```
%% Simulation loop start
warning('*** Simulation Loop Start')
while (kt<=ktmax)
    k_start = kt+1;

    if kt==ktmax
        k_end = kt + k_inc(ks);
    else
        k_end = kt + k_inc(ks) + 1;
    end

    for k = k_start:k_end
        %% step 3a: network solution

        % display k and t at k_inc and every ...th step - thad
        if ( mod(k,50)==0 ) || k == 1 || k == k_end
            fprintf('*** k = %5d, \tt(k) = %7.4f\n',k,t(k)) % DEBUG
        end

        % mach_ref(k) = mac_ang(syn_ref,k);
        mach_ref(k) = 0;
        pmech(:,k+1) = pmech(:,k);
        tmig(:,k+1) = tmig(:,k);

        if n_conv~=0
            cur_ord(:,k+1) = cur_ord(:,k);
        end

        % Trip gen - Copied from v2.3 06/01/20 - thad
        [f,mac_trip_states] = mac_trip_logic(mac_trip_flags,mac_trip_states,t,k);
        mac_trip_flags = mac_trip_flags | f;

        %% Flag = 1
        flag = 1;
        timestep = int2str(k);
        % network-machine interface
        mac_ind(0,k,bus_sim,flag);
        mac_igen(0,k,bus_sim,flag);
        mac_sub(0,k,bus_sim,flag);
        mac_tra(0,k,bus_sim,flag);
        mac_em(0,k,bus_sim,flag);
        mdc_sig(t(k),k);
        dc_cont(0,k,10*(k-1)+1,bus_sim,flag);

        %% Calculate current injections and bus voltages and angles
        if k >= sum(k_inc(1:3))+1
            %% fault cleared
            line_sim = line_pf2;
            bus_sim = bus_pf2;
            bus_int = bus_intpf2;
            Y1 = Y_gpf2;
            Y2 = Y_gncpf2;
            Y3 = Y_ncgpf2;
            Y4 = Y_ncpf2;
            Vr1 = V_rgpf2;
            Vr2 = V_rncpf2;
            bo = bopf2;
            % i_simu forms the network interface variables
            if k == 50 % DEBUG - showing of networ solution call
                warning('*** Performing network solution via i_simu')
            end
            %h_sol = i_simu(k,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
            % duplicate call?
            % h_sol calculated after this 'if' block...

        elseif k >=sum(k_inc(1:2))+1
            %% near bus cleared
            line_sim = line_pf1;
            bus_sim = bus_pf1;
            bus_int = bus_intpf1;
            Y1 = Y_gpf1;
            Y2 = Y_gncpf1;
            Y3 = Y_ncgpf1;
            Y4 = Y_ncpf1;
            Vr1 = V_rgpf1;
            Vr2 = V_rncpf1;
            bo = bopf1;

            %h_sol = i_simu(k,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);

        elseif k>=k_inc(1)+1
            %% fault applied
            line_sim = line_f;
            bus_sim = bus_f;
            bus_int = bus_intf;
            Y1 = Y_gf;
            Y2 = Y_gncf;
            Y3 = Y_ncgf;
            Y4 = Y_ncf;
            Vr1 = V_rgf;
            Vr2 = V_rncf;
            bo = bof;

            %h_sol = i_simu(k,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);

        elseif k<k_inc(1)+1
```

```
            %% pre fault
            line_sim = line;
            bus_sim = bus;
            bus_int = bus_intprf;
            Y1 = Y_gprf;
            Y2 = Y_gncprf;
            Y3 = Y_ncgprf;
            Y4 = Y_ncprf;
            Vr1 = V_rgprf;
            Vr2 = V_rncprf;
            bo = boprf;

            %h_sol = i_simu(k,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
        end

        %% apply gen trip - added from v2.3 - 06/01/20 - thad
        if sum(mac_trip_flags)>0.5
            genBuses = mac_con(mac_trip_flags==1,2);
            for kB=1:length(genBuses)
                nL = find(genBuses(kB)==line_sim(:,1) | genBuses(kB)==line_sim(:,2));
                if isempty(nL); error(' '); end
                line_sim(nL,4) = 1e7; %make reactance infinity
            end
            [Y1,Y2,Y3,Y4,Vr1,Vr2,bo] = red_ybus(bus_sim,line_sim);
            clear nL kB genBuses
        end

        %% solve
        if k == 50 % DEBUG - showing of network solution call
            warning('*** k == 50; Performing network solution via i_simu')
        end
        h_sol = i_simu(k,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);

        %% HVDC
        if ndcr_ud~=0
            % calculate the new value of bus angles rectifier user defined control
            tot_states=0;
            for jj = 1:ndcr_ud
                b_num1 = dcr_dc{jj,3};
                b_num2 = dcr_dc{jj,4};
                conv_num = dcr_dc{jj,2};
                angdcr(jj,k) = (theta(bus_int(b_num1),k)-theta(bus_int(b_num2),k));
                dcrd_sig(jj,k)=angdcr(jj,k);
                st_state = tot_states+1;
                dcr_states = dcr_dc{jj,7};
                tot_states = tot_states+dcr_states;
                ydcrmx=dcr_dc{jj,5};
                ydcrmn = dcr_dc{jj,6};
                dcr_dsig(jj,k) = ...

                     ↪  dcr_sud(jj,k,flag,dcr_dc{jj,1},dcrd_sig(jj,k),ydcrmx,ydcrmn,xdcr_dc(st_state:tot_s
            end
        end
        if ndci_ud~=0
            % calculate the new value of bus angles inverter user defined control
            for jj = 1:ndci_ud
                tot_states=0;
                b_num1 = dci_dc{jj,3};
                b_num2 = dci_dc{jj,4};
                conv_num = dci_dc{jj,2};
                angdci(jj,k)=theta(bus_int(b_num1),k)-theta(bus_int(b_num2),k);
                dcid_sig(jj,k)=(angdci(jj,k)-angdci(jj,k-1))/(t(k)-t(k-1));
                st_state = tot_states+1;
                dci_states = dci_dc{jj,7};
                tot_states = tot_states+dci_states;
                ydcimx=dci_dc{jj,5};
                ydcimn = dci_dc{jj,6};
                dci_dsig(jj,k) = ...

                     ↪  dci_sud(jj,k,flag,dci_dc{jj,1},dcid_sig(jj,k),ydcimx,ydcimn,xdci_dc(st_state:tot_s
            end
        end

        dc_cont(0,k,10*(k-1)+1,bus_sim,flag);

        %% network interface for control models
        dpwf(0,k,bus_sim,flag);
        mexc_sig(t(k),k);
        smpexc(0,k,bus_sim,flag);
        smppi(0,k,bus_sim,flag);
        exc_st3(0,k,bus_sim,flag);
        exc_dc12(0,k,bus_sim,flag);
        mtg_sig(k);
        tg(0,k,flag);
        tg_hydro(0,k,bus_sim,flag);

        if n_dcud~=0
            %% set the new line currents
            for jj=1:n_dcud
                l_num = svc_dc{jj,3};
                svc_num = svc_dc{jj,2};
                from_bus = bus_int(line_sim(l_num,1));
                to_bus = bus_int(line_sim(l_num,2));
                svc_bn = bus_int(svc_con(svc_num,2));
                V1 = bus_v(from_bus,k);
                V2 = bus_v(to_bus,k);
                R = line_sim(l_num,3);
                X = line_sim(l_num,4);
                B = line_sim(l_num,5);
                tap = line_sim(l_num,6);
                phi = line_sim(l_num,7);
                [l_if,l_it] = line_cur(V1,V2,R,X,B,tap,phi);

                if svc_bn == from_bus
                    d_sig(jj,k) = abs(l_if);
                elseif svc_bn == to_bus
                    d_sig(jj,k) = abs(l_it);
                end
            end
        end

        if n_tcscud~=0
            % set the new bus voltages
            for jj=1:n_tcscud
                b_num = tcsc_dc{jj,3};tcsc_num = tcsc_dc{jj,2};
                td_sig(jj,k)=abs(bus_v(bus_int(b_num),k));
            end
        end

        %% Live plot call
        livePlotFlag = 1; % for possible fugure sim flags
        if livePlotFlag
           livePlot
        end

        %% step 3b: compute dynamics and integrate
        flag = 2;
        sys_freq(k) = 1.0; % why?... 5/21/20
        mpm_sig(t(k),k);
        mac_ind(0,k,bus_sim,flag);
        mac_igen(0,k,bus_sim,flag);
        mac_sub(0,k,bus_sim,flag);
        mac_tra(0,k,bus_sim,flag);
        mac_em(0,k,bus_sim,flag);
        dpwf(0,k,bus_sim,flag);
        pss(0,k,bus_sim,flag);
        mexc_sig(t(k),k);
        smpexc(0,k,bus_sim,flag);
        smppi(0,k,bus_sim,flag);
        exc_st3(0,k,bus_sim,flag);
        exc_dc12(0,k,bus_sim,flag);
        mtg_sig(k);
        tg(0,k,flag);
        tg_hydro(0,k,bus_sim,flag);

        if n_svc~=0
            v_svc = abs(bus_v(bus_int(svc_con(:,2)),k));
            if n_dcud~=0
                tot_states=0;
                for jj = 1:n_dcud
                    ysvcmx = svc_dc{jj,4};
                    ysvcmn = svc_dc{jj,5};
                    svc_num = svc_dc{jj,2};
                    st_state = tot_states+1;
                    svc_states = svc_dc{jj,6};
                    tot_states = tot_states+svc_states;

                    ↪  [svc_dsig(svc_num,k),xsvc_dc(st_state:tot_states,k),dxsvc_dc(st_state:tot_states,k
                    ↪  =...

                        ↪  svc_sud(jj,k,flag,svc_dc{jj,1},d_sig(jj,k),ysvcmx,ysvcmn,xsvc_dc(st_state:tot_
                end
            end
            msvc_sig(t(k),k);
            svc(0,k,bus_sim,flag,v_svc);
        end
        if n_tcsc~=0
            if n_tcscud~=0
                tot_states=0;
                for jj = 1:n_tcscud
                    ytcscmx = tcsc_dc{jj,4};
                    ytcscmn = tcsc_dc{jj,5};
                    tcsc_num = tcsc_dc{jj,2};
                    st_state = tot_states+1;
                    tcsc_states = tcsc_dc{jj,6};
                    tot_states = tot_states+tcsc_states;

                    ↪  [tcsc_dsig(tcsc_num,k),xtcsc_dc(st_state:tot_states,k),dxtcsc_dc(st_state:tot_stat
                    ↪  =...

                        ↪  tcsc_sud(jj,k,flag,tcsc_dc{jj,1},td_sig(jj,k),ytcscmx,ytcscmn,xtcsc_dc(st_stat
                end
            end
            mtcsc_sig(t(k),k);
            tcsc(0,k,bus_sim,flag);
        end

        % modified in v2.3 - thad 06/01/20
        if g.lmod.n_lmod~=0
            ml_sig(k); % removed t - thad
            lmod(0,k,flag); % removed bus input - thad
        end

        if n_rlmod~=0
            rml_sig(t(k),k);
            rlmod(0,k,bus_sim,flag);
        end

        %% pwrmod - copied from v2.3 - 06/01/20 -thad
        if n_pwrmod~=0
            Pst = cell(n_pwrmod,1);
            Qst = Pst;
            for index=1:n_pwrmod
                Pst{index} = pwrmod_p_sigst{index}(:,k);
                Qst{index} = pwrmod_q_sigst{index}(:,k);
            end
            [~,~,dp,dq,~,~] = pwrmod_dyn(Pst,Qst,bus,t,k,flag,n_pwrmod);
            if (~iscell(dp) || ~iscell(dq))
                error('Error in pwrmod_dyn, dp and dq must be cells');
            end
            if (any(size(dp)-[n_pwrmod 1]) || any(size(dq)-[n_pwrmod 1]))
                error('Dimension error in pwrmod_dyn');
            end
            for index=1:n_pwrmod
                if ((size(dp{index},2)~=1) || (size(dq{index},2)~=1))
                    error('Dimension error in pwrmod_dyn');
                end
                if size(dp{index},1)~=size(dpwrmod_p_sigst{index},1)
                    error('Dimension error in pwrmod_dyn');
                end
                if size(dq{index},1)~=size(dpwrmod_q_sigst{index},1)
                    error('Dimension error in pwrmod_dyn');
                end
                dpwrmod_p_sigst{index}(:,k) = dp{index};
                dpwrmod_q_sigst{index}(:,k) = dq{index};
            end
            [P,Q,~,~] = pwrmod_dyn(Pst,Qst,bus,t,k,1,n_pwrmod); %update pwrmod_p_sig and pwrmod_q_sig
            if (length(P)~=n_pwrmod) || (length(Q)~=n_pwrmod)
                error('Dimension error in pwrmod_dyn');
            end
            pwrmod_p_sig(:,k) = P;
            pwrmod_q_sig(:,k) = Q;
            pwrmod_p(0,k,bus_sim,flag);
            pwrmod_q(0,k,bus_sim,flag);
            clear P Q Pst Qst dp dq index
        end

        %% ivm modulation - copied from v2.3 - 06/01/20 -thad
        if n_ivm>0
```

```
            dst = cell(n_ivm,1);
            est = dst;
            for index=1:n_ivm
                dst{index} = ivmmod_d_sigst{index}(:,k);
                est{index} = ivmmod_e_sigst{index}(:,k);
            end
            [d,e,~,~,~,~] = ivmmod_dyn(dst,est,bus,t,k,1); %get internal voltage signals
            if (length(d)~=n_ivm) || (length(e)~=n_ivm)
                error('Dimension error in ivmmod_dyn');
            end
            ivmmod_d_sig(:,k) = d;
            ivmmod_e_sig(:,k) = e;
            mac_ivm(0,k,bus_sim,flag);
            [~,~,dd,de,~,~] = ivmmod_dyn(dst,est,bus,t,k,flag);
            if (~iscell(dd) || ~iscell(de))
                error('Error in ivmmod_dyn, dd and de must be cells');
            end
            if (any(size(dd)-[n_ivm 1]) || any(size(de)-[n_ivm 1]))
                error('Dimension error in ivmmod_dyn');
            end
            for index=1:n_ivm
                if ((size(dd{index},2)~=1) || (size(de{index},2)~=1))
                    error('Dimension error in ivmmod_dyn');
                end
                if size(dd{index},1)~=size(divmmod_d_sigst{index},1)
                    error('Dimension error in ivmmod_dyn');
                end
                if size(de{index},1)~=size(divmmod_e_sigst{index},1)
                    error('Dimension error in ivmmod_dyn');
                end
                divmmod_d_sigst{index}(:,k) = dd{index};
                divmmod_e_sigst{index}(:,k) = de{index};
            end
            clear d e dd de dst est
        end


        %% integrate dc at ten times rate (DC Stuff? 5/14/20)
        mdc_sig(t(k),k);
        if n_conv~=0
            hdc_sol = h_sol/10;
            for kk = 1:10
                kdc=10*(k-1)+kk;
                    [xdcr_dc(:,kdc:kdc+1),dxdcr_dc(:,kdc:kdc+1),xdci_dc(:,kdc:kdc+1),dxdci_dc(:,kdc:kdc+1)
                    = ...
                    dc_sim(k,kk,dcr_dc,dci_dc,xdcr_dc(:,kdc),xdci_dc(:,kdc),bus_sim,hdc_sol);
            end
        else
            dc_cont(0,k,k,bus_sim,2);
            dc_line(0,k,k,bus_sim,2);
        end

        %% following statements are predictor steps
        j = k+1;
        mac_ang(:,j) = mac_ang(:,k) + h_sol*dmac_ang(:,k);
        mac_spd(:,j) = mac_spd(:,k) + h_sol*dmac_spd(:,k);
        edprime(:,j) = edprime(:,k) + h_sol*dedprime(:,k);
        eqprime(:,j) = eqprime(:,k) + h_sol*deqprime(:,k);
        psikd(:,j) = psikd(:,k) + h_sol*dpsikd(:,k);
        psikq(:,j) = psikq(:,k) + h_sol*dpsikq(:,k);
        Efd(:,j) = Efd(:,k) + h_sol*dEfd(:,k);
        V_R(:,j) = V_R(:,k) + h_sol*dV_R(:,k);
        V_As(:,j) = V_As(:,k) + h_sol*dV_As(:,k);
        R_f(:,j) = R_f(:,k) + h_sol*dR_f(:,k);
        V_TR(:,j) = V_TR(:,k) + h_sol*dV_TR(:,k);
        sdpw1(:,j) = sdpw1(:,k) + h_sol*dsdpw1(:,k);
        sdpw2(:,j) = sdpw2(:,k) + h_sol*dsdpw2(:,k);
        sdpw3(:,j) = sdpw3(:,k) + h_sol*dsdpw3(:,k);
        sdpw4(:,j) = sdpw4(:,k) + h_sol*dsdpw4(:,k);
        sdpw5(:,j) = sdpw5(:,k) + h_sol*dsdpw5(:,k);
        sdpw6(:,j) = sdpw6(:,k) + h_sol*dsdpw6(:,k);
        pss1(:,j) = pss1(:,k) + h_sol*dpss1(:,k);
        pss2(:,j) = pss2(:,k) + h_sol*dpss2(:,k);
        pss3(:,j) = pss3(:,k) + h_sol*dpss3(:,k);

        % modified to g - thad
        g.tg.tg1(:,j) = g.tg.tg1(:,k) + h_sol*g.tg.dtg1(:,k);
        g.tg.tg2(:,j) = g.tg.tg2(:,k) + h_sol*g.tg.dtg2(:,k);
        g.tg.tg3(:,j) = g.tg.tg3(:,k) + h_sol*g.tg.dtg3(:,k);
        g.tg.tg4(:,j) = g.tg.tg4(:,k) + h_sol*g.tg.dtg4(:,k);
        g.tg.tg5(:,j) = g.tg.tg5(:,k) + h_sol*g.tg.dtg5(:,k);

        vdp(:,j) = vdp(:,k) + h_sol*dvdp(:,k);
        vqp(:,j) = vqp(:,k) + h_sol*dvqp(:,k);
        slip(:,j) = slip(:,k) + h_sol*dslip(:,k);
        vdpig(:,j) = vdpig(:,k) + h_sol*dvdpig(:,k);
        vqpig(:,j) = vqpig(:,k) + h_sol*dvqpig(:,k);
        slig(:,j) = slig(:,k) + h_sol*dslig(:,k);
        B_cv(:,j) = B_cv(:,k) + h_sol*dB_cv(:,k);
        B_con(:,j) = B_con(:,k) + h_sol*dB_con(:,k);
        xsvc_dc(:,j) = xsvc_dc(:,k) + h_sol* dxsvc_dc(:,k);
        B_tcsc(:,j) = B_tcsc(:,k) + h_sol*dB_tcsc(:,k);
        xtcsc_dc(:,j) = xtcsc_dc(:,k) + h_sol* dxtcsc_dc(:,k);

        %lmod_st(:,j) = lmod_st(:,k) + h_sol*dlmod_st(:,k); % original line - thad
        g.lmod.lmod_st(:,j) = g.lmod.lmod_st(:,k) + h_sol*g.lmod.dlmod_st(:,k); % line using g

        rlmod_st(:,j) = rlmod_st(:,k)+h_sol*drlmod_st(:,k);

        %% Copied from v2.3 - 06/01/20 - thad
        pwrmod_p_st(:,j) = pwrmod_p_st(:,k)+h_sol*dpwrmod_p_st(:,k);
        pwrmod_q_st(:,j) = pwrmod_q_st(:,k)+h_sol*dpwrmod_q_st(:,k);
        %% pwrmod
        if n_pwrmod~=0
            for index=1:n_pwrmod
                pwrmod_p_sigst{index}(:,j) =
                 ↪  pwrmod_p_sigst{index}(:,k)+h_sol*dpwrmod_p_sigst{index}(:,k);
                pwrmod_q_sigst{index}(:,j) =
                 ↪  pwrmod_q_sigst{index}(:,k)+h_sol*dpwrmod_q_sigst{index}(:,k);
            end
        end
        %% ivmmod
        if n_ivm~=0
            for index=1:n_ivm
                ivmmod_d_sigst{index}(:,j) =
                 ↪  ivmmod_d_sigst{index}(:,k)+h_sol*divmmod_d_sigst{index}(:,k);
                ivmmod_e_sigst{index}(:,j) =
                 ↪  ivmmod_e_sigst{index}(:,k)+h_sol*divmmod_e_sigst{index}(:,k);
            end
        end


        %% Flag = 1
        flag = 1;
        % mach_ref(j) = mac_ang(syn_ref,j);
        mach_ref(j) = 0;
        % perform network interface calculations again with predicted states
        mpm_sig(t(j),j);
        mac_ind(0,j,bus_sim,flag);
        mac_igen(0,j,bus_sim,flag);
        mac_sub(0,j,bus_sim,flag);
        mac_tra(0,j,bus_sim,flag);
        mac_em(0,j,bus_sim,flag);
        % assume Vdc remains unchanged for first pass through dc controls interface
        mdc_sig(t(j),j);
        dc_cont(0,j,10*(j-1)+1,bus_sim,flag);

        % Calculate current injections and bus voltages and angles
        if j >= sum(k_inc(1:3))+1
            % fault cleared
            bus_sim = bus_pf2;
            bus_int = bus_intpf2;
            Y1 = Y_gpf2;
            Y2 = Y_gncpf2;
            Y3 = Y_ncgpf2;
            Y4 = Y_ncpf2;
            Vr1 = V_rgpf2;
            Vr2 = V_rncpf2;
            bo = bopf2;
            %h_sol = i_simu(j,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
        elseif j >=sum(k_inc(1:2))+1
            % near bus cleared
            bus_sim = bus_pf1;
            bus_int = bus_intpf1;
            Y1 = Y_gpf1;
            Y2 = Y_gncpf1;
            Y3 = Y_ncgpf1;
            Y4 = Y_ncpf1;
            Vr1 = V_rgpf1;
            Vr2 = V_rncpf1;
            bo = bopf1;
            %h_sol = i_simu(j,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
        elseif j>=k_inc(1)+1
            % fault applied
            bus_sim = bus_f;
            bus_int = bus_intf;
            Y1 = Y_gf;
            Y2 = Y_gncf;
            Y3 = Y_ncgf;
            Y4 = Y_ncf;
            Vr1 = V_rgf;
            Vr2 = V_rncf;
            bo = bof;
            %h_sol = i_simu(j,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
        elseif k<k_inc(1)+1  % JHC - DKF thinks k should be j
            % pre fault
            bus_sim = bus;
            bus_int = bus_intprf;
            Y1 = Y_gprf;
            Y2 = Y_gncprf;
            Y3 = Y_ncgprf;
            Y4 = Y_ncprf;
            Vr1 = V_rgprf;
            Vr2 = V_rncprf;
            bo = boprf;
            %h_sol = i_simu(j,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);
        end

        % apply gen trip - copied from v2.3 - 06/01/20 - thad
        if sum(mac_trip_flags)>0.5
            genBuses = mac_con(mac_trip_flags==1,2);
            for kB=1:length(genBuses)
                nL = find(genBuses(kB)==line_sim(:,1) | genBuses(kB)==line_sim(:,2));
                if isempty(nL)
                    error('nL is empty.');
                end
                line_sim(nL,4) = 1e7; %make reactance infinity
            end
            [Y1,Y2,Y3,Y4,Vr1,Vr2,bo] = red_ybus(bus_sim,line_sim);
            clear nL kB genBuses
        end

        %% solve
        if k == 50 % DEBUG - showing of network solution call
            warning('*** k == 50; Performing network solution via i_simu')
        end
        h_sol = i_simu(j,ks,k_inc,h,bus_sim,Y1,Y2,Y3,Y4,Vr1,Vr2,bo);

        vex(:,j)=vex(:,k);
        cur_ord(:,j) = cur_ord(:,k);
        % calculate the new value of bus angles rectifier user defined control
        if ndcr_ud~=0
            tot_states=0;
            for jj = 1:ndcr_ud
                b_num1 = dcr_dc{jj,3};
                b_num2 = dcr_dc{jj,4};
                conv_num = dcr_dc{jj,2};
                angdcr(jj,j) = theta(bus_int(b_num1),j)-theta(bus_int(b_num2),j);
                dcrd_sig(jj,j)=angdcr(jj,j);
                st_state = tot_states+1;
                dcr_states = dcr_dc{jj,7};
                tot_states = tot_states+dcr_states;
                ydcrmx=dcr_dc{jj,5};ydcrmn = dcr_dc{jj,6};
                dcr_dsig(jj,j) = ...

                    ↪  dcr_sud(jj,j,flag,dcr_dc{jj,1},dcrd_sig(jj,j),ydcrmx,ydcrmn,xdcr_dc(st_state:tot_s
            end
        end
        if ndci_ud~=0
            % calculate the new value of bus angles inverter user defined control
            for jj = 1:ndci_ud
                tot_states=0;
                b_num1 = dci_dc{jj,3};
                b_num2 = dci_dc{jj,4};
                conv_num = dci_dc{jj,2};
                angdci(jj,j) = theta(bus_int(b_num1),j)-theta(bus_int(b_num2),j);
                dcid_sig(jj,j) = (angdci(jj,j)-angdci(jj,k))/(t(j)-t(k));
                st_state = tot_states+1;
                dci_states = dci_dc{jj,7};
                tot_states = tot_states+dci_states;
                ydcimx=dci_dc{jj,5};
                ydcimn = dci_dc{jj,6};
                dci_dsig(jj,j) = ...

                    ↪  dci_sud(jj,j,flag,dci_dc{jj,1},dcid_sig(jj,j),ydcimx,ydcimn,xdci_dc(st_state:tot_s
            end
        end

        dc_cont(0,j,10*(j-1)+1,bus_sim,flag);
        dpwf(0,j,bus_sim,flag);
        pss(0,j,bus_sim,flag);
        mexc_sig(t(j),j);
        smpexc(0,j,bus_sim,flag);
        smppi(0,j,bus_sim,flag);
        exc_st3(0,j,bus_sim,flag);
        exc_dc12(0,j,bus_sim,flag);
        tg(0,j,flag);
        tg_hydro(0,j,bus_sim,flag);

        if n_dcud~=0
            % set the new line currents
            for jj=1:n_dcud
                l_num = svc_dc{jj,3};svc_num = svc_dc{jj,2};
                from_bus = bus_int(line_sim(l_num,1));
                to_bus = bus_int(line_sim(l_num,2));
                svc_bn = bus_int(svc_con(svc_num,2));
                V1 = bus_v(from_bus,j);
                V2 = bus_v(to_bus,j);
                R = line_sim(l_num,3);
                X = line_sim(l_num,4);
                B = line_sim(l_num,5);
                tap = line_sim(l_num,6);phi = line_sim(l_num,7);
                [l_if,l_it] = line_cur(V1,V2,R,X,B,tap,phi);
                if svc_bn == from_bus;
                    d_sig(jj,j)=abs(l_if);
                elseif svc_bn==to_bus;
                    d_sig(jj,j)=abs(l_it);
                end
            end
        end

        if n_tcscud~=0
            % set the new line currents
            for jj=1:n_tcscud
                b_num = tcsc_dc{jj,3};
                tcsc_num = tcsc_dc{jj,2};
                td_sig(jj,j) = abs(bus_v(bus_int(b_num),j));
            end
        end

        %% Flag = 2, for 'corrector step' d's
        flag = 2;
        mac_ind(0,j,bus_sim,flag);
        mac_igen(0,j,bus_sim,flag);
        mac_sub(0,j,bus_sim,flag);
        mac_tra(0,j,bus_sim,flag);
        mac_em(0,j,bus_sim,flag);
        dpwf(0,j,bus_sim,flag);
        pss(0,j,bus_sim,flag);
        mexc_sig(t(j),j);
        smpexc(0,j,bus_sim,flag);
        smppi(0,j,bus_sim,flag);
        exc_st3(0,j,bus_sim,flag);
        exc_dc12(0,j,bus_sim,flag);
        mtg_sig(j);
        tg(0,j,flag);
        tg_hydro(0,j,bus_sim,flag);

        if n_svc~=0
            msvc_sig(t(j),j);
            if n_dcud~=0
                tot_states=0;
                for jj = 1:n_dcud
                    ysvcmx = svc_dc{jj,4};
                    ysvcmn = svc_dc{jj,5};
                    svc_num = svc_dc{jj,2};
                    st_state = tot_states+1;
                    svc_states = svc_dc{jj,6};
                    tot_states = tot_states+svc_states;

                    ↪  [svc_dsig(svc_num,j),xsvc_dc(st_state:tot_states,j),dxsvc_dc(st_state:tot_states,
                    ↪  =...

                        ↪  svc_sud(jj,j,flag,svc_dc{jj,1},d_sig(jj,j),ysvcmx,ysvcmn,xsvc_dc(st_state:tot_
                end
            end
            v_svc = abs(bus_v(bus_int(svc_con(:,2)),j));
            bus_sim = svc(0,j,bus_sim,flag,v_svc);
        end

        if n_tcsc~=0
            mtcsc_sig(t(j),j); % this has changed since v 2.3...
            if n_tcscud~=0
                tot_states=0;
                for jj = 1:n_tcscud
```

```
                    ytcscmx = tcsc_dc{jj,4};
                    ytcscmn = tcsc_dc{jj,5};
                    tcsc_num = tcsc_dc{jj,2};
                    st_state = tot_states+1;
                    tcsc_states = tcsc_dc{jj,6};
                    tot_states = tot_states+tcsc_states;

                    ↪  [tcsc_dsig(tcsc_num,j),xtcsc_dc(st_state:tot_states,j),dxtcsc_dc(st_state:tot_stat
                    ↪  =...

                        ↪  tcsc_sud(jj,j,flag,tcsc_dc{jj,1},td_sig(jj,j),ytcscmx,ytcscmn,xtcsc_dc(st_stat
                end
            end
            tcsc(0,j,bus_sim,flag);
        end

        % modified to handle g - thad 06/01/20
        if g.lmod.n_lmod~=0
            ml_sig(j); % removed t - thad
            lmod(0,j,flag); % removed bus - thad
        end
        if n_rlmod~=0
            rml_sig(t(j),j);
            rlmod(0,j,bus_sim,flag);
        end

        % copied from v2.3 - thad - 06/01/20
        if n_pwrmod~=0
            Pst = cell(n_pwrmod,1);
            Qst = Pst;
            for index=1:n_pwrmod
                Pst{index} = pwrmod_p_sigst{index}(:,j);
                Qst{index} = pwrmod_q_sigst{index}(:,j);
            end
            [~,~,dp,dq,~,~] = pwrmod_dyn(Pst,Qst,bus,t,j,flag,n_pwrmod);
            if (~iscell(dp) || ~iscell(dq))
                error('Error in pwrmod_dyn, dp and dq must be cells');
            end
            if (any(size(dp)-[n_pwrmod 1]) || any(size(dq)-[n_pwrmod 1]))
                error('Dimension error in pwrmod_dyn');
            end

            for index=1:n_pwrmod
                if ((size(dp{index},2)~=1) || (size(dq{index},2)~=1))
                    error('Dimension error in pwrmod_dyn');
                end
                if size(dp{index},1)~=size(dpwrmod_p_sigst{index},1)
                    error('Dimension error in pwrmod_dyn');
                end
                if size(dq{index},1)~=size(dpwrmod_q_sigst{index},1)
                    error('Dimension error in pwrmod_dyn');
                end
                dpwrmod_p_sigst{index}(:,j) = dp{index};
                dpwrmod_q_sigst{index}(:,j) = dq{index};
            end
            [P,Q,~,~,~,~] = pwrmod_dyn(Pst,Qst,bus,t,j,1,n_pwrmod); %update pwrmod_p_sig and pwrmod_q_sig
            if (length(P)~=n_pwrmod) || (length(Q)~=n_pwrmod)
                error('Dimension error in pwrmod_dyn');
            end
            pwrmod_p_sig(:,j) = P;
            pwrmod_q_sig(:,j) = Q;
            pwrmod_p(0,j,bus_sim,flag);
            pwrmod_q(0,j,bus_sim,flag);
            clear P Q Pst Qst dp dq index
        end

        if n_ivm>0
            dst = cell(n_ivm,1);
            est = dst;
            for index=1:n_ivm
                dst{index} = ivmmod_d_sigst{index}(:,j);
                est{index} = ivmmod_e_sigst{index}(:,j);
            end
            [d,e,~,~,~,~] = ivmmod_dyn(dst,est,bus,t,j,1);
            if (length(d)~=n_ivm) || (length(e)~=n_ivm)
                error('Dimension error in ivmmod_dyn');
            end
            ivmmod_d_sig(:,j) = d;
            ivmmod_e_sig(:,j) = e;
            mac_ivm(0,j,bus_sim,flag);
            [~,~,dd,de,~,~] = ivmmod_dyn(dst,est,bus,t,j,flag);
            if (~iscell(dd) || ~iscell(de))
                error('Error in ivmmod_dyn, dd and de must be cells');
            end
            if (any(size(dd)-[n_ivm 1]) || any(size(de)-[n_ivm 1]))
                error('Dimension error in ivmmod_dyn');
            end
            for index=1:n_ivm
                if ((size(dd{index},2)~=1) || (size(de{index},2)~=1))
                    error('Dimension error in ivmmod_dyn');
                end
                if size(dd{index},1)~=size(divmmod_d_sigst{index},1)
                    error('Dimension error in ivmmod_dyn');
                end
                if size(de{index},1)~=size(divmmod_e_sigst{index},1)
```

```
                    error('Dimension error in ivmmod_dyn');
                end
                divmmod_d_sigst{index}(:,j) = dd{index};
                divmmod_e_sigst{index}(:,j) = de{index};
            end
            clear d e dd de dst est
        end
        % end copied from...

        if n_conv~=0
            hdc_sol = h_sol/10;
            for kk = 1:10
                jdc=10*(j-1)+kk;

                ↪  [xdcr_dc(:,jdc:jdc+1),dxdcr_dc(:,jdc:jdc+1),xdci_dc(:,jdc:jdc+1),dxdci_dc(:,jdc:jdc+1
                ↪  = ...
                    dc_sim(j,kk,dcr_dc,dci_dc,xdcr_dc(:,jdc),xdci_dc(:,jdc),bus_sim,hdc_sol);
            end
        else
            dc_cont(0,j,j,bus_sim,2);
            dc_line(0,j,j,bus_sim,2);
        end

        %% following statements are corrector steps
        mac_ang(:,j) = mac_ang(:,k) + h_sol*(dmac_ang(:,k)+dmac_ang(:,j))/2.;
        mac_spd(:,j) = mac_spd(:,k) + h_sol*(dmac_spd(:,k)+dmac_spd(:,j))/2.;
        edprime(:,j) = edprime(:,k) + h_sol*(dedprime(:,k)+dedprime(:,j))/2.;
        eqprime(:,j) = eqprime(:,k) + h_sol*(deqprime(:,k)+deqprime(:,j))/2.;
        psikd(:,j) = psikd(:,k) + h_sol*(dpsikd(:,k)+dpsikd(:,j))/2.;
        psikq(:,j) = psikq(:,k) + h_sol*(dpsikq(:,k)+dpsikq(:,j))/2.;
        Efd(:,j) = Efd(:,k) + h_sol*(dEfd(:,k)+dEfd(:,j))/2.;
        V_R(:,j) = V_R(:,k) + h_sol*(dV_R(:,k)+dV_R(:,j))/2.;
        V_As(:,j) = V_As(:,k) + h_sol*(dV_As(:,k)+dV_As(:,j))/2.;
        R_f(:,j) = R_f(:,k) + h_sol*(dR_f(:,k)+dR_f(:,j))/2.;
        V_TR(:,j) = V_TR(:,k) + h_sol*(dV_TR(:,k)+dV_TR(:,j))/2.;
        sdpw11(:,j) = sdpw1(:,k) +h_sol*(dsdpw1(:,k)+dsdpw1(:,j))/2.;
        sdpw12(:,j) = sdpw2(:,k) +h_sol*(dsdpw2(:,k)+dsdpw2(:,j))/2.;
        sdpw13(:,j) = sdpw3(:,k) +h_sol*(dsdpw3(:,k)+dsdpw3(:,j))/2.;
        sdpw14(:,j) = sdpw4(:,k) +h_sol*(dsdpw4(:,k)+dsdpw4(:,j))/2.;
        sdpw15(:,j) = sdpw5(:,k) +h_sol*(dsdpw5(:,k)+dsdpw5(:,j))/2.;
        sdpw16(:,j) = sdpw6(:,k) +h_sol*(dsdpw6(:,k)+dsdpw6(:,j))/2.;
        pss1(:,j) = pss1(:,k) +h_sol*(dpss1(:,k)+dpss1(:,j))/2.;
        pss2(:,j) = pss2(:,k) +h_sol*(dpss2(:,k)+dpss2(:,j))/2.;
        pss3(:,j) = pss3(:,k) +h_sol*(dpss3(:,k)+dpss3(:,j))/2.;

        % modified to g
        g.tg.tg1(:,j) = g.tg.tg1(:,k) + h_sol*(g.tg.dtg1(:,k) + g.tg.dtg1(:,j))/2.;
        g.tg.tg2(:,j) = g.tg.tg2(:,k) + h_sol*(g.tg.dtg2(:,k) + g.tg.dtg2(:,j))/2.;
        g.tg.tg3(:,j) = g.tg.tg3(:,k) + h_sol*(g.tg.dtg3(:,k) + g.tg.dtg3(:,j))/2.;
        g.tg.tg4(:,j) = g.tg.tg4(:,k) + h_sol*(g.tg.dtg4(:,k) + g.tg.dtg4(:,j))/2.;
        g.tg.tg5(:,j) = g.tg.tg5(:,k) + h_sol*(g.tg.dtg5(:,k) + g.tg.dtg5(:,j))/2.;

        vdp(:,j) = vdp(:,k) + h_sol*(dvdp(:,j) + dvdp(:,k))/2.;
        vqp(:,j) = vqp(:,k) + h_sol*(dvqp(:,j) + dvqp(:,k))/2.;
        slip(:,j) = slip(:,k) + h_sol*(dslip(:,j) + dslip(:,k))/2.;
        vdpig(:,j) = vdpig(:,k) + h_sol*(dvdpig(:,j) + dvdpig(:,k))/2.;
        vqpig(:,j) = vqpig(:,k) + h_sol*(dvqpig(:,j) + dvqpig(:,k))/2.;
        slig(:,j) = slig(:,k) + h_sol*(dslig(:,j) + dslig(:,k))/2.;
        B_cv(:,j) = B_cv(:,k) + h_sol*(dB_cv(:,j) + dB_cv(:,k))/2.;
        B_con(:,j) = B_con(:,k) + h_sol*(dB_con(:,j) + dB_con(:,k))/2.;
        xsvc_dc(:,j) = xsvc_dc(:,k) + h_sol*(dxsvc_dc(:,j) + dxsvc_dc(:,k))/2.;
        B_tcsc(:,j) = B_tcsc(:,k) + h_sol*(dB_tcsc(:,j) + dB_tcsc(:,k))/2.;
        xtcsc_dc(:,j) = xtcsc_dc(:,k) + h_sol*(dxtcsc_dc(:,j) + dxtcsc_dc(:,k))/2.;

        % modified to g
        g.lmod.lmod_st(:,j) = g.lmod.lmod_st(:,k) + h_sol*(g.lmod.dlmod_st(:,j) +
        ↪  g.lmod.dlmod_st(:,k))/2.; % modified line with g

        rlmod_st(:,j) = rlmod_st(:,k) + h_sol*(drlmod_st(:,j) + drlmod_st(:,k))/2.;

        % Copied from v2.3 - 06/01/20 - thad
        pwrmod_p_st(:,j) = pwrmod_p_st(:,k)+h_sol*(dpwrmod_p_st(:,j) + dpwrmod_p_st(:,k))/2;
        pwrmod_q_st(:,j) = pwrmod_q_st(:,k)+h_sol*(dpwrmod_q_st(:,j) + dpwrmod_q_st(:,k))/2;
        if n_pwrmod~=0
            for index=1:n_pwrmod
                pwrmod_p_sigst{index}(:,j) =
                ↪  pwrmod_p_sigst{index}(:,k)+h_sol*(dpwrmod_p_sigst{index}(:,j) +
                ↪  dpwrmod_p_sigst{index}(:,k))/2;
                pwrmod_q_sigst{index}(:,j) =
                ↪  pwrmod_q_sigst{index}(:,k)+h_sol*(dpwrmod_q_sigst{index}(:,j) +
                ↪  dpwrmod_q_sigst{index}(:,k))/2;
            end
        end
        if n_ivm~=0
            for index=1:n_ivm
                ivmmod_d_sigst{index}(:,j) =
                ↪  ivmmod_d_sigst{index}(:,k)+h_sol*(divmmod_d_sigst{index}(:,j) +
                ↪  divmmod_d_sigst{index}(:,k))/2;
                ivmmod_e_sigst{index}(:,j) =
                ↪  ivmmod_e_sigst{index}(:,k)+h_sol*(divmmod_e_sigst{index}(:,j) +
                ↪  divmmod_e_sigst{index}(:,k))/2;
            end
        end

    end
    % counter increment
    kt = kt + k_inc(ks);
    ks = ks+1;
end% end simulation loop
```