```
function
 [V,Q,s,d,w,eq1,eq2,ed1,ed2,psid,psiq,Pm,Ef,Vavrm,Vavrr,Vavrf,Vavrref,tgovg,tgovm,
    hemMachinePFSalientcontinueDyn(SimData,SysData,SysPara,x0)

% Core HE algorithm for solving DAEs (dynamic simulation)
%
% FUNCTION hemMachinePFSalientcontinueDyn
%
% Author: Rui Yao <ruiyao@ieee.org>
%
% Copyright (C) 2021, UChicago Argonne, LLC. All rights reserved.
%
% OPEN SOURCE LICENSE
%
% Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:
%
% 1. Redistributions of source code must retain the above copyright
 notice, this list of conditions and the following disclaimer.
% 2. Redistributions in binary form must reproduce the above copyright
 notice, this list of conditions and the following disclaimer in the
 documentation and/or other materials provided with the distribution.
% 3. Neither the name of the copyright holder nor the names of its
 contributors may be used to endorse or promote products derived from
 this software without specific prior written permission.
%
%
%
 ********************************************************************************
% DISCLAIMER
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 "AS IS" AND ANY EXPRESS OR IMPLIED
% WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A
% PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
% DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 DAMAGES (INCLUDING, BUT NOT LIMITED TO,
% PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
% CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
% OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
 IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%
 ********************************************************************************
%
% INPUT
%   SimData - Simulation parameters
%   SysData - System data for simulation
```

```
%   SysPara - Parameters representing the events happening in the
 system
%   x0 - Initial system state
%
% OUTPUT - (will be consolidated in a future version)
%
% TODO % Modify the output arguments
%

global IS_OCTAVE;



% import system data
[bus,sw,pv,pq,shunt,line,ind,zip,syn,exc,tg,agc,cac,cluster]=unfoldSysData(SysData
% nbus:total number of buses
nbus=size(bus,1);
nline=size(line,1);

%determine islanding
if
 isfield(SysPara,'nIslands')&&isfield(SysPara,'islands')&&isfield(SysPara,'refs')

 nIslands=SysPara.nIslands;islands=SysPara.islands;refs=SysPara.refs;
else
    [nIslands,islands,refs]=searchIslands(bus(:,1),line(:,1:2));
end

% improt initial condition
[V0,Q0,s0,d0,w0,eq10,eq20,ed10,ed20,psid0,psiq0,Pm0,Ef0,Vavrm0,Vavrr0,Vavrf0,Vavrr
% import simualtion data
[~,~,~,nlvl,taylorN,~,~,~,~]=unfoldSimData(SimData);
% import system parameters
[pqIncr,pvIncr,Rind0,Rind1,Reind0,Reind1,Rzip0,Rzip1,Ytr0,Ytr1,Ysh0,Ysh1,VspSq2,~,
%

% It seems that PQ incremental is given for every pq bus.
% need to figure how to utilize them
Pls=zeros(nbus,2);Pls(pq(:,1),1)=pqIncr(:,1);if
 ~isempty(pv);Pls(pv(:,1),1)=Pls(pv(:,1),1)-pvIncr;end
Qls=zeros(nbus,2);Qls(pq(:,1),1)=pqIncr(:,2);
% why additional PQ is given
if size(pqIncr,2)>=4
    Pls(pq(:,1),2)=pqIncr(:,3);
    Qls(pq(:,1),2)=pqIncr(:,4);
end

% formatting Ymatrix, here the default value of fault is empty
if isempty(Ytr0)
    [Y,Ytr0,Ysh,ytrfr,ytrto,yshfr,yshto]=getYMatrix(nbus,line);
end


% reshape the pv, pq shunt and swing buses if they are empty
```

```
busType=zeros(nbus,1);
if isempty(pv)
    pv=zeros(0,6);
end
if isempty(pq)
    pq=zeros(0,6);
end
if isempty(shunt)
    shunt=zeros(0,7);
end
if isempty(sw)
    sw=zeros(0,13);
end

% label pv and swing buses
% 1: PV bus, 0: PQ bus
busType(pv(:,1))=1;
busType(sw(:,1))=2;
% zip(busType(zip(:,1))~=0,10)=0;

% index of swing bus (isw), pv bus(ipv), and pq bus(ipq)
% is given: isw, ipv, and ipq
% Additionally, number of pv and pq buses are given:
%npv, npq respectively
isw=find(busType==2);
ipv=find(busType==1);
ipq=find(busType==0);
npq=size(ipq,1);
npv=size(ipv,1);

% shunt capacitator is initialized as yShunt which is a complex
 number.
% for every bus
yShunt=zeros(nbus,1);
yShunt(shunt(:,1))=shunt(:,5)+1j*shunt(:,6);

% check if zip load exists in the system
% and initialized zip load
if ~isempty(zip)%zipMode=0

 Ysh0=Ysh0+accumarray(zip(:,1),Rzip0.*(zip(:,5)+1j*zip(:,8)).*zip(:,12),
[nbus,1]);

 Ysh1=Ysh1+accumarray(zip(:,1),Rzip1.*(zip(:,5)+1j*zip(:,8)).*zip(:,12),
[nbus,1]);
end

% now zip load + shunt
Ysh0=Ysh0+yShunt;
%     Y=Y+sparse(1:nbus,1:nbus,yShunt,nbus,nbus);

% now zip load + shunt+ network Y matrix
Y=Ytr0+sparse(1:nbus,1:nbus,Ysh0,nbus,nbus);
```

```
%initialize P and Q for every bus
pVec=zeros(nbus,1);
qVec=zeros(nbus,1);
%     vSp=zeros(nbus,1);

% need to figure out the meaning of index 1, 4,5
% based Kaiyang's understanding, 1 is load side and 4&5 are
 generators'
% output
pVec(pv(:,1))=pVec(pv(:,1))+pv(:,4);
pVec(pq(:,1))=pVec(pq(:,1))-pq(:,4);
qVec(pq(:,1))=qVec(pq(:,1))-pq(:,5);
% account the zip load, i.e dynamic load
if ~isempty(zip)%zipMode=0, account for the PQ components in ZIP loads
    pVec=pVec-accumarray(zip(:,1),Rzip0.*zip(:,7).*zip(:,12),
[nbus,1]);
    qVec=qVec-accumarray(zip(:,1),Rzip0.*zip(:,10).*zip(:,12),
[nbus,1]);
end
% qVec(ipv)=qVec(ipv)+Q0(ipv);
%     vSp(ipv)=pv(:,5);

% so far, initialization of PQ for every bus and Y matrix is ready


% initiliza voltage V and W = 1/V for every bus
V=zeros(nbus,nlvl+1);
V(:,1)=V0;
W=zeros(nbus,nlvl+1);
W(:,1)=1./V0;


% initiliza magnitude of voltage V for every bus
Vmag=zeros(nbus,nlvl+1);
Vmag(:,1)=abs(V0);

% Power is initilized as we already cooked pVec and qVec
P=zeros(nbus,nlvl+1);
P(:,1)=pVec;
%     P(isw,2:end)=0;
% here we need to figure out what Q extra mean, and difference from Q
% notice that Q0 is initialized with sysmdata but not P0
Q=zeros(nbus,nlvl+1);
Qxtra=zeros(size(Q));
Q(:,1)=Q0;
Qxtra(:,1)=qVec;

% Also, the meaning of Pls and Qls need to be verified
P(:,2:(size(Pls,2)+1))=-Pls;
Qxtra(:,2:(size(Qls,2)+1))=-Qls;


% In the previous, pVec and qvec are considered zip load, here Pls and
 Qls
```

```
% are not, so we need to do it.
if ~isempty(zip)
    P(:,2)=P(:,2)-accumarray(zip(:,1),Rzip1.*zip(:,7).*zip(:,12),
[nbus,1]);
    Qxtra(:,2)=Qxtra(:,2)-
accumarray(zip(:,1),Rzip1.*zip(:,10).*zip(:,12),[nbus,1]);
end
% Qxtra(busType~=0,2:end)=Q(busType~=0,2:end);
% Q(busType~=0,2:end)=0;


% seperate real and image part of voltages and their inverse
% here V = C+1i*D
% and  W = 1./V = E + 1i*F
C0=real(V(:,1));
D0=imag(V(:,1));
E0=real(W(:,1));
F0=imag(W(:,1));

% Construct sparse matrix individually for C,D,E,F,P,Q
% Notice that Q = Q(:,1)+Qxtra(:,1) which is different from P
C0M=sparse(1:nbus,1:nbus,C0,nbus,nbus);
D0M=sparse(1:nbus,1:nbus,D0,nbus,nbus);
E0M=sparse(1:nbus,1:nbus,E0,nbus,nbus);
F0M=sparse(1:nbus,1:nbus,F0,nbus,nbus);
P0M=sparse(1:nbus,1:nbus,P(:,1),nbus,nbus);
Q0M=sparse(1:nbus,1:nbus,Q(:,1)+Qxtra(:,1),nbus,nbus);

% get real part and image part of Y matrix, not sure why do this
G=real(Y);
B=imag(Y);
% so Y = G + 1i*B


%-------------------------------------------------------------
% Determine the frequency model of each island
% 0:sw,1:syn,2:steady-state f
freqTypeTag=zeros(nIslands,1);%0:sw,1:syn,2:steady-state f
freqKeptTag=zeros(nbus,1);
frefs=refs;
fswTag=zeros(nbus,1);
fsynTag=zeros(nbus,1);
fswTag(isw)=1;
fswTagxD=fswTag;
fsynTag(syn(:,1))=1;
for isl=1:nIslands
    if isempty(find(fswTag(islands==isl)==1, 1))
        if isempty(find(fsynTag(islands==isl)==1, 1))
            freqTypeTag(isl)=2;
            busesInIsland=find(islands==isl);
            [~,imin]=min(abs(D0(busesInIsland)));
            frefs(isl)=busesInIsland(imin(1));
            fswTagxD(frefs(isl))=1;
            freqKeptTag(busesInIsland)=1;
```

```
        else
            freqTypeTag(isl)=1;
        end
    end
end
freqKeptTagxRef=freqKeptTag;
freqKeptTagxRef(frefs)=0;
nFreqKept=sum(freqKeptTag);
%------------------------------------------------------------




%--------------------------------------------------------------------
% this part is for initialling inductor
if ~isempty(ind)            % check if there is any inductor
    nInd=size(ind,1);       % determine the number of inductors
    indIdx=ind(:,1);        % store the index of inductors among all
 buses

    s=zeros(nInd,nlvl+1);   % slip
    s(:,1)=s0;              % initialize slip
    IL=zeros(nInd,nlvl+1);  %  |
    IR=zeros(nInd,nlvl+1);  %  |
    Vm=zeros(nInd,nlvl+1);  % initialization finished 0 value


   %-----------------parameters of inductors---------
   %-----------------START----------------
    R1=ind(:,7);
    X1=ind(:,8);
    Z1=ind(:,7)+1j*ind(:,8);
    Ze=1j*ind(:,13);
    R2=ind(:,9);
    X2=ind(:,10);
    T0=ind(:,15)+ind(:,16)+ind(:,17);
    T1=-ind(:,16)-2*ind(:,17);
    T2=ind(:,17);
    Hm=ind(:,14);
   %-----------------parameters of inductors---------
   %-----------------END----------------


    Rm=zeros(nInd,1);

    Am=sparse(indIdx,(1:nInd)',ones(1,nInd),nbus,nInd);

    % first order value of induction motor IL,VM,IR
    IL(:,1)=V0(indIdx)./(Z1+Ze.*(R2+1j*X2.*s0)./
(R2.*Reind0+(1j*X2.*Reind0+Ze).*s0));
    Vm(:,1)=V0(indIdx)-IL(:,1).*Z1;
    IR(:,1)=Vm(:,1).*s0./(R2+1j*X2.*s0);

    J0=real(IR(:,1));
```

```
    K0=imag(IR(:,1));
    JL0=real(IL(:,1));
    KL0=imag(IL(:,1));

    % prepare the algebric matrix
    Yeind0=Reind0./Ze;
    Yeind1=Reind1./Ze;
    Ye1ind0=Reind0.*Z1./Ze;
    Ye1ind1=Reind1.*Z1./Ze;
    Ge=real(Yeind0);
    Be=imag(Yeind0);
    kg1e=real(Ye1ind0);
    kb1e=imag(Ye1ind0);
    Ge1=real(Yeind1);
    Be1=imag(Yeind1);
    kg1e1=real(Ye1ind1);
    kb1e1=imag(Ye1ind1);

    %     LHS_MatInd_Shr_sqz=zeros(nInd,4);
    %     RHS_C_Shr_sqz=zeros(nInd,8);
    %     LHS_MatInd_Shr2_sqz=zeros(nInd,8);
    %
    %     LHS_MatInd_Shr=zeros(nInd,2,2);
    %     RHS_C_Shr=cell(nInd,1);
    %     LHS_MatInd_Shr2=cell(nInd,1); % A^-1B
    %     LHS_MatInd_Shr3=cell(nInd,1); % A^-1
    %
    %     for i=1:nInd
    %         LHS_MatInd=[R2(i),-X2(i)*s0(i),R1(i)*s0(i),-
X1(i)*s0(i),-s0(i),0;...
    %                     X2(i)*s0(i), R2(i),X1(i)*s0(i),
 R1(i)*s0(i),0,-s0(i);...
    %                     -1,0,1+kg1e(i),-kb1e(i),-Ge(i), Be(i);...
    %                     0,-1,kb1e(i), 1+kg1e(i),-Be(i),-Ge(i);];
    %         temp0=LHS_MatInd([3,4],[1,2])\eye(2);
    % A^-1
    %         LHS_MatInd_Shr2{i}=temp0*LHS_MatInd([3,4],[3,4,5,6]);
    % A^-1B
    %         LHS_MatInd_Shr3{i}=temp0;
 % A^-1
    %         temp1=LHS_MatInd([1,2],[1,2])/LHS_MatInd([3,4],[1,2]);
     % CA^-1
    %         temp2=LHS_MatInd([1,2],[3,4,5,6])-
temp1*LHS_MatInd([3,4],[3,4,5,6]); % L=D-CA^-1B
    %         LHS_MatInd_Shr(i,:,:)=-temp2(:,[1,2])\temp2(:,[3,4]);
     % -R\S
    %         RHS_C_Shr{i}=temp2(:,[1,2])\[eye(2),-temp1];
 % R\[I,-CA^-1]
    %
    %         LHS_MatInd_Shr_sqz(i,:)=reshape(LHS_MatInd_Shr(i,:,:),
[1,4]);
    %         RHS_C_Shr_sqz(i,:)=reshape(RHS_C_Shr{i},[1,8]);
    %         LHS_MatInd_Shr2_sqz(i,:)=reshape(LHS_MatInd_Shr2{i},
[1,8]);
```

```
    %     end
    %     LHS_MatInd_Bus=zeros(nbus,2,2);
     % \sum{-R\S} by buses
    %
 LHS_MatInd_Bus(:,1,1)=accumarray(indIdx,LHS_MatInd_Shr(:,1,1),
[nbus,1]);
    %
 LHS_MatInd_Bus(:,1,2)=accumarray(indIdx,LHS_MatInd_Shr(:,1,2),
[nbus,1]);
    %
 LHS_MatInd_Bus(:,2,1)=accumarray(indIdx,LHS_MatInd_Shr(:,2,1),
[nbus,1]);
    %
 LHS_MatInd_Bus(:,2,2)=accumarray(indIdx,LHS_MatInd_Shr(:,2,2),
[nbus,1]);

    MInd0=zeros(nInd,1);
    MInd1=ones(nInd,1);
    LHS_MatInd_sqz=[R2,X2.*s0,-MInd1,MInd0,...
        -X2.*s0,R2,MInd0,-MInd1,...
        R1.*s0,X1.*s0,MInd1+kg1e,kb1e,...
        -X1.*s0,R1.*s0,-kb1e,MInd1+kg1e,...
        -s0,MInd0,-Ge,-Be,...
        MInd0,-s0,Be,-Ge];   % 4*6 matrix [C,D;A,B]
    LHS_MatInd_idx=reshape((1:24)',[4,6]);
    temp0inv_sqz=LHS_MatInd_sqz(:,reshape(LHS_MatInd_idx([3,4],
[1,2]),1,[]));
    temp0inv_sqz_det=temp0inv_sqz(:,1).*temp0inv_sqz(:,4)-
temp0inv_sqz(:,2).*temp0inv_sqz(:,3);
    temp0_sqz=[temp0inv_sqz(:,4),-temp0inv_sqz(:,2),-
temp0inv_sqz(:,3),temp0inv_sqz(:,1)]./repmat(temp0inv_sqz_det,[1,4]);%
 A^-1
    indB_sqz=LHS_MatInd_sqz(:,reshape(LHS_MatInd_idx([3,4],
[3,4,5,6]),1,[]));

 LHS_MatInd_Shr2_sqz=[temp0_sqz(:,1).*indB_sqz(:,1)+temp0_sqz(:,3).*indB_sqz(:,2),

 temp0_sqz(:,1).*indB_sqz(:,3)+temp0_sqz(:,3).*indB_sqz(:,4),temp0_sqz(:,2).*indB_

 temp0_sqz(:,1).*indB_sqz(:,5)+temp0_sqz(:,3).*indB_sqz(:,6),temp0_sqz(:,2).*indB_

 temp0_sqz(:,1).*indB_sqz(:,7)+temp0_sqz(:,3).*indB_sqz(:,8),temp0_sqz(:,2).*indB_
 A^-1B
    indC_sqz=LHS_MatInd_sqz(:,reshape(LHS_MatInd_idx([1,2],[1,2]),1,
[]));

 temp1_sqz=[indC_sqz(:,1).*temp0_sqz(:,1)+indC_sqz(:,3).*temp0_sqz(:,2),indC_sqz(:

 indC_sqz(:,1).*temp0_sqz(:,3)+indC_sqz(:,3).*temp0_sqz(:,4),indC_sqz(:,2).*temp0_
 CA^-1
    temp2_sqz=LHS_MatInd_sqz(:,reshape(LHS_MatInd_idx([1,2],
[3,4,5,6]),1,[]))-...

 [temp1_sqz(:,1).*indB_sqz(:,1)+temp1_sqz(:,3).*indB_sqz(:,2),temp1_sqz(:,2).*indB
```

```
 temp1_sqz(:,1).*indB_sqz(:,3)+temp1_sqz(:,3).*indB_sqz(:,4),temp1_sqz(:,2).*indB_

 temp1_sqz(:,1).*indB_sqz(:,5)+temp1_sqz(:,3).*indB_sqz(:,6),temp1_sqz(:,2).*indB_

 temp1_sqz(:,1).*indB_sqz(:,7)+temp1_sqz(:,3).*indB_sqz(:,8),temp1_sqz(:,2).*indB_
 L=D-CA^-1B=[R,S]
    temp2_c12_sqz=temp2_sqz(:,1:4);
    temp2_c34_sqz=temp2_sqz(:,5:8);
    temp2_c12_sqz_det=temp2_c12_sqz(:,1).*temp2_c12_sqz(:,4)-
temp2_c12_sqz(:,2).*temp2_c12_sqz(:,3);
    temp2_c12_inv_sqz=[temp2_c12_sqz(:,4),-temp2_c12_sqz(:,2),-
temp2_c12_sqz(:,3),temp2_c12_sqz(:,1)]./repmat(temp2_c12_sqz_det,
[1,4]);
    LHS_MatInd_Shr_sqz=-
[temp2_c12_inv_sqz(:,1).*temp2_c34_sqz(:,1)+temp2_c12_inv_sqz(:,3).*temp2_c34_sqz(

 temp2_c12_inv_sqz(:,1).*temp2_c34_sqz(:,3)+temp2_c12_inv_sqz(:,3).*temp2_c34_sqz(
 -R\S
    RHS_C_Shr_sqz=[temp2_c12_inv_sqz,...
        -
[temp2_c12_inv_sqz(:,1).*temp1_sqz(:,1)+temp2_c12_inv_sqz(:,3).*temp1_sqz(:,2),tem

 temp2_c12_inv_sqz(:,1).*temp1_sqz(:,3)+temp2_c12_inv_sqz(:,3).*temp1_sqz(:,4),tem
 R\[I,-CA^-1]
    % will be used to calculate algebric variabls for motors
    LHS_MatInd_Bus_sqz=zeros(nbus,4);
  % \sum{-R\S} by buses
    LHS_MatInd_Bus_sqz(:,1)=accumarray(indIdx,LHS_MatInd_Shr_sqz(:,1),
[nbus,1]);
    LHS_MatInd_Bus_sqz(:,2)=accumarray(indIdx,LHS_MatInd_Shr_sqz(:,2),
[nbus,1]);
    LHS_MatInd_Bus_sqz(:,3)=accumarray(indIdx,LHS_MatInd_Shr_sqz(:,3),
[nbus,1]);
    LHS_MatInd_Bus_sqz(:,4)=accumarray(indIdx,LHS_MatInd_Shr_sqz(:,4),
[nbus,1]);
else
    s=zeros(0,nlvl+1);
end
% Initialization of inductors is finished
%----------------------------------------------------------------------


%------------------------------Initialization of ZIP load---------
if ~isempty(zip)
    nZip=size(zip,1);
    zipIdx=zip(:,1);
    IiL=zeros(nZip,nlvl+1);
    BiL=zeros(nZip,nlvl+1);

    % prepare the necessary matrix by blocks
    Bi0=abs(V0(zipIdx));
    JI=zip(:,6);
    KI=-zip(:,9);
```

```
    % current
    Ii0L=Rzip0.*(JI+1j*KI).*V0(zipIdx)./Bi0;
    Ji0L=real(Ii0L);
    Ki0L=imag(Ii0L);

    IiL(:,1)=Ii0L;
    BiL(:,1)=Bi0;
    % voltage
    Ci0=real(V0(zipIdx));
    Di0=imag(V0(zipIdx));

    LHS_MatZip=[Rzip0.*JI./Bi0-Ci0.*Ji0L./Bi0./Bi0,-Rzip0.*KI./Bi0-
Di0.*Ji0L./Bi0./Bi0,...
        Rzip0.*KI./Bi0-Ci0.*Ki0L./Bi0./Bi0,Rzip0.*JI./Bi0-Di0.*Ki0L./
Bi0./Bi0];
    Mat_BZip=[Ci0./Bi0,Di0./Bi0];
else
    IiL=zeros(0,nlvl+1);
end
%------------------------------Initialization of ZIP
 load------------------
%------------------------------Initialization of ZIP load is
 finished----------------


%------------------------------Initialization of GEN------------------
%------------------------------Start-----------------------
nSyn=size(syn,1);
if ~isempty(syn)
    synIdx =syn(:,1);% index number of Generators
    wgb    =syn(:,4);% maybe the base value
    modSyn =syn(:,5);% the order of generator models
    Xgl    =syn(:,6);
    Rga    =syn(:,7);
    Xgd    =syn(:,8);
    Xgd1   =syn(:,9);
    Xgd2   =syn(:,10);
    Tgd1   =syn(:,11);
    Tgd2   =syn(:,12);
    Xgq    =syn(:,13);
    Xgq1   =syn(:,14);
    Xgq2   =syn(:,15);
    Tgq1   =syn(:,16);
    Tgq2   =syn(:,17);
    Mg     =syn(:,18);
    Dg     =syn(:,19);
    TgAA   =syn(:,24);
    gammad =Tgd2./Tgd1.*Xgd2./Xgd1.*(Xgd-Xgd1);
    gammaq =Tgq2./Tgq1.*Xgq2./Xgq1.*(Xgq-Xgq1);

    d=zeros(nSyn,nlvl+1); % delta
    w=zeros(nSyn,nlvl+1); % omega
    eq1=zeros(nSyn,nlvl+1); %eq'
    eq2=zeros(nSyn,nlvl+1); %eq''
```

```
ed1=zeros(nSyn,nlvl+1); %ed'
ed2=zeros(nSyn,nlvl+1); %ed''
psiq=zeros(nSyn,nlvl+1); % not sure, only in 8th order model
psid=zeros(nSyn,nlvl+1); % not sure, only in 8th order model
JG=zeros(nSyn,nlvl+1);
KG=zeros(nSyn,nlvl+1);
IGq=zeros(nSyn,nlvl+1);
IGd=zeros(nSyn,nlvl+1);
VGq=zeros(nSyn,nlvl+1);
VGd=zeros(nSyn,nlvl+1);
Cd=zeros(nSyn,nlvl+1);
Sd=zeros(nSyn,nlvl+1);
Ef=zeros(nSyn,nlvl+1);
Pm=zeros(nSyn,nlvl+1);

cosd=cos(d0);
sind=sin(d0);
CG0=C0(synIdx);
DG0=D0(synIdx);
% the first value is given here, notice all are 8th order model
d(:,1)=d0;
w(:,1)=w0;
eq1(:,1)=eq10;
eq2(:,1)=eq20;
ed1(:,1)=ed10;
ed2(:,1)=ed20;
psiq(:,1)=psiq0;
psid(:,1)=psid0;

% transform between grid side and dq side

VGd(:,1)=sind.*CG0-cosd.*DG0;
VGq(:,1)=cosd.*CG0+sind.*DG0;
% now they are under dq side

Cd(:,1)=cosd; %  first order of cos(delta)
Sd(:,1)=sind; %  first order of sin(delta)
Ef(:,1)=Ef0;
Pm(:,1)=Pm0;

%check if controller exists
if ~isempty(Ef1)
    Ef(:,2)=Ef1;
end
if ~isempty(Eq11)
    eq1(:,2)=Eq11;
end
if ~isempty(Pm1)
    Pm(:,2)=Pm1;
end

% notice that here truncated taylor is applied
% and this is the key differnet from Dt rule
```

```
   % Here only at most 5 th order taylor series are considered for
 sin
   % and cos function
   [cosp,sinp,taylorN]=getTaylorPolynomials(d0,taylorN); % taylorN
 may be truncated

   Mats=zeros(nSyn,4);
   MatsR=zeros(nSyn,4);
   MatsRs=zeros(nSyn,4);

   % count the number for different kinds models
   % ex: modelTag = [ 0 0 0 0 0 10 0 0].'
   % ex: there are 10 gens using 6th order model
   modelTag=accumarray(modSyn,ones(nSyn,1),[8,1]);

   % determine the order of the model
   % Do we really need for loop?
   % the answer is yes since different gen may use different
   % order model
   for i=1:nSyn
       % 8th order, no need to change
       if modSyn(i)==8
           IGd(i,1)=(eq20(i)-psid0(i))/Xgd2(i);
           IGq(i,1)=(-ed20(i)-psiq0(i))/Xgq2(i);
           Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
       % 6th order
       elseif modSyn(i)==6
           % algebric equation to solve Id, Iq
           IGd(i,1)=((ed20(i)-VGd(i,1))*Rga(i)+(eq20(i)-
VGq(i,1))*Xgq2(i))/(Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
           IGq(i,1)=(-(ed20(i)-VGd(i,1))*Xgd2(i)+(eq20(i)-
VGq(i,1))*Rga(i))/(Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
           % transform matrix (inverse version)
           Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
           % Here matrix is the inverse matrix, to understand this
           % We have A*Ixy+B*Vxy = f => MatsR = A^-1, MatsRs = A^-1*B
 = MatsRs*B
           % so Ixy = MatsR*f-MatsRs*Vxy, which is used later to
           % eliminate Ixy when disturbance happens
           MatsR(i,:)=[sind(i)*Rga(i)-
cosd(i)*Xgd2(i),sind(i)*Xgq2(i)+cosd(i)*Rga(i),-cosd(i)*Rga(i)-
sind(i)*Xgd2(i),-cosd(i)*Xgq2(i)+sind(i)*Rga(i)]/...
               (Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
           MatsRs(i,:)=[MatsR(i,1)*sind(i)+MatsR(i,2)*cosd(i),-
MatsR(i,1)*cosd(i)+MatsR(i,2)*sind(i),...
               MatsR(i,3)*sind(i)+MatsR(i,4)*cosd(i),-
MatsR(i,3)*cosd(i)+MatsR(i,4)*sind(i)];
       % 5th order
       elseif modSyn(i)==5
           IGd(i,1)=((ed20(i)-VGd(i,1))*Rga(i)+(eq20(i)-
VGq(i,1))*Xgq2(i))/(Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
           IGq(i,1)=(-(ed20(i)-VGd(i,1))*Xgd2(i)+(eq20(i)-
VGq(i,1))*Rga(i))/(Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
           Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
```

```
            MatsR(i,:)=[sind(i)*Rga(i)-
cosd(i)*Xgd2(i),sind(i)*Xgq2(i)+cosd(i)*Rga(i),-cosd(i)*Rga(i)-
sind(i)*Xgd2(i),-cosd(i)*Xgq2(i)+sind(i)*Rga(i)]/...
                (Rga(i)*Rga(i)+Xgd2(i)*Xgq2(i));
            MatsRs(i,:)=[MatsR(i,1)*sind(i)+MatsR(i,2)*cosd(i),-
MatsR(i,1)*cosd(i)+MatsR(i,2)*sind(i),...
                MatsR(i,3)*sind(i)+MatsR(i,4)*cosd(i),-
MatsR(i,3)*cosd(i)+MatsR(i,4)*sind(i)];
        % 4th order
        elseif modSyn(i)==4
            IGd(i,1)=((ed10(i)-VGd(i,1))*Rga(i)+(eq10(i)-
VGq(i,1))*Xgq1(i))/(Rga(i)*Rga(i)+Xgd1(i)*Xgq1(i));
            IGq(i,1)=(-(ed10(i)-VGd(i,1))*Xgd1(i)+(eq10(i)-
VGq(i,1))*Rga(i))/(Rga(i)*Rga(i)+Xgd1(i)*Xgq1(i));
            Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
            MatsR(i,:)=[sind(i)*Rga(i)-
cosd(i)*Xgd1(i),sind(i)*Xgq1(i)+cosd(i)*Rga(i),-cosd(i)*Rga(i)-
sind(i)*Xgd1(i),-cosd(i)*Xgq1(i)+sind(i)*Rga(i)]/...
                (Rga(i)*Rga(i)+Xgd1(i)*Xgq1(i));
            MatsRs(i,:)=[MatsR(i,1)*sind(i)+MatsR(i,2)*cosd(i),-
MatsR(i,1)*cosd(i)+MatsR(i,2)*sind(i),...
                MatsR(i,3)*sind(i)+MatsR(i,4)*cosd(i),-
MatsR(i,3)*cosd(i)+MatsR(i,4)*sind(i)];
        % 3rd order
        elseif modSyn(i)==3
            IGd(i,1)=((-VGd(i,1))*Rga(i)+(eq10(i)-VGq(i,1))*Xgq(i))/
(Rga(i)*Rga(i)+Xgd1(i)*Xgq(i));
            IGq(i,1)=(-(-VGd(i,1))*Xgd1(i)+(eq10(i)-VGq(i,1))*Rga(i))/
(Rga(i)*Rga(i)+Xgd1(i)*Xgq(i));
            Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
            MatsR(i,:)=[sind(i)*Rga(i)-
cosd(i)*Xgd1(i),sind(i)*Xgq(i)+cosd(i)*Rga(i),-cosd(i)*Rga(i)-
sind(i)*Xgd1(i),-cosd(i)*Xgq(i)+sind(i)*Rga(i)]/...
                (Rga(i)*Rga(i)+Xgd1(i)*Xgq(i));
            MatsRs(i,:)=[MatsR(i,1)*sind(i)+MatsR(i,2)*cosd(i),-
MatsR(i,1)*cosd(i)+MatsR(i,2)*sind(i),...
                MatsR(i,3)*sind(i)+MatsR(i,4)*cosd(i),-
MatsR(i,3)*cosd(i)+MatsR(i,4)*sind(i)];
        % classical model
        elseif modSyn(i)==2
            IGd(i,1)=((-VGd(i,1))*Rga(i)+(Ef0(i)-VGq(i,1))*Xgq(i))/
(Rga(i)*Rga(i)+Xgd(i)*Xgq(i));
            IGq(i,1)=(-(-VGd(i,1))*Xgd(i)+(Ef0(i)-VGq(i,1))*Rga(i))/
(Rga(i)*Rga(i)+Xgd(i)*Xgq(i));
            Mats(i,:)=[sind(i),cosd(i),-cosd(i),sind(i)];
            MatsR(i,:)=[sind(i)*Rga(i)-
cosd(i)*Xgd(i),sind(i)*Xgq(i)+cosd(i)*Rga(i),-cosd(i)*Rga(i)-
sind(i)*Xgd(i),-cosd(i)*Xgq(i)+sind(i)*Rga(i)]/...
                (Rga(i)*Rga(i)+Xgd(i)*Xgq(i));
            MatsRs(i,:)=[MatsR(i,1)*sind(i)+MatsR(i,2)*cosd(i),-
MatsR(i,1)*cosd(i)+MatsR(i,2)*sind(i),...
                MatsR(i,3)*sind(i)+MatsR(i,4)*cosd(i),-
MatsR(i,3)*cosd(i)+MatsR(i,4)*sind(i)];
        end
```

```
    end
    % not sure how to use them now, but they are zeroth order of Ix
 and Iy
    JG(:,1)= sind.*IGd(:,1)+cosd.*IGq(:,1);
    KG(:,1)=-cosd.*IGd(:,1)+sind.*IGq(:,1);

    % put previous matrix in a right place in all buses instead of
 only
    % generator buses
    MatGCD=-
[sparse(synIdx,synIdx,MatsRs(:,1),nbus,nbus),sparse(synIdx,synIdx,MatsRs(:,2),nbus

 sparse(synIdx,synIdx,MatsRs(:,3),nbus,nbus),sparse(synIdx,synIdx,MatsRs(:,4),nbus
else
    d=zeros(0,nlvl+1);
    w=zeros(0,nlvl+1);
    eq1=zeros(0,nlvl+1);
    eq2=zeros(0,nlvl+1);
    ed1=zeros(0,nlvl+1);
    ed2=zeros(0,nlvl+1);
    psiq=zeros(0,nlvl+1);
    psid=zeros(0,nlvl+1);
    JG=zeros(0,nlvl+1);
    KG=zeros(0,nlvl+1);
    IGq=zeros(0,nlvl+1);
    IGd=zeros(0,nlvl+1);
    VGq=zeros(0,nlvl+1);
    VGd=zeros(0,nlvl+1);
    Cd=zeros(0,nlvl+1);
    Sd=zeros(0,nlvl+1);
    Ef=zeros(0,nlvl+1);
    Pm=zeros(0,nlvl+1);
end
%------------------------------Initialization of GEN-----------------
%------------------------------EnD-----------------------------------



%------------------------------Initialization of
 Exciter-----------------
%------------------------------
START-----------------------------------
if ~isempty(exc)
    nExc    =size(exc,1);
    % All Type 3 AVR, a 3rd order controller
    % for Type 3 AVR, avr0(:,1:3) are Vavrm, Vavrr, Vavrf,
    % and avr0(:,4) is reference Vref (input for secondary voltage
 control).
    excIdx  = exc(:,1);
    VavrMax = exc(:,3);
    VavrMin = exc(:,4);
    muavr0  = exc(:,5);
    Tavr1   = exc(:,7);
    Tavr2   = exc(:,6);
```

```
    vavrf0  = exc(:,8);
    Vavr0   = exc(:,9);
    Tavre   = exc(:,10);
    Tavrr   = exc(:,11);

    %here I need to check why Vavrref is time varing instead of
 constant
    % memory is given to state variables of EXC
    Vavrm  = zeros(nExc,nlvl+1);
    Vavrr  = zeros(nExc,nlvl+1);
    Vavrf  = zeros(nExc,nlvl+1);
    Vavrref= zeros(nExc,nlvl+1);

    % zeroth order value is given
    Vavrm(:,1)=real(Vavrm0);
    Vavrr(:,1)=real(Vavrr0);
    Vavrf(:,1)=real(Vavrf0);
    Vavrref(:,1)=real(Vavrref0);

    % here Varref1 is given with syspara
    if ~isempty(Varref1)
        Vavrref(:,2)=Varref1;
    end

    %  non-windup limiter, check the limit
    tavrMaxDiff=Vavrf(:,1)-VavrMax;
    tavrMinDiff=Vavrf(:,1)-VavrMin;

    % label values in different interval
    avrSt=zeros(nExc,1);
    avrSt(tavrMaxDiff>0)=1;
    avrSt(tavrMinDiff<0)=-1;

    % output after the limiter
    Ef(excIdx(avrSt==-1),1)=VavrMin(avrSt==-1);
    Ef(excIdx(avrSt== 1),1)=VavrMax(avrSt== 1);
    Ef(excIdx(avrSt== 0),1)=Vavrf(avrSt==0,1);

else
    Vavrm=zeros(0,nlvl+1);
    Vavrr=zeros(0,nlvl+1);
    Vavrf=zeros(0,nlvl+1);
    Vavrref=zeros(0,nlvl+1);
end
%------------------------------Initialization of
 Exciter------------------
%------------------------------END-----------------------------------




%------------------------------Initialization of Turbing
 Governor------------------
```

```
%------------------------------
START-----------------------------------
if ~isempty(tg)
    nTg  = size(tg,1);
    % Type 2 Turbing governor.
    % one DE, one AE and one limiter
    tgIdx  = tg(:,1);

    wtgref = tg(:,3);
    Rtg    = tg(:,4);
    Ttgmax = tg(:,5);
    Ttgmin = tg(:,6);
    Ttg2   = tg(:,7);
    Ttg1   = tg(:,8);

    tgovg  = zeros(nTg,nlvl+1); % tg
    tgovm  = zeros(nTg,nlvl+1); % Tmi*
    Tmech  = zeros(nTg,nlvl+1); % Tmi0

    % zeroth value is given
    tgovg(:,1)=real(tgovg0);
    tgovm(:,1)=real(tgovm0);
    Tmech(:,1)=real(tgovmech0);

    if ~isempty(Tmech1)
        Tmech(:,2)=Tmech1;
    end

    % check if limit is approached
    tgovMaxDiff=tgovm(:,1)-Ttgmax;
    tgovMinDiff=tgovm(:,1)-Ttgmin;

    govSt=zeros(nTg,1);
    govSt(tgovMaxDiff>0)=1;
    govSt(tgovMinDiff<0)=-1;
    % if limit is approached, set Pm to constant value
    Pm(tgIdx(govSt==0),1)=tgovm(govSt==0,1);
    Pm(tgIdx(govSt==1),1)=Ttgmax(govSt==1,1);
    Pm(tgIdx(govSt==-1),1)=Ttgmin(govSt==-1,1);
else
    tgovg=zeros(0,nlvl+1);
    tgovm=zeros(0,nlvl+1);
    Tmech=zeros(0,nlvl+1);
end
%------------------------------Initialization of Turbing
 Governor------------------
%------------------------------END-----------------------------------

% this part i don't quite understand. It looks like f denotes
 frequency
% on every bus, is it relevant with frequency dependant load?
% now i find that this is for dynamics of agc
f=zeros(nbus,nlvl+1);
f(:,1)=f0;
```

```
synTag=zeros(nbus,1);
synTag(syn(:,1))=1:nSyn;
numSynOnBus=accumarray(syn(:,1),1,[nbus,1]);
dpgTag=ones(nbus,1);
for islIdx=1:nIslands
    busIsland=find(islands==islIdx);
    synTagIsland=synTag(busIsland);
    wIsland=w(synTagIsland(synTagIsland~=0),1);
    if ~isempty(wIsland)
        f(busIsland,1)=mean(wIsland); % note that here the freq can be
 different
        dpgTag(busIsland)=0;
    end
end

%AGC part
if ~isempty(agc)
    agcExt=zeros(nbus,size(agc,2));
    agcExt(agc(:,1),:)=agc;
    dpg=zeros(nbus,nlvl+1);
    dpg(:,1)=dpg0;
    fdk=agcExt(:,2)+agcExt(:,3); %1/R+D
else
    dpg=zeros(nbus,nlvl+1);
    fdk=zeros(nbus,1);
end

% this is long term dynamic, it seems that not considered here
if ~isempty(cac)&&~isempty(cluster)

else
    qplt=zeros(0,nlvl+1);
    vg=zeros(0,nlvl+1);
end

% freq relevant part induced by AGC
FreqReal=sparse(1:nbus,1:nbus,-freqKeptTag.*fdk.*E0,nbus,nbus);
FreqImag=sparse(1:nbus,1:nbus,-freqKeptTag.*fdk.*F0,nbus,nbus);
Freq2freq=sparse([1:nbus,1:nbus],[1:nbus,frefs(islands)'],
[ones(1,nbus),-ones(1,nbus)],nbus,nbus);

Y11=-G;Y12=B;Y21=-B;Y22=-G;
% Influence to Origianl Power flow
YEF11=P0M+sparse(1:nbus,1:nbus,freqKeptTag.*(-
fdk.*f0+dpg0),nbus,nbus);YEF12=-Q0M;YEF21=-Q0M;YEF22=-P0M-
sparse(1:nbus,1:nbus,freqKeptTag.*(-fdk.*f0+dpg0),nbus,nbus);

% Counting influence of ZIP load into Y matrix
if ~isempty(zip)
    Y11=Y11-sparse(1:nbus,1:nbus,accumarray(zipIdx,LHS_MatZip(:,1),
[nbus,1]),nbus,nbus);
    Y12=Y12-sparse(1:nbus,1:nbus,accumarray(zipIdx,LHS_MatZip(:,2),
[nbus,1]),nbus,nbus);
```

```
    Y21=Y21-sparse(1:nbus,1:nbus,accumarray(zipIdx,LHS_MatZip(:,3),
[nbus,1]),nbus,nbus);
    Y22=Y22-sparse(1:nbus,1:nbus,accumarray(zipIdx,LHS_MatZip(:,4),
[nbus,1]),nbus,nbus);
end
YLHS=[Y11,Y12;Y21,Y22];

% Counting influence of Motors into small Y matrix
if ~isempty(ind)
    YLHS=YLHS-...

 [sparse(1:nbus,1:nbus,LHS_MatInd_Bus_sqz(:,1),nbus,nbus),sparse(1:nbus,1:nbus,LHS

 sparse(1:nbus,1:nbus,LHS_MatInd_Bus_sqz(:,2),nbus,nbus),sparse(1:nbus,1:nbus,LHS_
end

% Counting influence of generators into small Y matrix
if ~isempty(syn)
    YLHS=YLHS+MatGCD;
end

idxNonSw=find(busType~=2);
idxNonSwxD=find(fswTagxD==0);
idxNonSwD=find(busType~=2&fswTagxD==1);

% This is the left hand side matrix totally
LHS_mat=[YLHS([idxNonSw;idxNonSw+nbus],[idxNonSw;idxNonSw+nbus]),...
    [YEF11(idxNonSw,idxNonSw),YEF12(idxNonSw,idxNonSw),-
F0M(idxNonSw,ipv),FreqReal(idxNonSw,freqKeptTag==1);...
    YEF21(idxNonSw,idxNonSw),YEF22(idxNonSw,idxNonSw),-
E0M(idxNonSw,ipv),-FreqImag(idxNonSw,freqKeptTag==1)];...
    C0M(ipv,idxNonSw),D0M(ipv,idxNonSw),sparse(npv,2*npq+3*npv
+nFreqKept);...
    E0M(idxNonSw,idxNonSw),-
F0M(idxNonSw,idxNonSw),C0M(idxNonSw,idxNonSw),-
D0M(idxNonSw,idxNonSw),sparse(npq+npv,npv+nFreqKept);...

 F0M(idxNonSw,idxNonSw),E0M(idxNonSw,idxNonSw),D0M(idxNonSw,idxNonSw),C0M(idxNonSw
+npv,npv+nFreqKept);...

 sparse(sum(freqKeptTagxRef),size(idxNonSw,1)+size(idxNonSw,1)+2*npq
+3*npv),Freq2freq(freqKeptTagxRef==1,freqKeptTag==1);...

 sparse(size(idxNonSwD,1),size(idxNonSw,1)),sparse(1:size(idxNonSwD,1),idxNonSwD,o
+3*npv+nFreqKept)];

% if nbus<=500
%     [L_LHS_mat,U_LHS_mat,p_LHS_mat]=lu(LHS_mat,'vector');
% end


% deterine if we use LU factoration
% for this part, i assume the system algebrac equation is under a good
```

```
% dcondition number and the dimension is not very high, otherwise LU
 will
% be time consuming
useLU=isfield(SysPara,'iden')&&isfield(SysPara,'p_amd');

if useLU
    if isempty(SysPara.p_amd)
        p_amd = colamd (LHS_mat) ;
        save([SysPara.iden,'.mat'],'p_amd');
    else
        p_amd=SysPara.p_amd;
    end
    MxI = speye (size(LHS_mat)) ;
    MxQ = MxI (:, p_amd) ;
    if IS_OCTAVE
        [MxL,MxU,MxP,MxQx] = lu (LHS_mat*MxQ) ;
    else
        [MxL,MxU,MxP] = lu (LHS_mat*MxQ) ;
    end
end

Not enough input arguments.

Error in Copy_of_hemMachinePFSalientcontinueDyn (line 48)
[bus,sw,pv,pq,shunt,line,ind,zip,syn,exc,tg,agc,cac,cluster]=unfoldSysData(SysData

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
%%%this is the recursive part for computing high order of time series%
%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

% strat interations nlvl: order of Taylor series
for i=1:nlvl
    % seq2  provides two columns from 0 to i, and i to 0
    % seq2p provides two columns from 0 to i+1, and i+1 to 0
    % seq3  provides 3 columns, the summary of each row is equal to
 i(binominal coefficients)
    seq2=getseq(i,2);
    seq2p=getseq(i+1,2);
    seq3=getseq(i,3);
    idxSeq2=sum(seq2==i,2);
    idxSeq2x=sum(seq2(:,2)==i,2);
    idxSeq2p=sum(seq2p>=i,2);
    idxSeq3=sum(seq3==i,2);
    idxSeq3x=sum(seq3(:,[2,3])==i,2);

    % seq2R is usually used in constructing algebric equations
    % seq2R provides two columns from 1 to i-1, and i-1 to 1
    % seq2x provides two columns from 1 to i, and i-1 to 0
    % seq2m provides two columns from 0 to i-1, and i-1 to 0
    % seq2mm provides two columns from 0 to i-2, and i-2 to 0
    seq2R=seq2(idxSeq2==0,:);
```

```
    seq2x=seq2(idxSeq2x==0,:);
    seq2m=getseq(i-1,2);
    seq2mm=getseq(i-2,2);

    RHSILr=zeros(nbus,1);
    RHSILi=zeros(nbus,1);

    % This part is for induction motor
    if ~isempty(ind)
        % package right hand side vector at every iteration

 rhsM=sum(Vm(:,seq2R(:,1)+1).*s(:,seq2R(:,2)+1),2)-1j*X2.*sum(IR(:,seq2R(:,1)+1).*
        %         rhsI=-
real(sum(IR(:,seq2R(:,1)+1).*conj(IR(:,seq2R(:,2)+1)),2))+...
        %
 (T1.*sum(s(:,seq2R(:,1)+1).*s(:,seq2R(:,2)+1),2)+T2.*sum(s(:,seq3R(:,1)+1).*s(:,s
R2+...
        %
 (T0.*s(:,i)+T1.*sum(s(:,seq2m(:,1)+1).*s(:,seq2m(:,2)+1),2)+T2.*sum(s(:,seq3m(:,1
R2;

        %         s(:,i
+1)=(Rind0.*(T0.*s(:,i)+T1.*sum(s(:,seq2m(:,1)+1).*s(:,seq2m(:,2)+1),2)+T2.*sum(s(
        %             -
real(sum(IR(:,seq2m(:,1)+1).*conj(IR(:,seq2m(:,2)+1)),2)).*R2-2*Hm.*sum(repmat(seq
        %             ./(2*Hm.*s(:,1)*i);
        %         if i>=2
        %             s(:,i+1)=s(:,i+1)+...
        %
 Rind1.*(T0.*s(:,i-1)+T1.*sum(s(:,seq2mm(:,1)+1).*s(:,seq2mm(:,2)+1),2)+T2.*sum(s(
        %                 ./(2*Hm.*s(:,1)*i);
        %         end

        % update the high order of slip, a special setting is required
 for
        % low order when i<2
        s(:,i
+1)=(Rind0.*(T1.*s(:,i)+T2.*sum(s(:,seq2m(:,1)+1).*s(:,seq2m(:,2)+1),2))-
real(sum(Vm(:,seq2m(:,1)+1).*conj(IR(:,seq2m(:,2)+1)),2))./(2*Hm*i);
        if i>=2
            s(:,i+1)=s(:,i+1)+...

 Rind1.*(T1.*s(:,i-1)+T2.*sum(s(:,seq2mm(:,1)+1).*s(:,seq2mm(:,2)+1),2))...
                ./(2*Hm*i);
        end
        if i==1
            s(:,i+1)=s(:,i+1)+Rind0.*T0./(2*Hm*i);
        end
        if i==2
            s(:,i+1)=s(:,i+1)+Rind1.*T0./(2*Hm*i);
        end
        % for dynamic model, Right hand side vector is required a
 update
        addenRhs=Vm(:,1).*s(:,i+1)-1j*X2.*IR(:,1).*s(:,i+1);
```

```
        %         rhsBus=zeros(2,nInd);
        %         for j=1:nInd
        %
 rhsBus(:,j)=RHS_C_Shr{j}*[real(rhsM(j)+addenRhs(j));imag(rhsM(j)+addenRhs(j));0;0
        %         end

        % count the influence of dynamic of slip into rigt hand side
 vector
        tempRhsInd=rhsM+addenRhs;

 rhsBus=[RHS_C_Shr_sqz(:,1).*real(tempRhsInd)+RHS_C_Shr_sqz(:,3).*imag(tempRhsInd)

        %accumulate currents IL
        RHSILr=accumarray(indIdx,rhsBus(1,:)',[nbus,1]);
        RHSILi=accumarray(indIdx,rhsBus(2,:)',[nbus,1]);

        %             rhsBus=zeros(5,nInd);
        %
 rhsM=sum(Vm(:,seq2R(:,1)+1).*s(:,seq2R(:,2)+1),2)-1j*X2.*sum(IR(:,seq2R(:,1)+1).*
        %
 rhsImod=Rind1.*(T1.*s(:,i)+T2.*sum(s(:,seq2m(:,1)+1).*s(:,seq2m(:,2)+1),2))+Rind0
        %
 real(sum(V(indIdx,seq2R(:,1)+1).*conj(IR(:,seq2R(:,2)+1)),2))+...
        %
 real(sum(IL(:,seq2R(:,1)+1).*conj(IR(:,seq2R(:,2)+1)),2).*Z1);
        %             if i==1
        %                 rhsImod=rhsImod+Rind1.*T0;
        %             end
        %             rhsIL=V(indIdx,i).*Yeind1-IL(:,i).*Ye1ind1;
        %             for j=1:nInd
        %
 rhsBus(:,j)=squeeze(RHS_C_Shr(j,:,:))*[real(rhsM(j));imag(rhsM(j));rhsImod(j);rea
        %             end
        %             RHSILr=accumarray(indIdx,rhsBus(3,:)',[nbus,1]);
        %             RHSILi=accumarray(indIdx,rhsBus(4,:)',[nbus,1]);
    end

    % strat update ZIP load into currents
    RHSIiLr=zeros(nbus,1);
    RHSIiLi=zeros(nbus,1);
    if ~isempty(zip)

 RHS_BZip=(real(sum(V(zipIdx,seq2R(:,1)+1).*conj(V(zipIdx,seq2R(:,2)+1)),2))-
sum(BiL(:,seq2R(:,1)+1).*BiL(:,seq2R(:,2)+1),2))./Bi0/2;
        RHZ_BIConv=sum(IiL(:,seq2R(:,1)+1).*BiL(:,seq2R(:,2)+1),2);
        RHSILr_full=Rzip1.*(JI.*real(V(zipIdx,i))-
KI.*imag(V(zipIdx,i))))./Bi0-real(RHZ_BIConv)./Bi0-Ji0L.*RHS_BZip./Bi0;

 RHSILi_full=Rzip1.*(KI.*real(V(zipIdx,i))+JI.*imag(V(zipIdx,i)))./
Bi0-imag(RHZ_BIConv)./Bi0-Ki0L.*RHS_BZip./Bi0;
        RHSIiLr=accumarray(zipIdx,RHSILr_full,[nbus,1]);
        RHSIiLi=accumarray(zipIdx,RHSILi_full,[nbus,1]);
    end
```

```
    % Start uodate generators
    RHSIGr=zeros(nbus,1);
    RHSIGi=zeros(nbus,1);
    if ~isempty(syn)
        RhsEd=zeros(nSyn,1);
        RhsEq=zeros(nSyn,1);
        IGdAdd=zeros(nSyn,1);
        IGqAdd=zeros(nSyn,1);
        % select different models for generators
        if modelTag(8)>0
            d(modSyn==8,i+1)=(wgb(modSyn==8).*w(modSyn==8,i))/i;
            w(modSyn==8,i+1)=(Pm(modSyn==8,i)-...

 (sum(psid(modSyn==8,seq2m(:,1)+1).*IGq(modSyn==8,seq2m(:,2)+1),2)-
sum(psiq(modSyn==8,seq2m(:,1)+1).*IGd(modSyn==8,seq2m(:,2)+1),2))-...
                Dg(modSyn==8).*w(modSyn==8,i))./Mg(modSyn==8)/i;
            psid(modSyn==8,i
+1)=wgb(modSyn==8).*(Rga(modSyn==8).*IGd(modSyn==8,i)+psiq(modSyn==8,i)+VGd(modSyn
i;
            psiq(modSyn==8,i
+1)=wgb(modSyn==8).*(Rga(modSyn==8).*IGq(modSyn==8,i)-
psid(modSyn==8,i)+VGq(modSyn==8,i))/i;
            eq1(modSyn==8,i+1)=(-eq1(modSyn==8,i)-(Xgd(modSyn==8)-
Xgd1(modSyn==8)-gammad(modSyn==8)).*IGd(modSyn==8,i)+(1-
TgAA(modSyn==8)./Tgd1(modSyn==8)).*Ef(modSyn==8,i))./Tgd1(modSyn==8)/
i;
            ed1(modSyn==8,i+1)=(-ed1(modSyn==8,i)+(Xgq(modSyn==8)-
Xgq1(modSyn==8)-gammaq(modSyn==8)).*IGq(modSyn==8,i))./
Tgq1(modSyn==8)/i;
            eq2(modSyn==8,i+1)=(-eq2(modSyn==8,i)+eq1(modSyn==8,i)-
(Xgd1(modSyn==8)-
Xgd2(modSyn==8)+gammad(modSyn==8)).*IGd(modSyn==8,i)+TgAA(modSyn==8)./
Tgd1(modSyn==8).*Ef(modSyn==8,i))./Tgd2(modSyn==8)/i;
            ed2(modSyn==8,i+1)=(-
ed2(modSyn==8,i)+ed1(modSyn==8,i)+(Xgq1(modSyn==8)-
Xgq2(modSyn==8)+gammaq(modSyn==8)).*IGq(modSyn==8,i))./
Tgq2(modSyn==8)/i;
            IGdAdd(modSyn==8)=(eq2(modSyn==8,i+1)-psid(modSyn==8,i
+1))./Xgd2(modSyn==8);
            IGqAdd(modSyn==8)=(-ed2(modSyn==8,i+1)-psiq(modSyn==8,i
+1))./Xgq2(modSyn==8);
        end
        if modelTag(6)>0
            d(modSyn==6,i+1)=(wgb(modSyn==6).*w(modSyn==6,i))/i;
            w(modSyn==6,i+1)=(Pm(modSyn==6,i)-...

 (sum(VGq(modSyn==6,seq2m(:,1)+1).*IGq(modSyn==6,seq2m(:,2)+1),2)+sum(VGd(modSyn==

 Rga(modSyn==6).*(sum(IGq(modSyn==6,seq2m(:,1)+1).*IGq(modSyn==6,seq2m(:,2)+1),2)+
                Dg(modSyn==6).*w(modSyn==6,i))./Mg(modSyn==6)/i;
            eq1(modSyn==6,i+1)=(-eq1(modSyn==6,i)-(Xgd(modSyn==6)-
Xgd1(modSyn==6)-gammad(modSyn==6)).*IGd(modSyn==6,i)+(1-
```

```
TgAA(modSyn==6)./Tgd1(modSyn==6)).*Ef(modSyn==6,i))./Tgd1(modSyn==6)/
i;
            ed1(modSyn==6,i+1)=(-ed1(modSyn==6,i)+(Xgq(modSyn==6)-
Xgq1(modSyn==6)-gammaq(modSyn==6)).*IGq(modSyn==6,i))./
Tgq1(modSyn==6)/i;
            eq2(modSyn==6,i+1)=(-eq2(modSyn==6,i)+eq1(modSyn==6,i)-
(Xgd1(modSyn==6)-
Xgd2(modSyn==6)+gammad(modSyn==6)).*IGd(modSyn==6,i)+TgAA(modSyn==6)./
Tgd1(modSyn==6).*Ef(modSyn==6,i))./Tgd2(modSyn==6)/i;
            ed2(modSyn==6,i+1)=(-
ed2(modSyn==6,i)+ed1(modSyn==6,i)+(Xgq1(modSyn==6)-
Xgq2(modSyn==6)+gammaq(modSyn==6)).*IGq(modSyn==6,i))./
Tgq2(modSyn==6)/i;
            RhsEd(modSyn==6)=ed2(modSyn==6,i+1);
            RhsEq(modSyn==6)=eq2(modSyn==6,i+1);
        end
        if modelTag(5)>0
            d(modSyn==5,i+1)=(wgb(modSyn==5).*w(modSyn==5,i))/i;
            w(modSyn==5,i+1)=(Pm(modSyn==5,i)-...

 (sum(VGq(modSyn==5,seq2m(:,1)+1).*IGq(modSyn==5,seq2m(:,2)+1),2)+sum(VGd(modSyn==

 Rga(modSyn==5).*(sum(IGq(modSyn==5,seq2m(:,1)+1).*IGq(modSyn==5,seq2m(:,2)+1),2)+
                Dg(modSyn==5).*w(modSyn==5,i))./Mg(modSyn==5)/i;
            eq1(modSyn==5,i+1)=(-eq1(modSyn==5,i)-(Xgd(modSyn==5)-
Xgd1(modSyn==5)-gammad(modSyn==5)).*IGd(modSyn==5,i)+(1-
TgAA(modSyn==5)./Tgd1(modSyn==5)).*Ef(modSyn==5,i))./Tgd1(modSyn==5)/
i;
            eq2(modSyn==5,i+1)=(-eq2(modSyn==5,i)+eq1(modSyn==5,i)-
(Xgd1(modSyn==5)-
Xgd2(modSyn==5)+gammad(modSyn==5)).*IGd(modSyn==5,i)+TgAA(modSyn==5)./
Tgd1(modSyn==5).*Ef(modSyn==5,i))./Tgd2(modSyn==5)/i;
            ed2(modSyn==5,i+1)=(-ed2(modSyn==5,i)+(Xgq(modSyn==5)-
Xgq2(modSyn==5)).*IGq(modSyn==5,i))./Tgq2(modSyn==5)/i;
            RhsEd(modSyn==5)=ed2(modSyn==5,i+1);
            RhsEq(modSyn==5)=eq2(modSyn==5,i+1);
        end
        if modelTag(4)>0
            d(modSyn==4,i+1)=(wgb(modSyn==4).*w(modSyn==4,i))/i;
            w(modSyn==4,i+1)=(Pm(modSyn==4,i)-...

 (sum(VGq(modSyn==4,seq2m(:,1)+1).*IGq(modSyn==4,seq2m(:,2)+1),2)+sum(VGd(modSyn==

 Rga(modSyn==4).*(sum(IGq(modSyn==4,seq2m(:,1)+1).*IGq(modSyn==4,seq2m(:,2)+1),2)+
                Dg(modSyn==4).*w(modSyn==4,i))./Mg(modSyn==4)/i;
            eq1(modSyn==4,i+1)=(-eq1(modSyn==4,i)-(Xgd(modSyn==4)-
Xgd1(modSyn==4)).*IGd(modSyn==4,i)+Ef(modSyn==4,i))./Tgd1(modSyn==4)/
i;
            ed1(modSyn==4,i+1)=(-ed1(modSyn==4,i)+(Xgq(modSyn==4)-
Xgq1(modSyn==4)).*IGq(modSyn==4,i))./Tgq1(modSyn==4)/i;
            RhsEd(modSyn==4)=ed1(modSyn==4,i+1);
            RhsEq(modSyn==4)=eq1(modSyn==4,i+1);
        end
        if modelTag(3)>0
```

```
            d(modSyn==3,i+1)=(wgb(modSyn==3).*w(modSyn==3,i))/i;
            w(modSyn==3,i+1)=(Pm(modSyn==3,i)-...

 (sum(VGq(modSyn==3,seq2m(:,1)+1).*IGq(modSyn==3,seq2m(:,2)+1),2)+sum(VGd(modSyn==

 Rga(modSyn==3).*(sum(IGq(modSyn==3,seq2m(:,1)+1).*IGq(modSyn==3,seq2m(:,2)+1),2)+
                Dg(modSyn==3).*w(modSyn==3,i))./Mg(modSyn==3)/i;
            eq1(modSyn==3,i+1)=(-eq1(modSyn==3,i)-(Xgd(modSyn==3)-
Xgd1(modSyn==3)).*IGd(modSyn==3,i)+Ef(modSyn==3,i))./Tgd1(modSyn==3)/
i;
            RhsEd(modSyn==3)=0;
            RhsEq(modSyn==3)=eq1(modSyn==3,i+1);
        end
        if modelTag(2)>0
            d(modSyn==2,i+1)=(wgb(modSyn==2).*w(modSyn==2,i))/i;
            w(modSyn==2,i+1)=(Pm(modSyn==2,i)-...

 (sum(VGq(modSyn==2,seq2m(:,1)+1).*IGq(modSyn==2,seq2m(:,2)+1),2)+sum(VGd(modSyn==

 Rga(modSyn==2).*(sum(IGq(modSyn==2,seq2m(:,1)+1).*IGq(modSyn==2,seq2m(:,2)+1),2)+
                Dg(modSyn==2).*w(modSyn==2,i))./Mg(modSyn==2)/i;
            RhsEd(modSyn==2)=0;
            RhsEq(modSyn==2)=eq1(modSyn==2,i+1);
        end
        % this part may be different from DT
        AG0=cosp(:,2).*d(:,i+1);
        BG0=sinp(:,2).*d(:,i+1);
        % here multi-convolution is utilized as sine function is
        % approxiamted as a taylor series of delta
        if taylorN>=2

 AG0=AG0+cosp(:,3).*sum(d(:,seq2(:,1)+1).*d(:,seq2(:,2)+1),2);

 BG0=BG0+sinp(:,3).*sum(d(:,seq2(:,1)+1).*d(:,seq2(:,2)+1),2);
        end
        if taylorN>=3

 AG0=AG0+cosp(:,4).*sum(d(:,seq3(:,1)+1).*d(:,seq3(:,2)+1).*d(:,seq3(:,3)+1),2);

 BG0=BG0+sinp(:,4).*sum(d(:,seq3(:,1)+1).*d(:,seq3(:,2)+1).*d(:,seq3(:,3)+1),2);
        end
        if taylorN>=4
            seq4=getseq(i,4);

 AG0=AG0+cosp(:,5).*sum(d(:,seq4(:,1)+1).*d(:,seq4(:,2)+1).*d(:,seq4(:,3)+1).*d(:,

 BG0=BG0+sinp(:,5).*sum(d(:,seq4(:,1)+1).*d(:,seq4(:,2)+1).*d(:,seq4(:,3)+1).*d(:,
        end

        % high order coefficients of cos(delta) and sin(delta)
        Cd(:,i+1)=AG0;
        Sd(:,i+1)=BG0;
```

```
        VGdCr=sum(Cd(:,seq2x(:,1)+1).*VGd(:,seq2x(:,2)+1),2);%
 Vd*cosdta
        VGqCr=sum(Cd(:,seq2x(:,1)+1).*VGq(:,seq2x(:,2)+1),2);%
 Vq*cosdta
        VGdSr=sum(Sd(:,seq2x(:,1)+1).*VGd(:,seq2x(:,2)+1),2);%
 Vd*sindta
        VGqSr=sum(Sd(:,seq2x(:,1)+1).*VGq(:,seq2x(:,2)+1),2);%
 Vq*sindta
        JCr=sum(Cd(:,seq2x(:,1)+1).*JG(:,seq2x(:,2)+1),2);% similar,
 for currents
        KCr=sum(Cd(:,seq2x(:,1)+1).*KG(:,seq2x(:,2)+1),2);
        JSr=sum(Sd(:,seq2x(:,1)+1).*JG(:,seq2x(:,2)+1),2);
        KSr=sum(Sd(:,seq2x(:,1)+1).*KG(:,seq2x(:,2)+1),2);

        RHSIGxr=-(MatsRs(:,1).*(-VGdSr-VGqCr)+MatsRs(:,2).*(VGdCr-
VGqSr))+...
            (MatsR(:,1).*RhsEd+MatsR(:,2).*RhsEq)-(Mats(:,1).*(JSr-
KCr)+Mats(:,2).*(JCr+KSr))+(Mats(:,1).*IGdAdd+Mats(:,2).*IGqAdd);
        RHSIGxi=-(MatsRs(:,3).*(-VGdSr-VGqCr)+MatsRs(:,4).*(VGdCr-
VGqSr))+...
            (MatsR(:,3).*RhsEd+MatsR(:,4).*RhsEq)-(Mats(:,3).*(JSr-
KCr)+Mats(:,4).*(JCr+KSr))+(Mats(:,3).*IGdAdd+Mats(:,4).*IGqAdd);
        % current injections from generators IG
        RHSIGr=accumarray(synIdx,RHSIGxr,[nbus,1]);
        RHSIGi=accumarray(synIdx,RHSIGxi,[nbus,1]);
    end
    % update exciter, 3 state variables
    if ~isempty(exc)
        Vavrm(:,i+1)=(Vmag(synIdx(excIdx),i)-Vavrm(:,i))./Tavrr/i;
        Vavrr(:,i+1)=(muavr0.*(1-Tavr1./Tavr2).*(Vavrref(:,i)-
Vavrm(:,i))-Vavrr(:,i))./Tavr2/i;
        Vavrf(:,i+1)=((vavrf0.*Vmag(synIdx(excIdx),i)+...

 sum(Vavrr(:,seq2m(:,1)+1).*Vmag(synIdx(excIdx),seq2m(:,2)+1),2)+...
            muavr0.*Tavr1./Tavr2.*sum((Vavrref(:,seq2m(:,1)+1)-
Vavrm(:,seq2m(:,1)+1)).*Vmag(synIdx(excIdx),seq2m(:,2)+1),2))./Vavr0-
Vavrf(:,i))./Tavre/i;
        Ef(excIdx(avrSt==-1),i+1)=0;
        Ef(excIdx(avrSt== 1),i+1)=0;
        Ef(excIdx(avrSt== 0),i+1)=Vavrf(avrSt==0,i+1);
    end

    % update agc, one state variables
    if ~isempty(agc)
        dpg(:,i+1)=-f(:,i).*agcExt(:,4)/i;
        for islIdx=1:nIslands
            busIsland=find(islands==islIdx);
            synTagIsland=synTag(busIsland);
            wIsland=w(synTagIsland(synTagIsland~=0),i+1);
            if ~isempty(wIsland)
                f(busIsland,i+1)=mean(wIsland); % note that here the
 freq can be different
            end
        end % TODO: steady-state model
```

```
        % update generator participation part from agc
        if ~isempty(syn) %dynamic model (synchronous generators)
            if ~isempty(tg)
                Tmech(:,i+1)=Tmech(:,i+1)+dpg(syn(tg(:,1),1),i+1)./
numSynOnBus(syn(tg(:,1),1));
            end
            Pm(:,i+1)=Pm(:,i+1)+dpg(syn(:,1),i+1)./
numSynOnBus(syn(:,1));
        end
    end
    % update Turbine, 2 state variables
    if ~isempty(tg)
        tgovg(:,i+1)=(-(1-Ttg1./Ttg2).*w(tgIdx,i)./Rtg-tgovg(:,i))./
Ttg2/i;
        tgovm(:,i+1)=tgovg(:,i+1)-Ttg1./Ttg2.*w(tgIdx,i+1)./Rtg
+Tmech(:,i+1);

        Pm(tgIdx(govSt==0),i+1)=tgovm(govSt==0,i+1);
        Pm(tgIdx(govSt==1),i+1)=0;
        Pm(tgIdx(govSt==-1),i+1)=0;
    end

    % HEM Body
    RHS1=sum((-
P(:,seq2(:,1)+1)+1j*(Q(:,seq2(:,1)+1)+Qxtra(:,seq2(:,1)+1))).*conj(W(:,seq2(:,2)+1
        freqKeptTag.*sum(-
dpg(:,seq2(:,1)+1).*conj(W(:,seq2(:,2)+1)),2)+...

 freqKeptTag.*fdk.*sum(f(:,seq2R(:,1)+1).*conj(W(:,seq2R(:,2)+1)),2)+Ysh1.*V(:,i)+
    RHS2=-0.5*real(sum(V(:,seq2R(:,1)+1).*conj(V(:,seq2R(:,2)+1)),2));
    RHS3=sum(-W(:,seq2R(:,1)+1).*V(:,seq2R(:,2)+1),2);


    if i==1
        RHS2=RHS2+0.5*VspSq2(:,2);
    end

    compactRHS1=RHS1(busType~=2);
    compactRHS1=compactRHS1+Y(busType~=2,isw)*real(V(isw,i+1));
    % combine all current injection involing Motor, zip load, and
 Generators
    RHS=[real(compactRHS1)+RHSILr(busType~=2)+RHSIiLr(busType~=2)-
RHSIGr(busType~=2);...
        imag(compactRHS1)+RHSILi(busType~=2)+RHSIiLi(busType~=2)-
RHSIGi(busType~=2);...
        RHS2(ipv);...
        real(RHS3(busType~=2));...
        imag(RHS3(busType~=2));...
        zeros(sum(freqKeptTagxRef),1);...
        zeros(size(idxNonSwD,1),1)];
    % solve AE, notice that every time we need to solve Ax(k) =b(k),
 which
```

```
    % means that A in invariant for every order. so we only need to
 rebulid
    % b every iteration
    if useLU
        if IS_OCTAVE
            x = real(MxQ * MxQx* (MxU \ (MxL \ (MxP * RHS)))) ;
        else
            x =real( MxQ * (MxU \ (MxL \ (MxP * RHS)))) ;
        end
    else
        x=real(LHS_mat\RHS);
    end

    % x= [V;W;Q_pv;f]
    xC=real(V(:,i+1));
    xD=imag(V(:,i+1));
    xC(idxNonSw)=x(1:(npq+npv));
    xD(idxNonSw)=x(((npq+npv)+1):(2*(npq+npv)));
    V(:,i+1)=xC+1j*xD;
    W(busType~=2,i+1)=x((2*(npq+npv)+1):(3*(npq+npv)))+...
        1j*x((3*(npq+npv)+1):(4*(npq+npv)));
    Q(ipv,i+1)=x((4*(npq+npv)+1):(4*(npq+npv)+npv));
    f(freqKeptTag==1,i+1)=x((4*(npq+npv)+npv+1):end);

    Vmag(:,i+1)=(sum(V(:,seq2(:,1)+1).*conj(V(:,seq2(:,2)+1)),2)-
sum(Vmag(:,seq2R(:,1)+1).*Vmag(:,seq2R(:,2)+1),2))./Vmag(:,1)/2; %
 Calculate voltage magnitude

    % now update the Algebric variables for motors:IL,IR,VM
    if ~isempty(ind)
        %         for j=1:nInd
        %
 tempIL=squeeze(LHS_MatInd_Shr(j,:,:))*[real(V(indIdx(j),i
+1));imag(V(indIdx(j),i+1))]+rhsBus(:,j);
        %             tempIRs=-
LHS_MatInd_Shr2{j}*[tempIL;real(V(indIdx(j),i+1));imag(V(indIdx(j),i
+1))];
        %             IL(j,i+1)=tempIL(1)+1j*tempIL(2);
        %             IR(j,i+1)=tempIRs(1)+1j*tempIRs(2);
        %             Vm(j,i+1)=V(indIdx(j),i+1)-IL(j,i+1)*Z1(j);
        %         end
        tempILvr=LHS_MatInd_Shr_sqz(:,1).*real(V(indIdx,i
+1))+LHS_MatInd_Shr_sqz(:,3).*imag(V(indIdx,i+1))+rhsBus(1,:)';
        tempILvi=LHS_MatInd_Shr_sqz(:,2).*real(V(indIdx,i
+1))+LHS_MatInd_Shr_sqz(:,4).*imag(V(indIdx,i+1))+rhsBus(2,:)';
        tempIRsvr=-sum(LHS_MatInd_Shr2_sqz(:,
[1,3,5,7]).*[tempILvr,tempILvi,real(V(indIdx,i+1)),imag(V(indIdx,i
+1))],2);
        tempIRsvi=-sum(LHS_MatInd_Shr2_sqz(:,
[2,4,6,8]).*[tempILvr,tempILvi,real(V(indIdx,i+1)),imag(V(indIdx,i
+1))],2);
        IL(:,i+1)=tempILvr+1j*tempILvi;
        IR(:,i+1)=tempIRsvr+1j*tempIRsvi;
        Vm(:,i+1)=V(indIdx,i+1)-IL(:,i+1).*Z1;
```

```
    end

    % now update the Algebric variables for ZIP loads
    if ~isempty(zip)
        IiL(:,i
+1)=(LHS_MatZip(:,1)+1j*LHS_MatZip(:,3)).*real(V(zipIdx,i
+1))+(LHS_MatZip(:,2)+1j*LHS_MatZip(:,4)).*imag(V(zipIdx,i
+1))+(RHSILr_full+1j*RHSILi_full);
        BiL(:,i+1)=Mat_BZip(:,1).*real(V(zipIdx,i
+1))+Mat_BZip(:,2).*imag(V(zipIdx,i+1))+RHS_BZip;
    end

    % now update the Algebric variables for Generators: Vd,Vq, Id, Iq
    if ~isempty(syn)
        JG(:,i+1)=-MatsRs(:,1).*real(V(synIdx,i+1))-
MatsRs(:,2).*imag(V(synIdx,i+1))+RHSIGxr;
        KG(:,i+1)=-MatsRs(:,3).*real(V(synIdx,i+1))-
MatsRs(:,4).*imag(V(synIdx,i+1))+RHSIGxi;
        IGd(:,i+1)=JSr-KCr+sind.*JG(:,i+1)-cosd.*KG(:,i+1);
        IGq(:,i+1)=JCr+KSr+cosd.*JG(:,i+1)+sind.*KG(:,i+1);
        tempVGC=real(V(synIdx,i+1))-VGdSr-VGqCr;
        tempVGD=imag(V(synIdx,i+1))+VGdCr-VGqSr;
        VGd(:,i+1)=sind.*tempVGC-cosd.*tempVGD;
        VGq(:,i+1)=cosd.*tempVGC+sind.*tempVGD;
    end
end

% Output value: coefficients for every order.
Q=real(Q);
s=real(s);
d=real(d);
w=real(w);
eq1=real(eq1);
eq2=real(eq2);
ed1=real(ed1);
ed2=real(ed2);
psid=real(psid);
psiq=real(psiq);
Pm=real(Pm);
Ef=real(Ef);
Vavrm=real(Vavrm);
Vavrr=real(Vavrr);
Vavrf=real(Vavrf);
Vavrref=real(Vavrref);
tgovg=real(tgovg);
tgovm=real(tgovm);
Tmech=real(Tmech);
f=real(f);
dpg=real(dpg);
qplt=real(qplt);
vg=real(vg);

if ~isempty(exc)
    avr={Vavrm,Vavrr,Vavrf};
```

```
end

if ~isempty(tg)
    gov={tgovg,tgovm};
end

end
```

*Published with MATLAB® R2018b*