KTH ROYAL INSTITUTE OF TECHNOLOGY

Modelica Classes of the Norwegian Grid

Power systems components modeling and software-to-software validation

Zhang Mengjia[1], Maxime Baudette[1]

Prof. Luigi Vanfretti[1,2]

mengjia@kth.se , baudette@kth.se, luigiv@kth.se
Electric Power Systems Dept.
KTH
Stockholm, Sweden

luigi.vanfretti@statnett.no
Research and Development Division
Statnett SF
Oslo, Norway

Acknowledgments

- This work has been funded in part by the EU funded FP7 iTesla project: http://www.itesla-project.eu/ and Statnett SF, the Norwegian power system operator.

- Special thanks for ‘special training’ and support from
  - Prof. Peter Fritzson and his team at Linköping University
  - Prof. Bernhard Bachmann and his team at FH Bielefeld

Contents

Contents

- Background
- Introduction
- Model implementation
  - ✓ State equations
    - Round rotor generator
    - Three-winding transformer
  - ✓ Initialization procedure
    - Round rotor generator
    - Excitation System
- Model Validation
  - Principles
  - Results
- Conclusions
- Future work
- Discussion

Background

Power system modelling and simulation are facing new challenges

- The expansion of the grid and the penetration of new technologies results in considerable complex system model which cause heavily computation burden.
- Coordination between different transmission system operators (TSOs) requires faster and standard way for sharing information.

At the same time

- Simulation results are inconsistent across different software.
- Modelling method is solver dependent and written implicitly.
- Difficulties in models update and evaluation with existing simulation tools.

To cope with these challenges, a new power system library available for modification and maintenance is under development at **SmarTS Lab** within the FP7 ***iTesla***[1] project.

Power system modelling and simulation using Modelica

The nature of the Modelica modeling language supports model exchange at the "equation-level", this allows for unambiguous model exchange between different Modelica-based simulation tools without loss of information about the model[4]. The Modelica models can be used as standard for steady-state and dynamic information exchange !

iTesla[1]: Innovative Tools for Electrical System Security within Large Areas

Contents

Introduction

Objectives

- Make a contribution to the development of Modelica power system library by providing:
  - validated component models used in Norwegian power grid.
  - validated test system models to be used in future tasks.

Outcome

- The Modelica classes of the Norwegian grid component can be used as reference models for future evaluation and modification.

Introduction

Steps

- **Model implementation**
  - Study the reference models from Power System Simulation Tool for Engineer (PSS/E), which is an tool focus on Electro-mechanical transient simulation.
  - Implementation of the models in Modelica language, based on the PSS/E reference model.

- **Model validation**
  - Build identically system models in both simulation platforms.
  - Collect simulation results of power flow and dynamic responses for models from PSS/E.
  - Redefine the Modelica model until the simulation results match the correct initial steady-states conditions and the transient dynamic after a disturbance.
    - Manual tuning of the parameters

Introduction------Developed Modelica Classes

Regulators of the generator

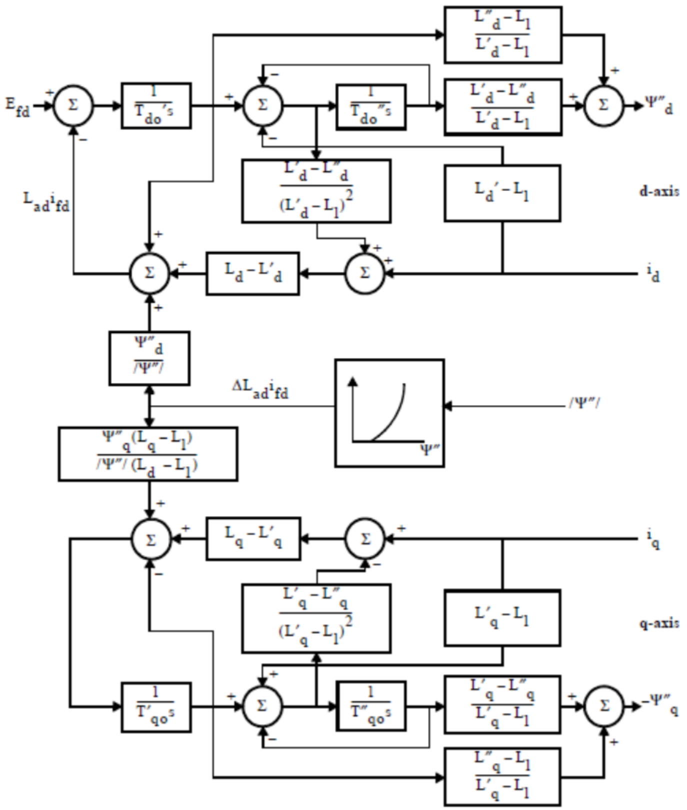
Model type	Name
Generator	Round rotor generator GENROU Salinet pole generator GENSAL Clasic generator model GENCLS
Transformer	2-winding and 3-winding transformer with phase-shift and on load tap changer (OLTC)
Excitation system	IEEET1, IEEET2, SCRX, SEX, URST5T, ESST4B, ESAC2A, EXST1
Governor system	HYGOV, GAST IEESGO, GGOV1
Stablizer	IEEEST, STAB2A
Load model	A composite load includes specific load characteristics from load model in PSS/E
FACTS devices	Static var compensator (SVC)

Contents

Model implementation------Round rotor generator

Swing equation and Stator circuit equations are not shown in the diagram

Figure 1: Block diagram of the rotor circuit in Genrou model[3]



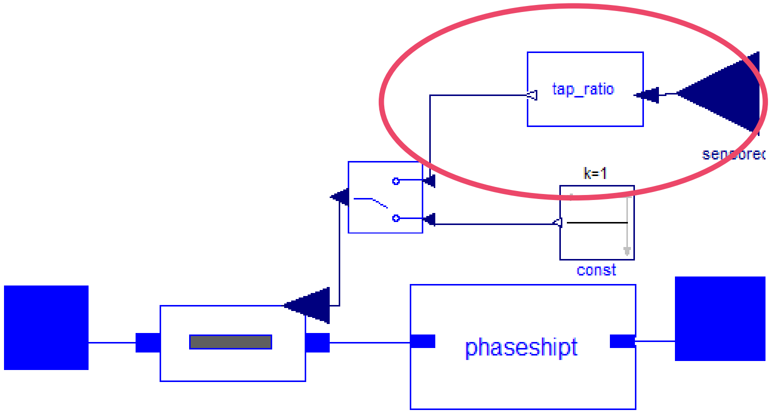
6 states

30 unknowns and equations

```
der(delta) = wbase * w;
der(w) = ((Pm - D * w) / (w + 1) - Te) / (2 * H);
der(Epq) = 1 / Tpd0 * (E_fd - XadIfd);
der(Epd) = 1 / Tpq0 * (-1) * XaqIlq;
der(PSIkd) = 1 / Tppd0 * (Epq - PSIkd - (Xpd - Xl) * id);
der(PSIkq) = 1 / Tppq0 * (Epd - PSIkq + (Xpq - Xl) * iq);
Te = PSId * iq - PSIq * id;
PSId = PSIppd - Xppd * id;
PSIq = PSIppq - Xppq * iq;
PSIppd = Epq * K3d + PSIkd * K4d;
PSIppq = (-Epd * K3q) - PSIkq * K4q;
PSIpp = sqrt(PSIppd * PSIppd + PSIppq * PSIppq);
ud = (-PSIq * (w + 1)) - Ra * id;
uq = PSId * (w + 1) - Ra * iq;
anglev = atan2(p.vi, p.vr);
Vt = sqrt(p.vr ^ 2 + p.vi ^ 2);
anglei = atan2(p.ii, p.ir);
I = sqrt(p.ii ^ 2 + p.ir ^ 2);
Delta_XadIfd = Se(PSIpp, s10, s12) * PSIppd;
Delta_XaqIlq = Se(PSIpp, s10, s12) * PSIppq * (Xq - Xl) / (Xd - Xl);
XadIfd = K1d * (Epq - PSIkd - (Xpd - Xl) * id) + Epq + id * (Xd - Xpd) + Delta_XadIfd;
XaqIlq = K1q * (Epd - PSIkq + (Xpq - Xl) * iq) + Epd - iq * (Xq - Xpq) - Delta_XaqIlq;
[p.ir; p.ii] = -[sin(delta), cos(delta); -cos(delta), sin(delta)] * [id; iq];
[p.vr; p.vi] = [sin(delta), cos(delta); -cos(delta), sin(delta)] * [ud; uq];
-P = p.vr * p.ir + p.vi * p.ii;
-Q = p.vi * p.ir - p.vr * p.ii;
```

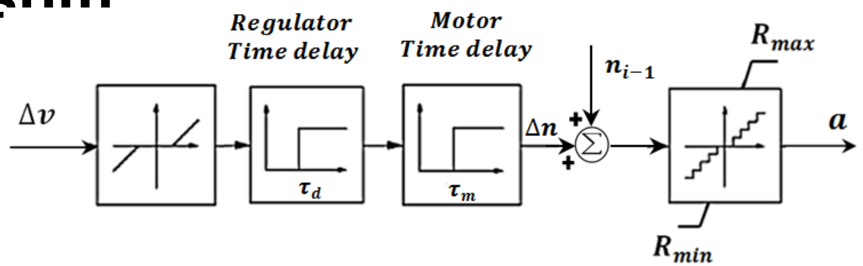
Model implementation------ 3-winding Transformer with OLTC and phase-shift

```
-
  Modelica.Blocks.Interfaces.RealInput t(start = 1) a;
  parameter Real R "Resistance p.u.";
  parameter Real X "Reactance p.u.";
equation
 t*(R * n.ir  - X * n.ii)= n.vr - p.vr*t;
 t*(R * n.ii  + X * n.ir)= n.vi- p.vi*t;
 -(R * p.ir  - X * p.ii) = n.vr- p.vr*t;
 -(R * p.ii  + X * p.ir)= n.vi- p.vi*t;
```



Mixed continuous/discrete models

Model implementation------ 3-winding Transformer with OLTC and phase-shift




```
//deadband
if Vmin - u > dV then
    p1 = -1;
  elseif u - Vmax > dV then
   p1= 1;
  else
    p1=0;
  end if;
// The voltage is controlled to stay within
// the range [Vmin, Vmax]

// Timer 1
if n1 > -tau and n1 < tau then
    der(n1) = p1;
 elseif n1 <= -tau and  p1 > 0 then
    der(n1) =  p1;
 elseif n1 >= tau and  p1 < 0 then
    der(n1) =  p1;
 else
    der(n1) = 0;
 end if;
```

```
// Delay 1: regulator delay
  if n1>=tau or n1<=-tau then
    action=true;
  else
    action=false;
  end if;
// Timer 2
  when action then
    Timer2 = time;
  end when;
// Delay 2: motor mechanism delay
  x2 = integer((time - Timer2) / TC);
  when change(x2) and action then
    m = pre(m) + dtap * pre(p1);
  end when;
```

```
// Reset timer if the voltage returns from
outside of the range or jumps across the
deadband
x1=integer(p1);
if change(x1) then
    action1=true;
else
    action1=false;
end if;
when action1 then
    reinit(n1,0);
  end when;
// Limiter
  if m>=Rmax then
  y  = Rmax;
  elseif m<=Rmin then
   y = Rmin;
  else
   y=m;
  end if;
// Adjust the transformer ratios in
// the range of [Rmin, Rmax]
```

Contents

Model implementation----initialization

- **Assume steady state operation**
  - The system is in a state where its numerous properties are unchanging in time, in other words, the behavior of the system will continue into its future.
  - It means there are no transient changes in power flow or voltage phasor and thus the system frequency is also assumed to be constant.
- **General initialization chain of power system model**

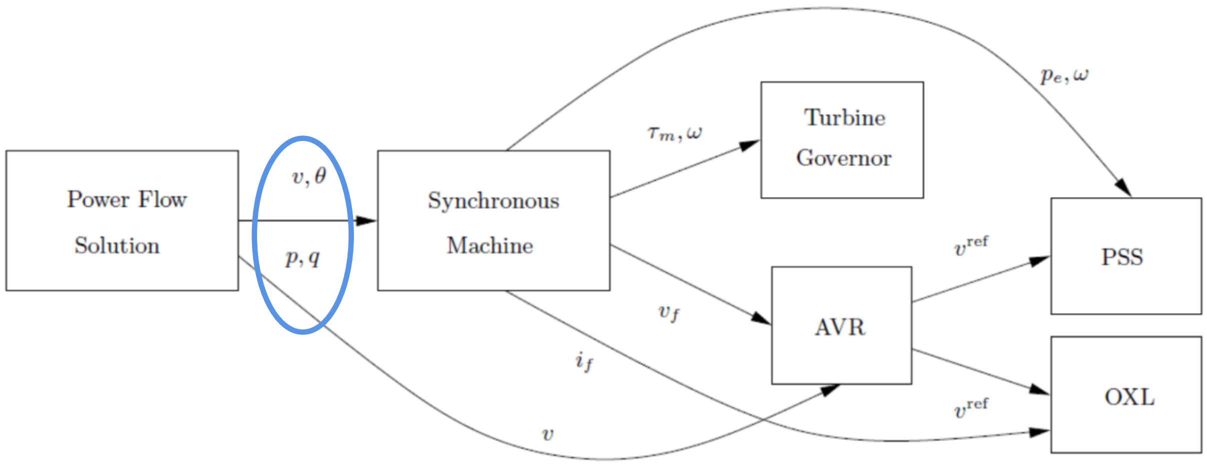


Figure 4: Initialization chain of power system model[5]

Model implementation----initialization

- **For variables**
  - The initial values of variables can be specified with an equation through the initial equation construct or by setting the (fixed=true, start=x0) attribute of the instance variables.
  - If nothing is specified, the default would be zero and if fixed=false, then the value will be viewed as a guess value.
- **For parameters**
  - By setting its attribute to be (fixed=false), the initial value of the parameter could be implicitly computed during initialization and then keep its value throughout the simulation.
- **The number of equations for initialization**
  - During initialization stages, all the derivities of the states (der(x)) and parameter with (fixed=false) are treated as unknown variables.
  - To keep a balance between the same number of unknowns and equations, for each of these unknowns, a extra equation should be provied under the initialization section.

Initialization------ Round rotor generator

- **Tips**
  - Provide the value of the derivatives of the states.
  - Initialize the interfaces of the models (connectors).
  - Provided good guess value for residue states (the value of all other states depends on these states.) in this case it is delta0.
  - Provided good guess value for parameters with attributes (fixed=false).
  - Case dependent !

```
PowerSystems.Connectors.PwPin p(vr(start = vr0), vi(start = vi0), ir(start = ir0), ii(start = ii0)) a;
Real delta(start=delta0) "load angle";
Real w(start=0) "machine speed deviation, p.u.";
parameter Real Pm0(fixed=false, start = p0);
parameter Real Efd0(fixed=false,start = dsat * PSIppq0 + PSIppq0 + (Xpd - Xpp) * id0 + (Xd - Xpd) * id0);
initial equation
  der(delta) = 0;
  der(w) = 0;
  der(Epd) = 0;
  der(Epq) = 0;
  der(PSIkd) = 0;
  der(PSIkq) = 0;
```

Initialization------ Round rotor generator

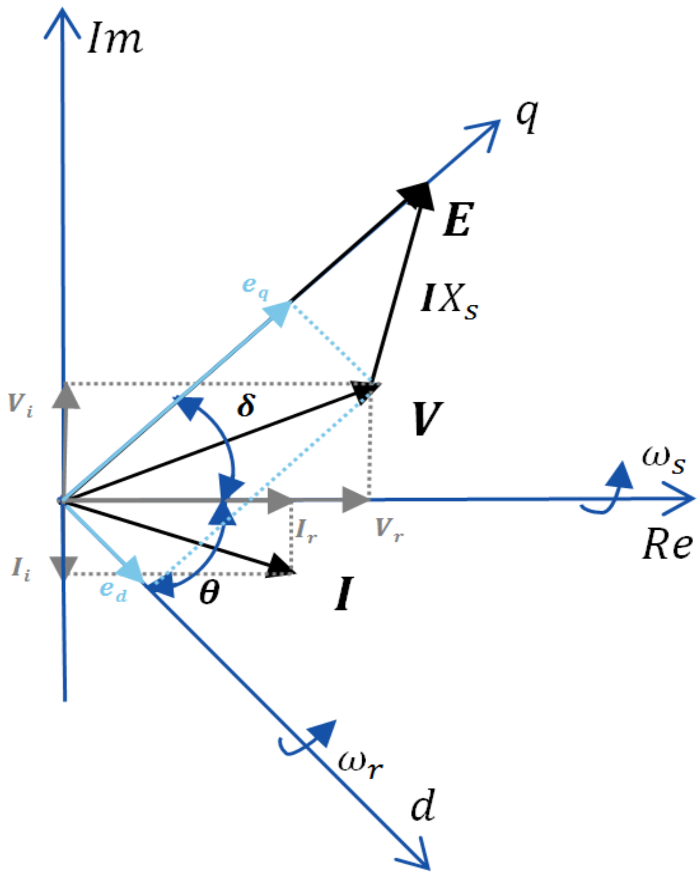


Figure 5: Phasor diagram of Genrou model

$$\begin{bmatrix} V_r \\ V_i \end{bmatrix} = \begin{bmatrix} sin(\delta) & cos(\delta) \\ -cos(\delta) & sin(\delta) \end{bmatrix} \times \begin{bmatrix} u_d \\ u_q \end{bmatrix}$$

$$\begin{bmatrix} I_r \\ I_i \end{bmatrix} = - \begin{bmatrix} sin(\delta) & cos(\delta) \\ -cos(\delta) & sin(\delta) \end{bmatrix} \times \begin{bmatrix} i_d \\ i_q \end{bmatrix}$$

$$\delta_0 = \arctan\left(\frac{|I_{t0}|(X'_q - X''_q)\cos(\theta'_{p0} - \theta_{i0})}{|I_{t0}|(X'_q - X''_q)\sin(\theta'_{p0} - \theta_{i0}) - K|\psi''_0|}\right) + \theta'_{p0}$$

$$P_{m0} = T_{e0} = P_{t0} + R_a(I_{t0})^2$$

$$E_{fd0} = i_{d0}(X_d - X'_d) + E'_{q0} + S_e|\psi''_0|\psi''_{d0}$$

Initialization------ Round pole generator

```
parameter Real anglev_rad = anglev0 * pi / 180
  "initial value of bus anglev in rad";
parameter Real p0 = pelec / mbase "initial value of bus active power in p.u.";
parameter Real q0 = qelec / mbase
  "initial value of bus reactive power in p.u.";
parameter Complex Zs(re = Ra, im = Xpp) "Equivation impedance";
parameter Complex VT(re = eterm * cos(anglev_rad), im = eterm * sin(anglev_rad));
parameter Complex S(re = p0, im = q0);
parameter Real div_SVTre = ((+S.re * VT.re) + S.im * VT.im) / (VT.re * VT.re + VT.im * VT.im);
parameter Real Ndiv_SVTim = -((-S.re * VT.im) + S.im * VT.re) / (VT.re * VT.re + VT.im * VT.im);
parameter Complex It(re = div_SVTre, im = Ndiv_SVTim);
//5*********************conj(S / VT);
parameter Real Itdiv_VTZsre = It.re + ((+VT.re * Zs.re) + VT.im * Zs.im) / (Zs.re * Zs.re + Zs.im * Zs.im);
parameter Real Itdiv_VTZsim = It.im + ((-VT.re * Zs.im) + VT.im * Zs.re) / (Zs.re * Zs.re + Zs.im * Zs.im);
parameter Complex Is(re = Itdiv_VTZsre, im = Itdiv_VTZsim);
//4*********************It + VT / Zs "Equivation current source";
//Zs.re * Is.re - Zs.im * Is.im
//Zs.re * Is.im + Zs.im * Is.re
parameter Complex fpp(re = Zs.re * Is.re - Zs.im * Is.im, im = Zs.re * Is.im + Zs.im * Is.re);
//3*********************Zs * Is "Flux linkage in synchronous reference frame";
parameter Real ang_P = arg(fpp);
parameter Real ang_I = arg(It);
parameter Real ang_PI = ang_P - ang_I;
parameter Real psi = abs(fpp);
//Include saturation factor during initialization
parameter Real dsat = Se(psi, s10, s12);
parameter Real a = psi + psi * dsat * (Xq - Xl) / (Xd - Xl);
parameter Real b = (It.re ^ 2 + It.im ^ 2) ^ 0.5 * (Xpp - Xq);
//***********************abs(It) * (Xpp - Xq);
//Initialize rotor angle position
parameter Real delta0 = atan(b * cos(ang_PI) / (b * sin(ang_PI) - a)) + ang_P;
parameter Complex DQ_dq(re = cos(delta0), im = -sin(delta0))
  "Change Reference Frame";
parameter Complex fpp_dq(re = fpp.re * DQ_dq.re - fpp.im * DQ_dq.im, im = fpp.re * DQ_dq.im + fpp.im * DQ_dq.re);
//2********************* fpp * DQ_dq "Flux linkage in rotor reference fram (dq axes)";
parameter Complex I_dq(re = It.re * DQ_dq.re - It.im * DQ_dq.im, im = (-1) * (It.re * DQ_dq.im + It.im * DQ_dq.re));
//1********************* conj(It * DQ_dq);
parameter Real PSIppq0 = real(fpp_dq);
parameter Real PSIppd0 = imag(fpp_dq);
```

Initialization------Excitation System

IEEET2

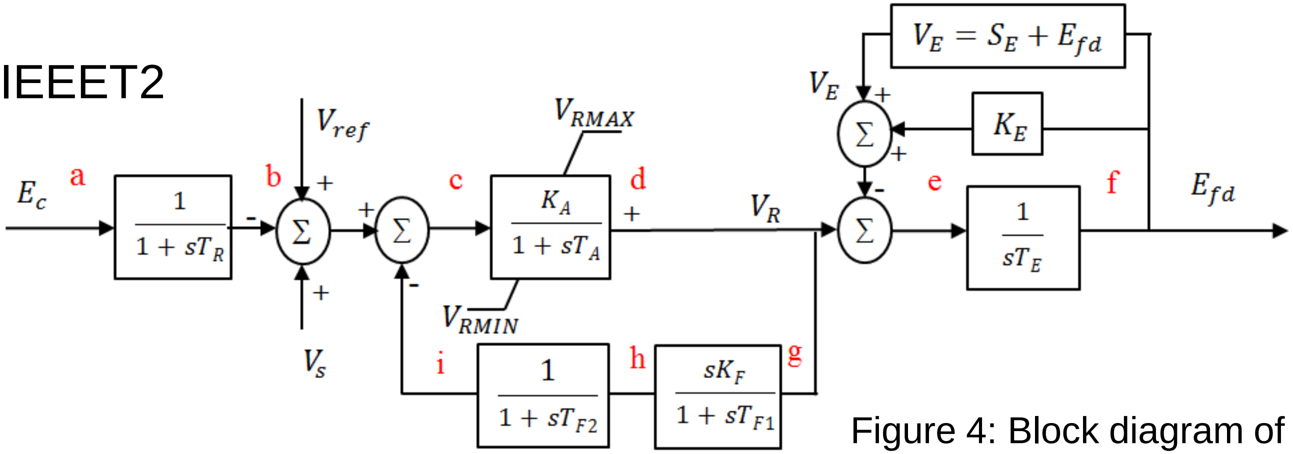


Figure 4: Block diagram of IEEET2

$$a = b + T_R\dot{b}$$
$$K_A c = d + T_A\dot{d}$$
$$e = T_E\dot{f}$$
$$K_F\dot{g} = h + T_{F1}\dot{h}$$
$$h = i + T_{F2}\dot{i}$$

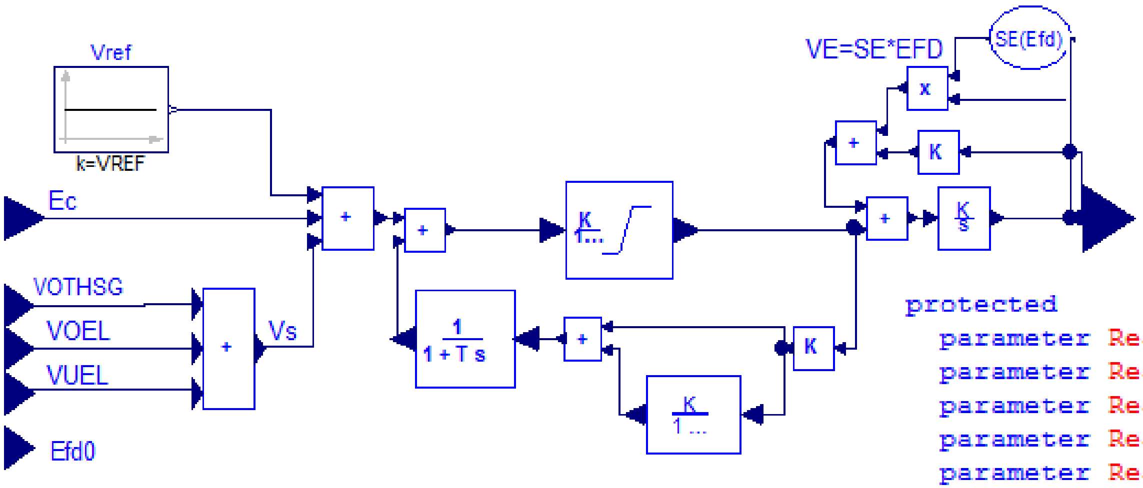
$$\Rightarrow$$

$$a_0 = V_{t0}$$
$$a_0 = b_0$$
$$K_A c_0 = d_0$$
$$e_0 = 0$$
$$h_0 = 0$$
$$i_0 = h_0$$
$$f_0 = E_{fd0}$$

$$\Rightarrow$$

$$V_{R0} = (S_e(E_{fd0}) + 1 + K_E)Efd0$$
$$V_{ref} = \frac{V_{R0}}{K_A} - V_{S0} + V_{t0}$$

Initialization------Excitation System




```
protected
  parameter Real VRMAX0(fixed = false);
  parameter Real VRMIN0(fixed = false);
  parameter Real KE0(fixed = false);
  parameter Real VT0(fixed = false);
  parameter Real VREF(fixed = false);
  parameter Real Efd0(fixed = false);
  parameter Real SE_Efd0(fixed = false);
  parameter Real VR0(fixed = false);
  parameter Real Switch(fixed = false);
  parameter Real T_R(fixed = false);
initial equation
  VT0 = Ec0;
  Efd0 = EFD0;
  SE_Efd0 = SE(EFD0, SE1, SE2, E1, E2);
  (VRMAX0, KE0) = ini0(VRMAX, KE, E2, SE2, Efd0, SE_Efd0);
  VRMIN0 = -VRMAX0;
  VR0 = Efd0 * (KE0 + SE_Efd0);
  VREF = VR0 / KA + VT0 + Vs.a0;
  Switch = TR;
  T_R = if not Switch == 0 then TR else 1;
```

Contents

- Background
- Introduction
- Model implementation
  - ✓ State equations
    - Round pole generator
    - Three-winding transformer
  - ✓ Initialization procedure
    - Round pole generator
    - Excitation System
- Model Validation
  - Principles
  - Results
- Conclusions
- Future work
- Discussion

Model Validation------Princeples

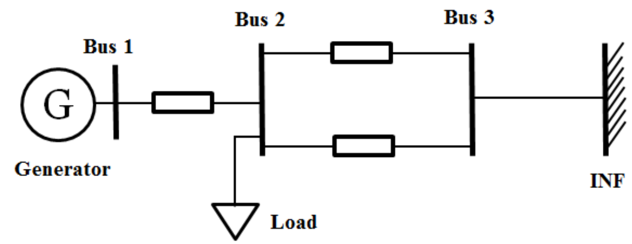
- **Completely testing**
  - Make sure the dynamic simulation starts from the same initial point.
  - Fully test the components to show the specific function of the models.

- **Control variables**
  - Design the test system with less components as possible.
  - Build identical systems in both software.
  - Implement the perturbations as similar as possible.
  - Use the same parameters for simulation setup (solver method, time steps, plot interval, Tolerance etc…)

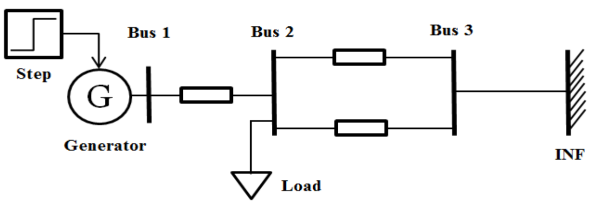
- **Compare data**
  - Collect and compare the states data and I/O data of the models.

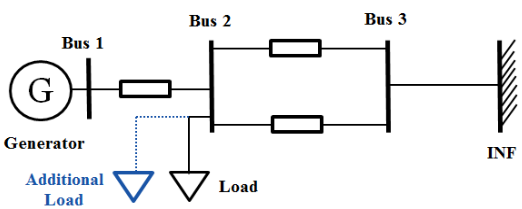
Model Validation-----Test system and perturbation

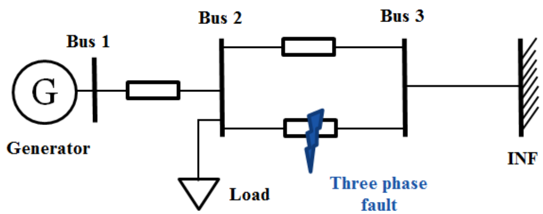
- Basic grid

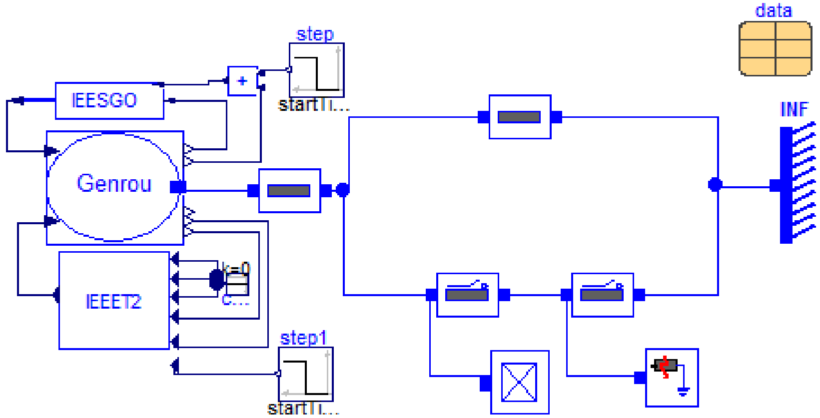


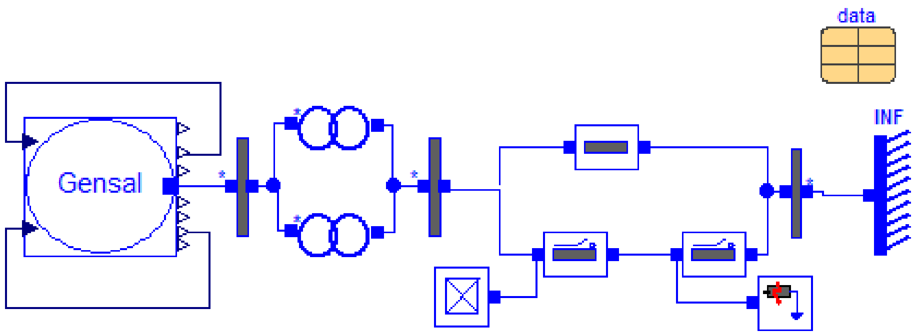
- Perturbation scenarios









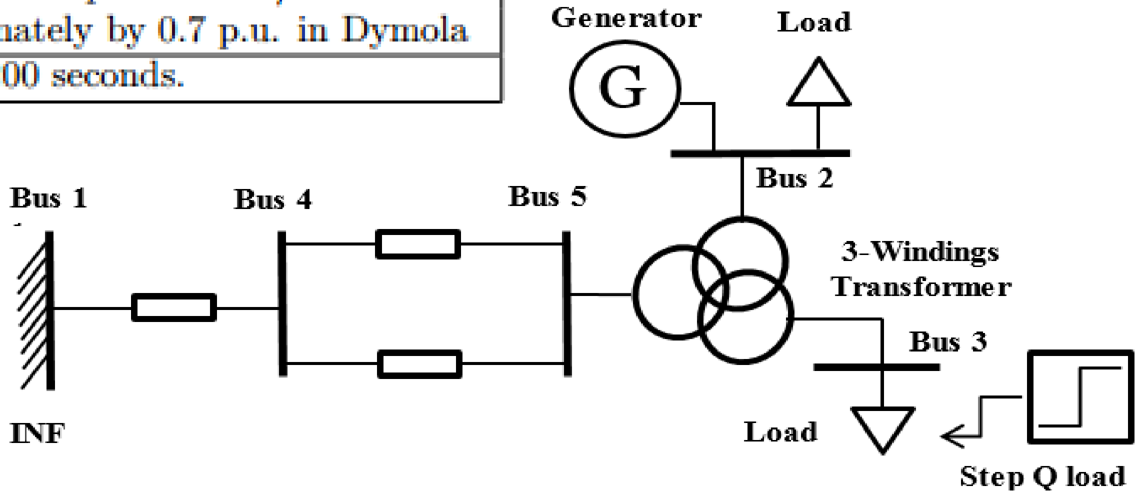


Contents

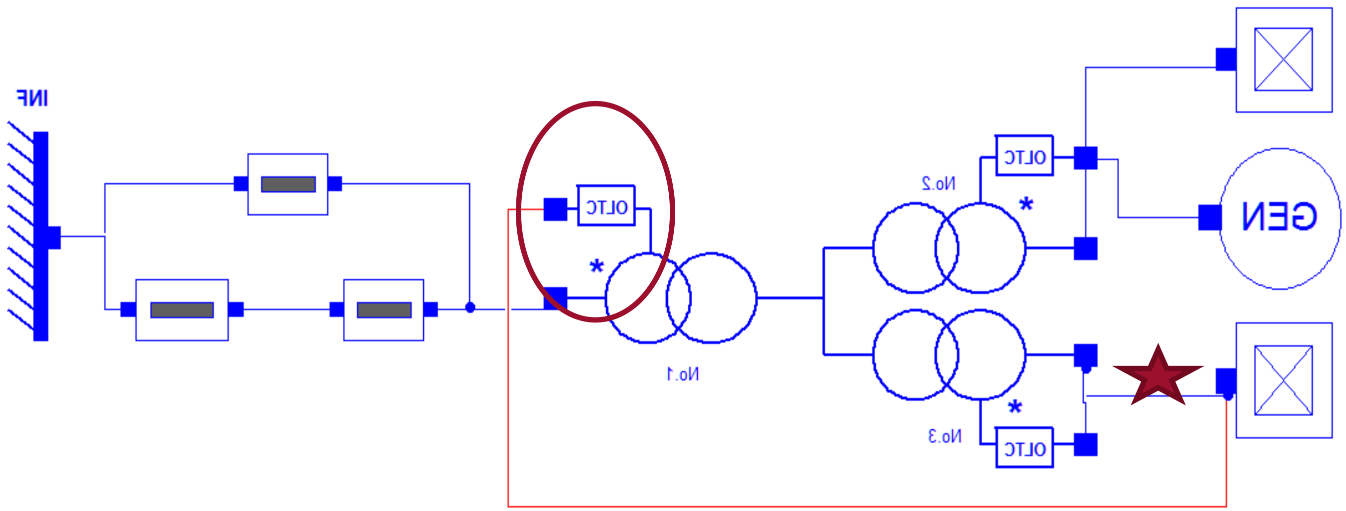
- Background
- Introduction
- Model implementation
  - ✓ State equations
    - Round pole generator
    - Three-winding transformer
  - ✓ Initialization procedure
    - Round pole generator
    - Excitation System
- Model Validation
  - Principles
  - Results
- Conclusions
- Future work
- Discussion

OLTC validation test

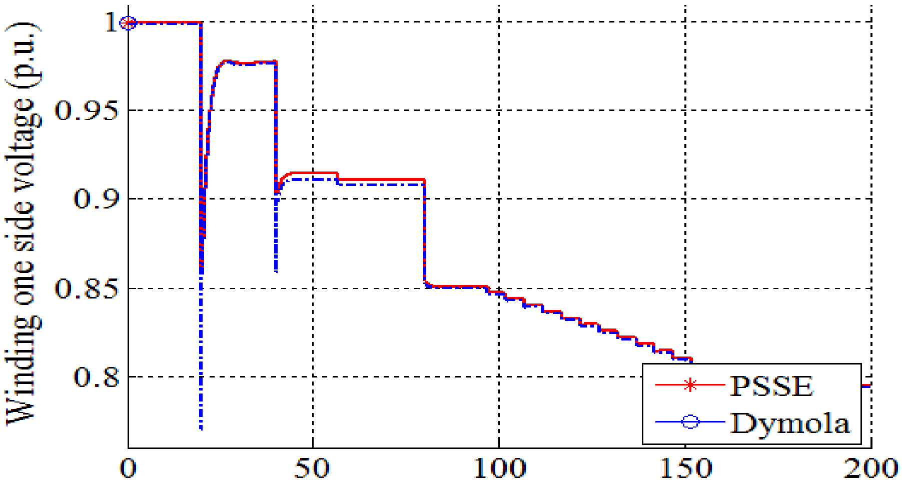
Step	Scenarios
1.	Running under steady state for 20 seconds.
2.	Increasing the Q load to 1.5 p.u. in PSS/E it means to add Q load approximately by 1.42 p.u. in Dymola
3.	Simulating to 40 seconds.
4.	Increasing the Q load to 2.2 p.u. in PSS/E it means to add Q load approximately by 0.85 p.u. in Dymola
5.	Simulating to 80 seconds.
6.	Increasing the Q load to 2.5 p.u. in PSS/E it means to add Q load approximately by 0.7 p.u. in Dymola
5.	Simulating to 200 seconds.

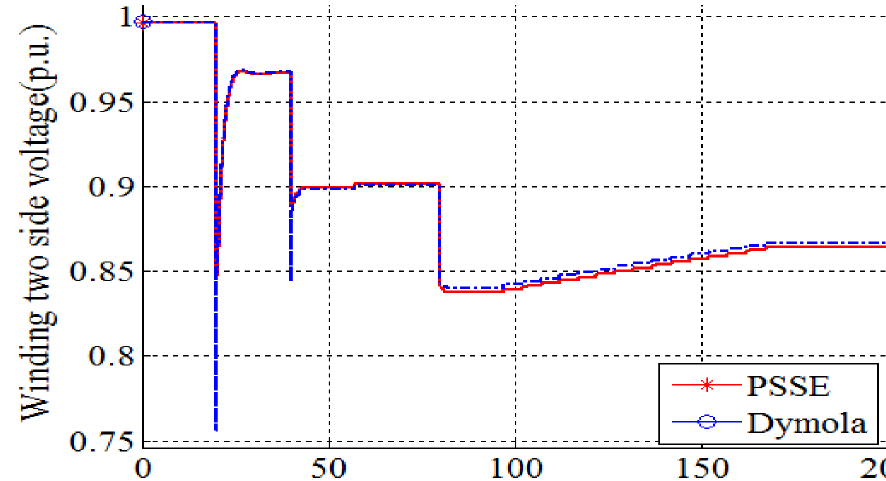


OLTC validation test

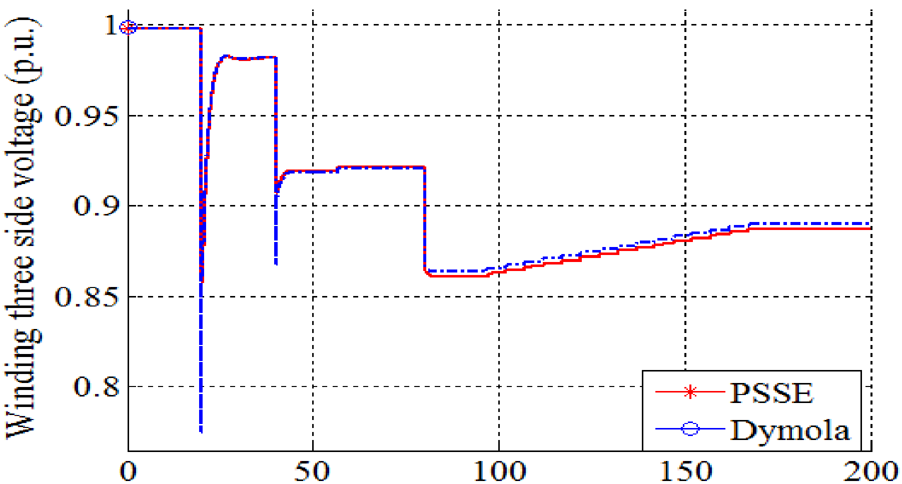


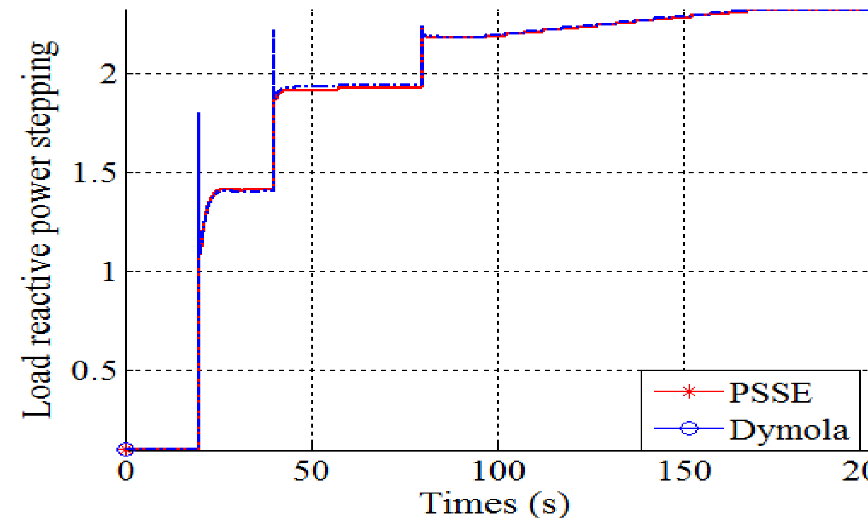
OLTC validation results





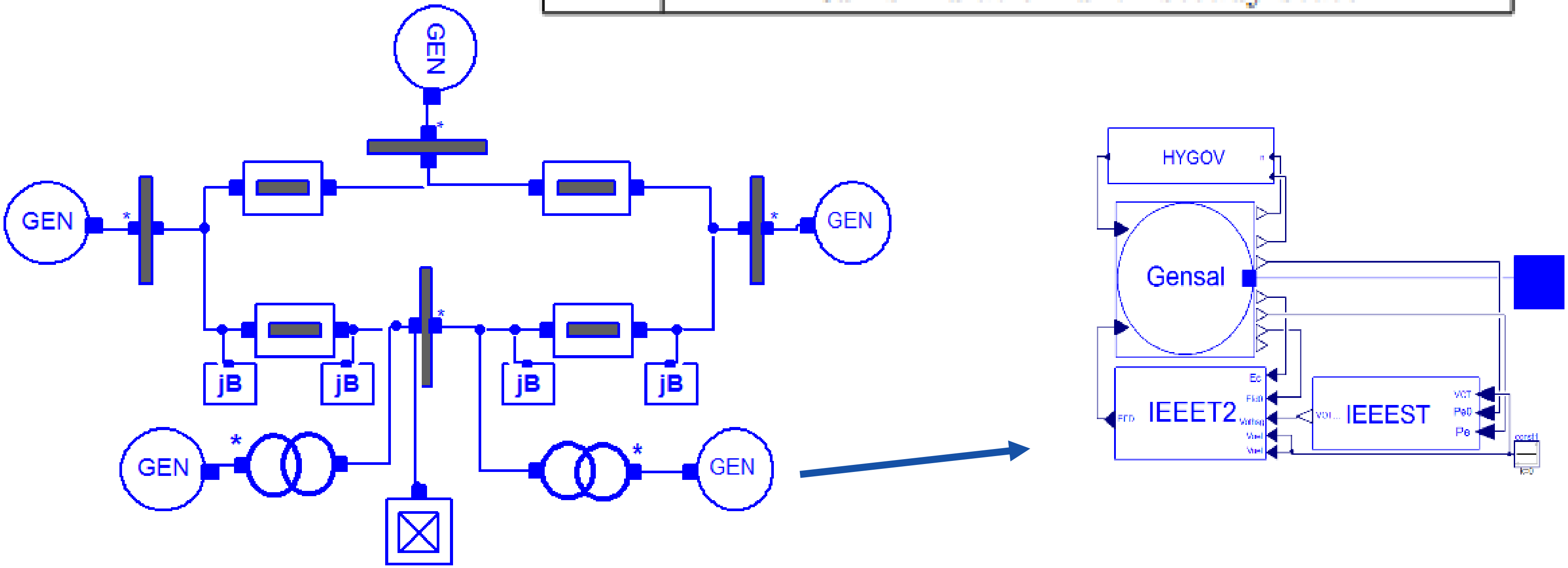
Modelica can handle hybrid system



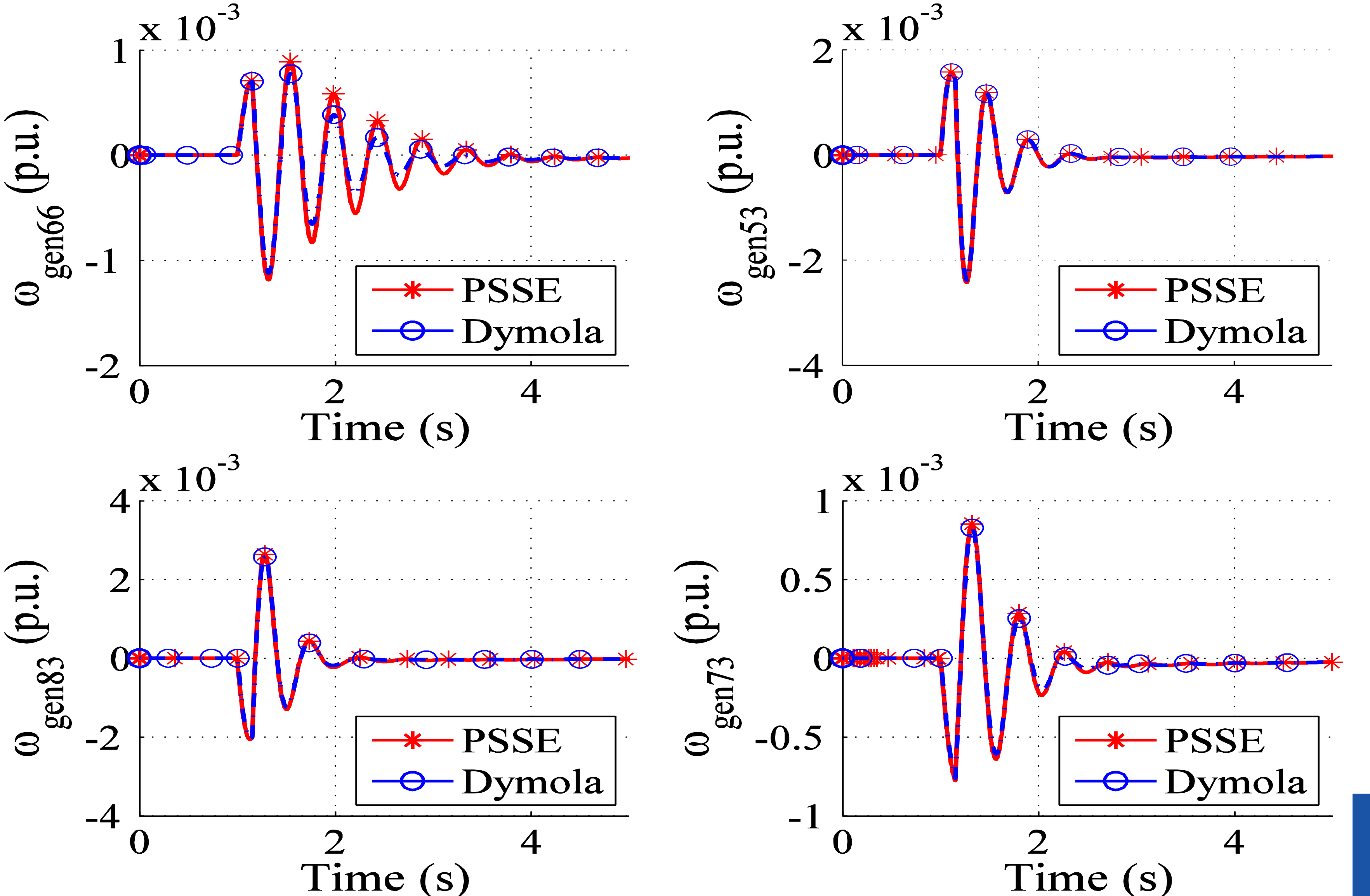


Subset of Norwegian grid model validation test

Step	Scenarios
1.	Running under steady state for 1s.
1.	Open the line for 0.15s.
1.	Apply 3phtgf at 7.5s; 0.15s later, clear the fault.
1.	Run simulation until steady state.



Subset of Norwegian grid model validation results



Contents

- Background
- Introduction
- Model implementation
  - ✓ State equations
    - Round pole generator
    - Three-winding transformer
  - ✓ Initialization procedure
    - Round pole generator
    - Excitation System
- Model Validation
  – Principles
  – Procedure
- Results
- Conclusions
- Future work
- Discussion

Conclusions

- All the component models used in Norwegian power grid have been successfully implemented in Modelica environment.

- The developed models have been tested using OpenModelica and Dymola. The validation results guarantee consistent simulation results among OM, Dymola and PSS/E.

- This work can serve as a proof of the feasibility and priority of utilizing equation-based Modelica tools in power system modeling and simulation.

- This work also provides a proof of simulation capabilities of OpenModelica to handle power system problems in terms of the complex controls and initialization problems.

Future work

- Test more Modelica models and larger size power system models in OpenModelica environment.

- Develop proper algorithm in order to utilize Modelica solver to solve initialization problem without providing explicitly entered power flow data.

Some results

During my Intership in FH Bielefeld

- Test more Modelica models and larger size power system models in OpenModelica environment.

- Develop proper algorithm in order to utilize Modelica solver to solve initialization problem without providing explicitly entered power flow data.

OpenModelica in Power System Simulation

- More modelica models from the power system library built by our SmarTS Lab have been tested in OpenModelica environment.

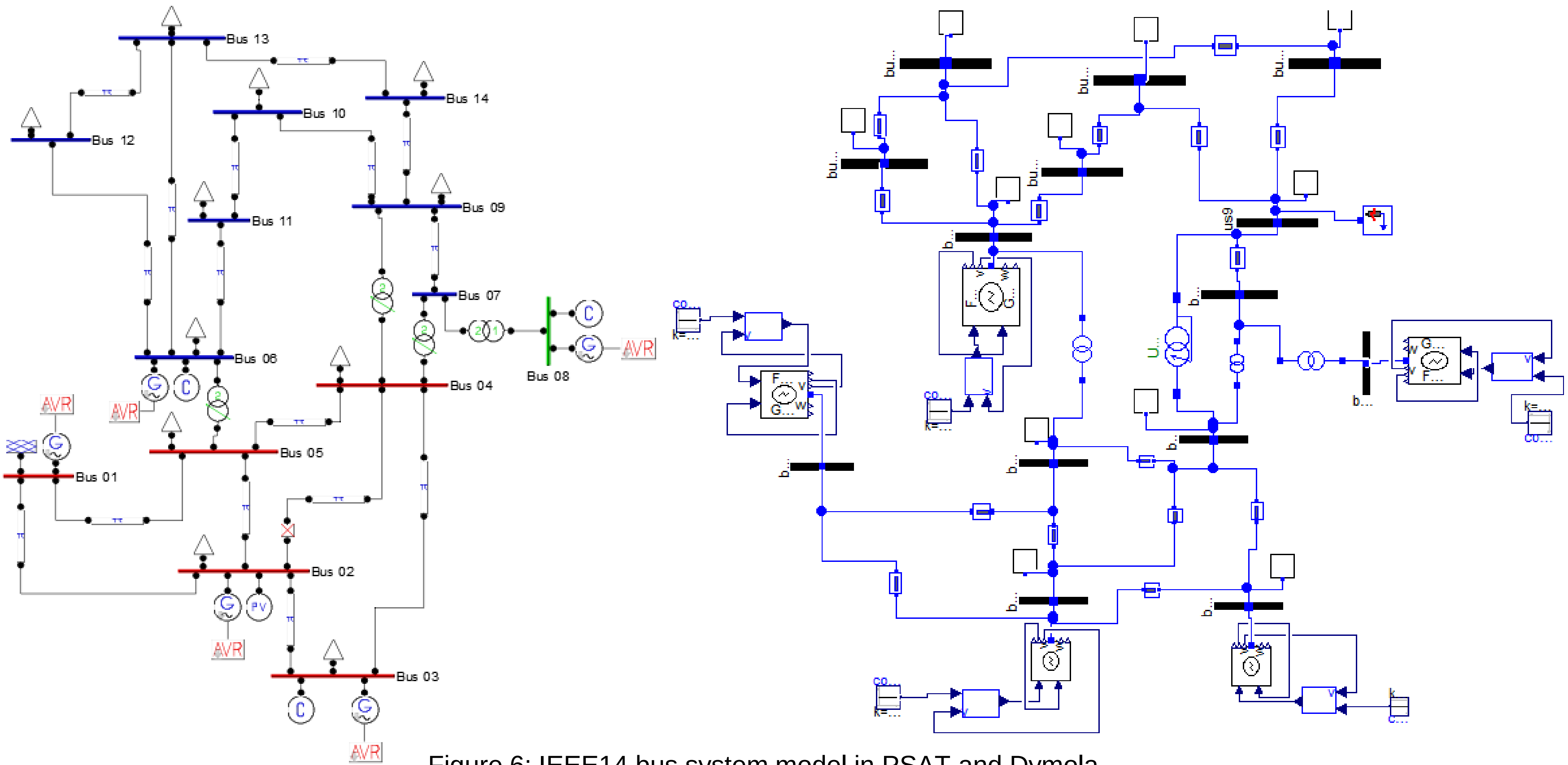
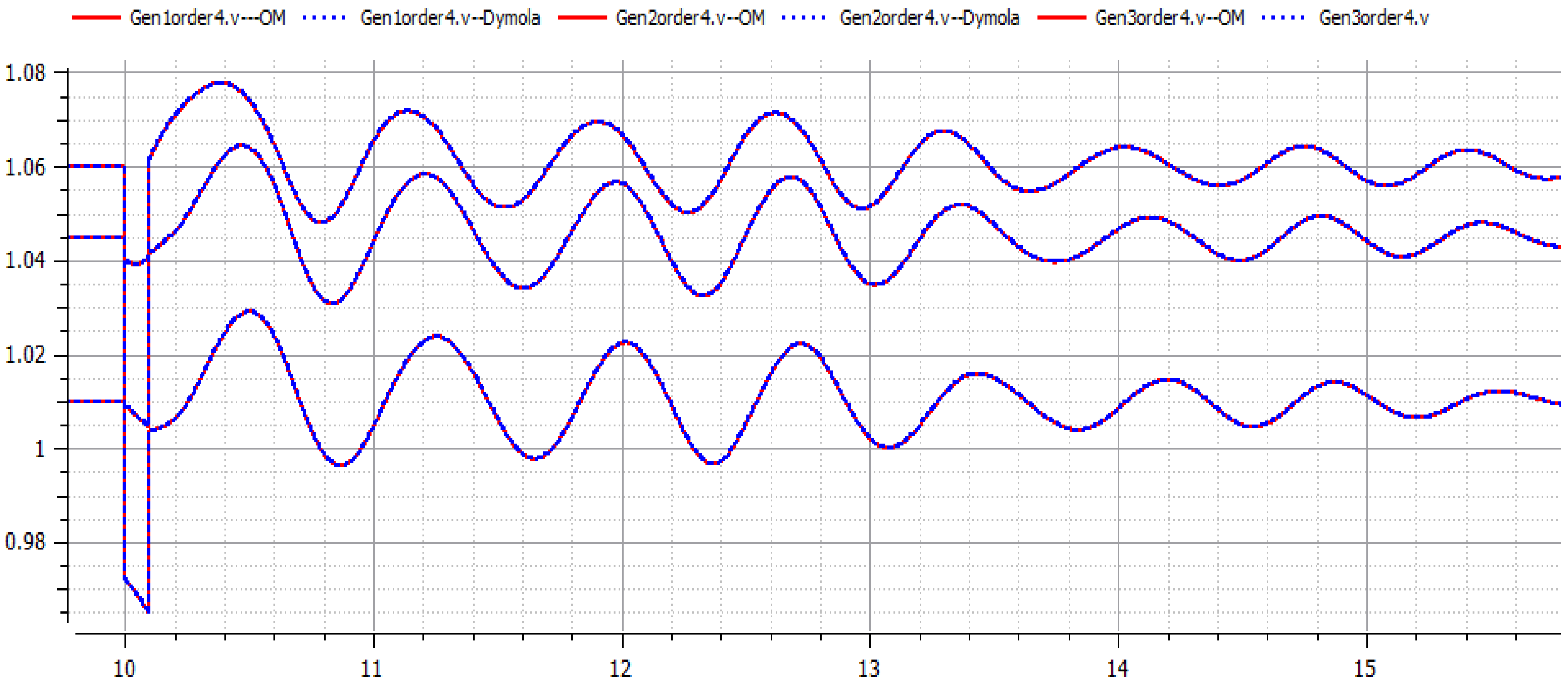
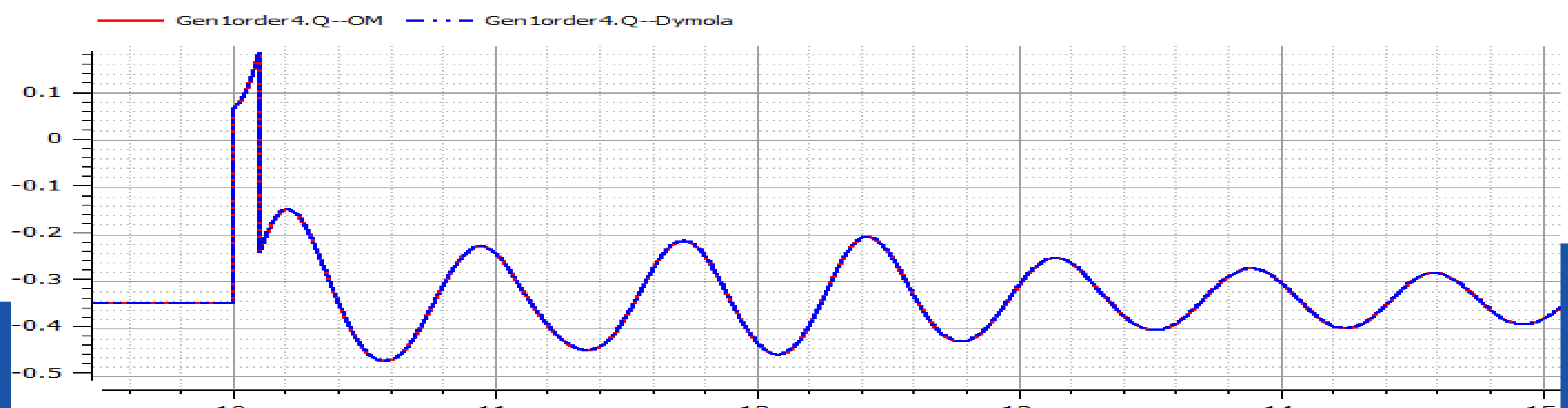


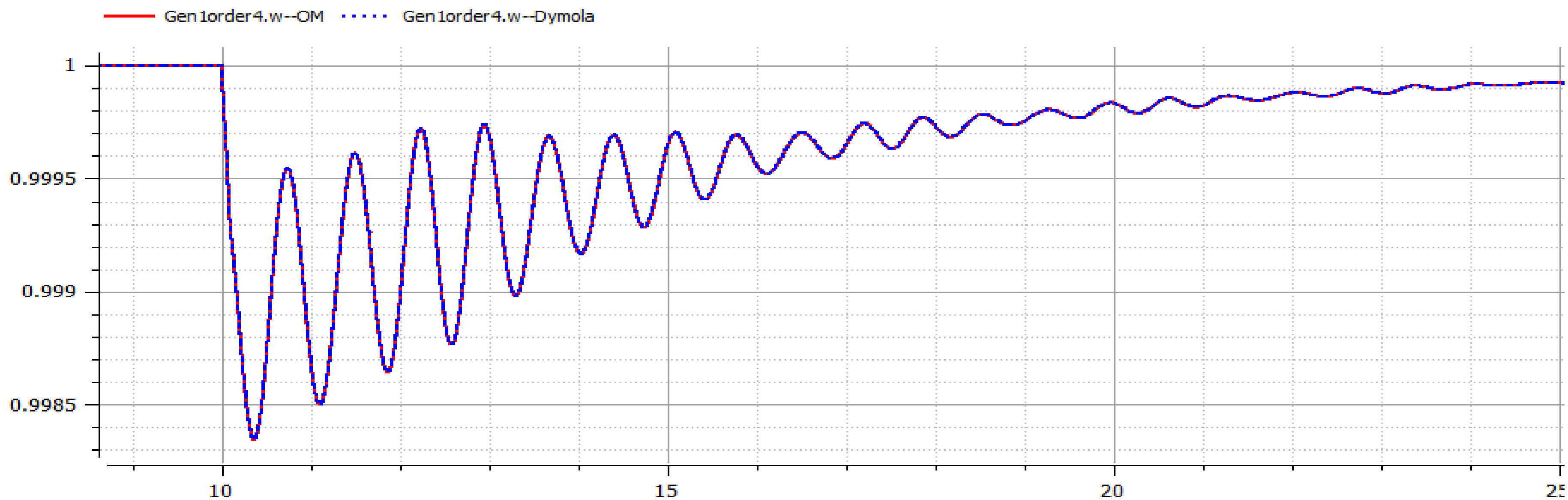
Figure 6: IEEE14 bus system model in PSAT and Dymola

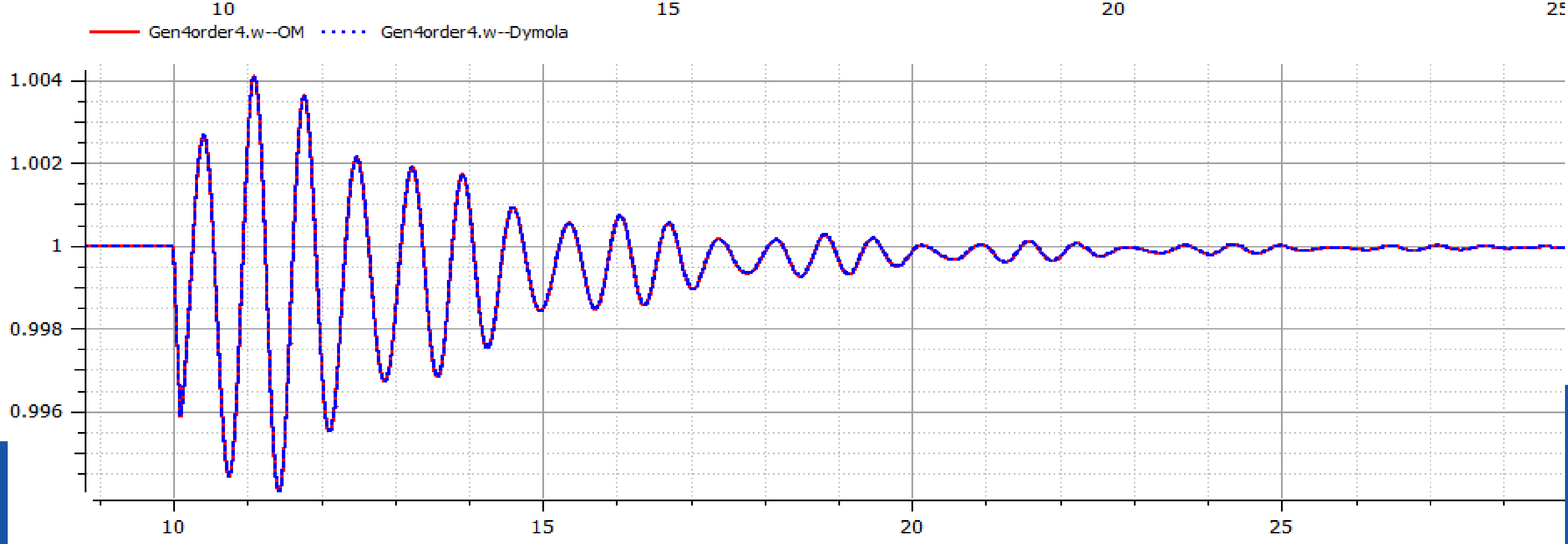
OpenModelica in Power System Simulation





OpenModelica in Power System Simulation





OpenModelica in Power System Simulation

- More modelica models from the power system library built by our SmarTS Lab have been tested in OpenModelica environment.

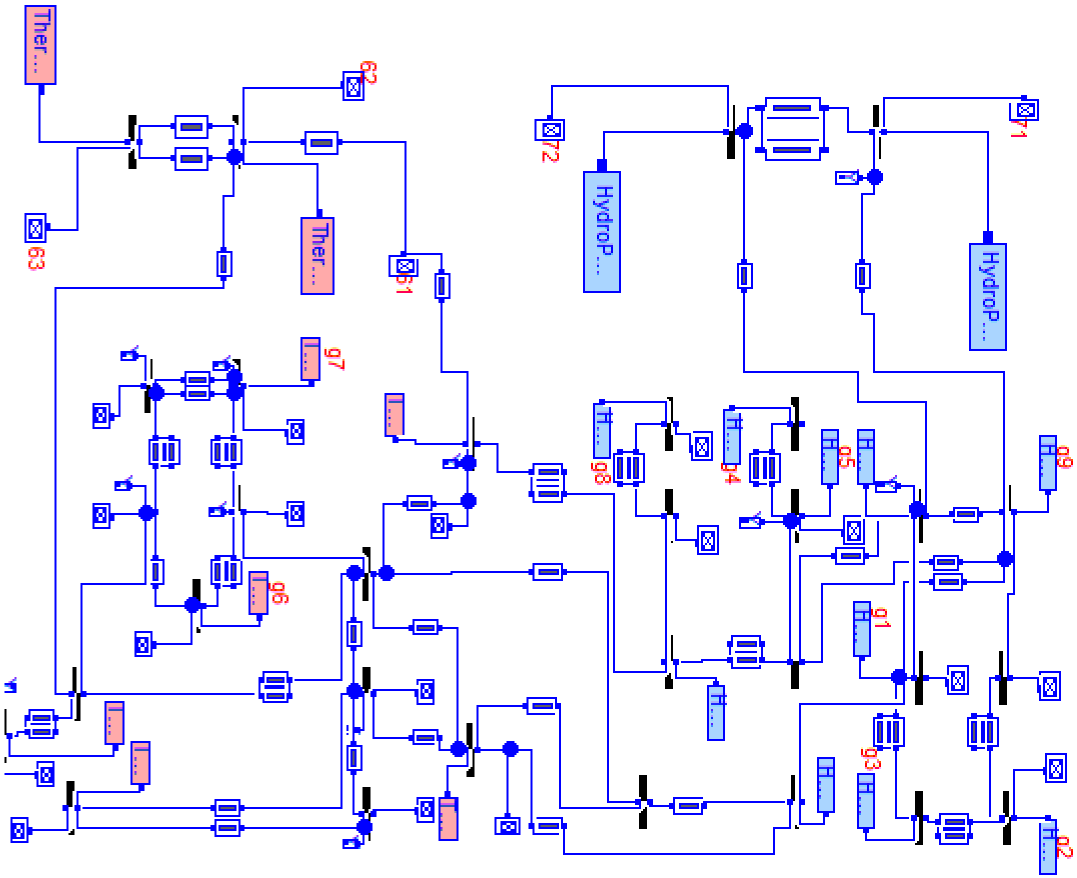
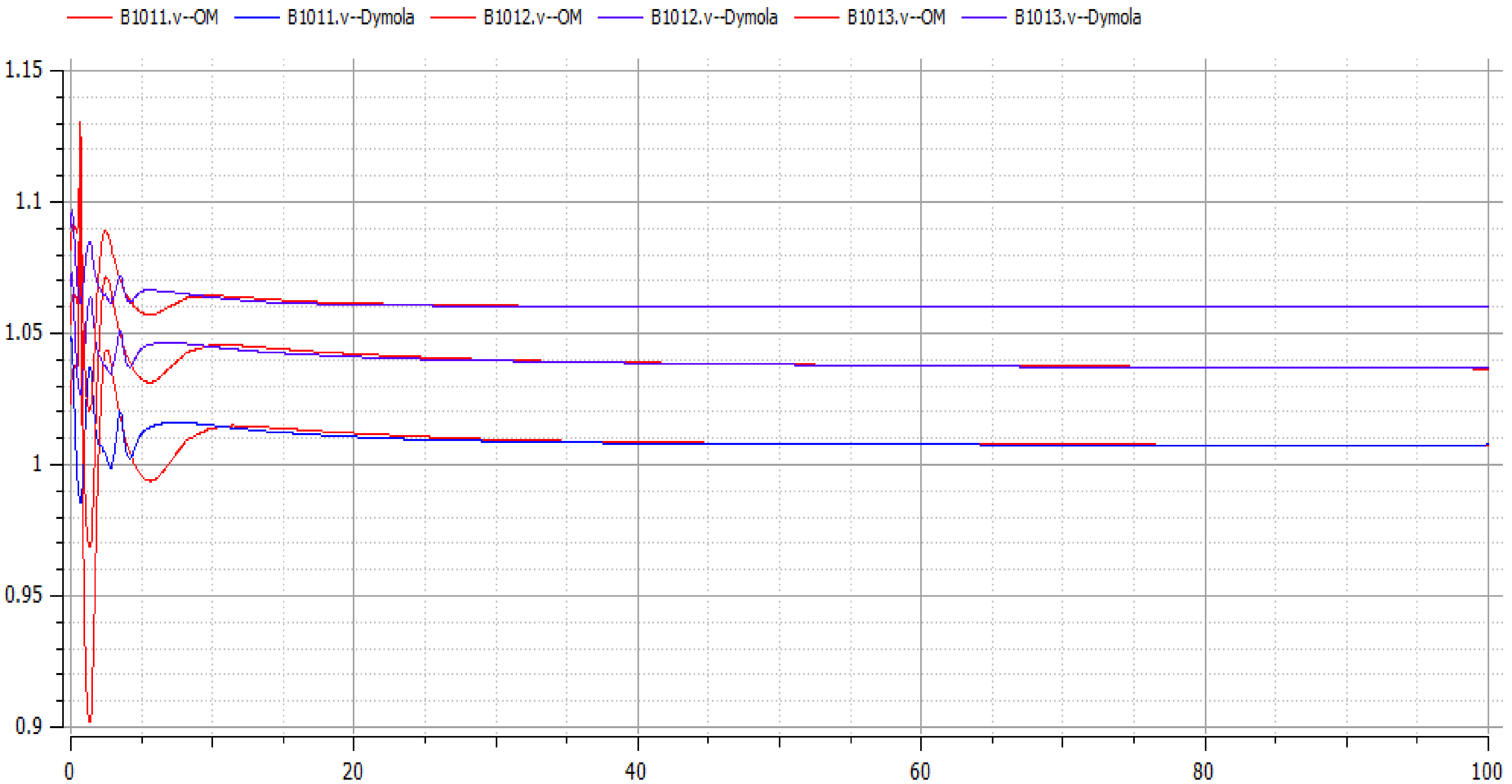


Figure 7: IEEE Nordic 32

OpenModelica in Power System Simulation



Future work

- Test more Modelica models and larger size power system models in OpenModelica environment.

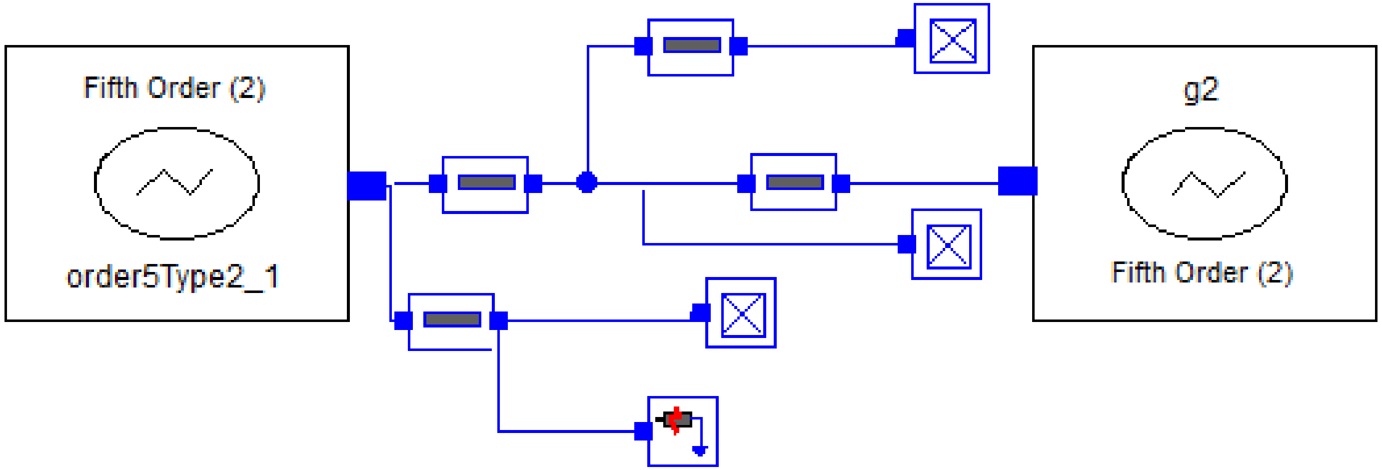
- Develop proper algorithm in order to utilize Modelica solver to solve initialization problem without providing explicitly entered power flow data.

Power flow algorithm in Modelica tools

- Power flow data as initial conditions should be specified at the system level.
- Free the related auxilary parameters such as initial active power, reactive power, terminal voltage and phase angle of the generator $(P_0, Q_0, V_0, angle_0)$ by setting (fixed=false).
- State the power balance relations at system level
  - At each node $S_{injected} = S_{exported}$
  - For the whole system $S_{generated} = S_{consumed} + S_{losses}$

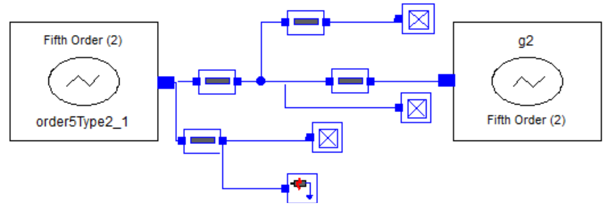
Power flow algorithm in Modelica tools

One of the test cases...



Power flow algorithm in Modelica tools

- Some results….



Translation of PF_in_Dymola.Order5_2test2_GOOD4_withfualt:

The DAE has 116 scalar unknowns and 116 scalar equations.

Statistics

Selected continuous time states

Statically selected continuous time states
g2.delta
g2.e1q
g2.e2d
g2.e2q
g2.w
order5Type2_1.delta
order5Type2_1.e1q
order5Type2_1.e2d
order5Type2_1.e2q
order5Type2_1.w

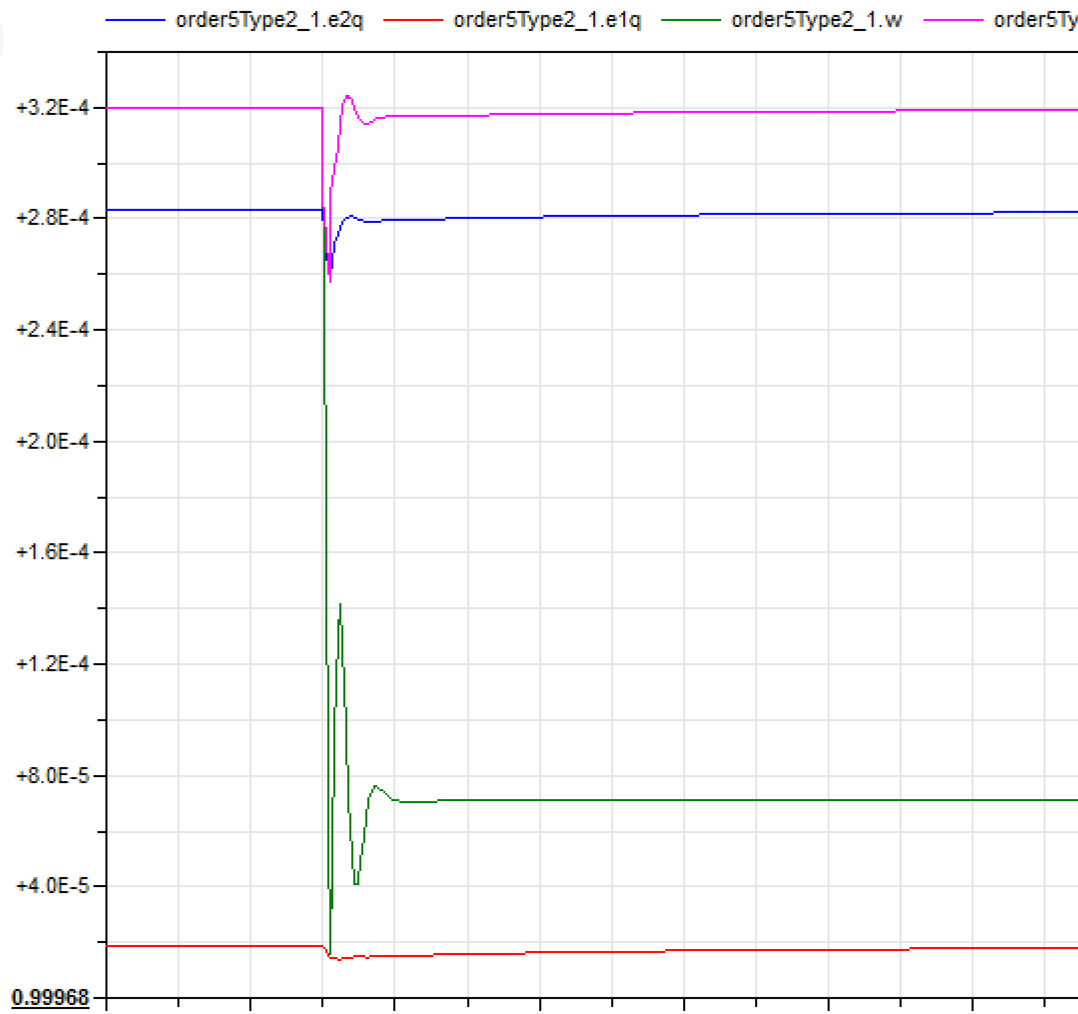
The following variables are iteration variables of the initialization problem:
g2.p.ii
g2.p.ir
pwLine1.p.ii
pwLine1.p.ir
pwLine2.p.ii
pwLine2.p.ir
pwLine3.n.ii
pwLine3.n.ir
pwLine3.p.ii
pwLine3.p.ir
pwLine4.n.ii
pwLine4.n.ir
pwLoadPQ1.p.ii
pwLoadPQ1.p.ir
pwLoadPQ2.p.ii
pwLoadPQ2.p.ir

but they are not given any explicit start values. Zero will be used.

Finished

// experiment StopTime=20

WARNING: 1 warning was issued



Citation

[1] *iTesla*, url=http://www.itesla-project.eu/

[2] Federico Milano, *Power System Modelling and Scripting*, Springer, 2010.

[3] J. Arrillaga and C.P. Arnold, Computer Analysis of Power Systems, John Wiley & Sons, 1990.

[4] L. Vanfretti, W. Li, T. Bogodorova, Unambiguous Power System Dynamic Modeling and Simulation using Modelica Tools, IEEE PES General Meeting, 2013.

KTH ROYAL INSTITUTE
OF TECHNOLOGY

Thank you for your attention

Back up slides

Genrou+IEEET1+STAB2A
validation results