

ObjectStab—An Educational Tool for Power System Stability Studies

Mats Larsson, *Member, IEEE*

Abstract—Traditionally, the simulation of transient and voltage stability in power systems has been constrained to domain-specific tools such as Simpow, PSS/E, ETMSP, and EuroStag. While being efficient and thereby able to simulate large systems, their component models are often encapsulated and difficult or impossible to examine and modify. Also, these simulators often require substantial training and are therefore not ideal for normal classroom use. For academic and educational use, it is more important that the component modeling be transparent and flexible, and that students quickly get started with their simulations. This paper describes a freely available power system library called ObjectStab intended for power system stability simulations written in Modelica, a general-purpose object-oriented modeling language. All component models are transparent and can easily be modified or extended. Power system topology and parameter data are entered in one-line diagram form using a graphical editor. The component library has been validated using comparative simulations with EuroStag.

Index Terms—Author, please supply your own keywords or send a blank e-mail to keywords@ieee.org to receive a list of suggested keywords.

I. INTRODUCTION

THE simulation of power systems using special purpose tools is now a well-established area and several commercial tools are available (e.g., PSS/E, Simpow, DigSilent, NEPLAN, and EuroStag). While most of these tools are computationally very efficient and reasonably user-friendly they have a closed architecture where it is very difficult or impossible to view or change most of the component models. Some tools (e.g., EuroStag and Simpow), provide the possibility of modeling controllers such as governors and exciters using block diagram representation or using their own textual modeling language, but this is often cumbersome. The Power System Toolbox [1] is a set of Matlab program files capable of dynamical simulation of power systems. Here, the component models are accessible, but the modification of these requires a proper understanding of the interaction of the model files in this specific environment. Graphical editing of the models is not possible. Power System Analysis toolbox (PSAT) [2] is a freely available Matlab implementation that contains not only time-domain simulation but also a set of routines for static analysis such as load-flow and continuation power-flow analysis. It also contains an interface where users can enter new models using equation-based modeling, and a one-line diagram editor where the power system topology can be entered. However,

source code is not available so that the files can be edited or extended.

Most of the above mentioned tools have the capability of exporting a linearized representation of the system for further analysis but the full nonlinear representation remains hidden to the user. Also, most industrial-grade tools require substantial training before they can be used productively. On the other hand, PowerWorld is a simulator suitable for classroom use. Power system diagrams are built using a graphical user interface and the simulation result is visualized during simulation. However, the modeling detail is limited to static load-flow-type models so the tool cannot be used to illustrate dynamic phenomena such as oscillatory or transient stability. The component modeling is also here encapsulated so it is not possible for the user to add dynamic generator or load models.

The aim of the ObjectStab tool is to provide an environment within which students can quickly get started and provides enough modeling flexibility to allow modification or addition of new generator, load, and control system models. Whereas ease of use is mostly important for coursework, the modeling flexibility is useful mostly for research applications. Since the tool is based on the standardized (tool-independent) modeling language Modelica [3], this environment could also be used to specify new component models and benchmark power systems.

The model library contains models suitable for load-flow and power system dynamic stability studies in power systems. It is designed for use by undergraduate and graduate students in the learning of power system dynamic stability and for rapid testing of research ideas. All component models are transparent and can easily be modified or extended. Emphasis has been given to keeping the component models transparent and simple. Power system topology and parameter data are entered in one-line diagram form using a graphical editor as shown in Fig. 1. Using the Dymola simulator, the full hybrid nonlinear or linearized equation system can be exported for use as a block in Simulink simulations and the full range of MATLAB tools for visualization and control or online optimization can then be used. The component library has been validated using comparative simulations with EuroStag [4].

II. MODELICA

Modelica [3] is an object-oriented modeling language that is under development in an international effort to define a unified language for modeling of physical systems. Modelica supports object-oriented modeling using inheritance and aggregation concepts similar to those used in programming languages such as Simula, Java, and C++, and this has been extensively used in the design of the language. It also supports noncausal

Manuscript received July 16, 2003.

The author was with Lund University, PLEASE PROVIDE CITY AND POSTAL CODE Sweden. He is now with Utility Solutions, Corporate Research, Baden-Dättwil 5405, Switzerland (e-mail: mats.larsson@ch.abb.com).

Digital Object Identifier 10.1109/TPWRS.2003.821001

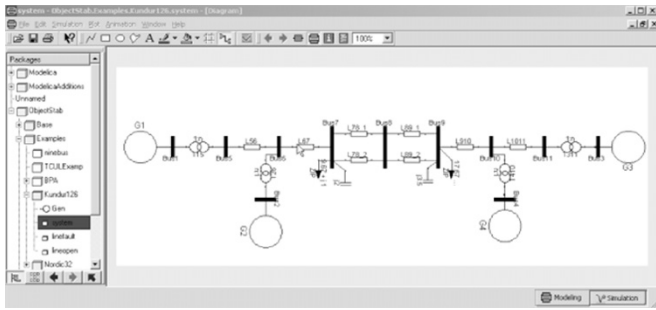


Fig. 1. Four-generator test system from [5].

modeling, meaning that a model's terminals do not necessarily have to be assigned an input or output role. Models can be entered as ordinary or differential-algebraic equations, state-machines and block diagrams, or any combination thereof. Modelica also supports discrete-event constructs for hybrid modeling.

While the object-oriented concepts enable proper structuring of models and organization of test cases, the capability of non-causal modeling makes it easy to model for example power lines that are very cumbersome to model using signal-oriented languages such as Simulink. Causality is generally not defined in electrical systems [6]. For example, when modeling a resistor, it is not evident ahead of time, whether an equation of the type

$$u = Ri \quad (1)$$

will be needed, or one of the form

$$i = \frac{u}{R}. \quad (2)$$

It depends on the environment in which the resistor is embedded. Consequently, the modeling tool should relax the causality constraint that has been imposed on the modeling equations in the past. Using causal modeling as in Simulink that does not have the capability of relaxing such constraints is illustrated in [7] where a model for transient stability simulation is described. With this approach, the network has to be modeled using the traditional admittance matrix method and the power system topology cannot be visualized in the graphical editor.

Presently, there are two commercial simulation tools supporting the Modelica language, Dymola [8] and MathModelica [9], although some other producers of general-purpose simulators have announced that future versions of their programs will support Modelica. There is already a number of free and commercial Modelica libraries for various application domains such as; three-dimensional (3-D) multibody systems, rotational and translational mechanics, hydraulics, magnetics, thermodynamical systems, and chemical processes [3].

III. COMPONENT LIBRARY

The ObjectStab library contains the following components:

- generators with constant frequency and voltage as slack or PV nodes, or using third- or sixth-order dq-models with detailed models of excitation and governor control systems, or as quasisteady state equivalents;

- transmission lines in pi-link or series impedance representation;
- reactive power compensation devices; shunt reactors, shunt capacitances, and series capacitances according to [10];
- fixed ratio transformers;
- onload tap changing transformers (OLTCs) modeled as detailed discrete models or using corresponding continuous approximations according to [11];
- static and dynamic loads, including generic exponential recovery loads [12];
- buses;
- faulted lines and buses with fault impedance;
- various example power systems, with sizes from two to 41 buses;
- various relay models such as underfrequency, under-voltage, overcurrent, and distance relays.

Standard assumptions for multimachine power system stability simulations are made (i.e., generator stator and network time constants are neglected and voltages and currents are assumed to be sinusoidal and symmetrical). Except where other references are given, the components are modeled according to the guidelines given in [10] but have been adapted to the object-oriented environment. Furthermore, each generator's set of equations is referred to an individual dq-frame and the stator equations are related to the system (common) reference frame through the so-called Kron's transformation [10].

All models can be modified and extended through inheritance. New models or extensions of old ones can be entered as differential and algebraic equations, block diagrams, or state-graphs.

Connectors are a special type of model in Modelica. Such models contain only variable declarations but are anyhow central to any device model library written in Modelica. The connector defines the interface variables, that is, the quantities that will be directly affected by the connection of two or more devices. In the ObjectStab library, all of the component models contain at least one connector. There can be two types of variables in a connector. The first type is used for potential variables whose values are set equal when two connectors are connected. Typical examples are electrical voltage, pressure, and rotational velocity. Second, there are flow variables which require summing to zero when two connectors are connected, such as electrical current, mass flow, and mechanical force.

In the ObjectStab library, voltages and currents are described by their phasor representation

$$\mathbf{I} = i_a + j i_b \quad (3)$$

$$\mathbf{V} = 1 + v_a + j v_b. \quad (4)$$

Using this representation, a connection point for power system components is defined using the following Modelica connector definition:

```
connector Pin
  Real va;
  Real vb;
  flow Real ia;
```

```

    flow Real ib;
end Pin;

```

By definition, all Pins use a per-unit system based on the system reference of 100-MVA, although the generator models support parameter entry on their own individual base. Voltage is defined as a potential variable and current as a flow variable. The one per-unit offset for the real part of the voltage in (4) has been introduced for historical reasons. Older versions of the Dymola tool did not support user-specified initialization of connector variables. With the default values of zero for both v_a and v_b , the initial value problem would most often become singular and the simulation could not be started. This issue was solved by adding the offset, but would no longer be necessary in a newly developed library, since the tool now supports such initializations.

A. Modeling Example: Impedance Line Model

The inheritance hierarchy for the impedance transmission line model is given below:

```

partial model TwoPin
    Pin T1;
    Pin T2;
end TwoPin;

```

The definition implies that a TwoPin is a model with two pins named T1 and T2 that will be used as external connectors for the line. This very basic class is placed in a package called Base along with definitions of different units such as voltage and current. It will be used as base class for transmission line and transformer models.

The Impedance model is defined using the TwoPin as a base class and, thereby, inherit its attributes. Furthermore, equations defining the real and imaginary voltage drop on the line have to be given:

```

model Impedance
    extends Base.TwoPin;
    parameter Base.Resistance R = 0.0;
    parameter Base.Reactance X = 0.1;
    equation
        [T1.va - T2.va; T1.vb - T2.vb] = [R, -X; X, R] *
        [T1.ia; T1.ib];
        [T1.ia; T1.ib] + [T2.ia; T2.ib] = [0; 0];
end Impedance;

```

B. Modeling Example: Third-Order Generator Model

The dynamic generator models inherit from the OnePin class and its declarations are split into three classes in order to simplify the implementation of new generator models. The class DetGen defines the swing equation and the coordinate transformations as follows:

```

partial model Partial.DetGen
    extends Base.OnePin;
    parameter Base.Voltage V0 = 1;

```

```

    parameter Base.ActivePower Pg0 = 1;
    parameter Base.VoltageAngle theta0 = 0;
    parameter Boolean isSlack=true;
    parameter Base.ApparentPower Sbase =
    100;
    parameter Base.InertiaConstant H = 6;
    parameter Base.DampingCoefficient
    D = 0;
    Base.AngularVelocity w(start = 1);
    Base.VoltageAngle delta;
    Base.MechanicalPower Pm;
    Base.VoltageAmplitude Efd;
    Base.Current id, iq;
    Base.Voltage vd, vq;
    Base.Current Iarm = sqrt(id^2 + iq^2);
    Base.ActivePower Pe;
    equation
        // swing equations
        2*H*der(w) = Pm - Pe - D/Base.ws*(w -
        Base.wref);
        der(delta) = Base.ws*(w - Base.wref);
        // Kron's transformation
        -[T.ia; T.ib] = [-sin(delta), cos(delta);
        cos(delta), sin(delta)]*[id; iq];
        [1 + T.va; T.vb] = [-sin(delta), cos(delta);
        cos(delta), sin(delta)]*[vd; vq];
        // Load-flow initialization equation
        initial equation
            w = 1;
            der(w) = 0;
            V = V0;
            if isSlack then
                theta = theta0;
            else
                Pg = Pg0;
            end if;
        end DetGen;

```

During the initialization, which is equivalent to a load-flow calculation, generators are represented as either PV-nodes or Slack nodes [10] depending on the attribute isSlack. For each dynamic state variable, that is, variable on which the der-operator has been applied, one equation has to be given in the initial equation section. For the generator, it is specified that it should be initialized at steady state and with rated speed. It is declared as a partial class (i.e., noninstantiable), since it lacks equations for the stator and **AUTHOR: "EMF" REFERS TO "ELECTROMOTIVE FORCE"?** EMF equations that are dependent on the type of generator model used.

For the third-order generator model with a single transient EMF in the q-axis, the definition is as follows:

```

model ObjectStab.Generators.Par-
    tials.DetGen3
    extends DetGen;
    parameter Base.Resistance ra = 0;
    parameter Base.Reactance xd = 0.8948;
    parameter Base.Reactance xq = 0.84;

```

```

parameter Base.Reactance xdp = 0.30;
parameter Base.Time Td0p = 7;
Base.Voltage Eqp(start = 1);
equation
// Transient EMF equation
Td0p*der(Eqp) = Efd - Eqp + id*(xd - xdp);
// stator equations
vd = -ra*id - xq*iq;
vq = Eqp + xdp*id - ra*iq;
// electrical power
Pe = Eqp*iq + (xdp - xq)*id*iq;
initial equation
der(Eqp) = 0;
end DetGen3;

```

Also, DetGen3 is declared as a partial class since it does not contain equations for the governor and excitation system. These are provided in the instantiable class GovExc3rdGen, which contains models of the governor and excitation system:

```

model GovExc3rdGen
  extends Partials.DetGen3;
  replaceable Controllers.ConstPm Gov;
  replaceable Controllers.ConstEfd Exc;
equation
// connection equations
Gov.u = w;
Pm = Gov.y;
Exc.u1 = sqrt(vd^2 + vq^2);
Exc.u2 = w - Base.wref;
Efd = Exc.y;
end GovExc3rdGen;

```

Since the Governor and Exciter models have been declared using the modifier `replaceable`, they can be replaced for models taken from a controller library or for user-defined models in instantiations of the GovExc3rdGen class.

IV. EXTENDING THE LIBRARY

The goal has been to provide a framework with the basic models necessary for power system stability studies. However, such a library can never be complete and users can create their own models inheriting from the (partial) base classes and providing the equations given below:

- generic current injector models OnePin and TwoPin—equations for each of the real and imaginary currents of each connector (Pin);
- loads—equations for active and reactive load;
- generators—equations for active and reactive production;
- governor, exciter—generator control systems, equations for the mechanical shaft power **AUTHOR: COMMA HERE?** or the field voltage must be given.

These equations can be defined by block diagrams, state machines, or differential or algebraic equations. Internal dynamic states in user-defined models should be initialized, normally by setting the state derivatives to zero, similar to what is done for the generator model in Section III-B.

A. Extension Example: Exciter IEEE ST1A [5]

New exciter models can be constructed by creating a new class that inherits from the partial class Exciter. This partial class contains the declarations necessary for communication with the generator models and calculation of the initial values of the field voltage and voltage reference. The block diagram can then be drawn using components from the block libraries as shown in Fig. 2, or alternatively, the corresponding differential or algebraic equations can be entered in the equation window of the new model. The design of new governor system is analogous. Note that also equations for dynamic blocks, which at the start of the simulation have nonzero output, must be given as shown in Fig. 2.

V. CASE STUDY: FOUR GENERATOR TEST SYSTEM

The following steps must be carried out to model and simulate a power system using the ObjectStab library and the Dymola simulator.

- 1) Draw the power system in one-line diagram form in the model window as shown in Fig. 1 using the graphical model editor. New components are created by dragging them from a library window into the model window. By double-clicking on a component in the model window, a dialog box appears and the parameters can be entered.
- 2) Switch Dymola into simulation mode.
- 3) Simulate the system by choosing “Simulate” from the “Simulation” menu of the simulator window. If desired, parameter values can be adjusted using a dialog box prior to starting the simulation.
- 4) Select variables to plot from the menu in Dymola.

When the model is built by hand, it is usually most convenient to use the graphical editor, but when systems are converted from data files in text form, it is easier to generate a textual representation of the system. The model of a particular power system is then written in Modelica similar to the component models described here.

If the system is built using the graphical editor, parameter values are entered using dialog boxes and the corresponding textual representation is automatically generated. Fig. 1 shows the graphical display of the four-generator test system [5] used for validation of the library. Fig. 3 shows a comparison of simulation results obtained using the ObjectStab library and EuroStag [4]. At simulation time 2 s, a ground fault occurs close to bus 7. The faulted line is disconnected at both ends at 2.07 s, and later, at 3.5 s, the fault has cleared and the line is reconnected. From the figure, we can see that there is a near-perfect agreement between the results of the two programs. The Modelica code for the system is:

```

model linefault
  extends Examples.Kundur126.system(
    redeclare Extras.FaultedPQPilink
  L67_5(
    FaultTime = 10, FaultR = 0, FaultX = 0,
    ClearTime = 0.07, RecloseTime = 3.5) );
end linefault;

```

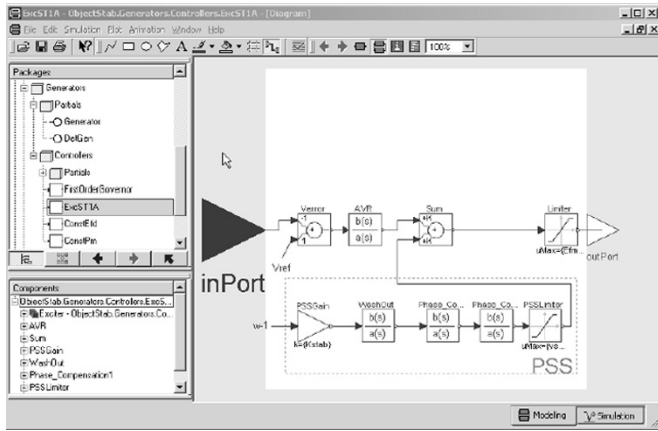


Fig. 2. Diagram representation of the IEEE ST1A excitation system model [5].

This example illustrates how different disturbance scenarios can be modeled by redeclaring components of the base model. The model `FaultedPQPilink`, which is used in the redeclaration, is a `Pilink` line model that has been extended by a fault and logic for fault clearing and reclosing. The model that is extended `Examples.Kundur126.system` is the basic power system model as shown in Fig. 1. Components that are to be replaced must be declared using the `replaceable` keyword.

In terms of simulation efficiency, the Dymola tool is typically faster than industrial grade tools such as PSS/E or EuroStag for small power system models because of the smaller amount of overhead required to start a simulation. Because of the lack of sparse solvers in Dymola, this speed advantage is, however, turned into a disadvantage for larger power systems. Table I lists the compilation and simulation time for some of the examples models that are included with ObjectStab. The test was made on a 900-MHz IBM Thinkpad laptop with Dymola 5.0 c and the Microsoft Visual C++ compiler. The simulations used the default settings in Dymola for the `lsodar` integration algorithm. For the two small examples, the time is clearly dominated by the compilation time for the model.

VI. CASE STUDY: SIMULTANEOUS PSS TUNING

For the design of controllers, it is often convenient to use Matlab/Simulink because of the wide variety of toolboxes available on that platform. We will illustrate such a procedure using a case study where Matlab is used to find the optimal tuning of the PSSs in the test system shown in Fig. 1. Note that the tuning is somewhat naive since we are making the design based on one single operating point, but as an example of how to programmatically access the power system model from Matlab it is adequate. Also, the optimization problem appears to have many local minima so some experimentation with the initial parameters is advisable. We here choose to start the optimization from the nominal parameters given in [5]. A more detailed study of simultaneous PSS tuning using a software for genetic algorithms and ObjectStab has been reported in [14].

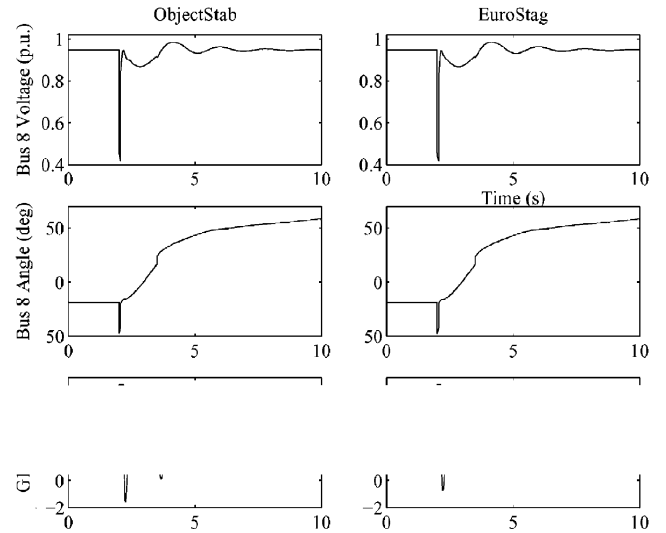


Fig. 3. Simulation results obtained using the ObjectStab library and EuroStag.

TABLE I
SAMPLE SIMULATION TIMES WITH
DYMOLA 5.0 AND OBJECTSTAB

	Kundur (Fig. 1)	CIGRE BPA	CIGRE Nordic
# of buses	11	11	41
# dynamic states	44	24	288
Simulated time range	0-20 s	0-158 s	0-233 s
Scenario	Bus fault+line trip	Line fault+voltage collapse	Gen trip+voltage collapse
Compilation time	6 s	5 s	31 s
Simulation time	1.77 s	4.05 s	283 s

A. Simulink Integration

Dymola includes a Matlab toolbox to translate Modelica models into Simulink S-Mex functions that can be simulated by Simulink. The inputs and outputs are then declared on the top level of the power system model as shown in the example below:

```

model TunePSS
  extends Examples.Kundur126.linefault;
  output Real w1, w2, w3, w4;
equation
  w1 = G1.w;
  w2 = G2.w;
  w3 = G3.w;
  w4 = G4.w;
end TunePSS;
    
```

The model simply extends the example power system model with the required outputs. These will become outputs also of the translated model. Here, the generator speeds are declared as outputs. Note that also inputs can be specified in an analogous way. After translation to a S-mex function, the power system model is included in a block diagram as shown in Fig. 4.

The Dymola compiler translates any Modelica model into the following ordinary differential equation (ODE) form:

$$\dot{x} = f(t, x, u) \quad (5)$$

$$y = g(t, x, u). \quad (6)$$

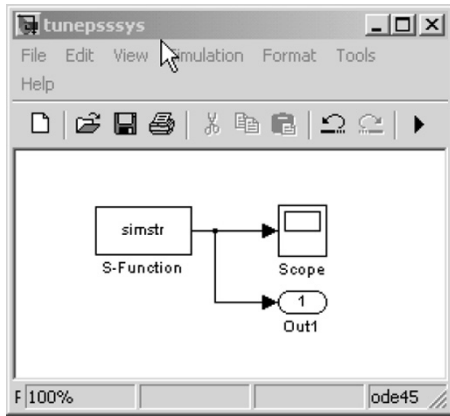


Fig. 4. Sample Simulink model that includes the S-function containing a power system model.

Note that the power system model typically contains also many algebraic loops, and that a power system model therefore usually has an differential-algebraic (DA) structure. Dymola applies symbolic index-reduction techniques [3] to reduce the DA structure to ODE form, thereby handling any issues with algebraic loops. Signal-oriented modeling tools, such as Simulink, usually have to rely on ad-hoc methods, such as adding time-delays, etc., to resolve algebraic loops. This often results in ill-conditioned equation systems that require small step sizes in the integration. A power system model generated by Dymola can therefore be expected to be considerably more efficient than an equivalent model implemented directly in Simulink.

The model (5), (6) can be programmatically accessed from Matlab via the S-mex function (e.g., to linearize the power system model).

B. Tuning Strategy

After linearization, the power system model can be described on the following form:

$$\Delta \dot{x} = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial u} \Delta u = \mathbf{A} \Delta x + \mathbf{B} \Delta u \quad (7)$$

$$\Delta y = \frac{\partial g}{\partial x} \Delta x + \frac{\partial g}{\partial u} \Delta u = \mathbf{C} \Delta x + \mathbf{D} \Delta u. \quad (8)$$

After computing the eigenvalues of the matrix \mathbf{A} , it is straightforward to obtain the damping and frequencies of a mode related to a complex eigenvalue pair $a_i = \alpha_i \pm j\beta_i$ as follows:

$$z_i = -\cos\left(\arctan\left(\frac{\beta_i}{\alpha_i}\right)\right). \quad (9)$$

We then determine the optimal parameter set p_{opt} based on the criterion

$$\begin{aligned} & \text{maximize} && \min(z_i) \\ & \text{subject to} && i \in 1 \dots N \end{aligned} \quad (10)$$

where N is the number of complex eigenvalue pairs. That is, the aim is to maximize the damping of the least damped mode.

C. Matlab Implementation

The objective function in (10) is coded as follows in Matlab:

```
function f = pss_obj(x);
global p x0 pnames pindex
% set PSS parameters from optimization
% the p-vector is passed to the
S-function
% in Fig. 5
p(pindex) = x;
% linearize
[A, B, C, D] = linmod('tunepsssys', x0);
% compute eigenvalues
eigs = eig(A);
% extract oscillatory modes
modes = eigs(find(imag(eigs) > eps));
% compute damping and frequency
wn = abs(modes);
z = -cos(atan2(imag(modes), real(modes)));
% return negative minimum damping
f = -min(z).
```

The Matlab command `linmod` is used to linearize the nonlinear model (5), (6). We will use the function `fmincon` from Matlab's optimization toolbox to solve the optimization problem. The minus sign on the last row of the objective function has been introduced since that function can only solve minimization problems.

A Matlab script that makes the optimization is shown in the Appendix. This illustrates how the Modelica model is translated and how it can be accessed through the Matlab/Simulink functions for simulation and linearization. First, the Modelica model is compiled by a call to the Dymola model, which is thereafter translated to a S-function by the call to `dymcomp`. After setting up the input to the environment, the optimization problem is then solved by the call to `fmincon`.

D. Results

Table II lists the PSS parameters used for three different simulations. First, with the nominal parameters from [5], second with the PSSs turned off and third with the optimal parameters given by the solution of (10). The damping of the critical mode is predicted as -0.017 with PSSs turned off, 0.15 with nominal parameters, and 0.46 with the optimal parameters. These predictions are confirmed by the results from time simulation in Fig. 5. Without PSSs, the system is locally unstable and sustained oscillation is observed following the tripping and reclosing of the line. Furthermore, we see that the oscillations are better damped and that the settling time is considerably smaller with the optimal tuning than for the nominal case.

VII. HOW TO GET OBJECTSTAB RUNNING

The ObjectStab library was constructed for use with the simulation tool Dymola, which runs on Microsoft Windows as well as SUN Solaris and Linux platforms. In addition to the component models, the library includes models of the four-generator test system used in this paper and other examples such as the

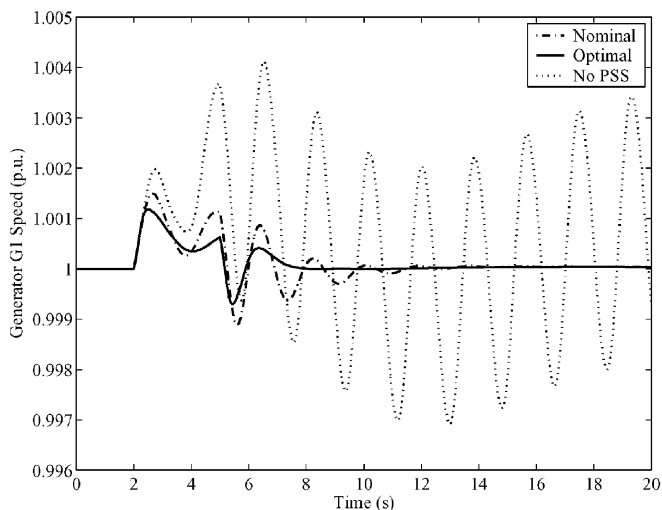


Fig. 5. Simulation results with PSSs turned off, nominal tuning and with the optimal tuning.

TABLE II
PSS PARAMETERS FOUND THROUGH OPTIMIZATION

	Nominal	No PSS	Optimal	G1	G2	G3	G4
Kstab	20	0.00	Kstab	19.98	20.00	19.98	19.98
T1	0.05	n/a	T1	0.74	0.54	0.53	1.02
T2	0.02	n/a	T2	0.75	0.13	0.94	0.28
T3	3	n/a	T3	3.61	3.50	3.65	3.60
T4	5.4	n/a	T4	5.08	5.12	5.03	5.10
max min(z_i)	0.15	-1.70E-02		0.46			

Nordic 32 test system from [13]. A free demo version of Dymola for Windows can be downloaded from the Dynasim.¹ Note that the demo version is fully functional, but can simulate only the prepared example power systems. The library itself can be downloaded from the ObjectStab group at Yahoo groups, which now has about 50 members,² and is free for educational use. The source codes for the PSS tuning example can also be downloaded from the Yahoo group website.

VIII. CONCLUSION

A component library called ObjectStab for power system transient and voltage stability simulations has been presented. Using the library, students can quickly enter their power system in one-line diagram form using a graphical editor, without having to type cryptic data files in text format or entering parameter data in countless dialog boxes. The library has an open structure and all models can be modified or extended using various Modelica constructs, such as state machines, block diagrams, or plain algebraic or differential equations. The models also have reasonable default parameter values defined such that students can play around with the models even before they have full understanding of the meaning of all system parameters. It is primarily intended as an educational tool, which can be used to experiment with and observe the effect of the different levels of modeling or trying out new types of controllers. As an example, it is shown how to combine the

tool with Matlab to make simultaneous PSS tuning using the optimization functions in Matlab.

APPENDIX

APPENDIX: MATLAB CODE TO SOLVE (10)

```

global p x0 pnames pindex
% compile Dymola model
dymola ('translateModel
('TunePSS.TuneSys');');
% create simulink mex function
dymcomp;
% load parameter values and names
[p,x0,pnames,x0names] = loaddsin;
pnom = p; % nominal parameters
% set parameter values
pindex = [
    tindex (pnames, 'G1.Exc.Kstab');
    tindex (pnames, 'G2.Exc.Kstab');
    tindex (pnames, 'G3.Exc.Kstab');
    tindex (pnames, 'G4.Exc.Kstab');
    tindex (pnames, 'G1.Exc.T1');
    tindex (pnames, 'G2.Exc.T1');
    tindex (pnames, 'G3.Exc.T1');
    tindex (pnames, 'G4.Exc.T1');
    tindex (pnames, 'G1.Exc.T2');
    tindex (pnames, 'G2.Exc.T2');
    tindex (pnames, 'G3.Exc.T2');
    tindex (pnames, 'G4.Exc.T2');
    tindex (pnames, 'G1.Exc.T3');
    tindex (pnames, 'G2.Exc.T3');
    tindex (pnames, 'G3.Exc.T3');
    tindex (pnames, 'G4.Exc.T3');
    tindex (pnames, 'G1.Exc.T4');
    tindex (pnames, 'G2.Exc.T4');
    tindex (pnames, 'G3.Exc.T4');
    tindex (pnames, 'G4.Exc.T4');
];
poff = p; poff(pindex(1:4)) = 0; % no pss
% simulate with nominal PSS tuning
[tnom,xnom,ynom] = sim('tunepsssys',[0 20]);
% get linearization point from simulation
x0 = xnom(size(xnom,1),:);
% set optimization parameter and run
opt = optimset ('Diagnostics','on', ...
'Display','iter',...
'MaxIter',1e5,...
'MaxFunEvals',1e5)%,...
% 'TolFun',1e-3);
Kmin = 0.01; Kmax = 20;
Tmin = 0.000 01; Tmax = 20;
I4 = ones(4,1); I16 = ones(4,1);
pmin = [I4*Kmin; I16*Tmin];
pmax = [I4*Kmax; I16*Tmax];
popt = fmincon('pss_obj',pnom,...
[ ],[ ],[ ],[ ],pmin,pmax,[ ],opt);

```

¹<http://www.dynasim.se/>

²<http://groups.yahoo.com/group/objectstab/>

REFERENCES

- [1] J. H. Chow and K. W. Cheung, "A toolbox for power system dynamics and control engineering education and research," *IEEE Trans. Power Syst.*, vol. 7, pp. 1559–1564, Nov. 1992.
- [2] F. Milano. (2003) Psat Website. [Online]<http://www.power.uwaterloo.ca/~fmlano/>
- [3] M. Tiller, *Introduction to Physical Modeling With Modelica*. Norwell, MA: Kluwer, 2001.
- [4] J. Deuse and M. Stubbe, "Dynamic simulation of voltage collapses," *IEEE Trans. Power Syst.*, vol. 8, pp. 894–904, Aug. 1993.
- [5] P. Kundur, *Power System Stability and Control*. New York: McGraw-Hill, 1994.
- [6] K. J. Åström, H. Elmqvist, and S. E. Mattson, "Evolution of continuous-time modeling and simulation," in *Proc. 12th Europe. Simulation Multiconf.*, Manchester, U.K., June 16–19, 1998.
- [7] T. Hiyama, Y. Fujimoto, and J. Hayashi, "Matlab/Simulink based transient stability simulation of electric power systems," in *Power Eng. Soc. Winter Meet.*. Piscataway, NJ: IEEE, 1999, vol. 1, pp. 249–253.
- [8] H. Elmqvist, D. Brück, and M. Otter, *Dymola, Users Manual*, Sweden: Dynasim AB, 2000.
- [9] Website, MathCore. (2003). <http://www.mathcore.com> [Online]
- [10] J. Machowski, J. W. Bialek, and J. R. Bumby, *Power System Dynamics and Stability*. New York: Wiley, 1997.
- [11] P. W. Sauer and M. A. Pai, "A comparison of discrete vs. continuous dynamic models of tap-changing-under-load transformers," in *Proc. Bulk Power Syst. Voltage Phenomena-III: Voltage Stability, Security and Control*, Davos, Switzerland, 1994.
- [12] D. Karlsson and D. J. Hill, "Modeling and identification of nonlinear dynamic loads in power systems," *IEEE Trans. Power Syst.*, vol. 9, pp. 157–163, Feb. 1994.
- [13] "Long Term Dynamics Phase II," CIGRE Task Force, Tech. Rep., 1995.
- [14] K. Hongesombut, Y. Mitani, and K. Tsuji, "An incorporated use of genetic algorithm and a Modelica library for simultaneous tuning of power system stabilizers," in *Modelica Workshop 2002*, Oberpfaffenhofen, Germany, Mar. 2002.

Mats Larsson (M'91) received the Master's degree in computer science and engineering, Licentiate in industrial automation, and the Ph.D. degree in industrial automation from Lund Institute of Technology, **PLEASE PROVIDE CITY** Sweden, in 1993, 1997, and 2001, respectively.

Currently, he is with Corporate Research, ABB Switzerland Ltd., working on the research and development of wide-area stability controls for power systems. His research interests include power system stability, power system modeling and simulation, optimal control, and artificial intelligence applications in power systems.