A

Symbolic Volterra Circuit Simulator Implementing Direct Calculation Method

A.1 User Guide

The Matlab implementation of the direct calculation method for symbolic Volterra circuit analysis is given here. The Matlab code is presented in a sequence as seen from the user's perspective. Fig. 5.2 can be used to graphically illustrate this sequence of code development.

First, the nonlinear current source information is generated using the main script in Section A.2. The required user inputs are the maximum nonlinear order to be considered and the number of tones in the excitation. The nonlinear current source information generated are stored in the nonlinear current data structures shown in Fig. 5.2, and recalled during circuit analysis. For the same type of excitation, these data structures can be saved to avoid redundant script execution. For completeness, the code for the functions called in the main script are also presented in Section A.2. The corresponding underlying theoretical development was presented in Section 5.2.2.

Next, the circuit to be analyzed needs to defined by the user. Two example scripts are given in Section A.3 to create the data structure describing the circuit. This data structure is accessed later by the circuit analysis program, and is implicitly part of the user input in Fig. 5.2. The two examples given correspond to the ones presented in Section 5.3 where circuit equations are derived and results from the circuit analysis program are presented. For other circuits, the user must first derive the circuit equations, then encode circuit information (such as the admittance matrix and controlling variables for the dependent nonlinear current sources) into appropriate data structures by following the examples scripts.

Finally, the user defined circuit is analyzed using the main script in Section A.4. The required user inputs are the IM response frequencies which are specified as frequency mix vectors. The circuit analysis results are stored in the circuit response data structure in Fig. 5.2. These results were presented in a tabular format for the two example circuits in Section 5.3. For completeness, the code for the functions called in the main script are also presented in Section A.4. The corresponding underlying theoretical development was presented in Section 5.2.3.

A.2 Generate Nonlinear Current Source

A.2.1 Main Script

```
%% ################################################### MIX, Wnr, Fp, Mn, Fm
%measure 1-tone + 2-tone response: find order N
N = 3;% max nonlinear order
Q = 2;% number of tones
[ MIX ] = Mix_Qtone_Sym( N, Q, struct('HD',{'wHD'}, ...
        'fListOrd',{'ascend'}), Opt_Disp);
%------------------------------ Poly
[ Wnr ]         = INL_Wnr( N, Opt_Disp);
[ Fp  Fp_ ]     = INL_Poly( N, MIX, Wnr, Opt_Disp);% Fp_{:}
[ Fp2 ]         = INL_Poly_Condense( Fp, Wnr, MIX, 'ALL', 0);
%------------------------------ Mult
[ Mn  Fm  Fm_ ] = INL_Mult( N, MIX, Opt_Disp);% Fm_{:}
[ Fm2 ]         = INL_Mult_Condense( Fm, Mn, MIX, 'ALL', 0);
```

```
%------------------------------
% save MIX Wnr Fp  Fp_ Fp2 Mn  Fm  Fm_ Fm2

[ MIX ] = Mix_BandDiv( MIX );
[ MIX ] = Mix_BandSYM2( MIX, 1);
```

A.2.2 Functions

Mix_Qtone_Sym Function

```
function [ MIX ] = Mix_Qtone_Sym( N, Q, Opt, Opt_Disp)
% for Q < 2-Tone only
%---------------------------
% mix_fSymVEC = [-f2 -f1 f1 f2]  Symbolic Freq
% fSymVEC       = [        f1 f2]   Symbols in Symbolic Freq
% fSymCEL       = {        f1 f2}
% fNumVEC       = [        #  # ]   Numeric-Val in Symbolic Freq
%---------------------------
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

    Nord    = N;
    Nfreq   = CumSum( Q );
    Ntone   = Q;
Buss_Sym
switch Q
    %=================================================================
    case{1} %Q=1 for 1-tone, thus mix_MAT = [m1 m2]
    MIX.mix_fSymVEC = [-fc fc];
    MIX.fNumVEC     = [ 200 ];%{-200,200}
    %=================================================================
    case{2} %Q=2 for 2-tone, thus mix_MAT = [m1 m2 m3 m4]
    MIX.mix_fSymVEC = [-f2 -f1  f1  f2];
    MIX.fNumVEC     = [ 200, 201 ];% {-201,-200,200,201}
    %=================================================================
    case{3} %Q = 3 for 3-tone, thus mix_MAT = [m1 m2 m3 m4 m5 m6]
    MIX.mix_fSymVEC = [-f3 -f2 -f1  f1  f2  f3];
    MIX.fNumVEC     = [ 200, 201, 202 ];% {-202,-201,-200,200,201,202}
    %=================================================================
    otherwise
        disp('Q > 3'); return;
    %=================================================================
end
    MIX.fSymVEC     = MIX.mix_fSymVEC( length(MIX.mix_fSymVEC)/2+1 : end);
    MIX.fSymCEL     = SYMvec2SYMcel( MIX.fSymVEC );

%% #####################################################
MIX.N   = N;
MIX.Q   = Q;
%----------------------------------------------------------------------
MIX.mix_MAT     = {};% {[4x4 double]  [10x4 double]  [20x4 double]}
MIX.fmix_MAT    = {};% {[4x2 double]  [10x2 double]  [20x2 double]}
MIX.fmixHD_VEC = {};% {[4x1 double]  [10x1 double]  [20x1 double]}
MIX.fmixSYM_VEC = {};% {[4x1 sym]     [10x1 sym]     [20x1 sym]}
MIX.mixCoef_VEC = {};% {[4x1 double]  [10x1 double]  [20x1 double]}
%----------------------------------------------------------------------
MIX.fmix_RCD        = {};% {[2x2 double]  [5x2 double]  [8x2 double]}
MIX.fmixHD_RCD     = {};% {[2x1 double]  [5x1 double]  [8x1 double]}
MIX.fmixSYM_RCD   = {};% {[2x1 sym]  [5x1 sym]  [8x1 sym]}
MIX.fmix_RCDIDX     = {};% {}
MIX.fmix_RCDIDX2    = {};% ptr to xxx_ALL  {[1 2]  [3 4 5 6 7]}
MIX.fmix_RCDIDXcel = {};% {}
%----------------------------------------------------------------------
MIX.fmix_ALL        = [];% [13x2 double]
MIX.fmixHD_ALL    = [];% [13x1 double]
MIX.fmixSYM_ALL     = [];% [13x1 sym]
MIX.fmix_ALLIDXcel = {};% ptr to xxx_RCD  {1x13 cell}
%----------------------------------------------------------------------

for n = 1 : N
    if(Opt_Disp)
        disp(['%########################## n=' int2str(n)])% DEBUG
    end
    [MIX.mix_MAT{n}     MIX.fmix_MAT{n}     MIX.fmixHD_VEC{n}       ...
     MIX.fmixSYM_VEC{n}      MIX.mixCoef_VEC{n} ...
     MIX.fmix_RCD{n}     MIX.fmixHD_RCD{n}       MIX.fmixSYM_RCD{n} ] = ...
        Mix_Qtone(Q, MIX.mix_fSymVEC, MIX.fSymVEC, MIX.fNumVEC, ...
            n, Opt, Opt_Disp);

    [ALL_IDX ALLXOR_IDX RCD_IDX     RCDXOR_IDX] = Find_VecIDX( ...
        MIX.fmix_ALL, MIX.fmix_RCD{n}, 'MATRIX', 'Not symSUBnum', [],0);
    MIX.fmix_RCDIDX2{n}(RCD_IDX)      = ALL_IDX;
    MIX.fmix_RCDIDX2{n}(RCDXOR_IDX)   = length( MIX.fmix_ALL ) ...
        + [1 : length(RCDXOR_IDX)];

    for nRCD = 1 : length( RCD_IDX )
        MIX.fmix_ALLIDXcel{ALL_IDX(nRCD)} = ...
            [MIX.fmix_ALLIDXcel{ALL_IDX(nRCD)}  [n; RCD_IDX(nRCD)] ];
    end
    for nRCD = 1 : length( RCDXOR_IDX )
        MIX.fmix_ALLIDXcel{length(MIX.fmixSYM_ALL) + nRCD} = ...
```

```
            [n; RCDXOR_IDX(nRCD)];
    end
    MIX.fmix_ALL = [MIX.fmix_ALL;   MIX.fmix_RCD{n}(RCDXOR_IDX,:) ];
    MIX.fmixHD_ALL = [MIX.fmixHD_ALL; MIX.fmixHD_RCD{n}(RCDXOR_IDX) ];
    MIX.fmixSYM_ALL = [MIX.fmixSYM_ALL; MIX.fmixSYM_RCD{n}(RCDXOR_IDX) ];

end
```

INL_Wnr Function

```
function [ Wnr ] = INL_Wnr( N, Opt_Disp)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

ord_FACTORIAL = FactorialVec([1:N]);
%----------------------------------------------------------
%Wnr.OrdPwr_MAT{n,r}                [ Vn^Pwr ] = [n1 ... nr]
%Wnr.c_VEC{n,r}                     ([Multiplicant])
%   .Ord_MAT(remove bc implicit)    position cC of non-zero in OrdPwr_MAT
%----------------------------------------------------------
%Wnr.PolyOrd_MAT{n,1}   k=[Multiplicant Vr1-Ord ... Vrn-Ord]=[n1 n1 n2 n2]
%    PolyDeg_MAT        implicit: [1...1]
%Wnr.cFCT_VEC{n,1}  calc iNL for NLTF: n!/([1!^k1...n!^kn]).*[k1! ... kn!])
%                                     * limited-Arithm-Avg( H(f1)H(f1,f2) )
%                       normally: n! * Arithm-Avg( H(f1)H(f1,f2) )
if(Opt_Disp)
sprintf('c \t cFCT \t OrdPwr')
end

% Wnr.OrdPwr_MAT: {3x3 cell}
%      Wnr.c_VEC: {3x3 cell}
%   Wnr.cFCT_VEC: {3x3 cell}
%Wnr.PolyOrd_MAT: {3x3 cell}
%% ########################### Wnr{n,r}.OrdPwr_MAT
for n = 1 : N
    if(Opt_Disp)
        disp(['%================================ n=' num2str(n)]);
    end
    Wnr.OrdPwr_MAT{n,1}   = zeros(1,N);   Wnr.OrdPwr_MAT{n,1}(n) = 1;
    Wnr.c_VEC{n,1}          = [1];
    Wnr.cFCT_VEC{n,1}    = factorial(n);
    Wnr.PolyOrd_MAT{n,1} = [n];
```

```
for r = 2 : n
    if(Opt_Disp) disp(['%---------------------(n,r)=' num2str([n r])]); end
    OrdPwr_MAT = [];
    c_VEC      = [];
cFCT_VEC    = [];
    PolyOrd_MAT = [];
%===================================================== OrdPwr_MAT, c_VEC
for nI = 1 : min([n-r+1 N])
    if(Opt_Disp)
        [[nI 1]   [n-nI r-1]]% DEBUG
    end
MAT1 = Wnr.OrdPwr_MAT{nI,1};
    MAT2    = Wnr.OrdPwr_MAT{n-nI,r-1};
    VEC1 = Wnr.c_VEC{nI,1};
    VEC2 = Wnr.c_VEC{n-nI,r-1};
    wH  = [];
    cH  = [];
for n1 = 1 : size(MAT1,1)
    for n2 = 1 : size(MAT2,1)
        % ADD vector of power of each Vn
        wH(size(wH,1)+1,:)  = MAT1(n1,:) + MAT2(n2,:);
        % MULTIPLY coef of prod
        cH(size(cH,1)+1,1) = VEC1(n1,1) * VEC2(n2,1);
    end
    end
    [wnr_IDX wnr_IDX_ wH_IDX wH_IDX_]    = Find_VecIDX( ...
        OrdPwr_MAT, wH,'MATRIX','Not symSUBnum',[],0);
    c_VEC(wnr_IDX) = c_VEC(wnr_IDX) + cH(wH_IDX);% existing
    OrdPwr_MAT     = [OrdPwr_MAT;  wH(wH_IDX_,:)];% new
    c_VEC          = [c_VEC;       cH(wH_IDX_,1)];% new
end
    Wnr.OrdPwr_MAT{n,r} = OrdPwr_MAT;
    Wnr.c_VEC{n,r}      = c_VEC;
%===================================================== PolyOrd_MAT, cFCT_VEC
for rC = 1 : size(OrdPwr_MAT,1)
        PolyOrd_VEC = [];
        %         ([n1! ... nr!] = [1!^k1 ... n!^kn]) .* [k1! ... kn!]
        cFCT = (ord_FACTORIAL .^ OrdPwr_MAT(rC,:)) ...
                       .* FactorialVec(OrdPwr_MAT(rC,:));
        cFCT = cFCT( find(cFCT ~= 0) );
        cFCT = prod( cFCT );
    for cC = 1 : size(OrdPwr_MAT,2)
        if(OrdPwr_MAT(rC,cC)~=0)
```

```
           PolyOrd_VEC = [PolyOrd_VEC repmat(cC,[1 OrdPwr_MAT(rC,cC)])];
        end
   end
        PolyOrd_MAT(rC,:)   = PolyOrd_VEC;
        cFCT_VEC            = [cFCT_VEC;    cFCT];
end
    Wnr.PolyOrd_MAT{n,r} = PolyOrd_MAT;
    Wnr.cFCT_VEC{n,r}       = cFCT_VEC;
%====================================================
    if(Opt_Disp)
        [c_VEC      cFCT_VEC    OrdPwr_MAT]% DEBUG
        PolyOrd_MAT% DEBUG
    end
end
end
```

INL_Poly Function

```
function [ Fp  Fp_ ] = INL_Poly( N, MIX, Wnr, ...
    Opt_Disp)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

%% ###################################################### initialize
for n = 1 : N
    Fp_{n} = zeros( size(MIX.fmix_RCD{n},1), 2);
for nRESP = 1 : size( MIX.fmix_RCD{n},1 )
%--------------------------------------------------------------- init .wnr
for r = 2 : n
for rC = 1 : size( Wnr.PolyOrd_MAT{n,r},1 )
    for nri = 1 : 2
        Fp{n}{nRESP,nri}.wnr{r}{rC,1} = [];% PolyOrd_nMIXMAT
        Fp{n}{nRESP,nri}.wnr{r}{rC,2} = [];% PolyOrd_riMAT
    end
end
end
%--------------------------------------------------------------- init .fmix
for k = 1 : N
    for nri = 1 : 2
        Fp{n}{nRESP,nri}.fmix{k,1} = zeros( size( MIX.fmix_RCD{k},1), 2);
    end
end
```

```
    %------------------------------ Lin-fResp for 2Tone
    if(n == 1)
        Fp{1}{nRESP,1}.fmix{1}(nRESP,:) = [1 0];
        Fp{1}{nRESP,2}.fmix{1}(nRESP,:) = [0 1];
    end
%-----------------------------------------------------------------
end
end

%% ######################################################
tic
for n = 2 : N
for r = 2 : n
%-----------------------------------------------------------------
for rC = 1 : size( Wnr.PolyOrd_MAT{n,r}, 1 )
    PolyOrd_IDX    = 1;% (current position in VEC)
PolyOrd_VEC     = Wnr.PolyOrd_MAT{n,r}(rC,:);% [ord ord ...]
    PolyOrd_nMIXVEC = [];% [nMIX nMIX ...]
    PolyOrd_riVEC = [];% [ri   ri   ...]
if(Opt_Disp)
      disp(’%=========================================’)
        PolyOrd_VEC
end
    %iterate all possible nMIX/nri @(current position in VEC)
    %Recurrsive func call bc: VEC is variable length, can’t use for-loop
    [ Fp{n} Fp_{n}] = INL_Fnr_ForRecur( MIX.fmix_RCD, PolyOrd_VEC, ...
        PolyOrd_IDX, PolyOrd_nMIXVEC, PolyOrd_riVEC, ...
        n, r, rC, Fp{n}, Fp_{n}, Opt_Disp);
end
%-----------------------------------------------------------------
end
        if(Opt_Disp)
            Fn_ = Fp_{n}
        end
end
toc

%% ###################################################### INL_Fnr_ForRecur
function [ Fn Fn_ ] = INL_Fnr_ForRecur( fmix_RCD, PolyOrd_VEC, ...
    PolyOrd_IDX, PolyOrd_nMIXVEC, PolyOrd_riVEC, ...
    n, r, rC, Fn, Fn_, Opt_Disp)
if ~exist(’Opt_Disp’,’var’)
    Opt_Disp = 0;
```

```
end

%ord @(current position in VEC)
nORD = PolyOrd_VEC( PolyOrd_IDX );
    if(Opt_Disp)
        disp('%---------------------------------------')
        sprintf('PolyOrd_IDX=%d     nORD=%d', PolyOrd_IDX, nORD)
    end
%iterate all possible nMIX/nri @(current position in VEC)
for ri = 1 : -2 : -1
    PolyOrd_riVEC( PolyOrd_IDX )    = ri;
    nMIXstart = 1;
    if (ri == -1) && ...
        isequal( fmix_RCD{nORD}(1,:), zeros( size(fmix_RCD{nORD}(1,:)) ) )
        nMIXstart = 2;% skip double-counting if DC (sorted to be 1st-pt)
    end
for nMIX = nMIXstart : size( fmix_RCD{nORD}, 1 )
    PolyOrd_nMIXVEC( PolyOrd_IDX )  = nMIX;
%-----------------------------------------------------------------
if( PolyOrd_IDX < length( PolyOrd_VEC ) )% not at final position in VEC
    %############### Don't forget to modify
    %iterate all possible nMIX/nri @(current position in VEC)
    %Recurrsive func call bc: VEC is variable length, can't use for-loop
[ Fn Fn_ ] = INL_Fnr_ForRecur( fmix_RCD, PolyOrd_VEC, ...
     PolyOrd_IDX+1, PolyOrd_nMIXVEC, PolyOrd_riVEC, ...
        n, r, rC, Fn, Fn_, Opt_Disp);
else% reaches final position in VEC: can calc Result fmix_VEC ~OUTPUT
    [ Fn    Fn_ ] = INL_Fnr_fmixVEC( fmix_RCD, PolyOrd_VEC, ...
        PolyOrd_nMIXVEC, PolyOrd_riVEC, ...
        n, r, rC, Fn, Fn_, Opt_Disp);
end
%-----------------------------------------------------------------
end
end

%% #####################################################
function [ Fn Fn_ ] = INL_Fnr_fmixVEC( fmix_RCD, PolyOrd_VEC, ...
    PolyOrd_nMIXVEC, PolyOrd_riVEC, ...
    n, r, rC, Fn, Fn_, Opt_Disp)

fmix_VEC = [];
for k = 1 : length( PolyOrd_VEC )
    Ord_k   = PolyOrd_VEC(k);
    Mix_k   = PolyOrd_nMIXVEC(k);
    ri_k    = PolyOrd_riVEC(k);
fmix_VEC = [fmix_VEC;       fmix_RCD{ Ord_k }(Mix_k,:) * ri_k];
end
    if(Opt_Disp)
        disp('%--------------------')
        fmix_VEC
    end
fmix_VEC = sum( fmix_VEC, 1 );
%----------------------------- find fmix_VEC in fmix_RCD{n} = nmix
%----------------------------- Fn{nmix,nri}.wnr{r}{rC,:}={nMIXVEC  riVEC}
[ nmix  nri ] = Mix_RCDsearch( fmix_VEC, fmix_RCD{n});
    Fn{nmix,nri}.wnr{r}{rC,1}=[Fn{nmix,nri}.wnr{r}{rC,1}; PolyOrd_nMIXVEC];
    Fn{nmix,nri}.wnr{r}{rC,2}=[Fn{nmix,nri}.wnr{r}{rC,2}; PolyOrd_riVEC];
    Fn{nmix,nri}.fmix{n,1}(nmix, nri) = 1;
    Fn_(nmix,nri) = 1;
%----------------------------- Fn{nmix,nri}.fmix(PolyOrd, nMIXVEC, riVEC)=1
for k = 1 : length( PolyOrd_VEC )
    Ord_k = PolyOrd_VEC(k);
    Mix_k = PolyOrd_nMIXVEC(k);
    ri_k    = PolyOrd_riVEC(k);
        if(ri_k == -1)
            nri_k = 2;
        else
            nri_k = 1;
        end
Fn{ nmix, nri}.fmix{Ord_k,1}(Mix_k, nri_k) = 1;
    %for NL-Cap, keep track 'jw' contribs (to the same fresp) in list?
    %can use same Wnr expansion: need further modification
end
```

INL_Poly_Condense Function

```
function [ Fp2 ] = INL_Poly_Condense( Fp, Wnr, MIX, Opt, Opt_Disp,...
    RCD_IDX, N)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

switch Opt
%% ==============================================
    case{'Select'}
for n = 1 : N
for nRESP = 1 : size( RCD_IDX.nMIX, 1 )
nmix = RCD_IDX.nMIX(nRESP,n);
nri  = RCD_IDX.ri(nRESP,n);
if(nmix ~= 0)
if(Opt_Disp)
        disp(sprintf(’==========n=%-5d  nMIX=%-5d  ri=%-5d fmixSYM=%-s’,...
        n, nmix, nri, char(MIX.fmixSYM_RCD{n}(nmix))) );
    end
%##########################################################
for r = 2 : n
    if(Opt_Disp)
        disp([’%======================================[r]=’ num2str([r])]);
        Wnr.OrdPwr_MAT{n,r}
        Wnr.PolyOrd_MAT{n,r}
        Wnr.c_VEC{n,r}
end
for rC = 1 : size(Wnr.PolyOrd_MAT{n,r}, 1)%size(Fp{n}{ nmix,nri }.wnr{r},1)
    PolyOrd_VEC       = Wnr.PolyOrd_MAT{n,r}(rC,:);
    PolyOrd_nMIXMAT     = Fp{n}{ nmix,nri }.wnr{r}{rC,1};
    PolyOrd_riMAT       = Fp{n}{ nmix,nri }.wnr{r}{rC,2};
    if(Opt_Disp)
        disp([’%-----------------------------------[rC]=’ num2str([rC])]);
    end
if(size( PolyOrd_nMIXMAT, 1 ) > 0)
%============================================================rMIX
[ PolyOrd_nMIXMAT_2  PolyOrd_riMAT_2  PolyOrd_coefVEC_2 ] = ...
    Poly_Condense( PolyOrd_VEC, PolyOrd_nMIXMAT, PolyOrd_riMAT, Opt_Disp);
Fp2{n}{ nmix,nri }.wnr{r}{rC,1} = PolyOrd_nMIXMAT_2;
Fp2{n}{ nmix,nri }.wnr{r}{rC,2} = PolyOrd_riMAT_2;
Fp2{n}{ nmix,nri }.wnr{r}{rC,3} = PolyOrd_coefVEC_2;
%============================================================
end
    if(Opt_Disp)
        PolyOrd_nMIXMAT
        PolyOrd_riMAT
        disp( sprintf(’%################mix = %d reduced to %d’, ...
            size(PolyOrd_nMIXMAT,1), size(PolyOrd_nMIXMAT_2,1)) );
        PolyOrd_nMIXMAT_2
        PolyOrd_riMAT_2
        PolyOrd_coefVEC_2
    end
end
end
```

```
%#############################################################
end
end
end
%% =============================================
    case{'ALL'}
for n           = 1 : length( Fp )
for nmix = 1 : size( Fp{n},1 )
for nri     = 1 : size( Fp{n},2 )
%############################################################
for r = 2 : n
    if(Opt_Disp)
        disp(['%======================================[r]=' num2str([r])]);
        Wnr.OrdPwr_MAT{n,r}
        Wnr.PolyOrd_MAT{n,r}
        Wnr.c_VEC{n,r}
end
for rC = 1 : size(Wnr.PolyOrd_MAT{n,r},1)%size(Fp{n}{ nmix,nri }.wnr{r}, 1)
    PolyOrd_VEC       = Wnr.PolyOrd_MAT{n,r}(rC,:);
    PolyOrd_nMIXMAT     = Fp{n}{ nmix,nri }.wnr{r}{rC,1};
    PolyOrd_riMAT       = Fp{n}{ nmix,nri }.wnr{r}{rC,2};
    if(Opt_Disp)
        disp(['%------------------------------------[rC]=' num2str([rC])]);
        PolyOrd_nMIXMAT
        PolyOrd_riMAT
    end
if(size( PolyOrd_nMIXMAT, 1 ) > 0)
%============================================================rMIX
[ PolyOrd_nMIXMAT_2  PolyOrd_riMAT_2  PolyOrd_coefVEC_2 ] = ...
    Poly_Condense( PolyOrd_VEC, PolyOrd_nMIXMAT, PolyOrd_riMAT, Opt_Disp);
Fp2{n}{ nmix,nri }.wnr{r}{rC,1} = PolyOrd_nMIXMAT_2;
Fp2{n}{ nmix,nri }.wnr{r}{rC,2} = PolyOrd_riMAT_2;
Fp2{n}{ nmix,nri }.wnr{r}{rC,3} = PolyOrd_coefVEC_2;
%============================================================
end
    if(Opt_Disp)
        disp(['%###############']);
        PolyOrd_nMIXMAT_2
        PolyOrd_riMAT_2
        PolyOrd_coefVEC_2
        disp( sprintf('#mix = %d reduced to %d', ...
            size(PolyOrd_nMIXMAT,1), size(PolyOrd_nMIXMAT_2,1)) );
    end
```

```
end
end
%##########################################################################
end
end
end
%% =============================================
end

%% ######################################################
function [ PolyOrd_nMIXMAT_2  PolyOrd_riMAT_2  PolyOrd_coefVEC_2 ] = ...
    Poly_Condense( PolyOrd_VEC, PolyOrd_nMIXMAT, PolyOrd_riMAT, Opt_Disp)

PolyOrd_nMIXMAT_2 = PolyOrd_nMIXMAT(1,:);% Init
PolyOrd_riMAT_2     = PolyOrd_riMAT(1,:);% Init
PolyOrd_coefVEC_2  = [1];% Init

for rMIX = 2 : size( PolyOrd_nMIXMAT, 1 )
    isFOUND = 0;
    %-----------------------------------------------------------------
    for rMIX_2 = 1 : size( PolyOrd_nMIXMAT_2, 1 )
        [ isFOUND ] = isSAME_Permute( PolyOrd_VEC, ...
            PolyOrd_nMIXMAT(rMIX,:),        PolyOrd_riMAT(rMIX,:), ...
            PolyOrd_nMIXMAT_2(rMIX_2,:),   PolyOrd_riMAT_2(rMIX_2,:) );
        if( isFOUND )% already exist in *_2
            break
        end
    end
    %-----------------------------------------------------------------
    if( isFOUND )% already exist in *_2
        PolyOrd_coefVEC_2( rMIX_2 ) = PolyOrd_coefVEC_2( rMIX_2 ) + 1;
        if( Opt_Disp )
            disp(['%##############[rMIX rMIX_2]=' num2str([rMIX rMIX_2])]);
        end
    else% NOT in *_2
        PolyOrd_nMIXMAT_2   = [PolyOrd_nMIXMAT_2; PolyOrd_nMIXMAT(rMIX,:)];
        PolyOrd_riMAT_2     = [PolyOrd_riMAT_2;   PolyOrd_riMAT(rMIX,:)];
        PolyOrd_coefVEC_2  = [PolyOrd_coefVEC_2; 1];
    end
    %-----------------------------------------------------------------
end

%% ######################################################
```

```
function [ isFOUND ] = isSAME_Permute( Ord_VEC, ...
    mixVEC, riVEC, mixVEC2, riVEC2 )

ordPERM = perms( Ord_VEC );
mixPERM = perms( mixVEC );
riPERM  = perms( riVEC );

isFOUND = 0;
for nPERM = 1 : size( mixPERM, 1)
ordBOOL = (ordPERM(nPERM,:) == Ord_VEC);
    if isempty(find(ordBOOL == 0))% all 1's
        %------------------------------
      mixBOOL = (mixPERM(nPERM,:) == mixVEC2);
        riBOOL  = (riPERM(nPERM,:)  == riVEC2);
        if isempty(find(mixBOOL == 0)) ...
            && isempty(find(riBOOL == 0))%all 1s
            isFOUND = 1;
            break;
        end
        %------------------------------
    end
end
```

INL_Mult Function

```
function [ Mn  Fm  Fm_ ] = INL_Mult( N, MIX, ...
    Opt_Disp)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

%% ############################################# Mn (like Wnr for INL_Poly)
for n = 1 : N
    Mn.PolyOrd_MAT{n} = [];% initialize
end

for nU = 1 : N
for nV = 1 : N
    n = nV + nU;
    if( n <= N )
        Mn.PolyOrd_MAT{n} = [Mn.PolyOrd_MAT{n};  [nV  nU]];
    end
end
```

```
end

%% ############################################# init
for n = 1 : N
    Fm_{n} = zeros( size(MIX.fmix_RCD{n},1), 2);
for nRESP = 1 : size( MIX.fmix_RCD{n},1 )
%--------------------------------------------------------------- init .wnr
for rC = 1 : size( Mn.PolyOrd_MAT{n},1 )
    for nRI = 1 : 2
        Fm{n}{nRESP,nRI}.Mn{rC,1} = [];% PolyOrd_nMIXMAT
        Fm{n}{nRESP,nRI}.Mn{rC,2} = [];% PolyOrd_riMAT
        Fm{n}{nRESP,nRI}.Mn{rC,3} = [];% PolyOrd_coefVEC
    end
end
%--------------------------------------------------------------- init .fmix
for k = 1 : N
    for nRI = 1 : 2
        Fm{n}{nRESP,nRI}.fmix{k,1} = zeros( size( MIX.fmix_RCD{k},1), 2);
    end
end
%-----------------------------------------------------------------
end
end

%% ###################################################### Fm  Fm_
for n = 2 : N
for rC = 1 : size( Mn.PolyOrd_MAT{n},1 )
    nV          = Mn.PolyOrd_MAT{n}(rC,1);
    nU          = Mn.PolyOrd_MAT{n}(rC,2);
%=================================================================
    %iterate all possible nmixRCD/nRI @(current position in VEC)
    %Recurrsive func call bc: VEC is variable length, can't use for-loop
for riV = 1 : -2 : -1
    mixStartV = 1;
    if (riV == -1) && isequal( MIX.fmix_RCD{nV}(1,:), ...
            zeros( size(MIX.fmix_RCD{nV}(1,:)) ) )
        mixStartV = 2;% skip double-counting if DC (sorted to be 1st-pt)
    end
for riU = 1 : -2 : -1
    mixStartU = 1;
    if (riU == -1) && isequal( MIX.fmix_RCD{nU}(1,:), ...
            zeros( size(MIX.fmix_RCD{nU}(1,:)) ) )
        mixStartU = 2;% skip double-counting if DC (sorted to be 1st-pt)
    end
%-----------------------------------------------------------------
for nmixRCDV = mixStartV : size( MIX.fmix_RCD{nV},1 )
for nmixRCDU = mixStartU : size( MIX.fmix_RCD{nU},1 )
    fmixVEC_V   = MIX.fmix_RCD{nV}(nmixRCDV,:);
    fmixVEC_U  = MIX.fmix_RCD{nU}(nmixRCDU,:);
    %---------------------------------------------------
    % calc Result fmix_VEC ~OUTPUT
    fmixVEC         = fmixVEC_V * riV + fmixVEC_U * riU;
    [ nmixRCD  nRI ] = Mix_RCDsearch( fmixVEC, MIX.fmix_RCD{n});
        Fm{n}{ nmixRCD, nRI}.Mn{rC,1} = [Fm{n}{ nmixRCD, nRI}.Mn{rC,1};...
               [nmixRCDV nmixRCDU]];
        Fm{n}{ nmixRCD, nRI}.Mn{rC,2} = [Fm{n}{ nmixRCD, nRI}.Mn{rC,2};...
               [riV        riU]];
        Fm{n}{ nmixRCD, nRI}.fmix{n,1}(nmixRCD, nRI) = 1;
        Fm_{n}( nmixRCD, nRI) = 1;% indicate fIM generated
    %---------------------------------------------------
        if(riV == -1)
            Fm{n}{nmixRCD, nRI}.fmix{nV,1}(nmixRCDV, 2) = 1;
        else
            Fm{n}{nmixRCD, nRI}.fmix{nV,1}(nmixRCDV, 1) = 1;
        end
        if(riU == -1)
            Fm{n}{nmixRCD, nRI}.fmix{nU,1}(nmixRCDU, 2) = 1;
        else
            Fm{n}{nmixRCD, nRI}.fmix{nU,1}(nmixRCDU, 1) = 1;
        end
    %---------------------------------------------------
end
end
%-----------------------------------------------------------------
end
end
%=================================================================
end
end
```

INL_Mult_Condense Function

```
function [ Fm2 ] = INL_Mult_Condense( Fm, Mn, MIX, Opt, ...
    Opt_Disp)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
```

```
end

switch Opt
%% =============================================
    case{'ALL'}
for n    = 1 : length( Fm )
for nmixRCD = 1 : size( Fm{n},1 )
for nRI     = 1 : size( Fm{n},2 )
  if(Opt_Disp)
    disp(sprintf('=========n=%-5d  nmixRCD=%-5d  nRI=%-5d  fmixSYM=%-s',...
    n, nmixRCD, nRI, char(MIX.fmixSYM_RCD{n}(nmixRCD))) );
    Mn.PolyOrd_MAT{n}
  end
%########################################################
for rC = 1 : size( Mn.PolyOrd_MAT{n}, 1 )%size( Fm{n}{nmixRCD, nRI}.Mn, 1 )
    PolyOrd_nMIXMAT     = Fm{n}{nmixRCD, nRI}.Mn{rC,1};
    PolyOrd_riMAT       = Fm{n}{nmixRCD, nRI}.Mn{rC,2};
    if(Opt_Disp)
        disp(['%----------------------------[rC]=' num2str([rC])]);
    end
if(size( PolyOrd_nMIXMAT, 1 ) > 0)
%============================================================rMIX
[PolyOrd_nMIXMAT_2  PolyOrd_riMAT_2  PolyOrd_coefVEC_2 ] =Mult_Condense(...
    PolyOrd_nMIXMAT, PolyOrd_riMAT, Opt_Disp);
Fm2{n}{ nmixRCD,nRI }.Mn{rC,1} = PolyOrd_nMIXMAT_2;
Fm2{n}{ nmixRCD,nRI }.Mn{rC,2} = PolyOrd_riMAT_2;
Fm2{n}{ nmixRCD,nRI }.Mn{rC,3} = PolyOrd_coefVEC_2;
%============================================================
end
    if(Opt_Disp)
        if( size(PolyOrd_nMIXMAT,1) ~= size(PolyOrd_nMIXMAT_2,1) )
            PolyOrd_nMIXMAT
            PolyOrd_riMAT
            disp( sprintf('%################mix = %d reduced to %d', ...
                size(PolyOrd_nMIXMAT,1), size(PolyOrd_nMIXMAT_2,1)) );
            PolyOrd_coefVEC_2
            PolyOrd_nMIXMAT_2
            PolyOrd_riMAT_2
        end
    end
end
%############################################################
end
```

```
end
end
%% =============================================
end

%% ######################################################
function [ PolyOrd_nMIXMAT_2  PolyOrd_riMAT_2  PolyOrd_coefVEC_2 ] = ...
    Mult_Condense( PolyOrd_nMIXMAT, PolyOrd_riMAT, Opt_Disp)

PolyOrd_nMIXMAT_2 = PolyOrd_nMIXMAT(1,:);% Init
PolyOrd_riMAT_2     = PolyOrd_riMAT(1,:);% Init
PolyOrd_coefVEC_2  = [1];% Init

for rMIX = 2 : size( PolyOrd_nMIXMAT, 1 )
    isFOUND = 0;
    %----------------------------------------------------- find in *_2
    for rMIX_2 = 1 : size( PolyOrd_nMIXMAT_2, 1 )
        mixBOOL = (PolyOrd_nMIXMAT(rMIX,:) == PolyOrd_nMIXMAT_2(rMIX_2,:));
        riBOOL  = (PolyOrd_riMAT(rMIX,:)   == PolyOrd_riMAT_2(rMIX_2,:));
        if isempty(find(mixBOOL == 0)) && isempty(find(riBOOL == 0))%all 1s
            isFOUND = 1;
            break;
        end
    end
    %------------------------------------------------------
    if( isFOUND )% IN *_2
        PolyOrd_coefVEC_2( rMIX_2 ) = PolyOrd_coefVEC_2( rMIX_2 ) + 1;
        if( Opt_Disp )
            disp([’%############### [rMIX  rMIX_2]=’ ...
                num2str([rMIX  rMIX_2])]);
        end
    else% NOT IN *_2
        PolyOrd_nMIXMAT_2   = [PolyOrd_nMIXMAT_2; PolyOrd_nMIXMAT(rMIX,:)];
        PolyOrd_riMAT_2     = [PolyOrd_riMAT_2;   PolyOrd_riMAT(rMIX,:)];
        PolyOrd_coefVEC_2  = [PolyOrd_coefVEC_2;   1];
    end
    %-----------------------------------------------------------------
end
```

A.3 Define Circuit

A.3.1 Differential Pair as Single-Balanced Mixer

```
%========================================================== Lin
%Ckt.CharCEL_Ymat    = { 'gL+s*cL'        '0'           '-gT'; ...
%                         '0'             'gL+s*cL'     '-gT'; ...
%                         '0'             '0'           '-2*gT' };
Ckt.CharCEL_Ymat    = { 'G'            '0'           '-gT'; ...
                         '0'             'G'             '-gT'; ...
                         '0'             '0'           '-2*gT' };
Ckt.CharCEL_ILIN = { '-gT*Vi/2'      'gT*Vi/2'       'Is' };
%----------------------------------------------------------
Ckt.CharCEL_Vdep  = { 'Vo1'           'Vo2'           'Ve' };
%--------------------------
Ckt.CharCEL_Vsrc = {'Vi' 'Is'};
Ckt.CharCEL_VsrcTone = {'Vi' 'Is'};
% {A1 A2}->creat voltage spectral (#source <= #tone),
%    {'Vin' 'Vin'} if 2tone in same source
%--------------------------
Ckt.CharCEL_R       = {'gT'};% freqIndep
%--------------------------
Ckt.CharCEL_Z       = {'G'};
% freqDep ('s' 'S' SubSYM/SubNUM as 'k*f1+l*f2', offlimits)
%========================================================== NL
Ckt.INL_types      = {'poly'};
Ckt.INL_Vdep_type = {{'poly'} {'poly'} {'poly' 'poly'}};% {} if no INL
Ckt.INL_Vdep_eo   = {{'eo'}   {'eo'}   {'eo'   'eo'}};
Ckt.INL_Vdep_Sign  = {[-1]     [-1]     [-1     -1]};
Ckt.INL_Vdep_gNL   = {{'gT'}   {'gT'}   {'gT'   'gT'}};% No Sign, only SYM
%----------
Ckt.INL_Vdep        = {{'Vi/2-Ve'} {'-Vi/2-Ve'} {'Vi/2-Ve'  '-Vi/2-Ve'}};
Ckt.INL_Vdep_SymCEL = {{{'Ve'}}    {{'Ve'}}     {{'Ve'}     {'Ve'}}};
Ckt.INL_Vdep_IDX = {{[3]}       {[3]}        {[3]         [3]}};
Ckt.INL_Vsrc_SymCEL = {{{'Vi'}}    {{'Vi'}}     {{'Vi'}     {'Vi'}}};
Ckt.INL_Vsrc_IDX = {{[1]}       {[1]}        {[1]         [1]}};
```

A.3.2 RF Front-End Behavioral Model

```
%========================================================== Lin
Ckt.CharCEL_Ymat    = ...
{ '1/Z1+1/Za''0'       '0'         '0'         '0'         '0'; ...
```

```
  ’A’           ’1/Zh’      ’0’         ’0’         ’0’         ’0’; ...
  ’0’        ’H’        ’-1’        ’0’         ’0’         ’0’; ...
  ’0’        ’0’        ’0’         ’1/Z2+1/Zb’’0’         ’0’; ...
  ’0’        ’0’        ’0’         ’B’         ’1/Zm2’    ’0’; ...
  ’0’        ’0’        ’0’         ’0’         ’0’         ’1’};
Ckt.CharCEL_ILIN = { ’V1/Z1’ ’0’ ’0’ ’V2/Z2’ ’0’ ’0’};
%Ckt.CharCEL_Ymat   = ...
%{ ’G1+Ga’     ’0’       ’0’         ’0’         ’0’         ’0’; ...
%  ’A’       ’Gh’      ’0’         ’0’         ’0’         ’0’; ...
%  ’0’        ’H’        ’-1’        ’0’         ’0’         ’0’; ...
%  ’0’        ’0’        ’0’        ’G2+Gb’     ’0’         ’0’; ...
%  ’0’        ’0’        ’0’         ’B’         ’Gm2’    ’0’; ...
%  ’0’        ’0’        ’0’         ’0’         ’0’         ’1’};
%Ckt.CharCEL_ILIN = { ’V1*G1’ ’0’ ’0’ ’V2*G2’ ’0’ ’0’};
%----------------------------------------------------------
Ckt.CharCEL_Vdep  = { ’Va’ ’Vh’ ’Vm1’ ’Vb’ ’Vm2’ ’Vm3’};
%--------------------------
Ckt.CharCEL_Vsrc = {’V1’ ’V2’};
Ckt.CharCEL_VsrcTone = {’V1’ ’V2’};
%--------------------------
Ckt.CharCEL_R       = {};% freqIndep
%--------------------------
Ckt.CharCEL_Z      = {’Z1’ ’Z2’ ’Za’ ’Zb’ ’Zh’ ’A’ ’B’ ’H’ ’Zm2’};
%Ckt.CharCEL_Z       = {’G1’ ’G2’ ’Ga’ ’Gb’ ’Gh’ ’A’ ’B’ ’H’ ’Gm2’};
%========================================================== NL
Ckt.INL_types      = {’poly’ ’multiply’};
Ckt.INL_Vdep_type = {{’poly’} {’poly’} {} {’poly’} {’poly’} {’multiply’}};
Ckt.INL_Vdep_eo    = {{’eo’}   {’eo’}   {}   {’eo’}   {’eo’}   {’eo’}};
Ckt.INL_Vdep_Sign  = {[-1]     [-1]     []   [-1]     [-1]     [1]};
Ckt.INL_Vdep_gNL   = {{’az’}   {’ag’}   {}   {’bz’}   {’bg’}   {’m’}};
%----------
Ckt.INL_Vdep        = {{’Va’} {’Va’} {} {’Vb’} {’Vb’} {’Vm1’; ’Vm2’}};
Ckt.INL_Vdep_SymCEL = {{{’Va’}} {{’Va’}} {{}} {{’Vb’}} {{’Vb’}} ...
    {{’Vm1’}; {’Vm2’}}};
Ckt.INL_Vdep_IDX = {{[1]} {[1]} {[]} {[4]} {[4]} {[3]; [5]}};
Ckt.INL_Vsrc_SymCEL = {{{}} {{}} {{}} {{}} {{}} {{}; {}}};
Ckt.INL_Vsrc_IDX = {{[]} {[]} {[]} {[]} {[]} {[]; []}};
```

A.4 Circuit Analysis

A.4.1 Main Script

```
%% ############################################### IM response of Interest
%Fresp.fRESP_MAT = [ 0    0; -1    1;  1    1;  2    0;  0    2];% 2nd
Fresp.fRESP_MAT = [ 1    0;  0    1;  2   -1;  2    1; ...
                    -1   2;  1    2;  3    0;  0    3];% 3rd
[ Fresp ]   = Fresp_SelectFMIX( Fresp, 'fRESP_MAT', ...
        Fp2, MIX, Wnr, 1,1);%Opt_Print, Opt_Disp

%% ############################################### Ckt
Ckt.N = N;
Ckt.Q = Q;
[ Ckt ]     = Ckt_Gen2( Ckt, Fresp, MIX, 1);
%------------------------------
[ Ckt ]     = Ckt_SolnLin3( Ckt, Fresp, MIX, 1);
%------------------------------
[ Ckt ]     = Ckt_SolnNL4( Ckt, Fresp,    MIX, Wnr, Mn, Fp2, Fm2, 1 );
%------------------------------
[ Ckt ]     = Ckt_Soln2( Ckt, Fresp, 1 );
```

A.4.2 Functions

Ckt_Gen2 Function

```
function [ Ckt ] = Ckt_Gen2( Ckt, Fresp, MIX, ...
    Opt_Disp)
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

Ckt.NumV = length( Ckt.CharCEL_Vdep );
%% ###########################
%CharCEL => SYM_[Ymat/ILIN/  INLvdep],  SYM_[R/gNL/  zBD.(Z)],  SYM_[A]
%========================================================== Lin
%[ Ckt.SYM_Vdep ] = CHARcel2SYMmat( Ckt.CharCEL_Vdep ).';
%[ Ckt.SYM_Vsrc ] = CHARcel2SYMmat( Ckt.CharCEL_Vsrc ).';
%------------------------------ usable as SymCEL
[ Ckt.SYM_Ymat ]    = CHARcel2SYMmat( Ckt.CharCEL_Ymat );
[ Ckt.SYM_ILIN ]    = CHARcel2SYMmat( Ckt.CharCEL_ILIN ).';
%------------------------- SYM_R
```

```
[ Ckt.SYM_R ]       = mat2cell( CHARcel2SYMmat( Ckt.CharCEL_R ), ...
                                ones(1,size(Ckt.CharCEL_R,1)), ...
                                ones(1,size(Ckt.CharCEL_R,2)) );
%------------------------- SYM_ZBD
for nz = 1 : length( Ckt.CharCEL_Z )
    Z = Ckt.CharCEL_Z{ nz };
for nBD = 1 : length( MIX.fmix_BD )
for nmixBD = 1 : size( MIX.fmix_BD{nBD},1 )
    Ckt.SYM_ZBD.( Z ){nBD}(nmixBD,:) = ...
        { sym([Z  MIX.fmixCHAR_BD{nBD}{nmixBD,1}]) ...
       sym([Z  MIX.fmixCHAR_BD{nBD}{nmixBD,2}]) };
end
end
end
%============================================================ NL
%------------------------- SYM_INLvdep
for nV = 1 : Ckt.NumV
for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
for k = 1 : size( Ckt.INL_Vdep{nV}, 1 )
if ~isempty( Ckt.INL_Vdep{nV}{k,nL} )
    Ckt.SYM_INLvdep{nV}{k,nL} = sym( Ckt.INL_Vdep{nV}{k,nL} );
else
    Ckt.SYM_INLvdep{nV}{k,nL} = [];% sym( [] )
end
end
end
end
%------------------------- SYM_gNL
    Ckt.SYM_gNL = struct;
for nV = 1 : Ckt.NumV
for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
    gNL = Ckt.INL_Vdep_gNL{nV}{nL};
if ~isfield( Ckt.SYM_gNL, gNL )
switch Ckt.INL_Vdep_type{nV}{nL}
%-------------------------------------------
case{'multiply'}% mixer
        Ckt.SYM_gNL.( gNL )(2) = sym( [gNL] );
%-------------------------------------------
case{'poly','capNL','indNL'}% gm, capNL, indNL
        for r = 2 : MIX.N
            switch Ckt.INL_Vdep_eo{nV}{nL}
                case{'eo'}
                    Ckt.SYM_gNL.( gNL )(r) =sym([gNL '_' num2str(r)]);
                case{’e’}
                    if IsEven_Func( r )
                        Ckt.SYM_gNL.( gNL )(r) =sym([gNL ’_’ num2str(r)]);
                    end
                case{’o’}
                    if ~IsEven_Func( r )
                        Ckt.SYM_gNL.( gNL )(r) =sym([gNL ’_’ num2str(r)]);
                    end
            end
        end
%--------------------------------------------
end
end
end
end
% SYM_gNL_FLAG
%========================================================== Mtone Source
for q = 1 : Ckt.Q
    Ckt.SYM_A(q,:) = { sym([’A’ num2str(q)]) sym([’A’ num2str(q) ’_’]) };
end

%% ########################### Soln.( Vdep, Vsrc )
if(Opt_Disp)
    disp([ ’%########################### Ckt.Soln( Vdep )’ ])
end
    Vdep1                = Ckt.CharCEL_Vdep{1};
    Ckt.Soln.( Vdep1 ) = Soln_Init( Fresp );
for nV = 2 : Ckt.NumV
    Vdep                 = Ckt.CharCEL_Vdep{ nV };
    Ckt.Soln.( Vdep ) = Ckt.Soln.( Vdep1 );% Soln_Init( Fresp )
end
for nSRC = 1 : length( Ckt.CharCEL_Vsrc )
    Vsrc                 = Ckt.CharCEL_Vsrc{ nSRC };
    Ckt.Soln.( Vsrc ) = Ckt.Soln.( Vdep1 );% Soln_Init( Fresp )
end

if(Opt_Disp)
    disp([ ’%########################### Ckt.Soln.( Vsrc )’ ])
end
for n = 1 : 1
for nmixRCD = 1 : length( Ckt.CharCEL_VsrcTone )
    %--------------------------
    [ nSRC xXORy_IDX yVEC_IDX yXORy_IDX] = Find_VecIDX_Func(...
            Ckt.CharCEL_Vsrc, ...
            Ckt.CharCEL_VsrcTone( nmixRCD ), 'VECTOR', [], 0);
    Vsrc = Ckt.CharCEL_Vsrc{ nSRC };
    Ckt.Soln.( Vsrc ){n}{nmixRCD,1} = Ckt.SYM_A{nmixRCD,1};% / 2
    Ckt.Soln.( Vsrc ){n}{nmixRCD,2} = Ckt.SYM_A{nmixRCD,2};% / 2
    %-------------------------
    if(Opt_Disp)
        disp([Vsrc '{' num2str(n) '}{' num2str(nmixRCD) ':}' ])
        Ckt.Soln.( Vsrc ){n}{nmixRCD,:}% DEBUG
    end
    %-------------------------
end
end

if(Opt_Disp)
    Ckt.Soln% DEBUG
end

%% ***************************************************
function [ V ] = Soln_Init( Fresp )

for n = 1 : length( Fresp.fmixFLAG_RCD )
for nmixRCD = 1 : size( Fresp.fmixFLAG_RCD{n},1 )
for nRI = 1 : 2% length( riVEC )
        V{n}{nmixRCD,nRI} = sym('0');
end
end
end
```

Ckt_SolnLin3 Function

```
function [ Ckt ] = Ckt_SolnLin3( Ckt, Fresp, MIX, ...
    Opt_Disp )
riFCT  = [1 -1];

%% ########################### Lin-Soln: Ckt.Soln.( Vdep ){n}{nmixRCD,nRI}
n = 1;
for nmixRCD = 1 : size( Fresp.fmixFLAG_RCD{n},1 )
    if(Opt_Disp)
        disp(['================ [nmixRCD n]=' num2str([nmixRCD  n])]);
    end
for nRI = 1 : 2
    if(Opt_Disp) disp(['-------- nRI=' num2str(nRI)]);end
```

```
%================================================ Lin Real/Imag resp
    nBD     = MIX.IdxBD_RCD{n}(nmixRCD,1);
    nmixBD = MIX.IdxBD_RCD{n}(nmixRCD,2);
Z_CEL   = {};
    for nz = 1 : length( Ckt.CharCEL_Z )
        Z               = Ckt.CharCEL_Z{ nz };
        Z_CEL{ nz }     = Ckt.SYM_ZBD.( Z ){nBD}{nmixBD,nRI};
    end
Vsrc_CEL = {};
    for nSRC = 1 : length( Ckt.CharCEL_Vsrc )
        Vsrc            = Ckt.CharCEL_Vsrc{ nSRC };
        Vsrc_CEL{ nSRC } = Ckt.Soln.( Vsrc ){n}{nmixRCD,nRI};
    end
%----------------------
    %{Ckt.CharCEL_Z{:}}
Ymat = subs( Ckt.SYM_Ymat, ...
        {Ckt.CharCEL_Z{:}  's'}, ...
        {Z_CEL{:}   j*2*pi * riFCT(nRI) * MIX.fmixSYM_RCD{n}(nmixRCD))} );
%----------------------
    %Ckt.SYM_ILIN
    %{ Ckt.CharCEL_Vsrc{:} }
    %[ Vsrc_CEL{:}         ]
Ilin_VEC = subs( Ckt.SYM_ILIN, ...
        {Ckt.CharCEL_Vsrc{:} Ckt.CharCEL_Z{:} 's'}, ...
        {Vsrc_CEL{:}            Z_CEL{:}          ...
            (j*2*pi * riFCT(nRI) * MIX.fmixSYM_RCD{n}(nmixRCD))} )
V_VEC = Ymat \ Ilin_VEC% V_VEC * 2, Ilin_VEC / 2
    for nV = 1 : Ckt.NumV
        Vdep = Ckt.CharCEL_Vdep{ nV };
        Ckt.Soln.( Vdep ){n}{nmixRCD,nRI} = V_VEC(nV);
    end
%=================================================
end
end
```

Ckt_SolnNL4 Function

```
function [ Ckt ] = Ckt_SolnNL4( Ckt, Fresp, ...
    MIX, Wnr, Mn, Fp2, Fm2, ...
    Opt_Disp )
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end
```

```
%=========================================== Output
Fresp.fmixFLAG_RCD3 = {};% indicat path traveled {n,1}{nmixRCD,:}=[R I]
    Fresp.fmixFLAG_RCD3{1} = Fresp.fmixFLAG_RCD{1};%LinSoln computed
    for n = 2 : MIX.N
        Fresp.fmixFLAG_RCD3{n} = zeros(size( MIX.fmix_RCD{n} ));%Init
    end

%% ########################### NL-Soln:    Ckt.Soln.( Vdep ){n}{nRESP,nRI}
for nRESP = 1 : Fresp.NumResp
if(Opt_Disp) ...
        disp(['======================= [nRESP]=' num2str([nRESP])]);
    end
%===========================================
for nn = 1 : size( Fresp.IdxRCD_fRESP{ nRESP },1 )
    n       = Fresp.IdxRCD_fRESP{ nRESP }(nn,1);
    nmixRCD = Fresp.IdxRCD_fRESP{ nRESP }(nn,2);
    nRI     = 1;% for measured
if(Opt_Disp) disp(['------------------[nRESP n]=' num2str([nRESP  n])' ...
          ' char( Fresp.fmixSYM_fRESP{ nRESP }(nn) )]);end
    %---------------------- Recur given (n, nmixRCD, nRI)
    if(Fresp.fmixFLAG_RCD3{n}(nmixRCD, nRI) == 0)% if path not traveled
        [ Ckt Fresp ] = Recur( n, nmixRCD, nRI, ...
            Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, Opt_Disp);
    end
%===========================================
end
end

%% ################################################# Fp2{n}{nmixRCD, nRI}
function [ Ckt Fresp ] = Recur( n, nmixRCD, nRI, ...
    Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, ...
    Opt_Disp)
if(Opt_Disp)  disp(['%###################[n, nmixRCD, nRI] ' ...
    num2str([n, nmixRCD, nRI])]); end
riFCT  = [1 -1];

if(n == 1)
%=================================================================
    % LinSoln computed already
%=================================================================
else%(n > 1)
%#################################################### calc INL_VEC
```

```
INL_VEC  = sym(zeros( Ckt.NumV,1 ));% (nV)
FlagPOLY = 0;%set once bc FuncCall find INL(gm, capNL, indNL) simultaneous
for nTYPE = 1 : length( Ckt.INL_types )
switch Ckt.INL_types{nTYPE}
        %-------------------------------------------
        case{'poly','capNL','indNL'}% gm, capNL, indNL
        if( FlagPOLY == 0 )
            [ INLpoly_VEC  INLcap_VEC  INLind_VEC ] = Poly( ...
                n, nmixRCD, nRI, ...
                Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, 1);
            INL_VEC     = INL_VEC + INLpoly_VEC + INLcap_VEC + INLind_VEC ;
            FlagPOLY = 1;
            %set once bc FuncCall find INL(gm, capNL, indNL) simultaneously
        end
        %-------------------------------------------
        case{'multiply'}% mixer
            [ INLmult_VEC ] = Multiply( n, nmixRCD, nRI, ...
                Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, 1);
            INL_VEC     = INL_VEC + INLmult_VEC;
        %-------------------------------------------
        otherwise
            disp('unknown Ckt.INL_types');
    end
end
%##################################################### calc V_VEC
    nBD     = MIX.IdxBD_RCD{n}(nmixRCD,1);
    nmixBD = MIX.IdxBD_RCD{n}(nmixRCD,2);
Z_CEL = {};
    for nz = 1 : length( Ckt.CharCEL_Z )
        Z         = Ckt.CharCEL_Z{ nz };
        Z_CEL{ nz } = Ckt.SYM_ZBD.( Z ){nBD}{nmixBD,nRI};
    end
Ymat = subs( Ckt.SYM_Ymat, ...
        {Ckt.CharCEL_Z{:}   's'}, ...
        {Z_CEL{:}  j*2*pi * riFCT(nRI) * MIX.fmixSYM_RCD{n}(nmixRCD)} );
V_VEC = Ymat \ INL_VEC;
    for nV = 1 : Ckt.NumV
        Vdep        = Ckt.CharCEL_Vdep{ nV };
        Ckt.Soln.( Vdep ){n}{nmixRCD,nRI} = V_VEC(nV);
    end
%#####################################################
%==================================================================
end
```

```
    Fresp.fmixFLAG_RCD3{n}(nmixRCD, nRI) = 1;% indicate path was traveled
return


%% ######################################################
function [ INLmult_VEC ] = Multiply( n, nmixRCD, nRI, ...
    Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, ...
    Opt_Disp)
INLmult_VEC = sym(zeros( Ckt.NumV,1 ));% (nV)

%#############################################################
r = 2;
  Vr_VecCEL    = Vrc_Init( 0, 1, Ckt.INL_Vdep );% {1,nV}{nL}
    C_r       = 1 / 2^(r-1);
%======================================================== OrdPwr_MAT, c_VEC
  Vc_MatCEL     = Vrc_Init( 0, size( Mn.PolyOrd_MAT{n}, 1 ), ...
        Ckt.INL_Vdep );% {rC,nV}{nL}
for rC = 1 : size( Mn.PolyOrd_MAT{n}, 1 )
    if(Opt_Disp)  disp(['%----------------------[rC] ' num2str([rC])]); end
PolyOrd_VEC         = Mn.PolyOrd_MAT{n}(rC,:);
    PolyOrd_nMIXMAT     = Fm2{n}{ nmixRCD,nRI }.Mn{rC,1};
    PolyOrd_riMAT       = Fm2{n}{ nmixRCD,nRI }.Mn{rC,2};
    PolyOrd_coefVEC     = Fm2{n}{ nmixRCD,nRI }.Mn{rC,3};
    if(Opt_Disp)
        PolyOrd_VEC% DEBUG
        PolyOrd_nMIXMAT% DEBUG
        PolyOrd_riMAT% DEBUG
        PolyOrd_coefVEC% DEBUG
    end
%------------------------------------------------------------ rMIX
  Vrc_MatCEL    = Vrc_Init( 1, size( PolyOrd_nMIXMAT,1), Ckt.INL_Vdep );
for rMIX = 1 : size( PolyOrd_nMIXMAT,1)
    if(Opt_Disp)  disp(['%----------------[rMIX] ' num2str([rMIX])]); end
C_rMIX = PolyOrd_coefVEC(   rMIX);
for k = 1 : length( PolyOrd_VEC )
    if(Opt_Disp)  disp(['%----------[k] ' num2str([k])]); end
Ord_k = PolyOrd_VEC(           k);
    Mix_k = PolyOrd_nMIXMAT( rMIX,k);
    ri_k    = PolyOrd_riMAT(    rMIX,k);
        if(ri_k == -1)
            nRI_k = 2;
        else
            nRI_k = 1;
        end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ calc Vdep (must call bc NonEmpty iNL)
if( Fresp.fmixFLAG_RCD3{Ord_k}(Mix_k, nRI_k) == 0)% if path not traveled
    [ Ckt Fresp ] = Recur( Ord_k, Mix_k, nRI_k, ...
        Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, Opt_Disp);
end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ calc INL
for nV = 1 : Ckt.NumV
for nL = 1 : length( Ckt.INL_Vdep_type{nV} )% effectiv take care of empty
switch Ckt.INL_Vdep_type{nV}{nL}
%--------------------------------------------
case{'multiply'}% mixer
    Vdep_CEL = {};
    Vsrc_CEL = {};
    for nIDX = 1 : length( Ckt.INL_Vdep_SymCEL{nV}{k,nL} )
        Vdep_IDX        = Ckt.INL_Vdep_IDX{nV}{k,nL}(nIDX);
        Vdep            = Ckt.CharCEL_Vdep{ Vdep_IDX };
        Vdep_CEL{nIDX} = Ckt.Soln.( Vdep ){Ord_k}{Mix_k,nRI_k};
    end
    for nIDX = 1 : length( Ckt.INL_Vsrc_SymCEL{nV}{k,nL} )
        Vsrc_IDX        = Ckt.INL_Vsrc_IDX{nV}{k,nL}(nIDX);
        Vsrc            = Ckt.CharCEL_Vsrc{ Vsrc_IDX };
        Vsrc_CEL(nIDX) = Ckt.Soln.( Vsrc ){Ord_k}{Mix_k,nRI_k};
    end
    if(Opt_Disp)
        %[ Vdep_CEL{:} ]% DEBUG
        %[ Vsrc_CEL{:} ]% DEBUG
    end
    Vtemp = subs( Ckt.SYM_INLvdep{nV}{k,nL}, ...
        {Ckt.INL_Vdep_SymCEL{nV}{k,nL}{:} ...
            Ckt.INL_Vsrc_SymCEL{nV}{k,nL}{:}}, ...
        {Vdep_CEL{:}                        Vsrc_CEL{:}} );
%Ckt.SYM_INLvdep{nV}{k,nL}% DEBUG subs
%Vtemp% DEBUG subs
        Vrc_MatCEL{rMIX,nV}{nL} = Vrc_MatCEL{rMIX,nV}{nL}     * Vtemp;
%--------------------------------------------
end
end
end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
end%----------------------[k]  Vrc_MatCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
```

```
        switch Ckt.INL_Vdep_type{nV}{nL}
            case{'multiply'}% mixer
                Vc_MatCEL{rC,nV}{nL} = Vc_MatCEL{rC,nV}{nL}...
                     + C_rMIX * Vrc_MatCEL{rMIX,nV}{nL};
        end
    end
    end
end%------------------------------------------- [rMIX]     Vc_MatCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
        switch Ckt.INL_Vdep_type{nV}{nL}
            case{'multiply'}% mixer
                Vr_VecCEL{1,nV}{nL}     = Vr_VecCEL{1,nV}{nL}...
                     + Vc_MatCEL{rC,nV}{nL};
        end
    end
    end
end%============================================= [rC]  Vr_VecCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
    switch Ckt.INL_Vdep_type{nV}{nL}
        %--------------------------------------------
        case{'multiply'}% mixer
            Sign    = Ckt.INL_Vdep_Sign{nV}(nL);
            gNL  = Ckt.INL_Vdep_gNL{nV}{nL};
            gNL_r   = Ckt.SYM_gNL.( gNL )(r);
            if(0)%disp('OK')
                %Ir_VEC(nV)
                %Vr_VecCEL{1,nV}{nL}
            end
            INLmult_VEC(nV)     = INLmult_VEC(nV)...
                + Sign * C_r * gNL_r  * Vr_VecCEL{1,nV}{nL};
        %--------------------------------------------
    end
    end
    end
%########################################### r = 2  Ir_VEC --> INLpoly_VEC
return


%% ######################################################
function [ INLpoly_VEC  INLcap_VEC  INLind_VEC ] = Poly( ...
    n, nmixRCD, nRI, ...
    Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, ...
    Opt_Disp)
INLpoly_VEC = sym(zeros( Ckt.NumV,1 ));% (nV)
INLcap_VEC = sym(zeros( Ckt.NumV,1 ));% (nV)
INLind_VEC = sym(zeros( Ckt.NumV,1 ));% (nV)
riFCT       = [1 -1];

%#############################################################
for r = 2 : length( Fp2{n}{nmixRCD, nRI}.wnr )
    if(Opt_Disp)  disp(['%===================[r] ' num2str([r])]); end
  Ir_VEC       = sym(zeros( Ckt.NumV,1 ));% (nV)
  Ir_VEC_CAP    = Ir_VEC;
  Ir_VEC_IND    = Ir_VEC;
  Vr_VecCEL    = Vrc_Init( 0, 1, Ckt.INL_Vdep );% {1,nV}{nL}
  Vr_VecCEL_CAP = Vr_VecCEL;
  Vr_VecCEL_IND = Vr_VecCEL;
%=============================================== OrdPwr_MAT, c_VEC
  Vc_MatCEL     = Vrc_Init( 0, size( Wnr.PolyOrd_MAT{n,r}, 1 ), ...
        Ckt.INL_Vdep );% {rC,nV}{nL}
  Vc_MatCEL_CAP = Vc_MatCEL;
  Vc_MatCEL_IND = Vc_MatCEL;
for rC = 1 : size( Wnr.PolyOrd_MAT{n,r}, 1 )
    if(Opt_Disp)  disp(['%-------------------[rC] ' num2str([rC])]); end
    C_rC              = Wnr.c_VEC{n,r}(rC) / 2^(r-1);
PolyOrd_VEC         = Wnr.PolyOrd_MAT{n,r}(rC,:);
    PolyOrd_nMIXMAT     = Fp2{n}{ nmixRCD,nRI }.wnr{r}{rC,1};
    PolyOrd_riMAT       = Fp2{n}{ nmixRCD,nRI }.wnr{r}{rC,2};
    PolyOrd_coefVEC     = Fp2{n}{ nmixRCD,nRI }.wnr{r}{rC,3};
    if(Opt_Disp)
        PolyOrd_VEC% DEBUG
        PolyOrd_nMIXMAT% DEBUG
        PolyOrd_riMAT% DEBUG
        PolyOrd_coefVEC% DEBUG
    end
%------------------------------------------------------------ rMIX
  Vrc_MatCEL    = Vrc_Init( 1, size( PolyOrd_nMIXMAT,1),...
     Ckt.INL_Vdep );% {rMIX,nV}{nL}
  Vrc_MatCEL_CAP= Vrc_MatCEL;
  Vrc_MatCEL_IND= Vrc_MatCEL;
for rMIX = 1 : size( PolyOrd_nMIXMAT,1)
    if(Opt_Disp)
        disp(['%----------------------[rMIX] ' num2str([rMIX])]);
    end
```

```
C_rMIX = PolyOrd_coefVEC(   rMIX);
for k = 1 : length( PolyOrd_VEC )
    if(Opt_Disp)  disp(['%--------------[k] ' num2str([k])]); end
Ord_k = PolyOrd_VEC(           k);
    Mix_k = PolyOrd_nMIXMAT( rMIX,k);
    ri_k    = PolyOrd_riMAT(    rMIX,k);
        if(ri_k == -1)
            nRI_k = 2;
        else
            nRI_k = 1;
        end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ calc Vdep (must call bc NonEmpty iNL)
if( Fresp.fmixFLAG_RCD3{Ord_k}(Mix_k, nRI_k) == 0)% if path not traveled
    [ Ckt Fresp ] = Recur( Ord_k, Mix_k, nRI_k, ...
        Ckt, Fresp, MIX, Wnr, Mn, Fp2, Fm2, Opt_Disp);
end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ calc INL
for nV = 1 : Ckt.NumV
for nL = 1 : length( Ckt.INL_Vdep_type{nV} )% effectiv take care of empty
switch Ckt.INL_Vdep_type{nV}{nL}
%------------------------------------------
case{'poly','capNL','indNL'}% gm, capNL, indNL
    Vdep_CEL = {};
    Vsrc_CEL = {};
    for nIDX = 1 : length( Ckt.INL_Vdep_SymCEL{nV}{1,nL} )
        Vdep_IDX        = Ckt.INL_Vdep_IDX{nV}{1,nL}(nIDX);
        Vdep            = Ckt.CharCEL_Vdep{ Vdep_IDX };
        Vdep_CEL{nIDX} = Ckt.Soln.( Vdep ){Ord_k}{Mix_k,nRI_k};
    end
    for nIDX = 1 : length( Ckt.INL_Vsrc_SymCEL{nV}{1,nL} )
        Vsrc_IDX        = Ckt.INL_Vsrc_IDX{nV}{1,nL}(nIDX);
        Vsrc            = Ckt.CharCEL_Vsrc{ Vsrc_IDX };
        Vsrc_CEL(nIDX) = Ckt.Soln.( Vsrc ){Ord_k}{Mix_k,nRI_k};
    end
    if(Opt_Disp)
        %[ Vdep_CEL{:} ]% DEBUG
        %[ Vsrc_CEL{:} ]% DEBUG
    end
    Vtemp = subs( Ckt.SYM_INLvdep{nV}{1,nL}, ...
        {Ckt.INL_Vdep_SymCEL{nV}{1,nL}{:} ...
            Ckt.INL_Vsrc_SymCEL{nV}{1,nL}{:}},...
        {Vdep_CEL{:}                        Vsrc_CEL{:}} );
        switch Ckt.INL_Vdep_type{nV}{nL}
```

```
            case{’poly’}
                Vrc_MatCEL{rMIX,nV}{nL}    =Vrc_MatCEL{rMIX,nV}{nL}    ...
                    * Vtemp;
            case{’capNL’}
                Vrc_MatCEL_CAP{rMIX,nV}{nL}=Vrc_MatCEL_CAP{rMIX,nV}{nL}...
                    * Vtemp;
            case{’indNL’}
                Vrc_MatCEL_IND{rMIX,nV}{nL}=Vrc_MatCEL_IND{rMIX,nV}{nL}...
                    * (Vtemp(j*2*pi*ri_k*MIX.fmixSYM_RCD{Ord_k}(Mix_k)));
        end
%--------------------------------------------
end
end
end
%@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
end%----------------------[k]                           Vrc_MatCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
        switch Ckt.INL_Vdep_type{nV}{nL}
            case{’poly’}
                Vc_MatCEL{rC,nV}{nL}     = Vc_MatCEL{rC,nV}{nL}     ...
                    + C_rMIX * Vrc_MatCEL{rMIX,nV}{nL};
            case{’capNL’}
                Vc_MatCEL_CAP{rC,nV}{nL} = Vc_MatCEL_CAP{rC,nV}{nL} ...
                    + C_rMIX * Vrc_MatCEL_CAP{rMIX,nV}{nL};
            case{’indNL’}
                Vc_MatCEL_IND{rC,nV}{nL} = Vc_MatCEL_IND{rC,nV}{nL} ...
                    + C_rMIX * Vrc_MatCEL_IND{rMIX,nV}{nL};
        end
    end
    end
end%---------------------------------------------- [rMIX]     Vc_MatCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
        switch Ckt.INL_Vdep_type{nV}{nL}
            case{’poly’}
                Vc_MatCEL{rC,nV}{nL}     = Vc_MatCEL{rC,nV}{nL}     * C_rC;
            case{’capNL’}
                Vc_MatCEL_CAP{rC,nV}{nL} = Vc_MatCEL_CAP{rC,nV}{nL} * C_rC;
            case{’indNL’}
                Vc_MatCEL_IND{rC,nV}{nL} = Vc_MatCEL_IND{rC,nV}{nL} * C_rC;
        end
    end
    end
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
        switch Ckt.INL_Vdep_type{nV}{nL}
            case{'poly'}
                Vr_VecCEL{1,nV}{nL}     = Vr_VecCEL{1,nV}{nL}     ...
                    + Vc_MatCEL{rC,nV}{nL};
            case{'capNL'}
                Vr_VecCEL_CAP{1,nV}{nL} = Vr_VecCEL_CAP{1,nV}{nL} ...
                    + Vc_MatCEL_CAP{rC,nV}{nL};
            case{'indNL'}
                Vr_VecCEL_IND{1,nV}{nL} = Vr_VecCEL_IND{1,nV}{nL} ...
                    + Vc_MatCEL_IND{rC,nV}{nL};
        end
    end
    end
end%========================================== [rC]       Vr_VecCEL
    for nV = 1 : Ckt.NumV
    for nL = 1 : length( Ckt.INL_Vdep_type{nV} )
    switch Ckt.INL_Vdep_type{nV}{nL}
        %---------------------------------------------
        case{'poly','capNL','indNL'}% gm, capNL, indNL
            Sign    = Ckt.INL_Vdep_Sign{nV}(nL);
            gNL  = Ckt.INL_Vdep_gNL{nV}{nL};
            gNL_r   = Ckt.SYM_gNL.( gNL )(r);
            if(0)%disp('OK')
                %Ir_VEC(nV)
                %Vr_VecCEL{1,nV}{nL}
            end
            switch Ckt.INL_Vdep_type{nV}{nL}
                case{'poly'}
                    Ir_VEC(nV)      = Ir_VEC(nV)    ...
                        + Sign * gNL_r  * Vr_VecCEL{1,nV}{nL};
                case{'capNL'}
                    Ir_VEC_CAP(nV) = Ir_VEC_CAP(nV) ...
                        + Sign * gNL_r  /r * Vr_VecCEL_CAP{1,nV}{nL};
                case{'indNL'}
                    Ir_VEC_IND(nV) = Ir_VEC_IND(nV) ...
                        + Sign * gNL_r     * Vr_VecCEL_IND{1,nV}{nL};
                otherwise
                    disp('unknown Ckt.INL_Vdep_type');
            end
        %---------------------------------------------
    end
    end
    end
    INLpoly_VEC = INLpoly_VEC   + Ir_VEC;
    INLcap_VEC  = INLcap_VEC    + Ir_VEC_CAP;
    INLind_VEC  = INLind_VEC    + Ir_VEC_IND;
end%###################################### [r]     Ir_VEC --> INLpoly_VEC
    INLcap_VEC  = INLcap_VEC * (j*2*pi * riFCT(nRI) ...
        * MIX.fmixSYM_RCD{n}(nmixRCD));
return


%% ###########################
function [ Vrc_MatCEL ] = Vrc_Init( const, NumMIX, INL_Vdep )

for rMIX = 1 : NumMIX
for nV = 1 : length( INL_Vdep )
for nL = 1 : size( INL_Vdep{nV}, 2 )
    Vrc_MatCEL{rMIX,nV}{nL} = sym( num2str(const) );
end
end
end
return
```

Ckt_Soln2 Function

```
function [ Ckt ] = Ckt_Soln2( Ckt, Fresp, ...
    Opt_Disp )
if ~exist('Opt_Disp','var')
    Opt_Disp = 0;
end

%% ######## Soln.( Vdep ){n}{nRESP,ri} => Soln2_[fSEPAR].( Vdep ){nRESP,1}
for nRESP = 1 : Fresp.NumResp
    Vresp_VEC  = sym( zeros([Ckt.NumV,1]) );
if(Opt_Disp) disp(['===================== [nRESP]=' num2str([nRESP])]);end
%===========================================
for nn = 1 : size( Fresp.IdxRCD_fRESP{ nRESP },1 )
    n       = Fresp.IdxRCD_fRESP{ nRESP }(nn,1);
    nmixRCD = Fresp.IdxRCD_fRESP{ nRESP }(nn,2);
    nRI     = 1;% for measured
if(Opt_Disp) disp(['------------------ [n  fmixSYM]=' num2str([n]) '...
      ' char( Fresp.fmixSYM_fRESP{ nRESP }(nn) )]);end
    %---------------------------------------------------- Soln2
    for nV = 1 : Ckt.NumV
        Vdep            = Ckt.CharCEL_Vdep{ nV };
        Vresp_VEC(nV) = Vresp_VEC(nV) + Ckt.Soln.( Vdep ){n}{nmixRCD,nRI};
        Vresp_nth(nV)  =                 Ckt.Soln.( Vdep ){n}{nmixRCD,nRI};
    end
end
%============================================
    for nV = 1 : Ckt.NumV
        Vdep        = Ckt.CharCEL_Vdep{ nV };
        Ckt.Soln2.( Vdep ){nRESP,1} = Vresp_VEC(nV);
    end
end
```