
    m: m - 1 = number of PV buses in the system
output:
    iterationRecord: max real and reactive power mismatches for the current
    iteration and where at which that occurs

    iterationNumber    max P     bus     max Q    bus
    -----------------------------------------------
        count             ~       ~        ~       ~
%}
```

```
    iterationRecord = [];
     iterationRecord(1) = count;
    [P_max, P_index] = max(abs(f_x(1 : N - 1)));
    [Q_max, Q_index] = max(abs(f_x(N : length(f_x))));

    dict = [1;dictionary];

    iterationRecord(2) = S_BASE * P_max;
    iterationRecord(3) = dict(P_index + 1);
    iterationRecord(4) = S_BASE * Q_max;
    iterationRecord(5) = dict(Q_index + m);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createJacobian.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function J = createJacobian(x, Y, N, m, PV, pCompFromMismatch, Vswing, thetaSwing)
%{
creates Jacobain matrix
inputs:
    x: column vector of unknowns [theta2,...,theta12,V6,...,V12]
    Y: admittance matrix (NxN)
    N: number of buses in the system
    m: m - 1 = number of PV buses in the system
    PV: values of PV from the generators in the system (2*m - 2)
    pCompFromMismatch: values of computed P values from mismatch.m
    Vswing: voltage at the swing bus (bus 1)
    thetaSwing: angle at the swing bus (bus 1)
outputs:
    J: Jacobian matrix
%}

    % Assign V values.
    % V1 is from swing bus.
    % V2 to V5 are known values from PV buses (renumbered).
    % V6 to V12 are initial values of x.
    V = [Vswing; PV(m : length(PV)); x(N : length(x))];

    % theta1 is the only known values from swing bus.
    % theta2 to theta12 are initial values of x.
    theta = [thetaSwing; x(1:(N-1))];

    % compute P values from the power flow equations.
    % add filler NaN value to make sure we don't use P1.
    % we calculated pComp in mismatch in createMismatch.m,
    % thus recycle it instead of recomputing it.
    pComp = [NaN; pCompFromMismatch];

    % compute Q values from the power flow equations.
    % add filler NaN value to make sure we don't use Q1.
    % we only calculated Q6 to Q12 in mismatch.
    % since we need Q2 to Q12 here, we use the same power flow
    % equation to compute Q values in computeQ.m
    qComp = [NaN; computeQ(Y, N, m, theta, V)];

    % J is J_size x J_size matrx  (18x18)
    J_size = 2*N-1-m;

    % four quadrants have different sizes
    J11 = zeros(N-1); %11x11
    J12 = zeros(N-1, J_size-N+1); %11x7
    J21 = zeros(J_size-N+1, N-1); %7x11
```

```
    J22 = zeros(J_size-N+1, J_size-N+1); %7x7

    % The following values for each quadrant are assigned based on
    % the equations from Lecture Note 10
    % J11 matrix
    for k = 2:N
        for j = 2:N
            if j == k
                J11(k-1,k-1) = -1*qComp(k)-(V(k)^2)*imag(Y(k,k));
            else
                J11(k-1,j-1) = V(k)*V(j)*(real(Y(k, j))*sin(theta(k)-theta(j))-...
                imag(Y(k,j))*cos(theta(k)-theta(j)));
            end
        end
    end

    % J21 matrix
    for k = m+1:N
        for j = 2:N
            if j == k
                J21(k-m,j-1) = pComp(k) - real(Y(k,k))*(V(k)^2);
            else
                J21(k-m,j-1) = -1*V(k)*V(j)*(real(Y(k,j))*cos(theta(k)-theta(j))+...
                imag(Y(k,j))*sin(theta(k)-theta(j)));
            end
        end
    end

    % J12 matrix
    for k = 2:N
        for j = m+1:N
            if j == k
                J12(k-1,j-m) = pComp(k)/V(k) + real(Y(k,k))*V(k);
            else
                J12(k-1,j-m) = V(k)*(real(Y(k,j))*cos(theta(k)-theta(j))+...
                imag(Y(k,j))*sin(theta(k)-theta(j)));
            end
        end
    end

    % J22 matrix
    for k = m+1:N
        for j = m+1:N
            if j == k
                J22(k-m,k-m) = qComp(k)/V(k) - imag(Y(k,k))*V(k);
            else
                J22(k-m,j-m) = V(k)*(real(Y(k,j))*sin(theta(k)-theta(j))-...
                imag(Y(k,j))*cos(theta(k)-theta(j)));
            end
        end
    end

    J = [J11, J12; J21, J22];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% computeQ.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function Qcomp = computeQ(Y, N, m, theta, V)
%{
computes Q values for bus 2 to bus 12
inputs:
    Y: admittance matrix (Nx)
```

```
    N: number of buses in the system
    m: m - 1 = number of PV buses in the system
    theta: values from the first 2:N entries in x
    V: values from the last N+1:length(x) entries in x
outputs:
    Qcomp: computed Q2 to Q12 values
%}

    % Qcomp is 11x1 vector
    Qcomp = zeros(N-1, 1);
    for k = 2:N
        Qsum = 0;
        for i = 1:N
            sum = V(k)*V(i) * (real(Y(k,i))*sin(theta(k) - theta(i)) ...
                            - (imag(Y(k,i))*cos(theta(k) - theta(i))));
            Qsum = Qsum + sum;
        end
        % index offset
        Qcomp(k-1) = Qsum;
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% newtonRaphson.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function x_new = newtonRaphson(J, f_x, x_old)
%{
newtonRaphson: performs newton raphson in order to update the x varialbes
and converge to a solution
inputs:
    J: the Jacobian matrix
    f_x: the mismatch equations
    x_old: the current x values, [theta2...thetaN;Vm+1...VN]
outputs: ...
    the updated x values
%}

    invJ = inv(J);
    negInvJ = -1*invJ;
    dx = negInvJ*f_x;
    x_new = x_old + dx;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% solveExplicitEquations.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function [theta, V, P, Q] = ...
solveExplicitEquations(x, Y, N, m, PV, PQ, ...
PV_buses, Vswing, thetaSwing, S_BASE, outputFile, powerInfoOutput)
%{
solveExplicitEquations
  solves the explicit equations that were not solved by newton raphson.
  this function also gathers all the data from the processing and
  compiles each value (theta, V, P, Q) at each bus to be returned to the
  user. this data is still renumbered. explicit values to be solved are
  P and Q at the swing bus as well as Qgen from the generators on the PV
  buses. This function also writes the P and Q values of the generators
  to the output excel file.

inputs:
```

```
    x: column vector of unknowns theta(2:N) and V(m+1:N) (
    Y: admittance matrix (N x N)
    N: number of buses in the system
    m: m - 1 = number of PV buses in the system
    PV: values of PV from the generators in the system (2*m - 2)
    PQ: values of PQ from the loads in the system (2*N - 2)
    PV_buses: list of buses in the system that are PV buses
    Vswing: voltage at the swing bus (bus 1)
    thetaSwing: angle at the swing bus (bus 1)
    S_BASE: apparent power base for per unit scaling
outputs:
    theta: theta values at buses 1-N
    V: voltage values at buses 1-N
    P: real power values at buses 1-N
    Q: reactive power values at buses 1-N
%}

    %voltage values are taken from PV buses if known. if not known, they are
    %taken from the values of x[]
    V      = [Vswing; PV(m : length(PV)); x(N : length(x))];
    %theta values are the first 2:N entries in x
    theta  =  [thetaSwing; x(1 : (N - 1))];
    %P values from PV will be positive injections into the system (ie a gen.)
    Pgen   = [PV(1 : (m - 1)); zeros(N-m,1)];
    %P values from PQ will be negative injections into the system (ie a load)
    Pload  = PQ(1 : (N - 1));
    %Q values taken from the "bottom" of PQ will be negative injections
    %positive Q injections for the generators will be explicit equations
    Qload  = PQ(N : length(PQ));
    %pKnown = pGen - pLoad
    Pknown = Pgen - Pload;

    Qgen = zeros(N - 1, 1);
    Pswing = 0;
    k = 1;

    %solve real power injection from the swing bus
    for i = 1:N
        sum = V(k)*V(i)*(real(Y(k, i))*cos(theta(k) - theta(i)) + ...
                   imag(Y(k, i))*sin(theta(k) - theta(i)));
        Pswing = Pswing + sum;
    end

    %solve reactive power injection from the swing bus
    Qswing = 0;
    for i = 1:N
        sum = V(k)*V(i)*(real(Y(k, i))*sin(theta(k) - theta(i)) - ...
                   imag(Y(k, i))*cos(theta(k) - theta(i)));
        Qswing = Qswing + sum;
    end

    %solve for reactive power injections from the PV buses
    for k = 2:m
        Qcomp = 0;
        for i = 1:N
            sum = V(k)*V(i)*(real(Y(k, i))*sin(theta(k) - theta(i)) - ...
                   imag(Y(k, i))*cos(theta(k) - theta(i)));
            Qcomp = Qcomp + sum;
        end
        Qgen(k - 1) = Qcomp + Qload(k - 1);
    end

    %compile this data into matrices and write to excel file
    Qknown = Qgen - Qload;
```

```
    P = [Pswing; Pknown];
    Q = [Qswing; Qknown];

    genLabels = ["Bus Number", "Real Power Injection (MW)", ...
                 "Reactive Power Injection (MVAr)"];
    genInfo = S_BASE.*[[Pswing; Pgen(1: (m-1))], [Qswing; Qgen(1: (m-1))]];
    genToExcel = [genLabels;[[1; PV_buses], genInfo]];
    xlswrite(outputFile, genToExcel, powerInfoOutput);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% recover.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function originalValues = recover (renumberedValues, dictionary, N)
%{
    recover_x rearrange a column vector so that the original bus numbering is
    satisfied
      inputs:
          renumberedValues: column vector that has been renumbered for
          processing
          dictionary: look up table for renumbering
          N: number of buses in the system
      output:
          originalValues: column vector returned to original numbering
%}
    originalValues = zeros(N, 1);
    for k = 1:N
        originalValues(k) = renumberedValues(dictionary(k));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% checkVLimit.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function VLimit = checkVLimit(V)
%{
checks if calculated V exceeds the V range:
0.95 < V < 1.05
input:
    V: calculated V vector
output:
    VLimit: 0 if V is within the limit, 1 otherwise
%}

    V_hi = 1.05;
    V_lo = 0.95;

    VLimit = zeros(length(V), 1);

    for i = 1:length(V)
        if V(i) < V_lo && V(i) > V_hi
            VLimit(i) = 1;
        else
            VLimit(i) = 0;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createBusNumber.m
```

```matlab
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function busNumber = createBusNumber(N)
%{
creates bus number column vector from 1 to N
inputs:
    N: number of buses in the system
outputs:
    busNumber: column vector from 1 to N
%}
    busNumber = zeros(N,1);
    for i = 1:N
        busNumber(i) = i;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createLineData.m
% % by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

function lineData = createLineData(S_BASE, V, theta, Ydata)
%{
computes powerflow data for each transmission line
inputs:
    S_BASE: base value for complex power
    V: final V values at all buses
    theta: final theta values at all buses (in rad)
    Ydata: original Line_data table
outputs:
    lineData: matrix filled with powerflow data (S,P,Q,FmaxCheck)...
        each line
%}

    % col 1 and col 2 are sending and receiving buses
    % col 3, 4, and 5 are magnitude(S), P, an Q
    % col 6 checks whether magnitue(S) exceeds given Fmax
    % thus, lineData is #ofLines x 6 big
    lineData = zeros(length(Ydata),6);
    sendIndex = 1;
    recIndex = 2;
    sIndex = 3;
    pIndex = 4;
    qIndex = 5;
    boundIndex = 6;

    % iterate through each line
    for r = 1:length(Ydata)
        currentRow = Ydata(r, :);
        sendingBus = currentRow(1);
        receivingBus = currentRow(2);
        V1 = V(sendingBus)*exp(j*theta(sendingBus));
        V2 = V(receivingBus)*exp(j*theta(receivingBus));
        Z = currentRow(3) + j*currentRow(4);
        dV = V1-V2;
        I = dV/Z;
        % S is the product of Vdrop and conj(I)
        S = dV*conj(I);
        lineData(r, sendIndex) = sendingBus;
        lineData(r, recIndex) = receivingBus;
        % multiply by S_BASE to get actual values
        lineData(r, sIndex) = S_BASE * abs(S);
        lineData(r, pIndex) = S_BASE * real(S);
        lineData(r, qIndex) = S_BASE * imag(S);

```

```
        % check if mag(S) exceeds given Fmax
        % 0 is no, 1 is yes
        if ~isnan(Ydata(r, 6))
            if S_BASE * abs(S) < Ydata(r,6)
                lineData(r, boundIndex) = 0;
            else
                lineData(r, boundIndex) = 1;
            end
        else % no Fmax given
             lineData(r, boundIndex) = 0;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The following are UnitTests for different function:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%unit test for the jacobian

%create random numbers (hope N > m)
N = round(30*rand());
m = round(5*rand());
Yreal = rand(N,N);
Yimag = rand(N,N);
Y = Yreal + j*Yimag;
PV = 1.2*rand(2*m - 2, 1);
PQ = 1.4*rand(2*N - 2, 1);
x = rand(2*N - m - 1, 1);
vswing = 1.05;
tswing = 0;

%run the function and see if reasonable
A = createJacobian(x, Y, N, m, PV, PQ(2:N), vswing, tswing);
%[theta_test, v_test] = recover_x(x, PV, dict, tswing, vswing, m, N);
Ainv = inv(A);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% createMismatchUnitTest.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

%unit test for creating mismatching
N = round(30*rand());
m = round(5*rand());
Yreal = rand(N,N);
Yimag = rand(N,N);
Y = Yreal + j*Yimag;
PV = 1.2*rand(2*m - 2, 1);
PQ = 1.4*rand(2*N - 2, 1);
x = rand(2*N - m - 1, 1);
vswing = 1.05;
tswing = 0;

A = createMismatch(x, Y, N, m, PV, PQ, vswing, tswing);
[theta_test, v_test] = recover_x(x, PV, dict, tswing, vswing, m, N);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% renumberUnitTest.m
% by Brad Dosch, Alex Htut, Bernardo Olivas, Muhammad Shafawi

%unit test for the renumbering scheme
```

```
%hardcode values that we know the answers for
PV_buses = [2;3;10;12];
N = 12;
m = 5;
dict = createDictionary(PV_buses, N);
PQ_orig = 1.4*rand(2*N - 2, 1);
PQ_renum = renumberPQ(PQ_orig, dict, N);
buses_orig1 = [1;1;2;2;2;3;4;4;5;6;6;6;7;7;8;10;11];
buses_orig2 = [2;5;3;4;5;4;5;7;6;9;10;11;8;12;9;11;12];
busesRenum1 = renumberBuses(buses_orig1, dict);
busesRenum2 = renumberBuses(buses_orig2, dict);

%test these values against our theoretical results

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%                   THE END                    %%%%%%%%%%%%%%%%%%
```