EE 523 Power System Stability and Control Algorithms

Preamble and Control Inputs

```
tic;
if strcmp(getenv('USERNAME'), 'aryan')
    cd C:\Users\aryan\Documents\documents_general\dablab_files\ee521_and_ee523
end
```

you may continue as usual

```
addpath functions\
systemName = "ieee11-caseTwo-"
```

```
systemName =
"ieee11-caseTwo-"
```

```
powerFlowMethod =  "NRPF"
```

```
powerFlowMethod =
"NRPF"
```

```
modelType = "Type3" %Synchronous Machine Model Type
```

```
modelType =
"Type3"
```

```
useReducedLoadsForPowerFlow = false; %only needed if Type 2 or 3
displayYGen = true; %Show YGen
displayYNet = false; %Show YNet
displayYBus = false;
saveYBus = true;
saveYGenMatrices = true; %Save YNet, Ygr, Yrg, Yrr, YGen
runTransientSimulation = false;
saveTransientRunValues = true;
saveTransientRunPlots = true;
showInternalMatrices = false; %Show Ygg, Ygr, Yrg, Yrr
numIterations = 20; %I don't wait for the system to converge,
printPowerFlowConvergenceMessages = false;
% neither do I care if the system converges earlier.
toleranceLimit = 1e-3; %mean of absolute values of
% corrections should be less than this for convergence to be achieved.
displayRawData = false;
displayRawDataForYNet = false;
displayTables = true; %show busData, branchData ybus,
% basically data structures which are not the final output.
saveBusDataAndBranchData = false;
saveBusDataAndBranchDataNet = false;
printJacobians = false  ; %Print Jacobians during NRPF iterations? Does not work if
displayTables is off.
printMismatches = false; %Print Mismatches during NRPF iterations? Does not work if
displayTables is off.
printCorrections = false;
```

```
disableTaps = false; %Disable Tap-changers when commputing YBus?
showPlots = false;
displayResults = false;
reducedBranchColumnsCDFReading = true;
verboseCDFReading = false; %Will give a verbose output when reading CDF files.
MVAb = 100; %Currently the same for all systems in database.
```

Transient Run Parameters

```
bus1 = 7;
bus2 = 8;
numLinesBeforeFault = 3;
relativeDistance = 0.5;

h = 1e-3;
clearingCycles = 3;
f0 = 60;
clearingTime = clearingCycles/f0;
numStepsFaultOn = ceil(clearingTime/h);
postFaultTime = 30;
numStepsPostFault = ceil(postFaultTime/h);
```

Housekeeping.

```
folder_rawData = "rawData/"; %location of CDF .txt file for the system
file_rawData = strcat(folder_rawData, systemName, "cdf.txt"); %Exact location of
CDF .txt file for the system
folder_processedData = "processedData/";
% Should configure it to be read from the CDF file later.
latex_interpreter %for LaTeX typesetting in plots

if contains(systemName, 'ieee11')
    systemName1 = 'ieee11';
else
    systemName1 = systemName;
end
```

Read CDF file and store the data in neat MATLAB `tables: busData` and `branchData`.

```
[busData, branchData, N, numBranchNet] = ...
    readCDF(file_rawData, reducedBranchColumnsCDFReading, verboseCDFReading);
```

Optionally display the system data.

```
if displayRawData
    displayRawDataAsTables(busData, branchData, N, numBranchNet);
end
if saveBusDataAndBranchData
    fileType = '.csv';
```

```
    filenameBusData = strcat(folder_processedData, systemName1, "/busData",
fileType);
    writetable(busData, filenameBusData);
    filenameBranchData = strcat(folder_processedData, systemName1, "/branchData",
fileType);
    writetable(branchData, filenameBranchData);
end
```

Synchronous Machine Parameters

```
[~, ~, ~, ~, ...
        lPQ, lPV, ~, nPV, ...
        ~] = initializeVectors(busData, MVAb);

kVB = 230;
kVB_Gen = 20;
kVB_Gen_side = 20;
MVAb_Gen = 900;

H = [6.5; 6.5; 6.175; 6.175] * (MVAb_Gen/MVAb);
Xd_prime = 0.3* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xq_prime = 0.55* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xq = 1.7* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xd = 1.8* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Tq0_prime = 0.4 * ones(nPV+1, 1);
Td0_prime = 8.0 * ones(nPV+1, 1);
w_s = 2*pi*60;
% K_D = 2*ones(nPV+1, 1) * (MVAb_Gen/MVAb);
K_D = 2*ones(nPV+1, 1);
k_A = 50*ones(nPV+1, 1);
T_A = 0.01*ones(nPV+1, 1);
EfdMax = 2.0*ones(nPV+1, 1);
Efdmin = 0.0*ones(nPV+1, 1);
VRmin = -4*ones(nPV+1, 1);
VRMax = 4*ones(nPV+1, 1);
Tsg = 100*ones(nPV+1, 1);
Ksg = 1*ones(nPV+1, 1);
PsgMax = 1*ones(nPV+1, 1);
Psgmin = 0*ones(nPV+1, 1);
R = 5*1/100*ones(nPV+1, 1);
```

PSS Parameters

```
Kwpss = 10;
Twpss = 10;
Kcomp = 0.1;
Tacomp1 = 0.49;
Tbcomp1 = 0.24;
Tacomp2 = Tacomp1;
Tbcomp2 = Tbcomp1;
```

Powerflow

Extract Y_{Bus}, Adjacency List E from the `branchData table` for Type I OR Make Y_{Bus} (reduced loads only), Y_{Net}, Y_{Gen} for Type II and Type III models.

Then Run Newton Raphson Power Flow and obtain a steady state snapshot of the system variables P_i, Q_i, V_i, δ_i $\forall$ buses $i \in [1, N], i \in \mathbb{N}$

```
if ~strcmp(modelType, "Type1")
    X_prime = 0.5* (Xd_prime + Xq_prime);

    [busDataNet, busDataReducedLoads, busDataGen, branchDataNet, NNet,
numBranchNet] = ...
        modifyForYNet(busData, branchData, N, ...
        numBranchNet, lPV, nPV, ...
        X_prime, MVAb, displayRawDataForYNet);

    if saveBusDataAndBranchDataNet
        fileType = '.csv';
        filenameBusDataNet = strcat(folder_processedData, systemName1, "/
busDataNet", fileType);
        writetable(busData, filenameBusData);
        filenameBranchDataNet = strcat(folder_processedData, systemName1, "/
branchDataNet", fileType);
        writetable(branchDataNet, filenameBranchDataNet);
    end

    [ybus, BMatrix, E, PSpecified, QSpecified, V, delta, listOfPQBuses, lPV,
nPQ, nPV, listOfNonSlackBuses] = generateYBusForType2(useReducedLoadsForPowerFlow,
busData, busDataReducedLoads, branchData, MVAb);

    [yNet, ~, ~, ~, ~, EdgesNet] = ybusGenerator(busDataNet, branchDataNet);

    [yGen, BMatrixGen, EdgesGen, listOfResidualBuses] = constructYGen(yNet, lPV,
NNet, displayYNet, showInternalMatrices, displayYGen, saveYGenMatrices);

    [P, Q, V, delta] = solveForPowerFlow(PSpecified, QSpecified, V, delta,
ybus, BMatrix, E, nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, numIterations,
toleranceLimit, powerFlowMethod, displayTables, printJacobians, printMismatches,
printPowerFlowConvergenceMessages);

    [busDataNetFaultOn, branchDataNetFaultOn, busDataNetPostFault,
branchDataNetPostFault] = modifySystemDuringAndPostFault(busDataNet, branchDataNet,
bus1, bus2, numLinesBeforeFault, relativeDistance);
    yNetFaultOn = ybusGenerator(busDataNetFaultOn, branchDataNetFaultOn);
    % saveAndDisplayYBus(yNetFaultOn, displayYBus, saveYGenMatrices,
folder_processedData, systemName1, 'yNet_FaultOn');
    yGenFaultOn = constructYGen(yNetFaultOn, lPV, NNet, displayYNet,
showInternalMatrices, displayYGen, saveYGenMatrices, 'FaultOn');
```

```
    yNetPostFault = ybusGenerator(busDataNetPostFault, branchDataNetPostFault);
    % saveAndDisplayYBus(yNetPostFault, displayYBus, saveYGenMatrices,
folder_processedData, systemName1, 'yNet_FaultOn');

    % PG_computedUsingYGen = array2table(double(subs(PG_Type3, theta,
theta0_Type3)), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses)
    yGenPostFault = constructYGen(yNetPostFault, lPV, NNet, displayYNet,
showInternalMatrices, displayYGen, saveYGenMatrices, 'PostFault');

else
    [ybus, BMatrix, ~, ~, ~, E] = ybusGenerator(busData, branchData);

    [PSpecified, QSpecified, V, delta, ...
        listOfPQBuses, lPV, nPQ, nPV, ...
        listOfNonSlackBuses] = initializeVectors(busData, MVAb);

    [P, Q, V, delta] = solveForPowerFlow(PSpecified, QSpecified, V, delta,
ybus, BMatrix, E, nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, numIterations,
toleranceLimit, powerFlowMethod, displayTables, printJacobians, printMismatches,
printPowerFlowConvergenceMessages);

    [busDataFaultOn, branchDataFaultOn, busDataPostFault, branchDataPostFault]
= modifySystemDuringAndPostFault(busData, branchData, bus1, bus2,
numLinesBeforeFault, relativeDistance);
    ybusFaultOn = ybusGenerator(busDataFaultOn, branchDataFaultOn);
    saveAndDisplayYBus(ybusFaultOn, displayYBus, saveYBus, folder_processedData,
systemName1, 'ybus_FaultOn');
    ybusPostFault = ybusGenerator(busDataPostFault, branchDataPostFault);
    saveAndDisplayYBus(ybusPostFault, displayYBus, saveYBus, folder_processedData,
systemName1, 'ybus_PostFault');
end
```

yGenTable = 4×4 table

	1	2	3	4
1 1	3.2168 -11.5866i	1.6434 + 7.9619i	0.9492 + 1.0023i	1.3592 + 1.3569i
2 2	1.6434 + 7.9619i	1.3830 -10.5241i	0.6581 + 0.6164i	0.9393 + 0.8316i
3 3	0.9492 + 1.0023i	0.6581 + 0.6164i	1.6473 - 8.1778i	1.9706 + 4.2053i
4 4	1.3592 + 1.3569i	0.9393 + 0.8316i	1.9706 + 4.2053i	2.9049 - 9.8799i

```
Convergence using NRPF achieved in 4 iterations.
```

yGenTable = 4×4 table

	1	2	3	4
1 1	1.8186 -14.5133i	0.6173 + 6.1083i	0.2689 + 0.3831i	0.3889 + 0.5223i
2 2	0.6173 + 6.1083i	0.6414 -11.6926i	0.1903 + 0.2393i	0.2740 + 0.3253i
3 3	0.2689 + 0.3831i	0.1903 + 0.2393i	1.4033 - 8.2672i	1.6277 + 4.0905i
4 4	0.3889 + 0.5223i	0.2740 + 0.3253i	1.6277 + 4.0905i	2.4236 -10.0261i

yGenTable = 4×4 table

	1	2	3	4
1 1	3.2766 -11.3047i	1.6935 + 8.1434i	0.8894 + 0.8893i	1.2717 + 1.2020i
2 2	1.6935 + 8.1434i	1.4228 -10.4077i	0.6147 + 0.5451i	0.8762 + 0.7340i
3 3	0.8894 + 0.8893i	0.6147 + 0.5451i	1.7186 - 8.1324i	2.0714 + 4.2656i
4 4	1.2717 + 1.2020i	0.8762 + 0.7340i	2.0714 + 4.2656i	3.0475 - 9.8000i

```
listOfGenBuses = [1; lPV];

saveAndDisplayYBus(ybus, displayYBus, saveYBus, folder_processedData, systemName1);
```

Compare obtained snapshot values of V_i and δ_i against the ones given in the CDF file.

```
resultTable = displayPowerFlowResults(P, Q, V, delta, displayResults);
plotPowerFlowResults(showPlots, V, busData, systemName, powerFlowMethod, delta);
```

Post-Powerflow

```
P_i_Gen_Vals = P(listOfGenBuses);
Q_i_Gen_Vals = Q(listOfGenBuses);
S_i_Gen_Vals = P_i_Gen_Vals + 1i*Q_i_Gen_Vals;

PGVals = P_i_Gen_Vals + busData.PL(listOfGenBuses)/MVAb;
QGVals = Q_i_Gen_Vals + busData.QL(listOfGenBuses)/MVAb;
SGVals = PGVals + 1i*QGVals;

ViVals = V(listOfGenBuses);
delta_i_Vals = delta(listOfGenBuses);

[Vi_CartesiansReal, Vi_Cartesian_Imag] = pol2cart(delta_i_Vals, ViVals);
V_Gen_Cartesian = Vi_CartesiansReal + 1i*Vi_Cartesian_Imag;
IGenVals = conj(SGVals./V_Gen_Cartesian);

theta = sym('theta', [nPV+1, 1]);
theta(1) = delta_i_Vals(1);

omega = sym('omega', [nPV+1, 1]);

PG = sym('P_G', [nPV+1, 1]);
QG = sym('Q_G', [nPV+1, 1]);

Eq_primeSymbols = arrayfun(@(i) sprintf('E_prime_q%d', i), 1:nPV+1,
'UniformOutput', false);
Eq_prime = str2sym(Eq_primeSymbols');
Eq_prime(1) = ViVals(1);

Ed_primeSymbols = arrayfun(@(i) sprintf('E_prime_d%d', i), 1:nPV+1,
'UniformOutput', false);
```

```
Ed_prime = str2sym(Ed_primeSymbols');
Ed_prime(1) = 0;

Pm = sym('P_m', [nPV+1, 1]);

Id = sym('I_d', [nPV+1, 1]);
Iq = sym('I_q', [nPV+1, 1]);

V_i = sym('V_i', [N, 1]);
V_i(1) = V(1);

delta_i = sym('delta_i', [N, 1]);
delta_i(1) = delta(1);

P_i = sym('P_i', [N, 1]);
Q_i = sym('Q_i', [N, 1]);

P_L = sym('P_L', [N, 1]);
Q_L = sym('Q_L', [N, 1]);

Vq = sym('V_q', [nPV+1, 1]);
Vd = sym('V_d', [nPV+1, 1]);

P_C = sym('P_C', [nPV+1, 1]);
V_R = sym('V_R', [nPV+1, 1]);
Vref = sym('V_ref', [nPV+1, 1]);
```

Dynamic Initialization

```
ode_theta = diff(theta) == (omega-1)*w_s;
ode_thetaEquilibrium = 0 == rhs(ode_theta(1PV));

namesGenBuses = arrayfun(@(i) sprintf('Gen %d', i), 1:nPV+1, 'UniformOutput',
false);

if strcmp(modelType, 'Type3')
    EprimeCartesian = V_Gen_Cartesian + 1i*X_prime.*IGenVals;
    [thetaVals, E_primeVals] = cart2pol(real(EprimeCartesian),
imag(EprimeCartesian));
    thetaVals(1) = delta_i_Vals(1);
    E_primeVals(1) = ViVals(1);

    resultGen = array2table([E_primeVals, thetaVals], 'VariableNames', {'E''',
'theta'}, 'RowNames', namesGenBuses)

    ode_omegaType3 = diff(omega) == (1./(2*H)).*(Pm - PG - K_D.*(omega - 1));
    ode_omegaType3_Equilibrium = 0 == rhs(ode_omegaType3(1PV));
    ode_omegaType3_Init = subs(ode_omegaType3_Equilibrium, PG, PGVals);

    ode_Type3_Init = [ode_thetaEquilibrium; ode_omegaType3_Init];
```

```
    display(ode_Type3_Init);
    unknowns_Type3 = [omega(lPV); Pm(lPV)];

    init_params_Type3 = [ones(nPV, 1); PGVals(lPV)];

    unknowns0_Type3_Vals = solveAndExtract(ode_Type3_Init, unknowns_Type3,
init_params_Type3);
    omega0_Type3 = unknowns0_Type3_Vals(genSVIndices(nPV, 1));
    Pm0_Type3 = unknowns0_Type3_Vals(genSVIndices(nPV, 2));
    theta0_Type3 = thetaVals;
    x0_Type3 = [theta0_Type3; omega0_Type3];


elseif strcmp(modelType, 'Type2') || strcmp(modelType, 'Type1')

    ode_omegaType2 = diff(omega) == ( 1./(2*H) ) .* (Pm - PG - K_D.*(omega - 1));
    ode_omegaType2_Equilibrium = 0 == rhs(ode_omegaType2(lPV));
    ode_omegaType2_Init = subs(ode_omegaType2_Equilibrium, PG, PGVals);
    ode_omegaType1_Init = subs(ode_omegaType2_Equilibrium, PG, PGVals.*(0.5 +
0.5*ViVals.^2));

    eqn_Id_PowerFlow = Id == abs(IGenVals).*sin(theta - delta_i_Vals);
    eqn_Id_PowerFlow_Init = 0 == lhs(eqn_Id_PowerFlow(lPV)) -
rhs(eqn_Id_PowerFlow(lPV));
    eqn_Iq_PowerFlow = Iq == abs(IGenVals).*cos(theta - delta_i_Vals);
    eqn_Iq_PowerFlow_Init = 0 == lhs(eqn_Iq_PowerFlow(lPV)) -
rhs(eqn_Iq_PowerFlow(lPV));

    eqn_Vq = Vq == ViVals.*cos(theta - delta_i_Vals);
    eqn_Vq_Init = 0 == lhs(eqn_Vq(lPV)) - rhs(eqn_Vq(lPV));

    eqn_Vd = Vd == ViVals.*sin(theta - delta_i_Vals);
    eqn_Vd_Init = 0 == lhs(eqn_Vd(lPV)) - rhs(eqn_Vd(lPV));

    eqn_Id_KVL = Id == (Eq_prime - Vq)./Xd_prime;
    eqn_Id_KVL_Init = 0 == lhs(eqn_Id_KVL(lPV)) - rhs(eqn_Id_KVL(lPV));

    eqn_Iq_KVL = Iq == (Ed_prime - Vd)./(-Xq_prime);
    eqn_Iq_KVL_Init = 0 == lhs(eqn_Iq_KVL(lPV)) - rhs(eqn_Iq_KVL(lPV));

    V_R_dot = (1./T_A).*( -V_R + k_A.*(Vref - V_i(listOfGenBuses)) );
    ode_VRType2 = diff(V_R) == V_R_dot;
    ode_VRType2_Equilibrium = 0 == rhs(ode_VRType2(lPV));
    ode_VRType2_Init = subs( ode_VRType2_Equilibrium, V_i(lPV), ViVals(lPV) );

    ode_EqprimeType2 = diff(Eq_prime) == (1./Td0_prime) .* ( -Eq_prime - (Xd -
Xd_prime).*Id + V_R );
    ode_EqprimeType2_Equilibrium = 0 == rhs(ode_EqprimeType2(lPV));
    ode_EqprimeType2_Init = ode_EqprimeType2_Equilibrium;
```

```
    ode_EdprimeType2 = diff(Ed_prime) == (1./Tq0_prime) .* ( -Ed_prime + (Xq -
Xq_prime).*Iq );
    ode_EdprimeType2_Equilibrium = 0 == rhs(ode_EdprimeType2(lPV));
    ode_EdprimeType2_Init = ode_EdprimeType2_Equilibrium;

    P_m_dotType2 = (1./Tsg) .* ( -Pm + Ksg.*( P_C - (1./R).*(omega - 1) ) );
    ode_PmType2 = diff(Pm) == P_m_dotType2;
    ode_PmType2_Equilibrium = 0 == rhs(ode_PmType2(lPV));
    ode_PmType2_Init = ode_PmType2_Equilibrium;

    ode_Type2_Init = [ode_thetaEquilibrium; ode_omegaType2_Init; ...
        ode_EqprimeType2_Init; ode_EdprimeType2_Init; ...
        ode_VRType2_Init; ode_PmType2_Init; ...
        eqn_Id_PowerFlow_Init; eqn_Iq_PowerFlow_Init; ...
        % eqn_Eq_prime_Init; eqn_Ed_prime_Init; ...
        eqn_Vq_Init; eqn_Vd_Init; ...
        eqn_Id_KVL_Init; eqn_Iq_KVL_Init];

    display(ode_Type2_Init)

    unknowns_Type2 = ...
        [theta(lPV); omega(lPV); ...
        Eq_prime(lPV); Ed_prime(lPV); ...
        V_R(lPV); Pm(lPV); ...
        Vref(lPV); P_C(lPV); ...
        Vq(lPV); Vd(lPV); ...
        Iq(lPV); Id(lPV)];

    init_params_Type2 = ...
        [delta_i_Vals(lPV); ones(nPV, 1); ...
        1.1*ViVals(lPV); 0.1*ViVals(lPV); ...
        1.8*ones(nPV, 1); PGVals(lPV); ...
        ViVals(lPV);  PGVals(lPV)./Ksg(lPV); ...
        1.0*ones(nPV, 1); 1.0*ones(nPV, 1); ...
        real(IGenVals(lPV)); imag(IGenVals(lPV))];

    unknowns0_Type2_Vals = solveAndExtract(ode_Type2_Init, unknowns_Type2,
init_params_Type2);
    unknowns0_Type2_Gens_Vals = sortByGenerators(unknowns0_Type2_Vals, nPV);
    namesUnknowns_Type2 = string([theta(lPV); omega(lPV); Eq_prime(lPV);
Ed_prime(lPV); V_R(lPV); Pm(lPV); Vref(lPV); P_C(lPV); Vq(lPV); Vd(lPV); Iq(lPV);
Id(lPV)]);
    namesUnknowns_Type2_Gens = sortByGenerators(namesUnknowns_Type2, nPV);
    unknowns0_Type2_Vals_Table = array2table(unknowns0_Type2_Vals, 'RowNames',
namesUnknowns_Type2, 'VariableNames', {'Values'});
    unknowns0_Type2_Gens_Vals_Table = array2table(unknowns0_Type2_Gens_Vals,
'RowNames', namesUnknowns_Type2_Gens, 'VariableNames', {'Values'});
    % display(unknowns0_Type2_Vals_Table);
    display(unknowns0_Type2_Gens_Vals_Table);
```

```
    theta0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 1));
    omega0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 2));
    Eq_prime0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 3));
    Ed_prime0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 4));
    V_R0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 5));
    Pm0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 6));

    Vref0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 7));
    P_C0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 8));

    Vq0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 9));
    Vd0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 10));
    Iq0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 11));
    Id0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 12));

    x0_Type2 = [theta0_Type2; omega0_Type2; Eq_prime0_Type2; Ed_prime0_Type2;
V_R0_Type2; Pm0_Type2];
    x0_Type2_Gens = sortByGenerators(x0_Type2, nPV);
    namesSVs = string([theta(lPV); omega(lPV); Eq_prime(lPV); Ed_prime(lPV);
V_R(lPV); Pm(lPV)]);
    namesSVs_Gens = sortByGenerators(namesSVs, nPV);
    x0_Type2Table = array2table(x0_Type2, 'RowNames', namesSVs, 'VariableNames',
{'Values'});
    x0_Type2_Gens_Table = array2table(x0_Type2_Gens, 'RowNames', namesSVs_Gens,
'VariableNames', {'Values'});
    % display(x0_Type2Table);
    display(x0_Type2_Gens_Table);
else
    error('This statement shouldn''t be reached.');
end
```

resultGen = 4×2 table

	E'	theta
1 Gen 1	1.0300	0
2 Gen 2	1.0745	0.1465
3 Gen 3	1.1392	-0.0383
4 Gen 4	1.1045	-0.2068

ode_Type3_Init =

$$\begin{pmatrix} 0 = 376.9911\,\omega_2 - 376.9911 \\ 0 = 376.9911\,\omega_3 - 376.9911 \\ 0 = 376.9911\,\omega_4 - 376.9911 \\ 0 = 0.0085\,P_{\mathrm{m2}} - 0.0171\,\omega_2 - 0.0427 \\ 0 = 0.0090\,P_{\mathrm{m3}} - 0.0180\,\omega_3 - 0.0467 \\ 0 = 0.0090\,P_{\mathrm{m4}} - 0.0180\,\omega_4 - 0.0450 \end{pmatrix}$$

Small Signal Stability Analysis

```
if strcmp(modelType, 'Type3')
    PG_Type3 = generateSymbolicPowerFlowEquations(nPV, PG, QG, yGen, E_primeVals,
EdgesGen, theta);
    ode_omegaType3_SS = subs(ode_omegaType3(lPV), PG, PG_Type3);
    ode_Type3_SS = [ode_thetaEquilibrium; ode_omegaType3_SS];
    display(ode_Type3_SS);
    x_Type3 = [theta(lPV); omega(lPV)];

    PG_computedUsingYGen = array2table(double(subs(PG_Type3, theta, theta0_Type3)),
'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    display(PG_computedUsingYGen);

    J_Type3 = jacobian(rhs(ode_Type3_SS), x_Type3);
    Je_Type3 = double(subs(J_Type3, [theta; omega(lPV)], x0_Type3));

    [VType3, DType3, WType3] = eig(Je_Type3);
    lambdas_Type3 = diag(DType3);
    display(lambdas_Type3);
    W_T_Type3 = WType3';

    for i = 1:size(VType3, 1)
        v_Type3_i = VType3(:, i);
        w_T_Type3_i = W_T_Type3(i, :);
        pMatrix_Type3 = v_Type3_i*w_T_Type3_i;
        pfactors_Type3 = abs(diag(pMatrix_Type3));
        pfactors_Type3_Normalized = pfactors_Type3./max(pfactors_Type3);
        fprintf('For mode λ = %f + 1i* %f', real(lambdas_Type3(i)),
imag(lambdas_Type3(i)));
        display(pfactors_Type3_Normalized);
    end

elseif strcmp(modelType, 'Type2')

    E_prime_Type2 = sqrt(Eq_prime.^2 + Ed_prime.^2);
    E_prime_Type2(1) = ViVals(1);

    gamma_Type2 = atan(Eq_prime./Ed_prime) + theta - pi/2;
    gamma_Type2(1) = delta_i_Vals(1);

    PG_Type2 = generateSymbolicPowerFlowEquations(nPV, PG, QG, yGen, E_prime_Type2,
EdgesGen, gamma_Type2);
    % display(PG_Type2)
    % PG_computedUsingYGen = array2table(double(subs(PG_Type2, [theta;
Eq_prime; Ed_prime], [ [0; theta0_Type2]; [ViVals(1); Eq_prime0_Type2]; [0;
Ed_prime0_Type2] ])), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    % display(PG_computedUsingYGen)
```

```
    [Id_Type2, Iq_Type2] = generate_dq_CurrentsFrom_dqVoltages(nPV, Id, Iq,
EdgesGen, yGen, theta, Eq_prime, Ed_prime);

    Id_computedUsingYGen = array2table(double(subs(Id_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [ theta0_Type2; Eq_prime0_Type2;
Ed_prime0_Type2] )), 'VariableNames', {'I_d'}, 'RowNames', namesGenBuses);
    Iq_computedUsingYGen = array2table(double(subs(Iq_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [ theta0_Type2; Eq_prime0_Type2;
Ed_prime0_Type2] )), 'VariableNames', {'I_q'}, 'RowNames', namesGenBuses);
    display(Iq_computedUsingYGen(lPV, :));
    display(Id_computedUsingYGen(lPV, :));
    display(Iq0_Type2);
    display(Id0_Type2);

    Vq_Type2 = V_i(listOfGenBuses) .* cos( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );
    Vd_Type2 = V_i(listOfGenBuses) .* sin( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );

    % V_i_Type2 = subs( (Ed_prime + Iq.*Xq_prime)./sin(theta - delta_i_Vals), Iq,
Iq_Type2);
    V_i_Type2 = subs( (Eq_prime - Id.*Xd_prime)./cos(theta - delta_i_Vals), Id,
Id_Type2);
    V_i_ComputedUsingFormula = array2table(double(subs(V_i_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [theta0_Type2; Eq_prime0_Type2; Ed_prime0_Type2] )),
'VariableNames', {'V_i'}, 'RowNames', namesGenBuses);
    display(V_i_ComputedUsingFormula(lPV, :));
    display(ViVals(lPV));

    % PG_Type2 = subs(Vd.*Id + Vq.*Iq, [Vd, Vq, Id, Iq], [Vd_Type2, Vq_Type2,
Id_Type2, Iq_Type2]);
    % PG_Type2 = subs(PG_Type2, V_i(listOfGenBuses), V_i_Type2);
    % PG_computedUsingFormula = array2table(double(subs(PG_Type2, [theta;
Eq_prime; Ed_prime], [ [0; theta0_Type2]; [ViVals(1); Eq_prime0_Type2]; [0;
Ed_prime0_Type2] ])), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    % display(PG_computedUsingFormula);

    % ode_thetaType2_SS = ode_thetaEquilibrium;
    ode_omegaType2_SS = subs(ode_omegaType2_Equilibrium, PG, PG_Type2);
    ode_EqprimeType2_SS = subs(ode_EqprimeType2_Equilibrium, Id, Id_Type2);
    % display(ode_EqprimeType2_SS);
    ode_EdprimeType2_SS = subs(ode_EdprimeType2_Equilibrium, Iq, Iq_Type2);
    % display(ode_EdprimeType2_SS);
    ode_VRType2_SS = subs(ode_VRType2_Equilibrium, [Vref(lPV); V_i(listOfGenBuses);
Iq], [Vref0_Type2; V_i_Type2; Iq_Type2]);
    % display(ode_VRType2_SS);
    ode_PmType2_SS = subs(ode_PmType2_Equilibrium, P_C(lPV), P_C0_Type2);
    % display(ode_PmType2_SS);
```

```
    ode_Type2_SS = [ode_thetaEquilibrium; ode_omegaType2_SS; ode_EqprimeType2_SS;
ode_EdprimeType2_SS; ode_VRType2_SS; ode_PmType2_SS];
    % display(ode_Type2_SS)

    x_Type2 = [theta(lPV); omega(lPV); Eq_prime(lPV); Ed_prime(lPV); V_R(lPV);
Pm(lPV)];
    x_Type2_Gens = sortByGenerators(x_Type2, nPV);
    % display(x_Type2_Gens);
    ode_Type2_SS_Gens = sortByGenerators(ode_Type2_SS, nPV);
    display(ode_Type2_SS_Gens)
    J_Type2 = jacobian(rhs(ode_Type2_SS), x_Type2);
    x0_Type2_Gens = sortByGenerators(x0_Type2, nPV);
    J_Type2_Gens = jacobian(rhs(ode_Type2_SS_Gens), x_Type2_Gens);
    % display(J_Type2)
    format shortG
    Je_Type2 = double(subs(J_Type2, x_Type2, x0_Type2));
    Je_Type2_Gens = double(subs(J_Type2_Gens, x_Type2_Gens, x0_Type2_Gens));
    % display(Je_Type2);
    Je_Type2_Table = array2table(Je_Type2, 'VariableNames', namesSVs, 'RowNames',
namesSVs);
    % display(Je_Type2_Table)
    format default
    % display(Je_Type2_Gens)
    Je_Type2_Gens_Table = array2table(Je_Type2_Gens, 'VariableNames',
namesSVs_Gens, 'RowNames', namesSVs_Gens);
    display(Je_Type2_Gens_Table)
    [VType2, DType2, WType2] = eig(Je_Type2);
    lambdasType2 = diag(DType2);
    [VType2_Gens, DType2_Gens, WType2_Gens] = eig(Je_Type2_Gens);
    lambdasType2_Gens = diag(DType2_Gens);
    format shortG
    display(lambdasType2);
    display(lambdasType2_Gens)
    format default
    W_T_Type2 = WType2';
else

    Vq_Type1 = V_i(listOfGenBuses) .* cos( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );
    Vd_Type1 = V_i(listOfGenBuses) .* sin( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );

    Id_Type1 = subs( (Eq_prime - Vq)./Xd_prime, Vq, Vq_Type1);
    Iq_Type1 = subs( (Ed_prime - Vd)./(-Xq_prime), Vd, Vd_Type1);

    PG_Type1 = subs(Vd.*Id + Vq.*Iq, [Vd, Vq, Id, Iq], [Vd_Type1, Vq_Type1,
Id_Type1, Iq_Type1]);
    QG_Type1 = subs(Vq.*Id - Vd.*Iq, [Vd, Vq, Id, Iq], [Vd_Type1, Vq_Type1,
Id_Type1, Iq_Type1]);
```

```
    ode_omegaType1_SS = subs(ode_omegaType2_Equilibrium, PG, PG_Type1);
    ode_EqprimeType1_SS = subs(ode_EqprimeType2_Equilibrium, Id, Id_Type1);
    ode_EdprimeType1_SS = subs(ode_EdprimeType2_Equilibrium, Iq, Iq_Type1);
    ode_VRType1_SS = subs(ode_VRType2_Equilibrium, Vref(lPV), Vref0_Type2);
    ode_PmType1_SS = subs(ode_PmType2_Equilibrium, P_C(lPV), P_C0_Type2);

    ode_Type1_SS = [ode_thetaEquilibrium; ode_omegaType1_SS; ode_EqprimeType1_SS;
ode_EdprimeType1_SS; ode_VRType1_SS; ode_PmType1_SS];
    % display(ode_Type1_SS)

    IL = sym('I_L', [1, 1]);
    delta_IL1 = sym('I_L1', [1, 1]);
    delta_IL2 = sym('I_L2', [1, 1]);
    V_PSS = sym('V_PSS', [4, 1]);

    ode_deltaIL1_Equilibrium = 0 == (1./Twpss) .* ( -delta_IL1 + Kwpss*Kcomp.*IL );
    ode_deltaIL2_SS = subs( ode_deltaIL1_Equilibrium, IL, -ybus(7, 8).*( V_i(7) -
V_i(8) ) );
    % ode_deltaIL2 = 0 == (1./Tbcomp1) .* ( -delta_IL2 + delta_IL1 +
Tacomp1*diff(delta_IL1) );
    ode_deltaIL2_Equilibrium = 0 == (1./Tbcomp1) .* ( -delta_IL2 + delta_IL1  +
Tacomp1 .* rhs(ode_deltaIL1_Equilibrium) );
    % ode_deltaIL2_SS = subs(ode_deltaIL2_Equilibrium, )
    % ode_VPSS = 0 == (1./Tbcomp2) .* ( -V_PSS + delta_IL2 +
Tacomp2.*diff(delta_IL2) );
    ode_VPSS_Equilibrium = 0 == (1./Tbcomp2) .* ( -V_PSS + delta_IL2 + Tacomp2 .*
rhs(ode_deltaIL2_Equilibrium) );


    x_Type1 = x_Type2;
    x0_Type1 = x0_Type2;
    % x_Type1_Gens = sortByGenerators(x_Type1, nPV);
    % x0_Type1_Gens = sortByGenerators(x0_Type1, nPV);

    y_Type1 = [delta_i(2:N); V_i(2:N)];
    % y_Type1_Gens = sortByGenerators(y_Type1, N-1);
    y0_Type1 = [delta(2:N); V(2:N)];
    % y0_Type1_Gens = sortByGenerators(y0_Type1, N-1);

    % ode_Type1_SS_Gens = sortByGenerators(ode_Type1_SS, nPV);

    A_Type1 = jacobian(rhs(ode_Type1_SS), x_Type1);
    % display(A_Type1)
    format shortG
    Ae_Type1 = double( subs(A_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]) );
    % display(Ae_Type1);
    % Ae_Type1_Table = array2table(Ae_Type1, 'VariableNames', namesSVs, 'RowNames',
namesSVs);
    % display(Ae_Type1_Table)
    format default
```

```
    B_Type1 = jacobian(rhs(ode_Type1_SS), y_Type1);
    Be_Type1 = double(subs(B_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));
    % display(Be_Type1);
    % Be_Type1_Table = array2table(Be_Type1, 'VariableNames', namesSVs, 'RowNames',
namesSVs);

    [P_i_Type1, Q_i_Type1] = generateSymbolicPowerFlowEquations(N-1, P_i, Q_i,
ybus, V_i, E, delta_i);

    P_L_Type1 = busData.PL/MVAb .* (0.5 + 0.5*V_i.^2);
    Q_L_Type1 = busData.QL/MVAb .* (0.5 + 0.5*V_i.^2);

    g = sym('g', [2*(N-1), 1]);
    % g(lPV-1) = subs( PG(lPV) - busData.PL(lPV)/MVAb - P_i(lPV), [PG; P_i],
[PG_Type1; P_i_Type1]);
    g(lPV-1) = subs( PG(lPV) - P_L(lPV) - P_i(lPV), [PG; P_L; P_i], [PG_Type1;
P_L_Type1; P_i_Type1]);

    % g(lPQ-1) = subs( -busData.PL(lPQ)/MVAb - P_i(lPQ), P_i, P_i_Type1);
    g(lPQ-1) = subs( -P_L(lPQ) - P_i(lPQ), [P_L; P_i], [P_L_Type1; P_i_Type1]);

    % g(lPV-1+N-1) = subs(QG(lPV) - busData.QL(lPV)/MVAb - Q_i(lPV), [QG; Q_i],
[QG_Type1; Q_i_Type1]);
    g(lPV-1+N-1) = subs( QG(lPV) - Q_L(lPV) - Q_i(lPV), [QG; Q_L; Q_i], [QG_Type1;
Q_L_Type1; Q_i_Type1]);

    % g(lPQ-1+N-1) = subs( -busData.QL(lPQ)/MVAb - Q_i(lPQ), Q_i, Q_i_Type1);
    g(lPQ-1+N-1) = subs( -Q_L(lPQ) - Q_i(lPQ), [Q_L; Q_i], [Q_L_Type1; Q_i_Type1]);

    C_Type1 = jacobian(g, x_Type1);
    Ce_Type1 = double(subs(C_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));

    D_Type1 = jacobian(g, y_Type1);
    De_Type1 = double(subs(D_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));

    Je_Type1 = Ae_Type1 - Be_Type1/De_Type1 * Ce_Type1;
    % display(Je_Type1);

    [V_Type1, D_Type1, W_Type1] = eig(Je_Type1);
    lambdas_Type1 = diag(D_Type1);
    display(lambdas_Type1);

    W_T_Type1 = W_Type1';

    for i = 1:size(V_Type1, 1)
        v_Type1_i = V_Type1(:, i);
        w_T_Type1_i = W_T_Type1(i, :);
        pMatrix_Type1 = v_Type1_i*w_T_Type1_i;
        pfactors_Type1 = diag(pMatrix_Type1);
```

```
        pfactors_Type1_Normalized = pfactors_Type1./max(abs(pfactors_Type1));
        fprintf('For mode λ = %f + 1i* %f', real(lambdas_Type1(i)),
imag(lambdas_Type1(i)));
        display(pfactors_Type1_Normalized);
    end

end
```

ode_Type3_SS =

$$
\begin{cases}
0 = 376.9911\,\omega_2 - 376.9911 \\
0 = 376.9911\,\omega_3 - 376.9911 \\
0 = 376.9911\,\omega_4 - 376.9911 \\
0 = 0.0085\,P_{m2} - 0.0171\,\omega_2 - 0.0127\cos(\theta_4 - \theta_2 + 0.7246) - 0.0094\cos(\theta_3 - \theta_2 + 0.7527) - 0.0769\cos \\
0 = 0.0090\,P_{m3} - 0.0180\,\omega_3 - 0.0526\cos(\theta_4 - \theta_3 + 1.1326) - 0.0099\cos(\theta_2 - \theta_3 + 0.7527) - 0.0146\cos \\
0 = 0.0090\,P_{m4} - 0.0180\,\omega_4 - 0.0526\cos(\theta_3 - \theta_4 + 1.1326) - 0.0134\cos(\theta_2 - \theta_4 + 0.7246) - 0.0197\cos
\end{cases}
$$

PG_computedUsingYGen = 4×1 table

	P_G
1 Gen 1	6.9143
2 Gen 2	6.9998
3 Gen 3	7.1904
4 Gen 4	7.0018

```
lambdas_Type3 = 6×1 complex
  -0.0090 + 6.6736i
  -0.0090 - 6.6736i
  -0.0086 + 5.6071i
  -0.0086 - 5.6071i
  -0.0090 + 2.8447i
  -0.0090 - 2.8447i
For mode λ = -0.008999 + 1i* 6.673622
pfactors_Type3_Normalized = 6×1
    0.0024
    0.6885
    1.0000
    0.0024
    0.6885
    1.0000
For mode λ = -0.008999 + 1i* -6.673622
pfactors_Type3_Normalized = 6×1
    0.0024
    0.6885
    1.0000
    0.0024
    0.6885
    1.0000
For mode λ = -0.008562 + 1i* 5.607053
pfactors_Type3_Normalized = 6×1
    1.0000
    0.0227
    0.0044
    1.0000
    0.0227
    0.0044
For mode λ = -0.008562 + 1i* -5.607053
pfactors_Type3_Normalized = 6×1
```

```
    1.0000
    0.0227
    0.0044
    1.0000
    0.0227
    0.0044
For mode λ = -0.008989 + 1i* 2.844733
pfactors_Type3_Normalized = 6×1
    0.0437
    1.0000
    0.7086
    0.0437
    1.0000
    0.7086
For mode λ = -0.008989 + 1i* -2.844733
pfactors_Type3_Normalized = 6×1
    0.0437
    1.0000
    0.7086
    0.0437
    1.0000
    0.7086
```

Transient Stability Analysis

Need to fix it to take modelType as argument (currently assumes that only Type 3 is running).

```
if runTransientSimulation
    transientSimulationScript(clearingCycles, saveTransientRunValues,
saveTransientRunPlots, modelType, nPV, PG, QG, yGenFaultOn, E_primeVals, EdgesGen,
theta, yGenPostFault, ode_omegaType3, lPV, Pm, PGVals, ode_thetaEquilibrium,
numStepsFaultOn, numStepsPostFault, x0_Type3, omega, h, folder_processedData,
systemName1);
end
```

Have a nice day!

In case you encounter a Java Heap Memory error, delete the above gif, or go to `Preferences->General->Java Heap Memory` and increase the allocated size.