APÉNDICES

Apéndice I

Formato de Archivos de Datos en MatPower

Dentro de los archivos de casos mostrados en MatPower se pueden identificar los atributos de los buses, ramas y generadores en las estructuras que se muestran a continuación:

Data de buses

name	column	description
BUS_I	1	bus number (positive integer)
BUS_TYPE	2	bus type (1 = PQ, 2 = PV, 3 = ref, 4 = isolated)
PD	3	real power demand (MW)
QD	4	reactive power demand (MVAr)
GS	5	shunt conductance (MW demanded at $V = 1.0$ p.u.)
BS	6	shunt susceptance (MVAr injected at $V = 1.0$ p.u.)
BUS_AREA	7	area number (positive integer)
VM	8	voltage magnitude (p.u.)
VA	9	voltage angle (degrees)
BASE_KV	10	base voltage (kV)
ZONE	11	loss zone (positive integer)
VMAX	12	maximum voltage magnitude (p.u.)
VMIN	13	minimum voltage magnitude (p.u.)
LAM_P†	14	Lagrange multiplier on real power mismatch (u/MW)
LAM_Q†	15	Lagrange multiplier on reactive power mismatch (u/MVAr)
MU_VMAX†	16	Kuhn-Tucker multiplier on upper voltage limit (u/p.u.)
MU_VMIN†	17	Kuhn-Tucker multiplier on lower voltage limit (u/p.u.)

† Included in OPF output, typically not included (or ignored) in input matrix. Here we assume the objective function has units u.

Data de Generadores

name	column	description
GEN_BUS	1	bus number
PG	2	real power output (MW)
QG	3	reactive power output (MVAr)
QMAX	4	maximum reactive power output (MVAr)
QMIN	5	minimum reactive power output (MVAr)
VG	6	voltage magnitude setpoint (p.u.)
MBASE	7	total MVA base of machine, defaults to baseMVA
GEN_STATUS	8	machine status, > 0 = machine in-service; ≤ 0 = machine out-of-service
PMAX	9	maximum real power output (MW)
PMIN	10	minimum real power output (MW)
PC1*	11	lower real power output of PQ capability curve (MW)
PC2*	12	upper real power output of PQ capability curve (MW)
QC1MIN*	13	minimum reactive power output at PC1 (MVAr)
QC1MAX*	14	maximum reactive power output at PC1 (MVAr)
QC2MIN*	15	minimum reactive power output at PC2 (MVAr)
QC2MAX*	16	maximum reactive power output at PC2 (MVAr)
RAMP_AGC*	17	ramp rate for load following/AGC (MW/min)
RAMP_10*	18	ramp rate for 10 minute reserves (MW)
RAMP_30*	19	ramp rate for 30 minute reserves (MW)
RAMP_Q*	20	ramp rate for reactive power (2 sec timescale) (MVAr/min)
APF*	21	area participation factor
MU_PMAX†	22	Kuhn-Tucker multiplier on upper P_g limit (u/MW)
MU_PMIN†	23	Kuhn-Tucker multiplier on lower P_g limit (u/MW)
MU_QMAX†	24	Kuhn-Tucker multiplier on upper Q_g limit (u/MVAr)
MU_QMIN†	25	Kuhn-Tucker multiplier on lower Q_g limit (u/MVAr)

* Not included in version 1 case format.
† Included in OPF output, typically not included (or ignored) in input matrix. Here we assume the objective function has units u.

Data de Ramas

name	column	description
F_BUS	1	"from" bus number
T_BUS	2	"to" bus number
BR_R	3	resistance (p.u.)
BR_X	4	reactance (p.u.)
BR_B	5	total line charging susceptance (p.u.)
RATE_A	6	MVA rating A (long term rating)
RATE_B	7	MVA rating B (short term rating)
RATE_C	8	MVA rating C (emergency rating)
TAP	9	transformer off nominal turns ratio, (taps at "from" bus, impedance at "to" bus, i.e. if $r = x = 0$, $tap = \frac{\|V_f\|}{\|V_t\|}$)
SHIFT	10	transformer phase shift angle (degrees), positive $\Rightarrow$ delay
BR_STATUS	11	initial branch status, 1 = in-service, 0 = out-of-service
ANGMIN*	12	minimum angle difference, $\theta_f - \theta_t$ (degrees)
ANGMAX*	13	maximum angle difference, $\theta_f - \theta_t$ (degrees)
PF†	14	real power injected at "from" bus end (MW)
QF†	15	reactive power injected at "from" bus end (MVAr)
PT†	16	real power injected at "to" bus end (MW)
QT†	17	reactive power injected at "to" bus end (MVAr)
MU_SF‡	18	Kuhn-Tucker multiplier on MVA limit at "from" bus (u/MVA)
MU_ST‡	19	Kuhn-Tucker multiplier on MVA limit at "to" bus (u/MVA)
MU_ANGMIN‡	20	Kuhn-Tucker multiplier lower angle difference limit (u/degree)
MU_ANGMAX‡	21	Kuhn-Tucker multiplier upper angle difference limit (u/degree)

* Not included in version 1 case format. The voltage angle difference is taken to be unbounded below if ANGMIN < -360 and unbounded above if ANGMAX > 360. If both parameters are zero, the voltage angle difference is unconstrained.
† Included in power flow and OPF output, ignored on input.
‡ Included in OPF output, typically not included (or ignored) in input matrix. Here we assume the objective function has units u.

Apéndice II

Datos del sistema de la IEEE de nueve barras

En las tablas siguientes se muestran los datos del sistema de la IEEE de nueve barras:

Tabla II.1 Carga

Bus	MW	MVAR
5	125	50
6	90	30
8	100	35

Tabla II.2 Líneas y transformadores

Bus	Bus	R	X	½ B
1	4	0	0.0576	0
4	6	0.017	0.092	0.079
4	5	0.010	0.085	0.088
6	9	0.039	0.170	0.179
3	9	0	0.0586	0
8	9	0.0119	0.1008	0.1045
7	8	0.085	0.072	0.0745
2	7	0	0.0625	0
5	7	0.032	0.161	0.153

Tabla II.3 Programación de Generación

Bus	Magnitud V	Generación MW	Mvar Min	Mvar Max
1	1.06	-	-	-
2	1.04	150	0	140
3	1.03	100	0	190

Tabla II.4 Datos de las Máquinas

Gen	R_a	X_d'	H
1	0	0.0608	23.64
2	0	0.1198	6.40
3	0	0.1813	3.01

Apéndice III

Programa MatDyn Modificado para el estudio de estabilidad debido a fallas monofásicas

```
function [Angles,Speeds,Eq_tr,Ed_tr,Efd,PM,Voltages,Stepsize,Errest,Time] =
rundynmodf(casefile_pf, casefile_dyn, casefile_ev,casefilesec0,casefilesec2, mdopt)

% [Angles,Speeds,Eq_tr,Ed_tr,Efd,PM,Voltages,Stepsize,Errest,Time] =
% rundyn(casefile_pf, casefile_dyn, casefile_ev, mdopt)
%
% Runs dynamic simulation
%
% INPUTS
% casefile_pf = m-file with power flow data
% casefile_dyn = m-file with dynamic data
% casefile_ev = m-file with event data
% casefilesec0 = m-file con data de secuencia cero
% casefilesec2 = m-file con data de secuencia negativa


% mdopt = options vector
%
% OUTPUTS
% Angles = generator angles
% Speeds = generator speeds
% Eq_tr = q component of transient voltage behind reactance
% Ed_tr = d component of transient voltage behind reactance
% Efd = Excitation voltage
% PM = mechanical power
% Voltages = bus voltages
% Stepsize = step size integration method
% Errest = estimation of integration error
% Failed = failed steps
% Time = time points

% MatDyn
% Copyright (C) 2009 Stijn Cole
% Katholieke Universiteit Leuven
% Dept. Electrical Engineering (ESAT), Div. ELECTA
% Kasteelpark Arenberg 10
% 3001 Leuven-Heverlee, Belgium

%% Begin timing
tic;
%% Add subdirectories to path

addpath([cd '/Solvers/']);
addpath([cd '/Models/Generators/']);
addpath([cd '/Models/Exciters/']);
addpath([cd '/Models/Governors/']);
addpath([cd '/Cases/Powerflow/']);
addpath([cd '/Cases/Dynamic/']);
addpath([cd '/Cases/Events/']);


%% define named indices into bus, gen, branch matrices
[PQ, PV, REF, NONE, BUS_I, BUS_TYPE, PD, QD, GS, BS, BUS_AREA, VM, ...
    VA, BASE_KV, ZONE, VMAX, VMIN, LAM_P, LAM_Q, MU_VMAX, MU_VMIN] = idx_bus;
[F_BUS, T_BUS, BR_R, BR_X, BR_B, RATE_A, RATE_B, RATE_C, ...
    TAP, SHIFT, BR_STATUS, PF, QF, PT, QT, MU_SF, MU_ST, ...
    ANGMIN, ANGMAX, MU_ANGMIN, MU_ANGMAX] = idx_brch;
[GEN_BUS, PG, QG, QMAX, QMIN, VG, MBASE, GEN_STATUS, PMAX, PMIN, ...
    MU_PMAX, MU_PMIN, MU_QMAX, MU_QMIN, PC1, PC2, QC1MIN, QC1MAX, ...
    QC2MIN, QC2MAX, RAMP_AGC, RAMP_10, RAMP_30, RAMP_Q, APF] = idx_gen;
```

```
%% Options
if nargin < 6
    mdopt = Mdoption;
end
method = mdopt(1);
tol = mdopt(2);
minstepsize = mdopt(3);
maxstepsize = mdopt(4);
output = mdopt(5);
plots = mdopt(6);

%% Load all data

% Load dynamic simulation data
if output; disp('> Loading dynamic simulation data...'); end
global freq
[freq,stepsize,stoptime] = Loaddyn(casefile_dyn);

% Load generator data
Pgen0 = Loadgen(casefile_dyn, output);

% Load exciter data
Pexc0 = Loadexc(casefile_dyn);

% Load governor data
Pgov0 = Loadgov(casefile_dyn);

% Load event data
if ~isempty(casefile_ev)
    [event,buschange,linechange] = Loadeventsmod(casefile_ev)
pause
else
    event=[];
end


genmodel = Pgen0(:,1);
excmodel = Pgen0(:,2);
govmodel = Pgen0(:,3);
pf=buschange(:,2); % vector que indica las barras donde ocurren los eventos


%% Initialization: Power Flow

% Power flow options
mpopt=mpoption;
% mpopt(31)=0;
% mpopt(32)=0;

% Run power flow
[baseMVA, bus, gen, branch, success] = runpf(casefile_pf,mpopt);
if ~success
    fprintf('> Error: Power flow did not converge. Exiting...\n')
    return;
else
    if output;
fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b>
Power flow converged\n'); end
end

U0=bus(:,VM).*(cos(bus(:,VA)*pi/180) + j.*sin(bus(:,VA)*pi/180));
U00=U0;
% Get generator info
on = find(gen(:, GEN_STATUS) > 0);      %% which generators are on?
gbus = gen(on, GEN_BUS);                %% what buses are they at?
ngen = length(gbus);
```

```
nbus = length(U0);


%% Construct augmented Ybus
if output; disp('> Constructing augmented admittance matrix...'); end
Pl=bus(:,PD)./baseMVA;                  %% load power
Ql=bus(:,QD)./baseMVA;

% Se toman los datos de secuencia positiva de los generadores
xd_tr = zeros(ngen,1);
xd_tr(genmodel==2) = Pgen0(genmodel==2,12); % 4th order model: xd_tr column 8
xd_tr(genmodel==1) = Pgen0(genmodel==1,12); % classical model: xd_tr column 7
[Ly, Uy, Py] = AugYbus(baseMVA, bus, branch, xd_tr, gbus, Pl, Ql, U0); % Calculo de la
matriz aumentada de secuencia positiva


% Se toman los datos de secuencia negativa de los generadores
xd_tr2 = zeros(ngen,1);
xd_tr2(genmodel==2) = Pgen0(genmodel==2,14); % 4th order model: xd_tr column 8
xd_tr2(genmodel==1) = Pgen0(genmodel==1,14); % classical model: xd_tr column 7

% Se toman los datos de secuencia cero de los generadores
xd_tr0 = zeros(ngen,1);
xd_tr0(genmodel==2) = Pgen0(genmodel==2,13); % 4th order model: xd_tr column 8
xd_tr0(genmodel==1) = Pgen0(genmodel==1,13); % classical model: xd_tr column 7


%Introduccion de datos de secuencia negativa
P2=bus(:,PD)./baseMVA;                  %% load power
Q2=bus(:,QD)./baseMVA;
mpc2 = loadcase(casefilesec2);
mpc2 = ext2int(mpc2);
[baseMVA, bus2, gen2, branch2] = deal(mpc2.baseMVA, mpc2.bus, mpc2.gen, mpc2.branch);
[Ly2, Uy2, Py2] = AugYbus(baseMVA,bus2,branch2,xd_tr2,gbus,P2,Q2,U0); % Calculo de la
matriz aumentada de secuencia negativa

Z2 = Zsecth(Ly2,Uy2,pf,nbus); %Impedancia equvalente de secuencia negativa en el punto
de falla

%Introduccion de datos de secuencia cero
P0=bus(:,PD)./baseMVA;                  %% load power
Q0=bus(:,QD)./baseMVA;
mpc0 = loadcase(casefilesec0);
mpc0 = ext2int(mpc0);
[baseMVA, bus0, gen0, branch0] = deal(mpc0.baseMVA, mpc0.bus, mpc0.gen, mpc0.branch);
[Ly0, Uy0, Py0] = AugYbusSec0(baseMVA,bus0,branch0,xd_tr0,gbus,P0,Q0,U0); % Calculo de
la matriz aumentada de secuencia cero

Z0 = Zsecth(Ly0,Uy0,pf,nbus); %Impedancia equvalente de secuencia cero en el punto de
falla

Ztheq=Z0+Z2; % suma de las impedancias de thevenin de secuencia negativa y cero

Rp=(abs(Ztheq)^2)/real(Ztheq); % Convierte las resistencia serie equivalente a
paralelo
Xp=(abs(Ztheq)^2)/imag(Ztheq); % Convierte las reactancia serie equivalente a paralelo


%% Calculate Initial machine state
if output;
fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\
b\b\b\b\b\b\b\b> Calculating initial state...\n'); end
[Efd0, Xgen0] = GeneratorInit(Pgen0, U0(gbus), gen, baseMVA, genmodel);

omega0 = Xgen0(:,2);

[Id0,Iq0,Pe0] = MachineCurrents(Xgen0, Pgen0, U0(gbus), genmodel);
Vgen0 = [Id0, Iq0, Pe0];
```

```
%% Exciter initial conditions

Vexc0 = [abs(U0(gbus))];
[Xexc0,Pexc0] = ExciterInit(Efd0, Pexc0, Vexc0, excmodel);


%% Governor initial conditions

Pm0 = Pe0;
[Xgov0, Pgov0] = GovernorInit(Pm0, Pgov0, omega0, govmodel);
Vgov0 = omega0;


%% Check Steady-state

Fexc0 = Exciter(Xexc0, Pexc0, Vexc0, excmodel);
Fgov0 = Governor(Xgov0, Pgov0, Vgov0, govmodel);
Fgen0 = Generator(Xgen0, Xexc0, Xgov0, Pgen0, Vgen0, genmodel);

% Check Generator Steady-state
if sum(sum(abs(Fgen0))) > 1e-6
    fprintf('> Error: Generator not in steady-state\n> Exiting...\n')
    return;
end
% Check Exciter Steady-state
if sum(sum(abs(Fexc0))) > 1e-6
    fprintf('> Error: Exciter not in steady-state\n> Exiting...\n')
    return;
end
% Check Governor Steady-state
if sum(sum(abs(Fgov0))) > 1e-6
    fprintf('> Error: Governor not in steady-state\n> Exiting...\n')
    return;
end

if output; fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b>
System in steady-state\n'); end


%% Initialization of main stability loop
t=-0.02; % simulate 0.02s without applying events
errest=0;
failed=0;
eulerfailed = 0;

if method==3 || method==4
    stepsize = minstepsize;
end


if ~output
    fprintf('                    ')
end

ev=1;
% if event(:,2)==3
% buschange(ev,4)=1/Xp;
% buschange(ev,6)=1/Rp;
% end
eventhappened = false;
i=0;


%% Allocate memory for variables

if output; fprintf('> Allocate memory..'); end
```

```
chunk = 5000;

Time = zeros(chunk,1); Time(1,:) = t;
Errest = zeros(chunk,1); Errest(1,:) = errest;
Stepsize = zeros(chunk,1); Stepsize(1,:) = stepsize;

% System variables
Voltages = zeros(chunk, length(U0)); Voltages(1,:) = U0.';

% Generator
Angles = zeros(chunk,ngen); Angles(1,:) = Xgen0(:,1).*180./pi;
Speeds = zeros(chunk,ngen); Speeds(1,:) = Xgen0(:,2)./(2.*pi.*freq);
Eq_tr = zeros(chunk,ngen); Eq_tr(1,:) = Xgen0(:,3);
Ed_tr = zeros(chunk,ngen); Ed_tr(1,:) = Xgen0(:,4);

% Exciter and governor
Efd = zeros(chunk,ngen); Efd(1,:) = Efd0(:,1);
PM = zeros(chunk,ngen); PM(1,:) = Pm0(:,1);


%% Main stability loop
while t < stoptime + stepsize

    %% Output
    i=i+1;
    if mod(i,45)==0 && output
       fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b> %6.2f%% completed',
t/stoptime*100)
    end

    %% Numerical Method
    switch method
        case 1
            [Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0, t,
newstepsize] = ModifiedEuler(t, Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0,
Pgov0, Vgov0, Ly, Uy, Py, gbus, genmodel, excmodel, govmodel, stepsize);
        case 2
            [Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0, t,
newstepsize] = RungeKutta(t, Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0,
Vgov0, Ly, Uy, Py, gbus, genmodel, excmodel, govmodel, stepsize);
        case 3
            [Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0,
errest, failed, t, newstepsize] = RungeKuttaFehlberg(t, Xgen0, Pgen0, Vgen0, Xexc0,
Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0, Ly, Uy, Py, gbus, genmodel, excmodel, govmodel,
tol, maxstepsize, stepsize);
        case 4
            [Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0,
errest, failed, t, newstepsize] = RungeKuttaHighamHall(t, Xgen0, Pgen0, Vgen0, Xexc0,
Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0, Ly, Uy, Py, gbus, genmodel, excmodel, govmodel,
tol, maxstepsize, stepsize);
        case 5
            [Xgen0, Pgen0, Vgen0, Xexc0, Pexc0, Vexc0, Xgov0, Pgov0, Vgov0, U0, t,
eulerfailed, newstepsize] = ModifiedEuler2(t, Xgen0, Pgen0, Vgen0, Xexc0, Pexc0,
Vexc0, Xgov0, Pgov0, Vgov0, Ly, Uy, Py, gbus, genmodel, excmodel, govmodel, stepsize);
    end

    if eulerfailed
        fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b> Error: No solution found. Try
lowering tolerance or increasing maximum number of iterations in ModifiedEuler2.
Exiting... \n')
        return;
    end

    if failed
        t = t-stepsize;
    end

    % End exactly at stop time
```

```
    if t + newstepsize > stoptime
        newstepsize = stoptime - t;
    elseif stepsize < minstepsize
        fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b> Error: No solution found with
minimum step size. Exiting... \n')
        return;
    end;


    %% Allocate new memory chunk if matrices are full
    if i>size(Time,1)
        Stepsize = [Stepsize; zeros(chunk,1)];Errest = [Errest; zeros(chunk,1)];Time =
[Time; zeros(chunk,1)];
        Voltages = [Voltages; zeros(chunk,length(U0))];Efd = [Efd;
zeros(chunk,ngen)];PM = [PM; zeros(chunk,ngen)];

Angles=[Angles;zeros(chunk,ngen)];Speeds=[Speeds;zeros(chunk,ngen)];Eq_tr=[Eq_tr;zeros
(chunk,ngen)];Ed_tr=[Ed_tr;zeros(chunk,ngen)];
    end


    %% Save values
    Stepsize(i,:) = stepsize.';
    Errest(i,:) = errest.';
    Time(i,:) = t;

    Voltages(i,:) = U0.';

    % exc
    Efd(i,:) = Xexc0(:,1).*(genmodel>1); % Set Efd to zero when using classical
generator model

    % gov
    PM(i,:) = Xgov0(:,1);

    % gen
    Angles(i,:) = Xgen0(:,1).*180./pi;
    Speeds(i,:) = Xgen0(:,2)./(2.*pi.*freq);
    Eq_tr(i,:) = Xgen0(:,3);
    Ed_tr(i,:) = Xgen0(:,4);


    %% Adapt step size if event will occur in next step
    if ~isempty(event) && ev <= size(event,1) && (method == 3 || method == 4)
        if t + newstepsize >= event(ev,1)
            if event(ev,1) - t < newstepsize
                newstepsize = event(ev,1) - t;
            end
        end
    end


    %% Check for events
    if ~isempty(event) && ev <= size(event,1)

        for k=ev:size(event,1) % cycle through all events ..


            if abs(t-event(ev,1))>10*eps ||  ev > size(event,1) %.. that happen on
time t
                break;
            else
                eventhappened = true;
            end

                switch event(ev,2)

                    case 1
```

```
                        bus(buschange(ev,2),buschange(ev,3)) = buschange(ev,4); %Falla
trifasica en bus
                    case 2
                        branch(linechange(ev,2),linechange(ev,3)) = linechange(ev,4);
%Salida de linea de forma trifasica
                    case 3 % Se incluye evento tipo falla monofasica a tierra
                           bus(buschange(ev,2),buschange(ev,3)) = baseMVA/Rp
                           bus(buschange(ev,2),buschange(ev,5)) = -baseMVA/Xp
                    case 4   %Se cambia atributos del bus para quitar la falla
monofasica
                            bus(buschange(ev,2),buschange(ev,3)) = buschange(ev,4);
                            bus(buschange(ev,2),buschange(ev,5)) = buschange(ev,6);

                end
                ev=ev+1;
        end

        if eventhappened
            % Refactorise
            [Ly, Uy, Py] = AugYbus(baseMVA, bus, branch, xd_tr, gbus,
bus(:,PD)./baseMVA, bus(:,QD)./baseMVA, U00);
            U0 = SolveNetwork(Xgen0, Pgen0, Ly, Uy, Py, gbus, genmodel);

            [Id0,Iq0,Pe0] = MachineCurrents(Xgen0, Pgen0, U0(gbus), genmodel);
            Vgen0 = [Id0,Iq0,Pe0];
            Vexc0 = abs(U0(gbus));

            % decrease stepsize after event occured
            if method==3 || method==4
                newstepsize = minstepsize;
            end

            i=i+1; % if event occurs, save values at t- and t+

            %% Save values
            Stepsize(i,:) = stepsize.';
            Errest(i,:) = errest.';
            Time(i,:) = t;

            Voltages(i,:) = U0.';

            % exc
            Efd(i,:) = Xexc0(:,1).*(genmodel>1); % Set Efd to zero when using
classical generator model

            % gov
            PM(i,:) = Xgov0(:,1);

            % gen
            Angles(i,:) = Xgen0(:,1).*180./pi
            Speeds(i,:) = Xgen0(:,2)./(2.*pi.*freq);
            Eq_tr(i,:) = Xgen0(:,3);
            Ed_tr(i,:) = Xgen0(:,4);

            eventhappened = false;
        end
    end

    %% Advance time
    stepsize = newstepsize;
    t = t + stepsize;

end % end of main stability loop


%% Output
if output
    fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b> 100%% completed')
```

```
else
    fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b')
end
simulationtime=toc;
if output; fprintf('\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b> Simulation completed in %5.2f seconds\n', simulationtime); end

% Save only the first i elements
[m,n]=size(Angles);

Angles2 = Angles(1:i,:);
Angles = Angles(1:i,:)-Angles(1:i,1)*ones(1,n); % Datos de los ángulos con referencia al generador principal.
Speeds = Speeds(1:i,:);
Eq_tr = Eq_tr(1:i,:);
Ed_tr = Ed_tr(1:i,:);

Efd = Efd(1:i,:);
PM = PM(1:i,:);

Voltages = Voltages(1:i,:);

Stepsize = Stepsize(1:i,:);
Errest = Errest(1:i,:);
Time = Time(1:i,:);

%% Clean up
rmpath([cd '/Solvers/']);
rmpath([cd '/Models/Generators/']);
rmpath([cd '/Models/Exciters/']);
rmpath([cd '/Models/Governors/']);
rmpath([cd '/Cases/Powerflow/']);
rmpath([cd '/Cases/Dynamic/']);
rmpath([cd '/Cases/Events/']);

%% Plot
if plots

close all

figure
xlabel('Time [s]')
ylabel('Angle [deg]')
hold on
plot(Time,Angles)
axis([0 Time(end) -1 1])
axis 'auto y'
grid on

aux=load('E:/dataasyp4.txt');
plot(aux(:,3),aux(:,4),'b')
hold on
plot(aux(:,3),aux(:,6),'b')
hold on


figure
xlabel('Time [s]')
ylabel('Angle [deg]')
hold on
plot(Time,Angles2) % Grafica de los ángulos con referencia al generador principal.
axis([0 Time(end) -1 1])
axis 'auto y'
grid on

figure
xlabel('Time [s]')
ylabel('Speed [pu]')
```

```
hold on
plot(Time,Speeds)
axis([0 Time(end) -1 1])
axis 'auto y'

% figure
% xlabel('Time [s]')
% ylabel('Voltage [pu]')
% hold on
% plot(Time,abs(Voltages))
% axis([0 Time(end) -1 1])
% axis 'auto y'
%
% figure
% xlabel('Time [s]')
% ylabel('Excitation voltage [pu]')
% hold on
% plot(Time,Efd)
% axis([0 Time(end) -1 1])
% axis 'auto y'
%
% figure
% xlabel('Time [s]')
% ylabel('Turbine Power [pu]')
% hold on
% plot(Time,PM)
% axis([0 Time(end) -1 1])
% axis 'auto y'
%
% figure
% hold on
% xlabel('Time [s]')
% ylabel('Step size')
% plot(Time,Stepsize,'-o')
% axis([0 Time(end) -1 1])
% axis 'auto y'

end
return;
```

Apéndice IV

Funciones *MakeYbus.m* y *MakeYbusSec0.m*

A continuación se muestran las funciones que calculan las matrices de admitancia de red Y_{bus} (*MakeYbus.m*) y su versión modificada para poder determinar la misma matriz de red, pero incluyendo el tipo de conexión de los transformadores en la red de secuencia cero (*MakeYbusSec0.m*).

IV.a Función *MakeYbus*

```
function [Ybus, Yf, Yt] = makeYbus(baseMVA, bus, branch)
%MAKEYBUS   Builds the bus admittance matrix and branch admittance matrices.
%   [YBUS, YF, YT] = MAKEYBUS(MPC)
%   [YBUS, YF, YT] = MAKEYBUS(BASEMVA, BUS, BRANCH)
%
%   Returns the full bus admittance matrix (i.e. for all buses) and the
%   matrices YF and YT which, when multiplied by a complex voltage vector,
%   yield the vector currents injected into each line from the "from" and
%   "to" buses respectively of each line. Does appropriate conversions to p.u.
%   Inputs can be a MATPOWER case struct or individual BASEMVA, BUS and
%   BRANCH values. Bus numbers must be consecutive beginning at 1
%   (i.e. internal ordering).
%
%   See also MAKEJAC, MAKESBUS, EXT2INT.
%   MATPOWER
%   Copyright (c) 1996-2016 by Power System Engineering Research Center (PSERC)
%   by Ray Zimmerman, PSERC Cornell
%
%   This file is part of MATPOWER.
%   Covered by the 3-clause BSD License (see LICENSE file for details).
%   See http://www.pserc.cornell.edu/matpower/ for more info.

%% extract from MPC if necessary
if nargin < 3
    mpc     = baseMVA;
    baseMVA = mpc.baseMVA;
    bus     = mpc.bus;
    branch  = mpc.branch;
end
%% constants
nb = size(bus, 1);          %% number of buses
nl = size(branch, 1);       %% number of lines

%% define named indices into bus, branch matrices
[PQ, PV, REF, NONE, BUS_I, BUS_TYPE, PD, QD, GS, BS, BUS_AREA, VM, ...
    VA, BASE_KV, ZONE, VMAX, VMIN, LAM_P, LAM_Q, MU_VMAX, MU_VMIN] = idx_bus;
[F_BUS, T_BUS, BR_R, BR_X, BR_B, RATE_A, RATE_B, RATE_C, ...
    TAP, SHIFT, BR_STATUS, PF, QF, PT, QT, MU_SF, MU_ST, ...
    ANGMIN, ANGMAX, MU_ANGMIN, MU_ANGMAX] = idx_brch;

%% check that bus numbers are equal to indices to bus (one set of bus numbers)
if any(bus(:, BUS_I) ~= (1:nb)')
    error('makeYbus: buses must be numbered consecutively in bus matrix; use ext2int()
to convert to internal ordering')
end

%% for each branch, compute the elements of the branch admittance matrix where
%%
%%      | If |   | Yff  Yft |   | Vf |
%%      |    | = |          | * |    |
%%      | It |   | Ytf  Ytt |   | Vt |
%%
```

```
stat = branch(:, BR_STATUS);                    %% ones at in-service branches
Ys = stat ./ (branch(:, BR_R) + 1j * branch(:, BR_X));  %% series admittance
Bc = stat .* branch(:, BR_B);                           %% line charging susceptance
tap = ones(nl, 1);                              %% default tap ratio = 1
i = find(branch(:, TAP));                       %% indices of non-zero tap ratios
tap(i) = branch(i, TAP);                        %% assign non-zero tap ratios
tap = tap .* exp(1j*pi/180 * branch(:, SHIFT)); %% add phase shifters
Ytt = Ys + 1j*Bc/2;
Yff = Ytt ./ (tap .* conj(tap));
Yft = - Ys ./ conj(tap);
Ytf = - Ys ./ tap;

%% compute shunt admittance
%% if Psh is the real power consumed by the shunt at V = 1.0 p.u.
%% and Qsh is the reactive power injected by the shunt at V = 1.0 p.u.
%% then Psh - j Qsh = V * conj(Ysh * V) = conj(Ysh) = Gs - j Bs,
%% i.e. Ysh = Psh + j Qsh, so ...
Ysh = (bus(:, GS) + 1j * bus(:, BS)) / baseMVA; %% vector of shunt admittances

%% build connection matrices
f = branch(:, F_BUS);                           %% list of "from" buses
t = branch(:, T_BUS);                           %% list of "to" buses
Cf = sparse(1:nl, f, ones(nl, 1), nl, nb);      %% connection matrix for line & from
buses
Ct = sparse(1:nl, t, ones(nl, 1), nl, nb);      %% connection matrix for line & to
buses

%% build Yf and Yt such that Yf * V is the vector of complex branch currents injected
%% at each branch's "from" bus, and Yt is the same for the "to" bus end
i = [1:nl; 1:nl]';                              %% double set of row indices
Yf = sparse(i, [f; t], [Yff; Yft], nl, nb);
Yt = sparse(i, [f; t], [Ytf; Ytt], nl, nb);
% Yf = spdiags(Yff, 0, nl, nl) * Cf + spdiags(Yft, 0, nl, nl) * Ct;
% Yt = spdiags(Ytf, 0, nl, nl) * Cf + spdiags(Ytt, 0, nl, nl) * Ct;

%% build Ybus
Ybus = Cf' * Yf + Ct' * Yt + ...                %% branch admittances
        sparse(1:nb, 1:nb, Ysh, nb, nb);  %% shunt admittance
```

IV.b Función *MakeYbusSec0.m*

```
function [Ybus, Yf, Yt] = makeYbussec0(baseMVA, bus, branch)
%MAKEYBUS   Builds the bus admittance matrix and branch admittance matrices.
%   [YBUS, YF, YT] = MAKEYBUS(MPC)
%   [YBUS, YF, YT] = MAKEYBUS(BASEMVA, BUS, BRANCH)
%
%   Returns the full bus admittance matrix (i.e. for all buses) and the
%   matrices YF and YT which, when multiplied by a complex voltage vector,
%   yield the vector currents injected into each line from the "from" and
%   "to" buses respectively of each line. Does appropriate conversions to p.u.
%   Inputs can be a MATPOWER case struct or individual BASEMVA, BUS and
%   BRANCH values. Bus numbers must be consecutive beginning at 1 (internal
%   ordering).
%
%   See also MAKEJAC, MAKESBUS, EXT2INT.

%   MATPOWER
%   Copyright (c) 1996-2015 by Power System Engineering Research Center (PSERC)
%   by Ray Zimmerman, PSERC Cornell
%
%   $Id: makeYbus.m 2644 2015-03-11 19:34:22Z ray $
%
%   This file is part of MATPOWER.
%   Covered by the 3-clause BSD License (see LICENSE file for details).
%   See http://www.pserc.cornell.edu/matpower/ for more info.

if nargin < 3
```

```
    mpc     = baseMVA;
    baseMVA = mpc.baseMVA;
    bus     = mpc.bus;
    branch  = mpc.branch;
end

%% constants
nb = size(bus, 1);          %% number of buses
nl = size(branch, 1);       %% number of lines
%% define named indices into bus, branch matrices
[PQ, PV, REF, NONE, BUS_I, BUS_TYPE, PD, QD, GS, BS, BUS_AREA, VM, ...
    VA, BASE_KV, ZONE, VMAX, VMIN, LAM_P, LAM_Q, MU_VMAX, MU_VMIN] = idx_bus;

[F_BUS, T_BUS, BR_R, BR_X, BR_B, YTD, DYT, RATE_C, ...
    TAP, SHIFT, BR_STATUS, PF, QF, PT, QT, MU_SF, MU_ST, ...
    ANGMIN, ANGMAX, MU_ANGMIN, MU_ANGMAX] = idx_brch;

%% check that bus numbers are equal to indices to bus (one set of bus numbers)
if any(bus(:, BUS_I) ~= (1:nb)')
    error('buses must appear in order by bus number')
end

%% for each branch, compute the elements of the branch admittance matrix where
%%
%%      | If |   | Yff  Yft |   | Vf |
%%      |    | = |          | * |    |
%%      | It |   | Ytf  Ytt |   | Vt |
%%
stat = branch(:, BR_STATUS);%% ones at in-service branches

Ys = stat ./ (branch(:, BR_R) + 1j * branch(:, BR_X));%% series admittance
Bc = stat .* branch(:, BR_B);%% line charging susceptance

tap = ones(nl, 1);%% default tap ratio = 1
i = find(branch(:, TAP));  %% indices of non-zero tap ratios
tap(i) = branch(i, TAP); %% assign non-zero tap ratios
tap = tap .* exp(1j*pi/180 * branch(:, SHIFT)); %% add phase shifters

        Ytt = (Ys + 1j*Bc/2);
        Yff = (Ytt ./ (tap .* conj(tap)));
        Yft =( - Ys ./ conj(tap));
        Ytf =(( - Ys ./ tap));

        YtD= find(branch(:,YTD));
        pause
        if YtD~=0
           Ytt(YtD) = 0;
           Yff = (Ys ./ (tap .* conj(tap)));
           Yft(YtD) =0;
           Ytf(YtD) =0;

        end
        DYt= find(branch(:,DYT));

        if DYt~=0
           Ytt = Ys;
           Yff(DYt) = 0;
           Yft(DYt) = 0;
           Ytf(DYt) = 0;
        end



%% compute shunt admittance
%% if Psh is the real power consumed by the shunt at V = 1.0 p.u.
%% and Qsh is the reactive power injected by the shunt at V = 1.0 p.u.
%% then Psh - j Qsh = V * conj(Ysh * V) = conj(Ysh) = Gs - j Bs,
%% i.e. Ysh = Psh + j Qsh, so ...
```

```
Ysh = (bus(:, GS) + 1j * bus(:, BS)) / baseMVA; %% vector of shunt admittances

%% build connection matrices
f = branch(:, F_BUS);                           %% list of "from" buses
t = branch(:, T_BUS);                           %% list of "to" buses
Cf = sparse(1:nl, f, ones(nl, 1), nl, nb);      %% connection matrix for line & from
buses
Ct = sparse(1:nl, t, ones(nl, 1), nl, nb);      %% connection matrix for line & to
buses

%% build Yf and Yt such that Yf * V is the vector of complex branch currents injected
%% at each branch's "from" bus, and Yt is the same for the "to" bus end
i = [1:nl; 1:nl]';                              %% double set of row indices
Yf = sparse(i, [f; t], [Yff; Yft], nl, nb);
Yt = sparse(i, [f; t], [Ytf; Ytt], nl, nb);
% Yf = spdiags(Yff, 0, nl, nl) * Cf + spdiags(Yft, 0, nl, nl) * Ct;
% Yt = spdiags(Ytf, 0, nl, nl) * Cf + spdiags(Ytt, 0, nl, nl) * Ct;

%% build Ybus
Ybus = Cf' * Yf + Ct' * Yt + ...                %% branch admittances
        sparse(1:nb, 1:nb, Ysh, nb, nb);%% shunt admittance
```

Apéndice V

Archivos *file.m* para falla monofásica a tierra en caso de estudio del IEEE de nueve barras

A continuación se muestran los archivos *file.m* para poder ejecutar el análisis de estabilidad para una falla monofásica en el caso de estudio del IEEE nueve barras.

V.I Archivo de secuencia positiva *Case9sec1.m*

```
%% bus data
%   bus_i   type    Pd  Qd  Gs  Bs  area    Vm  Va  baseKV  zone    Vmax    Vmin
mpc.bus = [
    1   3   0   0   0   0   1   1   0   345 1   1.1 0.9;
    2   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    3   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    4   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    5   1   125 50  0   0   1   1   0   345 1   1.1 0.9;
    6   1   90  30  0   0   1   1   0   345 1   1.1 0.9;
    7   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    8   1   100 35  0   0   1   1   0   345 1   1.1 0.9;
    9   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
];

%% generator data
%   bus Pg  Qg  Qmax    Qmin    Vg  mBase   status  Pmax    Pmin    Pc1 Pc2 Qc1m
mpc.gen = [
    1   0   0   300 -300    1.04    100 1   250 10  0   0   0   0   0   0   0
    2   163 0   300 -300    1.025   100 1   300 10  0   0   0   0   0   0   0
    3   85  0   300 -300    1.025   100 1   270 10  0   0   0   0   0   0   0
];

%% branch data
%   fbus    tbus    r   x   b   rateA   rateB   rateC   ratio   angle   status
mpc.branch = [
    1   4   0   0.0576  0   250 250 250 0   0   1   -360    360;
    4   6   0.017   0.092   0.158   250 250 250 0   0   1   -360    360;
    6   9   0.039   0.17    0.358   150 150 150 0   0   1   -360    360;
    3   9   0   0.0586  0   300 300 300 0   0   1   -360    360;
    8   9   0.0119  0.1008  0.209   150 150 150 0   0   1   -360    360;
    7   8   0.0085  0.072   0.149   250 250 250 0   0   1   -360    360;
    2   7   0   0.0625  0   250 250 250 0   0   1   -360    360;
    5   7   0.032   0.161   0.306   250 250 250 0   0   1   -360    360;
    4   5   0.01    0.085   0.176   250 250 250 0   0   1   -360    360;
];
```

V.II Archivo de secuencia negativa *Case9sec1.m*

```
%% system MVA base
mpc.baseMVA = 100;

%% bus data
%   bus_i   type    Pd  Qd  Gs  Bs  area    Vm  Va  baseKV  zone    Vmax    Vmin
mpc.bus = [
    1   3   0   0   0   0   1   1   0   345 1   1.1 0.9;
    2   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    3   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    4   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    5   1   125 50  0   0   1   1   0   345 1   1.1 0.9;
    6   1   90  30  0   0   1   1   0   345 1   1.1 0.9;
    7   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    8   1   100 35  0   0   1   1   0   345 1   1.1 0.9;
    9   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
];

%% generator data
%   bus Pg  Qg  Qmax    Qmin    Vg  mBase   status  Pmax    Pmin    Pc1 Pc2 Qc1mi
mpc.gen = [
    1   0   0   300 -300    1.04    100 1   250 10  0   0   0   0   0   0   0   0
    2   163 0   300 -300    1.025   100 1   300 10  0   0   0   0   0   0   0   0
    3   85  0   300 -300    1.025   100 1   270 10  0   0   0   0   0   0   0   0
];

%% branch data
%   fbus    tbus    r   x   b   rateA   rateB   rateC   ratio   angle   status  a
mpc.branch = [
    1   4   0   0.0576  0   250 250 250 0   -30 1   -360    360;
    4   6   0.017   0.092   0.158   250 250 250 0   0   1   -360    360;
    6   9   0.039   0.17    0.358   150 150 150 0   0   1   -360    360;
    3   9   0   0.0586  0   300 300 300 0   -30 1   -360    360;
    8   9   0.0119  0.1008  0.209   150 150 150 0   0   1   -360    360;
    7   8   0.0085  0.072   0.149   250 250 250 0   0   1   -360    360;
    2   7   0   0.0625  0   250 250 250 0   -30 1   -360    360;
    5   7   0.032   0.161   0.306   250 250 250 0   0   1   -360    360;
    4   5   0.01    0.085   0.176   250 250 250 0   0   1   -360    360;
```

V.II Archivo de secuencia cero *Case9sec0.m*

```
%% system MVA base
mpc.baseMVA = 100;

%% bus data
%   bus_i   type    Pd  Qd  Gs  Bs  area    Vm  Va  baseKV  zone    Vmax    Vmin
mpc.bus = [
    1   3   0   0   0   0   1   1   0   345 1   1.1 0.9;
    2   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    3   2   0   0   0   0   1   1   0   345 1   1.1 0.9;
    4   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    5   1   125 50  0   0   1   1   0   345 1   1.1 0.9;
    6   1   90  30  0   0   1   1   0   345 1   1.1 0.9;
    7   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
    8   1   100 35  0   0   1   1   0   345 1   1.1 0.9;
    9   1   0   0   0   0   1   1   0   345 1   1.1 0.9;
];

%% generator data
%   bus Pg  Qg  Qmax    Qmin    Vg  mBase   status  Pmax    Pmin    Pc1 Pc2 Qc1min
mpc.gen = [
    1   0   0   300 -300    1.04    100 1   250 10  0   0   0   0   0   0   0   0
    2   163 0   300 -300    1.025   100 1   300 10  0   0   0   0   0   0   0   0
    3   85  0   300 -300    1.025   100 1   270 10  0   0   0   0   0   0   0   0
];

%% branch data
%   fbus    tbus    r   x   b     D-Yt Yt-D rateC   ratio   angle   status  angmin
mpc.branch = [
    1   4   0   0.0576  0           0   1   250 0   0   1   -360    360;
    4   6   0.0425  0.23    0.158   0   0   250 0   0   1   -360    360;
    6   9   0.0975  0.425   0.358   0   0   150 0   0   1   -360    360;
    3   9   0   0.0586  0           0   1   300 0   0   1   -360    360;
    8   9   0.02975 0.252   0.209   0   0   150 0   0   1   -360    360;
    7   8   0.02150 0.18    0.149   0   0   250 0   0   1   -360    360;
    2   7   0   0.0625  0           0   1   250 0   0   1   -360    360;
    5   7   0.08    0.4025  0.306   0   0   250 0   0   1   -360    360;
    4   5   0.025   0.2125  0.176   0   0   250 0   0   1   -360    360;
```