**Assignment 4**

Submitted by Athul Jose P

WSU ID 011867566

**Executive Summary**

***State Estimation***

Steps in main.m function

1. Initializing 14 bus and importing data
2. Making Ybus by calling y\_bus\_calc.m (with taps or without taps should be mentioned)
3. Calculating the scheduled active power (P) and reactive power (Q)
4. Finding bus types and assigning to vectors
5. Initializing Voltage magnitude and angles
6. Calling Newton Raphson Function (NR.m)
7. Calculating P & Q after convergence by calling PQ\_calc.m
8. Case1: Doing economic dispatch by calling ED.m
9. Calculating Ptotal for case 2. Then does ED
10. Case3: Doing OPF without line limit by calling OPF.m
11. Case4: Doing OPF with line limit

Steps in y\_bus\_calc.m function

1. Initializing Ybus with zeros
2. Calculating diagonal and off diagonal elements
3. Changing terms if tap is present

Steps in NR.m function

1. Initializing indexes and deltas
2. Starting iteration loop which will terminate either if converged or 100 iterations
3. Calling dpdq.m for calculating mismatch
4. Calling J\_calc.m for calculating Jacobian
5. Calling fwd\_bwd.m for calculating the ΔV and Δδ
6. Updating del\_V and del\_T (magnitude and angle) for next iteration
7. Updating the error. Here the error is taken as the maximum of absolute of deltas (ΔV and Δδ)

Steps in dpdq.m function

1. Initializing P & Q as zeros
2. Calculating P for PV bus and P & Q for PQ bus

Steps in J\_calc.m function

1. Calculating J1 with loops according to limits (n\_bus-1, n\_bus-1)
2. Calculating J2 with loops according to limits (n\_bus-1, n\_pq)
3. Calculating J3 with loops according to limits (n\_pq, n\_bus-1)
4. Calculating J4 with loops according to limits (n\_pq, n\_pq)
5. Combining all to make J

Steps in fwd\_bwd.m function

1. Calling LU.m for calculating Lower and Upper elements
2. Doing the backward substitution
3. Doing the forward substitution

Steps in LU.m function

1. Making the Q matrix
2. Dividing it into L & U matrices

Steps in PQ\_calc.m function

1. Calculates the P & Q of each bus

Steps in ED.m function

1. Creating cost functions
2. Creating functions in sym environment
3. Finding the solution

Steps in OPF.m function

1. Initializing OPF parameters and syms variables
2. Starting loop for OPF with error tolerance as 0.1
3. Calculating different P&Q injections
4. Calculating different differential terms
5. Correcting the error values

**Results**

1. Economic Dispatch with P total = 259MW

**Economic Dispatch with total P = 259MW**

**Share of Generator-1 is 50.3636MW**

**Share of Generator-2 is 208.6364MW**

**Cost of Generation is 8.4029/MWhr**

**Total Cost is $1957.300000/hr**

2. Economic Dispatch with P total = P1 + P2

**Economic Dispatch with total P = P1 + P2**

**Share of Generator-1 is 57.6691MW**

**Share of Generator-2 is 214.7242MW**

**Cost of Generation is 8.4614/MWhr**

**Total Cost is $2070.200000/hr**

3. OPF without line limit

**Optimal Power Flow without line limits**

**Share of Generator-1 is 57.6691MW**

**Share of Generator-2 is 214.6958MW**

**Total Cost is $2070.000000/hr**

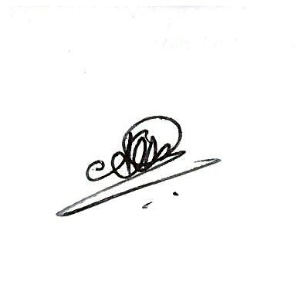
4. OPF with line limit

**Optimal Power Flow with line limits**

**Share of Generator-1 is 50.7659MW**

**Share of Generator-2 is 217.0886MW**

**Total Cost is $2032.000000/hr**

Statement

I, Athul Jose P, states that all the code written and submitted here is completely done by me. I have not taken any help from others or any online resources.

Athul Jose P

*Appendix*

**% main.m**

**clc**

**clear all; close all;**

**% Initializing 14 bus and importing data**

**n\_bus = 14;**

**bus\_data = importdata('ieee14bus.txt').data;**

**branch\_data = importdata('ieee14branch.txt').data;**

**% Ybus formation**

**t = 1; % 0 for without tap, 1 for with tap**

**Y = y\_bus\_calc(n\_bus,bus\_data,branch\_data,t);**

**% Scheduled power calculation**

**base\_MVA = 100;**

**P\_inj = (bus\_data(:,8) - bus\_data(:,6)) / base\_MVA;**

**Q\_inj = (bus\_data(:,9) - bus\_data(:,7)) / base\_MVA;**

**% Creating indexes for functions**

**pv\_i = find(bus\_data(:,3) == 2);**

**pq\_i = find(bus\_data(:,3) == 0);**

**n\_pv = length(pv\_i);**

**n\_pq = length(pq\_i);**

**fr = branch\_data(:,1);**

**to = branch\_data(:,2);**

**n\_br = length(fr);**

**B = branch\_data(:,9);**

**% Initializing Voltage magnitude and angles**

**V = bus\_data(:,11);**

**V(find(V(:)==0)) = 1;**

**T = zeros(n\_bus,1);**

**% Executing NR for Power Flow results**

**[V\_data,T\_data,T1] = NR(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y,n\_pq,pq\_i);**

**V = V\_data(:,size(V\_data,2));**

**T = T\_data(:,size(T\_data,2));**

**% P,Q calculation after convergence**

**[P,Q] = PQ\_calc(V,T,Y);**

**% Case 1: Total P = 259 MW**

**P\_tot = 259;**

**P\_eq = [8.0 0.004; 6.4 0.0048];**

**[P\_1, P\_2, cost, x] = ED(P\_eq,P\_tot);**

**disp('Economic Dispatch with total P = 259MW')**

**disp(['Share of Generator-1 is ' num2str(P\_1) 'MW'])**

**disp(['Share of Generator-2 is ' num2str(P\_2) 'MW'])**

**disp(['Cost of Generation is ' num2str(x) '/MWhr'])**

**fprintf('Total Cost is $%f/hr\n', cost)**

**disp('------------------------------------------------------------')**

**% Case 2: Total P = P1 + P2**

**P\_tot = (P(1) + P(2) + bus\_data(2,6)/base\_MVA) \* base\_MVA;**

**[P\_1, P\_2, cost, x] = ED(P\_eq,P\_tot);**

**disp('Economic Dispatch with total P = P1 + P2')**

**disp(['Share of Generator-1 is ' num2str(P\_1) 'MW'])**

**disp(['Share of Generator-2 is ' num2str(P\_2) 'MW'])**

**disp(['Cost of Generation is ' num2str(x) '/MWhr'])**

**fprintf('Total Cost is $%f/hr\n', cost)**

**disp('------------------------------------------------------------')**

**syms T\_2**

**PG0 = [P\_1; P\_2];**

**QG0 = [Q(1); Q(2)]\*base\_MVA;**

**bus\_no = [1 2];**

**eq = P\_tot == bus\_data(2,6) + base\_MVA\*(real(Y(2,2))\*V(2)^2 + abs(Y(2,1)\*V(1)\*V(2))\*cos(angle(Y(2,1)) + T\_2));**

**T\_2\_soln= solve(eq, T\_2);**

**T\_2\_Val = T\_2\_soln(1);**

**% Case 3: OPF without line limits**

**constraint = 0;**

**[P\_1,P\_2, cost] = OPF(V,T,Y,PG0,QG0,bus\_no,T\_2\_Val,base\_MVA,constraint,bus\_data,n\_bus,n\_pq,pq\_i,P\_inj,Q\_inj,P\_eq);**

**disp('Optimal Power Flow without line limits')**

**disp(['Share of Generator-1 is ' num2str(P\_1) 'MW'])**

**disp(['Share of Generator-2 is ' num2str(P\_2) 'MW'])**

**fprintf('Total Cost is $%f/hr\n', cost)**

**disp('------------------------------------------------------------')**

**% Case 4: OPF with line limits**

**constraint = 1;**

**[P\_1,P\_2, cost] = OPF(V,T,Y,PG0,QG0,bus\_no,T\_2\_Val,base\_MVA,constraint,bus\_data,n\_bus,n\_pq,pq\_i,P\_inj,Q\_inj,P\_eq);**

**disp('Optimal Power Flow with line limits')**

**disp(['Share of Generator-1 is ' num2str(P\_1) 'MW'])**

**disp(['Share of Generator-2 is ' num2str(P\_2) 'MW'])**

**fprintf('Total Cost is $%f/hr\n', cost)**

**disp('------------------------------------------------------------')**

**% y\_bus\_calc.m**

**function Y = y\_bus\_calc(N\_bs,D\_bs,D\_br,t)**

**Y = zeros(N\_bs);**

**% Calculating elements of Ybus**

**for k = 1:size(D\_br,1)**

**Y(D\_br(k,1),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,1)) + 1/(D\_br(k,7) + i\*D\_br(k,8)) + i\*D\_br(k,9)/2;**

**Y(D\_br(k,2),D\_br(k,2)) = Y(D\_br(k,2),D\_br(k,2)) + 1/(D\_br(k,7) + i\*D\_br(k,8)) + i\*D\_br(k,9)/2;**

**Y(D\_br(k,1),D\_br(k,2)) = -1/(D\_br(k,7) + i\*D\_br(k,8));**

**Y(D\_br(k,2),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,2));**

**end**

**for k = 1:N\_bs**

**Y(k,k) = Y(k,k) + D\_bs(k,14) + i\*D\_bs(k,15);**

**end**

**% adjusting for taps**

**if(t == 1)**

**for k = 1:size(D\_br,1)**

**if(D\_br(k,15) ~= 0)**

**t = D\_br(k,15);**

**((t^2) / i\*D\_br(k,8));**

**Y(D\_br(k,1),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,1)) + Y(D\_br(k,1),D\_br(k,2)) - (Y(D\_br(k,1),D\_br(k,2)))/(t^2);**

**Y(D\_br(k,1),D\_br(k,1));**

**Y(D\_br(k,1),D\_br(k,2)) = Y(D\_br(k,1),D\_br(k,2))/t;**

**Y(D\_br(k,2),D\_br(k,1)) = Y(D\_br(k,1),D\_br(k,2));**

**end**

**end**

**end**

**end**

**% NR.m**

**function [V\_data,T\_data] = NR(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y,n\_pq,pq\_i)**

**% Initializing index**

**i = 0;**

**Tol = 1;**

**del\_T = zeros(n\_bus,1);**

**del\_V = zeros(n\_bus,1);**

**% Iteration loop**

**while(Tol > 1e-5 & i < 100)**

**i = i+1**

**V = V+del\_V;**

**T = T+del\_T;**

**T\_data(:,i) = T;**

**V\_data(:,i) = V;**

**[del\_P, del\_Q] = dpdq\_calc(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y);**

**dpdq = [del\_P, del\_Q]; % mismatch calculation**

**J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i); % Jacobian calculation**

**delta = fwd\_bwd(J,dpdq); % finding errors**

**del\_T = [0 delta(1:n\_bus-1)]';**

**for j = 1:n\_pq**

**del\_V(pq\_i(j)) = delta(n\_bus+j-1);**

**end**

**Tol = max(abs(delta)) % updating error for convergence**

**end**

**end**

**% dpdq.m**

**function [del\_P, del\_Q] = dpdq\_calc(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y)**

**P = zeros(n\_bus,1);**

**Q = zeros(n\_bus,1);**

**Pi = 1;**

**Qi = 1;**

**for i = 1:n\_bus**

**if(bus\_data(i,3) ~= 3)**

**for j = 1:n\_bus**

**P(i) = P(i) + V(i)\*V(j)\*abs(Y(i,j))\*cos(T(i)-T(j)-angle(Y(i,j)));**

**Q(i) = Q(i) + V(i)\*V(j)\*abs(Y(i,j))\*sin(T(i)-T(j)-angle(Y(i,j)));**

**end**

**del\_P(Pi) = P\_inj(i) - P(i);**

**Pi = Pi+1;**

**if(bus\_data(i,3) == 0)**

**del\_Q(Qi) = Q\_inj(i) - Q(i);**

**Qi = Qi+1;**

**end**

**end**

**end**

**end**

**% J\_calc.m**

**function J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i)**

**% J1 calculation**

**J1 = zeros(n\_bus-1);**

**for i = 1:n\_bus**

**for j = 1:n\_bus**

**if(bus\_data(i,3) ~=3 & bus\_data(j,3) ~=3)**

**if(i==j)**

**for k = 1:n\_bus**

**J1(i-1,j-1) = J1(i-1,j-1)+(V(i)\*V(k)\*abs(Y(i,k))\*sin(angle(Y(i,k))-T(i)+T(k)));**

**end**

**J1(i-1,j-1) = J1(i-1,j-1) - ((V(i)^2) \* (imag(Y(i,i))));**

**else**

**J1(i-1,j-1) = -V(i)\*V(j)\*abs(Y(i,j))\*sin(angle(Y(i,j))-T(i)+T(j));**

**end**

**end**

**end**

**end**

**J1;**

**% J2 calculation**

**J2 = zeros(n\_bus-1,n\_pq);**

**for i = 2:n\_bus**

**for j = 1:n\_pq**

**n = pq\_i(j);**

**if(n == i)**

**for k = 1:n\_bus**

**J2(i-1,j) = J2(i-1,j)+(V(i)\*V(k)\*abs(Y(i,k))\*cos(angle(Y(i,k))-T(i)+T(k)));**

**end**

**J2(i-1,j) = J2(i-1,j) + ((V(i)^2) \* (real(Y(i,i))));**

**else**

**J2(i-1,j) = V(i)\*V(n)\*abs(Y(i,n))\*cos(angle(Y(i,n))-T(i)+T(n));**

**end**

**end**

**end**

**J2;**

**% J3 calculation**

**J3 = zeros(n\_pq,n\_bus-1);**

**for i = 1:n\_pq**

**n = pq\_i(i);**

**for j = 2:n\_bus**

**if(n==j)**

**for k = 1:n\_bus**

**J3(i,j-1) = J3(i,j-1)+(V(n)\*V(k)\*abs(Y(n,k))\*cos(angle(Y(n,k))-T(n)+T(k)));**

**end**

**J3(i,j-1) = J3(i,j-1) - ((V(n)^2) \* (real(Y(n,n))));**

**else**

**J3(i,j-1) = -V(n)\*V(j)\*abs(Y(n,j))\*cos(angle(Y(n,j))-T(n)+T(j));**

**end**

**end**

**end**

**J3;**

**% J4 calculation**

**J4 = zeros(n\_pq);**

**for i = 1:n\_pq**

**n1 = pq\_i(i);**

**for j = 1:n\_pq**

**n2 = pq\_i(j);**

**if(n1==n2)**

**for k = 1:n\_bus**

**J4(i,j) = J4(i,j)+(V(n1)\*V(k)\*abs(Y(n1,k))\*sin(angle(Y(n1,k))-T(n1)+T(k)));**

**end**

**J4(i,j) = - J4(i,j) - ((V(n1)^2) \* (imag(Y(n1,n1))));**

**else**

**J4(i,j) = -V(n1)\*V(n2)\*abs(Y(n1,n2))\*sin(angle(Y(n1,n2))-T(n1)+T(n2));**

**end**

**end**

**end**

**J4;**

**J = [J1, J2; J3, J4];**

**end**

**% fwd\_bwd.m**

**function x = fwd\_bwd(A,b)**

**[L, U] = LU(A);**

**% Forward Substitution**

**for k = 1:length(A)**

**s = 0;**

**for j = 1:k-1**

**s = s + (L(k,j)\*y(j));**

**end**

**y(k) = (b(k) - s) / L(k,k);**

**end**

**% Backward Substitution**

**for k = length(A):-1:1**

**s = 0;**

**for j = k+1:length(A)**

**s = s + (U(k,j)\*x(j));**

**end**

**x(k) = y(k) - s;**

**end**

**end**

**% LU.m**

**function [L, U] = LU(a)**

**Q = zeros(length(a));**

**for j = 1:length(a)**

**for k = j:length(a)**

**s = 0;**

**for m = 1:j-1**

**s = s + (Q(k,m)\*Q(m,j));**

**end**

**Q(k,j) = a(k,j) - s;**

**end**

**if j < length(a)**

**for k = j+1:length(a)**

**s = 0;**

**for m = 1:j-1**

**s = s + (Q(j,m)\*Q(m,k));**

**end**

**Q(j,k) = (a(j,k) - s) / Q(j,j);**

**end**

**end**

**end**

**L = tril(Q);**

**U = Q - L + eye(length(a));**

**end**

**% PQ\_calc.m**

**function [P,Q] = PQ\_calc(V,T,Y)**

**n\_bus = size(V,1);**

**P = zeros(n\_bus,1);**

**Q = zeros(n\_bus,1);**

**for i = 1:n\_bus**

**for j = 1:n\_bus**

**P(i) = P(i) + V(i)\*V(j)\*abs(Y(i,j))\*cos(T(i)-T(j)-angle(Y(i,j)));**

**Q(i) = Q(i) + V(i)\*V(j)\*abs(Y(i,j))\*sin(T(i)-T(j)-angle(Y(i,j)));**

**end**

**end**

**end**

**% ED.m**

**function [P\_1, P\_2, cost, x] = ED(P\_eq,P\_total)**

**y = sym('y', [2 1]);**

**syms P\_tot x**

**y\_x = num2cell(y);**

**C\_1(y) = P\_eq(1, 2)\*y(1)^2 + P\_eq(1, 1)\*y(1);**

**C\_2(y) = P\_eq(2, 2)\*y(2)^2 + P\_eq(2, 1)\*y(2);**

**yfull = [y; x];**

**f(y) = C\_1(y\_x{:}) + C\_2(y\_x{:});**

**c(y) = sum([y\_x{:}].\*[1 1]) - P\_tot;**

**c(y) = subs(c(y\_x{:}), P\_tot, P\_total);**

**y\_x\_all = num2cell(yfull);**

**C(yfull) = C\_1(y\_x{:}) + C\_2(y\_x{:}) - x\*c(y\_x{:});**

**Delta\_C(yfull) = jacobian(subs(C(y\_x\_all{:}), P\_tot, P\_total), yfull).';**

**y\_x\_soln = solve(Delta\_C(y\_x\_all{:}), y\_x\_all{:});**

**P\_1 = double(y\_x\_soln.y1);**

**P\_2 = double(y\_x\_soln.y2);**

**x = double(y\_x\_soln.x);**

**cost = f(P\_1, P\_2);**

**end**

**% OPF.m**

**function [P1,P2, cost] = OPF(V,T,Y,PG0,QG0,bus\_no,T\_2\_Val,base\_MVA,limit,bus\_data,n\_bus,n\_pq,pq\_i,P\_inj,Q\_inj,P\_eq)**

**syms PG\_i QG\_i P12**

**u = sym('u', [1 1]);**

**u\_r = num2cell(u);**

**x0 = sym('x', [n\_bus+n\_pq 1]);**

**x = x0(2:end);**

**% initializing OPF parameters**

**r = 0.06;**

**p\_coef = 1;**

**corr = 100;**

**tol = 0.1;**

**P12\_x = 100.0;**

**P12\_Limit = 5;**

**iter = 1;**

**i = bus\_no(1);**

**j = bus\_no(2);**

**u\_x = PG0(j);**

**xVals = [T\_2\_Val; T(3:end); V(pq\_i)];**

**P1 = PG0(i);**

**Q1\_Val = QG0(i)/base\_MVA;**

**J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i);**

**V = V(2:end);**

**C\_i = P\_eq(1, 2)\*PG\_i^2 + P\_eq(1, 1)\*PG\_i;**

**C\_j(u) = P\_eq(2, 2)\*u^2 + P\_eq(2, 1)\*u;**

**f(u\_r{:}) = C\_i + C\_j(u\_r{:});**

**if limit == 1**

**f(u\_r{:}) = f(u\_r{:}) + p\_coef\*(P12 - P12\_Limit)^2;**

**end**

**display(f(u\_r{:}));**

**% loop for OPF**

**while corr > tol || P12\_x\*limit > P12\_Limit**

**if iter > 1**

**bus\_data(2,8) = u\_x;**

**P\_inj = (bus\_data(:,8) - bus\_data(:,6)) / base\_MVA;**

**V = bus\_data(:,11);**

**V(find(V(:)==0)) = 1;**

**T = zeros(n\_bus,1);**

**[V\_data,T\_data,T1] = NR(bus\_data,V,T,P\_inj,Q\_inj,n\_bus,Y,n\_pq,pq\_i);**

**V = V\_data(:,size(V\_data,2));**

**T = T\_data(:,size(T\_data,2));**

**[P,Q] = PQ\_calc(V,T,Y);**

**xVals = [T(2:n\_bus); V(pq\_i)];**

**P1 = P(1)\*base\_MVA;**

**Q1\_Val = Q(1)\*base\_MVA;**

**J = J\_calc(bus\_data,V,T,Y,n\_bus,n\_pq,pq\_i);**

**P12\_x = abs(abs(V(2)\*V(1)\*Y(1, 2))\*cos(-T(2) - angle(Y(1, 2))) - real(Y(1, 2))\*(V(1))^2);**

**P12\_x = P12\_x\*base\_MVA;**

**dfdxP12 = p\_coef\*[2\*(P12\_x-5)\*abs(V(1)\*V(2)\*Y(1,2))\*sin(T(1)-T(2)-angle(Y(1,2))); zeros(n\_bus+n\_pq-2, 1)];**

**end**

**dgdu = [1; zeros(n\_bus + n\_pq - 2, 1)];**

**dgdx = - J;**

**df\_du\_temp = jacobian(f(u\_r{:}), u);**

**dfdu = subs(df\_du\_temp, u, u\_x);**

**P\_inj\_1\_temp = Pinj\_calc(x, V, Y);**

**P1\_temp = P\_inj\_1\_temp + bus\_data(2,6)/base\_MVA;**

**Q\_inj\_1\_temp = Qinj\_calc(x, V, Y);**

**Q1\_temp = Q\_inj\_1\_temp + bus\_data(1,7)/base\_MVA;**

**df\_dP1\_temp = transpose(jacobian(f(u\_r{:}), PG\_i));**

**dfdP1 = subs(df\_dP1\_temp, PG\_i, P1);**

**dP1\_dx\_temp = jacobian(P1\_temp, x);**

**dP1dx = subs(dP1\_dx\_temp, x, xVals);**

**df\_dQ1\_temp = transpose(jacobian(f(u\_r{:}), QG\_i));**

**dfdQ1 = subs(df\_dQ1\_temp, QG\_i, Q1\_Val);**

**dQ1\_dx\_temp = jacobian(Q1\_temp, x(14:end));**

**dQ1dx = subs(dQ1\_dx\_temp, x, xVals);**

**dfdx = dfdP1\*dP1dx;**

**if limit == 1**

**if iter > 1**

**dfdx = dfdx + transpose(dfdxP12);**

**end**

**end**

**del\_C = dfdu - transpose(dgdu)\*inv(transpose(dgdx))\*transpose(dfdx);**

**corr = r\*del\_C;**

**u\_x = u\_x - corr;**

**cost = subs(f(u\_x), PG\_i, P1);**

**iter = iter + 1**

**if iter > 50**

**break;**

**end**

**end**

**P2 = double(u\_x);**

**P12\_x = abs(abs(V(2)\*V(1)\*Y(1, 2))\*cos(-T(2) - angle(Y(1, 2))) - real(Y(1, 2))\*(V(1))^2);**

**P12\_x = P12\_x\*base\_MVA**

**if limit == 1**

**cost = subs(f(P2), [PG\_i P12], [P1 P12\_x]);**

**else**

**cost = subs(f(P2), PG\_i, P1);**

**end**

**end**

**% Pinj\_calc.m**

**function P\_inj\_temp = Pinj\_calc(x, V, Y)**

**P\_inj\_temp = real(Y(1, 1))\*abs(V(1)^2);**

**listOfNeighbours = [2,5];**

**for k = listOfNeighbours**

**if k == 5**

**Vk = x(15);**

**else**

**Vk = V(k);**

**end**

**P\_inj\_temp = P\_inj\_temp + abs( Y(1, k)\*Vk\* V(1) )\*cos( angle(Y(1, k)) + x(k-1));**

**end**

**P\_inj\_temp;**

**end**