

**دانشکده مهندسی برق**

**تمرین متلب پخش توان به روش نیوتون رافسون**

**تهیه کننده و نویسنده:**

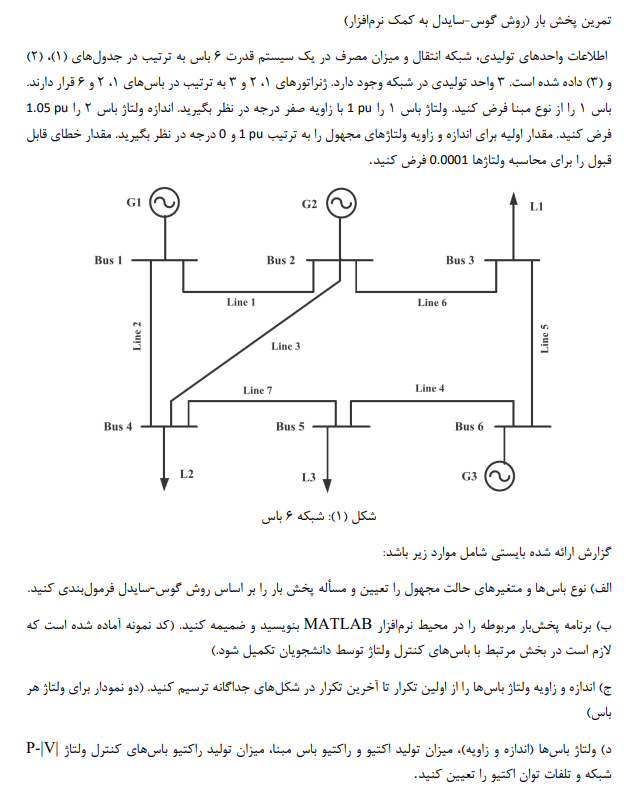
**رضا آدینه پور**

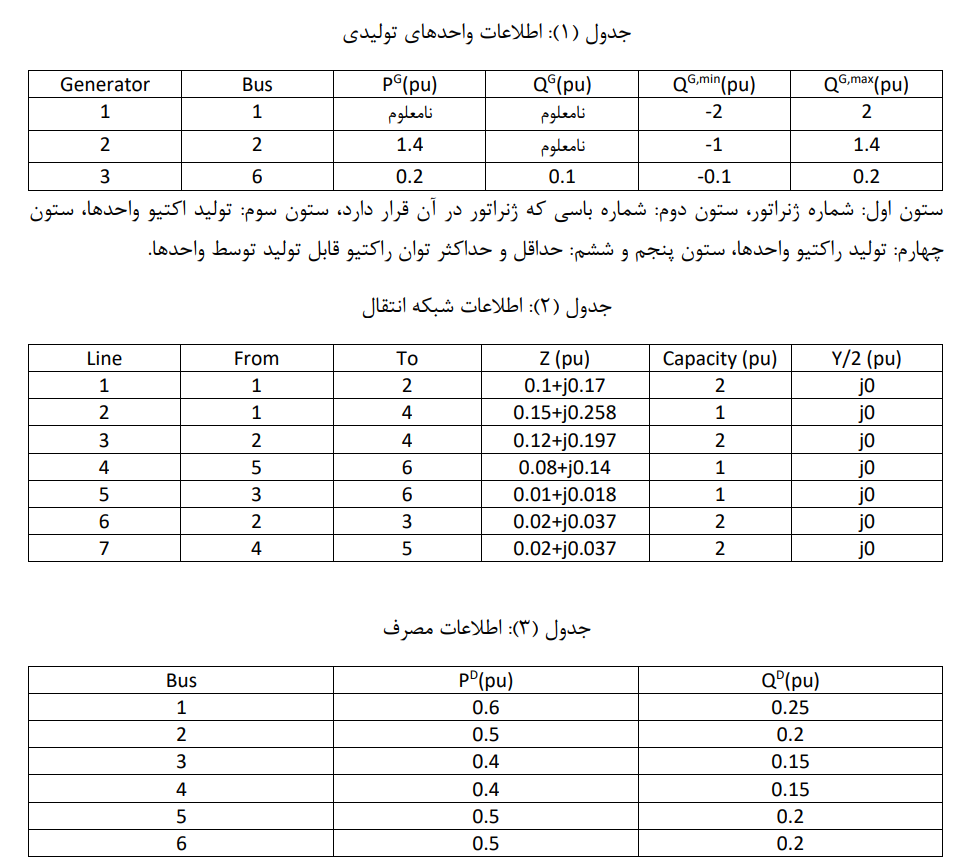
**استاد مربوطه:**

**جناب اقای دکتر رحیمیان**

**تاریخ تهیه و اراﺋﻪ:**

**دی ماه 1400**

****

****

**الف) ابتدا نوع باس ها و متغیر های مجهول را مشخص کنید؟**

**باس1:** مبنا (طبق صورت مسئله) **معلومات**: ولتاژ(اندازه و زاویه) **مجهولات**: توان اکتیو و راکتیو تزریقی

**باس2:** PV، **معلومات**: اندازه ولتاژ و توان اکتیو تزریقی **مجهولات**: زاویه ولتاژ و توان راکتیو تزریقی

**باس3 تا 6:** PQ، **معلومات**: توان اکتیو و راکتیو تزریقی **مجهولات**: ولتاژ (اندازه و زاویه)

**کد نوشته شده بر اساس کد همان توضیحی توسط اقای دکتر تکمیل شده است و فایل های آن ضمیمه فایل ارسالی شده است.**

اطلاعات باس ها و خطوط به صورت زیر در فایل data.m تکمیل شده است.

Type:

1: slack bus

2: voltage controlled bus

3: consumption bus

% input data (line and bus)

% bus types: type-1=slack, type-2=PV, type-3=PQ

function [linedata,busdata,tolerence,npv,nlb,lpv] = data()

% | From | To | R | X | Y/2 | Capacity |

% | Bus | Bus | | | | |

linedata = [ 1 2 0.10 0.170 0.0 2.0;

1 4 0.15 0.258 0.0 1.0;

2 4 0.12 0.197 0.0 2.0;

5 6 0.08 0.140 0.0 1.0;

3 6 0.01 0.018 0.0 1.0;

2 3 0.02 0.037 0.0 2.0;

4 5 0.02 0.37 0.0 2.0;];

% |Bus | Type | Vsp | theta | PGi | QGi | PLi | QLi | Qmin | Qmax | B\_shunt |

busdata = [ 1 1 1.00 0 0.0 0.0 0.6 0.25 -2.0 2.0 0.0;

2 2 1.05 0 0.0 0.0 0.5 0.20 -1.0 1.4 0.0;

3 3 1.00 0 0.0 0.0 0.4 0.15 0.0 0.0 0.0;

4 3 1.00 0 0.0 0.0 0.4 0.15 0.0 0.0 0.0;

5 3 1.00 0 0.0 0.0 0.5 0.20 0.0 0.0 0.0;

6 3 1.00 0 0.1 0.1 0.5 0.20 -0.1 0.2 0.0;];

nlb=4; % number of LOAD-buses

npv=1; % number of PV-buses

lpv=2; % lowest value of PV-bus number

tolerence=0.0001; % Defining the value of accepted tolerence

end

کد نوشته شده برای 10 تکرار به صورت زیر است:

clear;clc;close all;

MVA\_base=100.0; %Defining the Base-MVA

[linedata,busdata,tolerence,nPV,nLB,lPV] = data(); %Calling the function containing the input data

fb=linedata(:,1); % Storing the from-bus numbers

tb=linedata(:,2); % Storing the to-bus numbers

r=linedata(:,3); % Storing the resistance(s)

x=linedata(:,4); % Storing the reactance(s)

b=linedata(:,5); % Storing the half-line charging(shunt-admittance)

a=linedata(:,6); % Storing the off-nominal trans-ratio

bus\_type=(busdata(:,2)); % Storing the bus-type(s)

v\_bus\_initial=(busdata(:,3)); % Storing the initial bus-voltage(s) in (p.u.)

P\_gen=((busdata(:,5))\*(MVA\_base)); % Storing the active power generated at bus(s)

Q\_gen=((busdata(:,6))\*(MVA\_base)); % Storing the reactive power generated at bus(s)

S\_gen=((P\_gen)+(1i)\*(Q\_gen)); % Storing the complex power generated at bus(s)

P\_load=((busdata(:,7))\*(MVA\_base)); % Storing the active power at load(s) at bus(s)

Q\_load=((busdata(:,8))\*(MVA\_base)); % Storing the reactive power at load(s) at bus(s)

S\_load=((P\_load)+(1i)\*(Q\_load)); % Storing the complex power at load(s) at bus(s)

Q\_min=((busdata(:,9))\*(MVA\_base)); % Storing the minimum Q-range of the PV-bus(s)

Q\_max=((busdata(:,10))\*(MVA\_base)); % Storing the maximum Q-range of the PV-bus(s)

B\_shunt=((busdata(:,11))); % Storing the data corresponding to the static shunt capacitance

z\_elementary=((r)+(1i)\*(x)); % Calculating the 'z'

b\_elementary=((1i)\*(b)); % Calculating the 'b'

nbus = max(max(fb),max(tb)); % no. of buses

nbranch = length(fb); % no. of branches

z=zeros(nbus,nbus); % Initialising the elementary [z]

b=zeros(nbus,nbus); % Initialising the elementary [b]

Ybus=zeros(nbus,nbus); % Initialising the elementary [Y] (Bus-admittance matrix)

tolerence\_checker=1; % Initialising the 'tolerence\_checker' as '1'

iteration=0; % Initialising the 'iteration' as '0'

V\_new=v\_bus\_initial; % Initialising the 'V\_new[]' as 'V\_initial[]'

v\_scheduled=v\_bus\_initial; % Initialising the 'v\_scheduled[]' as 'V\_initial[]' (to get the scheduled voltages at the PV-bus(s))

V\_new\_accelerated=zeros(1,nbus); % Initialising the 'V\_new\_accelerated[]'

V\_new\_del=zeros(1,nbus); % Initialising the 'v\_new\_del[]' (to get the difference(s) of bus-voltage(s) of two(02) consecutive iteration(s))

difference=zeros(1,nbus); % Iniialising the 'difference[]'

real\_diff=zeros(1,nbus); % Initialising the 'real\_diff[]' (to store the real part of the difference)

imag\_diff=zeros(1,nbus); % Initialising the 'imag\_diff[]' (to store the imaginary part of the difference)

delta=zeros(1,nbus); % Initialising the 'delta[]' (for getting the angle at PV-bus(s))

E\_new=zeros(1,nbus); % Initialising the 'E\_new[]' (to store the real part of the voltage at PV-bus)

F\_new=zeros(1,nbus); % Initialising the 'F\_new[]' (to store the imaginary part of the voltage at PV-bus)

Q\_intermediate=zeros(1,nbus); % Initialising the 'Q\_intermediate'

Q\_final=zeros(1,nbus); % Initialising 'Q\_final[]'

complex\_flow\_line=zeros(1,nbranch); % Initialising 'complex\_flow\_line[]'

line\_flows=zeros(nbus,nbus); % Initialising 'line\_flows[]'

active\_flow\_line=zeros(1,nbranch); % Initialising 'active\_flow\_line[]'

reactive\_flow\_line=zeros(1,nbranch); % Initialising 'reactive\_flow\_line[]'

bus\_power\_injection=zeros(1,nbus); % Initialising 'bus\_power\_injection[]'

bus\_power\_mismatch=zeros(1,nbus); % Initialising 'bus\_power\_mismatch[]'

line\_loss=zeros(nbus,nbus); % Initialising 'line\_loss[]'

flow\_count=1; % Initialising the 'flow\_count' as (1) //(for tracking the total no. of lines)

sum\_line\_loss=((0.0)+(1i)\*(0.0)); % Initialising 'sum\_line\_loss' as '0'

sum=zeros(1,nbus); % Initialising the 'sum' (used in calculating the digaonal [Ybus] elements)

shunt\_fb\_onr=zeros(1,nbranch); % storing the ((a^2)/(1-a)) as seen from the FB(From Bus)

shunt\_tb\_onr=zeros(1,nbranch); % storing the ((a)/(a-1)) as seen from the TB(To Bus)

P\_injected\_sum=zeros(nbus,1); % Initialising 'P\_injected\_sum[]' for intermediate calculation(s) of injected (P)

Q\_injected\_sum=zeros(nbus,1); % Initialising 'Q\_injected\_sum[]' for intermediate calculation(s) of injected (Q)

P\_injected\_bus=zeros(nbus,1); % for storing the intermediate 'active power injection(s)' at different bus(s)

Q\_injected\_bus=zeros(nbus,1); % for storing the intermediate 'reactive power injection(s)' at different bus(s)

partial\_P\_delta=zeros(nbus-1,nbus-1); % Initialsing 'partial\_P\_delta[]'corresponding to the [J11]

partial\_Q\_delta=zeros(nbus-nPV-1,nbus-1); % Initialsing 'partial\_Q\_delta[]'corresponding to the [J21]

partial\_P\_vol\_mag=zeros(nbus-1,nbus-nPV-1); % Initialising 'partial\_P\_vol\_mag[]'corresponding to the [J12]

partial\_Q\_vol\_mag=zeros(nbus-nPV-1,nbus-nPV-1); % Initialising 'partial\_Q\_vol\_mag[]'corresponding to the [J12]

J=zeros(((2\*(nbus-1))-nPV),((2\*(nbus-1))-nPV)); % Initialising 'J[]' i.e. THE COMPLETE JACOBIAN MATRIX

inj\_active\_pow\_mismatch\_vector=zeros(nbus-1,1); % Initialising the 'inj\_active\_pow\_mismatch\_vector[]'

inj\_reactive\_pow\_mismatch\_vector=zeros(nbus-nPV-1,1); % Initialising the 'inj\_reactive\_pow\_mismatch\_vector[]'

inj\_pow\_mismatch\_vector=zeros(((2\*(nbus-1))-nPV),1); % Initialising the 'inj\_pow\_mismatch\_vector[]'

correction\_vector=zeros(((2\*(nbus-1))-nPV),1); % Initialising the 'correction\_vector[]'

correction\_voltage\_angle=zeros((nbus-1),1); % Initialising the 'correction\_voltage\_angle[]'

correction\_voltage\_magnitude=zeros((nbus-1-nPV),1); % Initialising the 'correction\_voltage\_magnitude[]'

mismatch\_active\_vector\_element\_count=1; % Initialising the 'mismatch\_active\_vector\_element\_count' as (1)

mismatch\_reactive\_vector\_element\_count=1; % Initialising the 'mismatch\_reactive\_vector\_element\_count' as (1)

mismatch\_vector\_element\_count=0;% Initialising the 'mismatch\_active\_vector\_element\_count' as (0)

real\_mismatch=zeros(nbus-1,1);

imag\_mismatch=zeros(nbus-1-nPV,1);

for u=1:((2\*(nbus-1))-nPV)

correction\_vector(u)=1.0; % Initialising the correction vector element(s) as (1.0); so that loop can be started

end

% Forming the elementary z-matrix & b-matrix

for u=1:nbranch

z(fb(u),tb(u))=z\_elementary(u); % Storing the mutual element(s) (Z(i,j))

z(tb(u),fb(u))=z(fb(u),tb(u)); % Storing the similar element(s) in (Z(j,i))

b(fb(u),tb(u))=b\_elementary(u); % Storing the mutual element(s) (B(i,j))

b(tb(u),fb(u))=b(fb(u),tb(u)); % Storing the similar element(s) in (B(j,i))

end

% Modification(s) in the element(s) of 'z' & 'b' due to OFF-NOMINAL TRANS RATIO

for u=1:nbranch

if((a(u,1))~=1.0)

shunt\_fb\_onr(1,u)=(((a(u,1))^(2))/(1-(a(u,1)))); % Calculating the (a^2/(1-a))

b(fb(u),tb(u))=(1/((shunt\_fb\_onr(1,u))\*(z(fb(u),tb(u)))))+(b(fb(u),tb(u))); % Modifying the element 'b' matrix w.r.t. 'fb'

shunt\_tb\_onr(1,u)=((a(u,1))/((a(u,1))-1)); % Calculating the (a/(a-1))

b(tb(u),fb(u))=(1/((shunt\_tb\_onr(1,u))\*(z(fb(u),tb(u))))); % Modifying the element 'b' matrix w.r.t. 'tb'

z(fb(u),tb(u))=z(fb(u),tb(u))\*(a(u,1)); % Modifying the 'z' matrix

z(tb(u),fb(u))=z(fb(u),tb(u)); % Storing the similar element(s) in (Z(j,i))

end

end

% Including the effect(s) of static shunt-capacitance

for u=1:nbus

if(B\_shunt(u)~=0.0)

sum(1,u)=B\_shunt(u); % Updating the 'sum(1,u)' with the data for static shunt capacitance

end

end

%% -------------------------- STEP-1 - CALCULATION OF BUS-ADMITTANCE MATRIX --------------------------

for u=1:nbus

for j=1:nbus

if(z(u,j)==0.0)

Ybus(u,j)=0.0; % No connection between (u) & (j)

else

Ybus(u,j)=-(1/z(u,j)); % Calculating the mutual-admittances

end

end

end

for u=1:nbus

for j=1:nbus

sum(1,u)=sum(1,u)+b(u,j)-Ybus(u,j);

if(j==nbus)

Ybus(u,u)=sum(1,u); % Calculating the self-admittances

end

end

end

fprintf('THE BUS-ADMITTANCE IS GIVEN BELOW\n'); %DISPLAYING THE RESULTS GOT FROM STEP-1

disp(Ybus);

Gbus=real(Ybus); % Generating the conducatance matrix[G] by taking the real part of the [Ybus] element(s); (y=g+jb)

Bbus=imag((+1)\*(Ybus)); % Generating the susceptance matrix[B] by taking the imaginary part of the [Ybus] element(s); (y=g+jb)

% ----------------------------------------------END OF STEP-1-------------------------------------------

%%

while((tolerence\_checker>0)&&(iteration<10))

mismatch\_vector\_element\_count=0;% Resetting the 'mismatch\_active\_vector\_element\_count' as (0)

mismatch\_active\_vector\_element\_count=1; % Resetting the 'mismatch\_active\_vector\_element\_count' as (1)

mismatch\_reactive\_vector\_element\_count=1; % Resetting the 'mismatch\_reactive\_vector\_element\_count' as (1)

P\_injected\_sum=zeros(nbus,1); % Resetting 'P\_injected\_sum[]' for intermediate calculation(s) of injected (P)

Q\_injected\_sum=zeros(nbus,1); % Resetting 'Q\_injected\_sum[]' for intermediate calculation(s) of injected (Q)

tolerence\_checker=0; % resetting the 'tolerence\_checker' to (0)

%% --------------------------------------------STARTING OF STEP-2----------------------------------------

% CALCULATING THE ACTIVE POWER INJECTION(s) AND REACTIVE POWER INJECTION(s) AT DIFFEENT BUS(s) AND

% THE MISMATCH VECTOR FOR THE ACTIVE POWER INJECTION(s) & REACTIVE POWER INJECTION(s)

% (ACTIVE POWER INJECTION MISMATCH=(SCHEDULED ACTIVE POWER INJECTION-CALCULATED ACTIVE POWER INJECTION))

% (REACTIVE POWER INJECTION MISMATCH=(SCHEDULED REACTIVE POWER INJECTION-CALCULATED REACTIVE POWER INJECTION))

%--------------------------------------------------------------------------------------------------------

P\_injected\_scheduled=((P\_gen-P\_load)/(MVA\_base)); % calculating the scheduled injected active power at each bus

Q\_injected\_scheduled=((Q\_gen-Q\_load)/(MVA\_base)); % calculating the scheduled injected reactive power at each bus

for u=1:nbus

for j=1:nbus

if((u~=j)&&(abs(Ybus(u,j)))~=0.0)

P\_injected\_sum(u,1)=((P\_injected\_sum(u,1))+(((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1)))))))); % calculating the mutual sumation part

end

if((u~=j)&&((abs(Ybus(u,j)))~=0.0)&&(bus\_type(u)~=2))

Q\_injected\_sum(u,1)=((Q\_injected\_sum(u,1))+(((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(sin((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1)))))))); % calculating the mutual sumation part

end

if(j==nbus)

P\_injected\_bus(u,1)=((((abs(V\_new(u,1)))^(2))\*(Gbus(u,u)))+(P\_injected\_sum(u,1))); % calculating the injecetd 'P' at bus-(u)

end

if((j==nbus)&&(bus\_type(u)~=2))

Q\_injected\_bus(u,1)=(-((((abs(V\_new(u,1)))^(2))\*(Bbus(u,u)))+(Q\_injected\_sum(u,1)))); % calculating the injecetd 'Q' at bus-(u)

end

end

end

for u=1:nbus

if(bus\_type(u)~=1)

inj\_active\_pow\_mismatch\_vector(mismatch\_active\_vector\_element\_count,1)=((P\_injected\_scheduled(u,1))-(P\_injected\_bus(u,1))); % calculating the mismatch vector for active power injecttion

mismatch\_active\_vector\_element\_count=mismatch\_active\_vector\_element\_count+1; % updating the value of 'mismatch\_vector\_element\_count'

end

if((bus\_type(u)~=1)&&(bus\_type(u)~=2))

inj\_reactive\_pow\_mismatch\_vector(mismatch\_reactive\_vector\_element\_count,1)=((Q\_injected\_scheduled(u,1))-(Q\_injected\_bus(u,1))); % calculating the mismatch vector for reactive power injecttion

mismatch\_reactive\_vector\_element\_count=mismatch\_reactive\_vector\_element\_count+1; % updating the value of 'mismatch\_vector\_element\_count'

end

end

for u=2:nbus

if(mismatch\_vector\_element\_count<=((mismatch\_active\_vector\_element\_count-1)+(mismatch\_reactive\_vector\_element\_count-1)))

inj\_pow\_mismatch\_vector((u-1),1)=inj\_active\_pow\_mismatch\_vector(u-1,1); % storing the mismatch in the active power

mismatch\_vector\_element\_count=mismatch\_vector\_element\_count+1; % updating the 'mismatch\_vector\_element\_count'

end

if((mismatch\_vector\_element\_count<((mismatch\_active\_vector\_element\_count-1)+(mismatch\_reactive\_vector\_element\_count-1)))&&((u-1)<=(nbus-1-nPV)))

inj\_pow\_mismatch\_vector((u+nbus-2),1)=inj\_reactive\_pow\_mismatch\_vector(u-1,1); % storing the mismatch in the reactive power

mismatch\_vector\_element\_count=mismatch\_vector\_element\_count+1; % updating the 'mismatch\_vector\_element\_count'

end

end

fprintf('\nThe mismatch-vector is :\n');

disp(inj\_pow\_mismatch\_vector); % displaying the 'inj\_pow\_mismatch\_vector[]'

for u=1:nbus-1

real\_mismatch(u,1)=inj\_pow\_mismatch\_vector(u,1); % storing the real-power mismatch(s)

if((u+nbus-1)<=((2\*(nbus-1))-nPV))

imag\_mismatch(u,1)=inj\_pow\_mismatch\_vector(u+nbus-1,1); % storing the imaginary-power mismatch(s)

end

end

for u=1:((2\*(nbus-1))-nPV)

if((abs(inj\_pow\_mismatch\_vector(u,1)))>(tolerence))

tolerence\_checker=tolerence\_checker+1;

end

end

%----------------------------------------------END OF STEP-2-------------------------------------------

%% --------------------------------------------STARTING OF STEP-3----------------------------------------

% STEP-3: CALCULATES THE [J] OR THE JACOBIAN MATRIX ([J]=[[J11] [J12] : [J21] [J22]])

% UNDER THIS STEP SUB-STEP HAS BEEN PRFORMED FOR CALCULATING THE [J11], [J12], [J21], [J22] IN INDIVIDUAL

%--------------------------------------------------------------------------------------------------------

% STEP-(3.a): CALCULATING THE [J11]

%--------------------------------------------------------------------------------------------------------

sum\_pv=0; % initialising 'sum\_pv' for getting the [J] elements dependent on PV-bus (Q)

for u=1:nbus-1

for j=1:nbus-1

if((u+1)~=(j+1))

partial\_P\_delta(u,j)=(-((abs((V\_new(u+1,1))\*(V\_new(j+1,1))\*(Ybus(u+1,j+1))))\*(sin((angle(Ybus(u+1,j+1)))+(angle(V\_new(j+1,1)))-(angle(V\_new(u+1,1))))))); % calculating the off-diagonal element(s)

else

if((bus\_type(u+1))~=2)

partial\_P\_delta(u,j)=(-((((abs(V\_new(u+1,1)))^(2))\*(Bbus(u+1,u+1)))+(Q\_injected\_bus(u+1,1)))); % calculating the diagonal element(s)

else

for k=1:nbus

if(((u+1)~=(k))&&(abs(Ybus(u+1,k))~=0.0))

%sum\_pv=sum\_pv-partial\_P\_delta(u-1,k-1);

sum\_pv=sum\_pv+((abs((V\_new(u+1,1))\*(V\_new(k,1))\*(Ybus(u+1,k)))\*(sin((angle(Ybus(u+1,k))+(angle(V\_new(k,1)))-(angle(V\_new(u+1,1))))))));

end

end

partial\_P\_delta(u,j)=sum\_pv; % calculating the (partial\_P\_delta(3,3)) for the PV-bus

sum\_pv=0; % resetting the 'sum\_pv' to (0)

end

end

end

end

fprintf('\n The [J11] is : \n');

disp(partial\_P\_delta); % diplaying the [J11]

%--------------------------------------------------------------------------------------------------------

% STEP-(3.b): CALCULATING THE [J21]

%--------------------------------------------------------------------------------------------------------

m=1;n=1;element\_count=0;

for u=2:nbus

for j=2:nbus

if((m~=n)&&((bus\_type(u))~=2)&&(u~=j))

partial\_Q\_delta(m,n)=(-((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))))); % calculating the off-diagonal element(s)

n=n+1;

element\_count=element\_count+1;

elseif((m~=n)&&((bus\_type(u))~=2)&&(u==j))

partial\_Q\_delta(m,n)=((P\_injected\_bus(u,1))-(((abs(V\_new(u,1)))^(2))\*(Gbus(u,u))));

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&((bus\_type(u))~=2)&&(u~=j))

partial\_Q\_delta(m,n)=(-((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))))); % calculating the off-diagonal element(s)

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&((bus\_type(u))~=2)&&(u==j))

partial\_Q\_delta(m,n)=((P\_injected\_bus(u,1))-(((abs(V\_new(u,1)))^(2))\*(Gbus(u,u))));

n=n+1;

element\_count=element\_count+1;

end

end

if((j==nbus)&&(element\_count>0))

m=m+1;n=1;element\_count=0;

end

end

fprintf('\n The [J21] is : \n');

disp(partial\_Q\_delta); % diplaying the [J21]

%--------------------------------------------------------------------------------------------------------

% STEP-(3.c): CALCULATING THE [J12]

%--------------------------------------------------------------------------------------------------------

m=1;n=1;element\_count=0;

for u=2:nbus

for j=2:nbus

if((m==n)&&(bus\_type(j)~=2)&&(u~=j)&&(bus\_type(u)~=2))

%partial\_P\_vol\_mag(m,n)=(-(partial\_Q\_delta(u-nPV-1,j-1))); % calculating the off-diagonal element(s)

partial\_P\_vol\_mag(m,n)=((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))));

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&(bus\_type(j)~=2)&&(u==j)&&(bus\_type(u)~=2))

partial\_P\_vol\_mag(m,n)=((P\_injected\_bus(u,1))+(((abs(V\_new(u,1)))^(2))\*(Gbus(u,u))));

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&(bus\_type(j)~=2)&&(u~=j)&&(bus\_type(u)==2))

partial\_P\_vol\_mag(m,n)=((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))));

n=n+1;

element\_count=element\_count+1;

elseif((m~=n)&&(bus\_type(j)~=2)&&(u~=j)&&(bus\_type(u)~=2))

partial\_P\_vol\_mag(m,n)=((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))));

n=n+1;

element\_count=element\_count+1;

elseif((m~=n)&&(bus\_type(j)~=2)&&(u==j)&&(bus\_type(u)~=2))

partial\_P\_vol\_mag(m,n)=((P\_injected\_bus(u,1))+(((abs(V\_new(u,1)))^(2))\*(Gbus(u,u))));

n=n+1;

element\_count=element\_count+1;

elseif((m~=n)&&(bus\_type(j)~=2)&&(u~=j)&&(bus\_type(u)==2))

partial\_P\_vol\_mag(m,n)=((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(cos((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1))))));

n=n+1;

element\_count=element\_count+1;

end

end

if((j==nbus)&&(element\_count>0))

m=m+1;n=1;element\_count=0;

end

end

fprintf('\n The [J12] is : \n');

disp(partial\_P\_vol\_mag); % diplaying the [J12]

%--------------------------------------------------------------------------------------------------------

% STEP-(3.d): CALCULATING THE [J22]

%--------------------------------------------------------------------------------------------------------

m=1;n=1;element\_count=0;

for u=2:nbus

for j=2:nbus

if((m~=n)&&(u~=j)&&((bus\_type(u))~=2)&&((bus\_type(j))~=2))

partial\_Q\_vol\_mag(m,n)=(-((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(sin((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1)))))));

n=n+1;

element\_count=element\_count+1;

elseif((m~=n)&&(u==j)&&((bus\_type(u))~=2)&&((bus\_type(j))~=2))

partial\_Q\_vol\_mag(m,n)=((Q\_injected\_bus(u,1))-(((abs(V\_new(u,1)))^(2))\*(Bbus(u,u))));

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&(u~=j)&&((bus\_type(u))~=2)&&((bus\_type(j))~=2))

%partial\_Q\_vol\_mag(m,n)=partial\_P\_delta(u-nPV-1,j-nPV-1);

partial\_Q\_vol\_mag(m,n)=(-((abs((V\_new(u,1))\*(V\_new(j,1))\*(Ybus(u,j))))\*(sin((angle(Ybus(u,j)))+(angle(V\_new(j,1)))-(angle(V\_new(u,1)))))));

n=n+1;

element\_count=element\_count+1;

elseif((m==n)&&(u==j)&&((bus\_type(u))~=2)&&((bus\_type(j))~=2))

partial\_Q\_vol\_mag(m,n)=((Q\_injected\_bus(u,1))-(((abs(V\_new(u,1)))^(2))\*(Bbus(u,u))));

n=n+1;

element\_count=element\_count+1;

end

end

if((j==nbus)&&(element\_count>0))

m=m+1;n=1;element\_count=0;

end

end

fprintf('\n The [J22] is : \n');

disp(partial\_Q\_vol\_mag); % diplaying the [J22]

%--------------------------------------------------------------------------------------------------------

% STEP-(3.e): CALCULATING THE COMPLETE [J]

%--------------------------------------------------------------------------------------------------------

J=[partial\_P\_delta partial\_P\_vol\_mag ; partial\_Q\_delta partial\_Q\_vol\_mag];

fprintf('\nThe complete JACOBIAN matrix is : \n');

disp(J);

%----------------------------------------------END OF STEP-3-------------------------------------------

%% --------------------------------------------STARTING OF STEP-4----------------------------------------

% STEP-4: CALCULATING THE CORRECTION VECTOR BY ((inv(J))\*(MISMATCH-VECTOR)) & THE UPDATED VALUES OF THE

% STATE VARIABLES i.e. (|v\_i|,DELTA\_i)

%--------------------------------------------------------------------------------------------------------

for u=1:(2\*(nbus-1))

correction\_vector(u,1)=0.0; % Initialising the 'correction\_vector' as (0.0)

end

correction\_vector=(inv(J))\*(inj\_pow\_mismatch\_vector); % calculating the 'correction\_vector[]'

fprintf('\n Inverse of the [J] matrix : \n');

disp(inv(J)); % displaying the inverse of the [J] matrix

for u=1:(nbus-1)

correction\_voltage\_angle(u,1)=correction\_vector(u,1); % storing the correction(s) in the voltage angle(s)

if(u<=(nbus-1-nPV))

correction\_voltage\_magnitude(u,1)=correction\_vector((u+nbus-1),1); % storing the correction(s) in the voltage angle(s)

end

end

correction\_voltage\_magnitude\_count=1; % initialising the 'correction\_voltage\_magnitude\_count' as (1)

for u=1:(nbus-1)

voltage\_angle\_updated=angle((V\_new(u+1,1)))+(correction\_voltage\_angle(u,1)); % calculating the updated voltage-angle

if(bus\_type(u+1)~=2)

voltage\_magnitude\_updated=abs((V\_new(u+1,1)))+((correction\_voltage\_magnitude(correction\_voltage\_magnitude\_count,1))\*(abs((V\_new(u+1,1))))); % calculating the updated voltage-magnitude

correction\_voltage\_magnitude\_count=correction\_voltage\_magnitude\_count+1; % updating 'correction\_voltage\_magnitude\_count' by (1)

end

if(bus\_type(u+1)~=2)

V\_new(u+1,1)=voltage\_magnitude\_updated\*((cos(voltage\_angle\_updated))+(sin(voltage\_angle\_updated)\*(1i))); % calculating the updated voltage

end

if(bus\_type(u+1)==2)

V\_new(u+1,1)=abs(V\_new(u+1,1))\*((cos(voltage\_angle\_updated))+(sin(voltage\_angle\_updated)\*(1i))); % calculating the updated voltage

end

end

%--------------------------------------------------------------------------------------------------------

iteration=iteration+1; % updating the value of iteration-count by (1)

fprintf('\n The updated state variables after the (%d)-th iteration is :\n',iteration);

fprintf(' BUS RECTANGULAR-FORM POLAR-FORM\n');

fprintf('------------------------------------------------------------------------\n');

for u=1:nbus

fprintf(' %d (%f)+j(%f) (%f)at an angle(%f)deg.\n',u,real(V\_new(u,:)),imag(V\_new(u,:)),abs(V\_new(u,:)),((180/pi)\*(angle(V\_new(u,:)))));

% figure(u);

% subplot(2,1,1);

% plot(abs(V\_new(u,:)), '-o');

% xlabel('Iteration');

% ylabel('Voltage amplitude (p.u.)');

%

% subplot(2,1,2);

% plot(((180/pi)\*(angle(V\_new(u,:)))), '-o r');

% xlabel('Iteration');

% ylabel('Voltage angle (degree)');

end

end

%% STEP-4 - CALCULATING THE LINE-FLOW(s) & THE BUS-INJECTION(s) & THE BUS-POWER MISMATCH(s) & THE LINE-LOSS(s)

%------------------------------------------------------------------------------------------------------------

fprintf('The Line-flows are as follows : \n');

fprintf('Line-code Active-flow(p.u.) Reactive-flow(p.u.)\n');

fprintf('------------------------------------------------------------\n');

for u=1:nbus

for j=1:nbus

if((u~=j)&&(Ybus(u,j))~=0.0)

complex\_flow\_line(1,flow\_count)=((conj(V\_new(u,1))\*((V\_new(u,1))-(V\_new(j,1))))\*(-Ybus(u,j)))+(conj(V\_new(u,1))\*(V\_new(u,1))\*(b(u,j))); % Calculating the line-flows

line\_flows(u,j)=conj(complex\_flow\_line(1,flow\_count)); % Storing the power flow in the line-flow matrix

active\_flow\_line(1,flow\_count)=real(complex\_flow\_line(1,flow\_count)); % P.U. active power flow from (u->j)

reactive\_flow\_line(1,flow\_count)=(-(imag(complex\_flow\_line(1,flow\_count)))); % P.U. reactive power flow from (u->j)

fprintf(' %d->%d',u,j); % Printing the line-code

fprintf(' %f %f\n',active\_flow\_line(1,flow\_count),reactive\_flow\_line(1,flow\_count)); % Printing the active & reactive power flows

bus\_power\_injection(1,u)=bus\_power\_injection(1,u)+conj(complex\_flow\_line(1,flow\_count)); % Calculating the bus-power-injection(s)

bus\_power\_mismatch(1,u)=((S\_gen(u,1)-S\_load(u,1))/MVA\_base)-(bus\_power\_injection(1,u)); % Calculating the bus-power-mismach

flow\_count=flow\_count+1; % Updating the flow\_count by (1)

end

end

end

fprintf('The bus-power injection(s) are :\n');

fprintf('Bus-code Power-injection(p.u.)\n');

fprintf('----------------------------------------\n');

for u=1:nbus

fprintf(' %d (%f)+j(%f)\n',u,real(bus\_power\_injection(u)),imag(bus\_power\_injection(u))); % Displaying the bus-power injections

end

fprintf('The bus-power mismatch(s) are :\n');

fprintf('Bus-code Bus-power-mismatch(p.u.)\n');

fprintf('----------------------------------------\n');

for u=2:nbus

fprintf(' %d (%f)+j(%f)\n',u,real(bus\_power\_mismatch(u)),imag(bus\_power\_mismatch(u))); % Displaying the bus-power injections

end

for u=1:nbus

for j=1:nbus

if((u<j)&&(abs(line\_flows(u,j))~=0.0))

line\_loss(u,j)=line\_flows(u,j)+line\_flows(j,u); % Calculating the individual line-loss

real\_line\_loss=abs(real(line\_loss(u,j))); % Absolute value of the active-line-loss

imag\_line\_loss=abs(imag(line\_loss(u,j))); % Absolute value of the reactive-line-loss

line\_loss(u,j)=(real\_line\_loss)+(imag\_line\_loss)\*(1i);

sum\_line\_loss=sum\_line\_loss+line\_loss(u,j); % Calculating the total line-loss

end

end

end

fprintf('Individual line-loss is as follows = \n');

fprintf('Line-code Line-loss \n');

fprintf('-----------------------------------\n');

for u=1:nbus

for j=1:nbus

if((u<j)&&(abs(line\_flows(u,j))~=0.0))

fprintf(' %d->%d (%f)+j(%f)\n',u,j,real(line\_loss(u,j)),imag(line\_loss(u,j)));

end

end

end

fprintf('Summation of all the losses = (%f)+j(%f)\n',real(sum\_line\_loss),imag(sum\_line\_loss)); % Displaying the summation of all the line-losses

خروجی کد به صورت زیر است:

