Building Power System Models for Stability and Control Design Analysis using Modelica and the OpenIPSL

Srijita Bhattacharjee[1] Luigi Vanfretti[1] Fernando Fachini[2]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, NY, USA
{bhatts10, vanfrl}@rpi.edu
[2]Electric Transmission Strategic Initiatives, Dominion Energy, VA, USA
fernando.fachini@dominionenergy.com

Abstract

Ensuring the stability of complex power system models is a critical challenge in the field of electrical power engineering, and the tuning of Power System Stabilizers (PSS) plays a pivotal role in this endeavor. Modelica, an open-access modeling language, emerges as a powerful tool for this purpose, due to its distinctive features that facilitate efficient power system modeling. This paper explores the capabilities of Modelica using the OpenIPSL library to create models to analyze control system designs developed for a multi-machine power system model. It particularly focuses on using the features of Modelica for the linearization, control-oriented analysis, and time-simulation of the model. The results demonstrate the effectiveness of using Modelica for control system design analysis and performing linear model-based analysis. This work aims to show how Modelica can be used to perform these tasks on a single platform efficiently, thereby streamlining the process of power system design and analysis.

Keywords: Power System Modeling, Linearization, Stability Analysis, Controller Design Analysis, OpenIPSL

1 Introduction

1.1 Motivation

Modern power systems exhibit a complex architecture that requires the use of both physics-based models for sophisticated control system designs. The design of robust control systems is crucial to ensure reliable grid operation, which facilitates the management of complex power system dynamics. A significant aspect of this involves conducting stability analysis and tuning of Power System Stabilizers (PSS), which are essential to damp electromechanical oscillations that can adversely affect system stability (F. J. De Marco, Martins, and Ferraz 2012). To address this need of developing models for control design and analysis, Modelica, in conjunction with the OpenIPSL library, has been used effectively to create a detailed University Campus Microgrid model, demonstrating its effectiveness in linear analysis, which is often challenging with traditional power system analysis tools (Fachini, Bhattacharjee, et al. 2023).

This work explores the capabilities of Modelica (Fritzson and Engelson 1998), and the Dymola tool (Brück et al. 2002), to develop power system models that are suitable for control system design and analysis. The model developed here emerges from the power system literature (F. De Marco, Rullo, and Martins 2021), which can be used to address intra-plant and inter-area oscillations when considering a power plant with multiple machines. The developed can be further explored to create a more detailed design for specific control tasks beyond those for which it was originally developed (F. J. De Marco, Martins, and Ferraz 2012).

1.2 Background and Related Works

The evolution of power system analysis has advanced computing technologies, notably through the development of software tools designed to improve the accuracy and efficiency of modeling and simulation (Isaacs 2017). This transition has been marked by significant shifts from traditional methods to more sophisticated software-oriented approaches that integrate the capabilities of modern computing frameworks (Guironnet et al. 2018). These advances have facilitated a detailed analysis of the dynamics of the power system, setting the stage for the addressing of the complex engineering challenges that arise from the adoption of renewable energy sources (Fachini, Luigi Vanfretti, et al. 2021; Plietzsch et al. 2022). The widespread commodification of computing technologies in the 1900s and 2000s led to the commercialization of domain-specific proprietary software and the rise of open-source software for power system analysis, often exploiting proprietary general-purpose computing languages and environments, that is, mainly tools based on `MATLAB` (Chow and Cheung 1992; Milano and Luigi Vanfretti 2009). This technological evolution set the stage for the addressing of more complex system challenges. One such challenge is linearization of power system models, a task that is complex due to the limitations of domain-specific tools, many of which lack symbolic linearization capabilities.

Many industry standard tools such as Siemens PSS/E depend on additional tools to perform numerical perturbations for linearization (Nikolaev et al. 2020). Likewise, CEPEL in Brazil has developed two independent tools, one for nonlinear time simulation and another for linear analysis (Martins et al. 2000). However, developers of both

tools need to provide symbolic expressions, which presents challenges in maintaining modeling consistency between the internal model descriptions within each tool (Luigi Vanfretti et al. 2013). Researchers have developed certain tools, such as PSAT, that support symbolic linearization (Milano 2005). However, they require users to input symbolic expressions and have a complete understanding of the source code of the software to modify or expand it (Li, Luigi Vanfretti, and Chompoobutrgool 2012). Other software tools such as DOME have been developed that utilize Python for power system analysis, demonstrating the viability and utility of scripting languages in this field, particularly for their modularity, ease of integration with various libraries and suitability for academia (Milano 2013).

In contrast, Modelica offers a compelling alternative, providing robust support for graphical modeling through software such as OpenModelica (al. 2020) and Dymola, thus significantly improving user experience and accessibility. Modelica emerges as a formidable language for power system modeling, especially when integrated with the Open-Instance Power System Library (OpenIPSL) (Baudette et al. 2018; De Castro et al. 2023), as elaborated in (Winkler 2017). Unlike the conventional power system approach of building a monolithic simulation tool, Modelica serves as a language that numerous software programs can implement, including proprietary options such as Dymola, Modelon Impact, Wolfram System Modeler, etc.

Along with these compliant tools, what the Modelica language offers is a unique advantage: it facilitates model linearization (including symbolic-based linearization) without the need of users or developers to specify additional (linear) models, excelling over other alternatives. This work explores the capabilities of Modelica symbolic analysis to *automatically* derive the linear model from *exactly* the same model used for non-linear time-domain simulation.

1.3 Paper Contribution

The main contributions of this paper are:

- To construct a multi-machine power system model by utilizing Modelica and the OpenIPSL library, specially designed to study intra-plant and inter-area oscillations (F. De Marco, Rullo, and Martins 2021).
- To extend the model in applying a control system design derived from the literature (F. J. De Marco, Martins, and Ferraz 2012).
- To demonstrate the benefits of object-oriented modeling for complex power system models.
- To demonstrate the application of the Modelica language and the OpenIPSL library for control system design analysis as a strong alternative to domain-specific power system tools with simulation results.

While the article aims to illustrate how Modelica and OpenIPSL can be used for the purposes stated above, some familiarity with the Modelica language would be beneficial to the reader. When necessary, the paper briefly introduces some language constructs and concepts used to guide the reader.

1.4 Paper Structure

The structure of the paper is as follows: Section 2 demonstrates the application of the object-oriented modeling technique to construct the different components of the system. Section 3 explains the process of creating the model used for linearization. Section 4 describes the nonlinear simulation of the multimachine system. The simulation results for the designed control system are presented in Section 5. Finally, Section 6 concludes the work and outlines the future direction of the work.

2 Object-Oriented System Modeling

Figure 1 shows the package structure of the three-machine infinite bus package `ThreeMIB` with several sub-packages, namely `Generation Units`, `Networks`, `Systems`, `PF_Data`, etc. Due to space constraints, only the `Generation Units` package is expanded in Figure 1 to show its internal structure. The sub-packages `Generation Units`, `Networks`, `Systems` are further explained below to describe the process of system modeling. This package is available in the Github repository: `https://github.com/ALSETLab/AMCONF2024_ThreeMIB`

2.1 Component Modeling

The OpenIPSL contains different component models that are built using object orientation. For components, object-oriented modeling can be illustrated using the instance for the `bus` component. As observed in the Modelica code excerpt in Listing 1, the `Bus` model extends a `partial model` named `pfComponent` from the `*.Electrical.Essentials` package. This inheritance approach is a hallmark of object-oriented modeling in Modelica, allowing the bus model to reutilize and extend predefined functionalities, such as initial parameter setups for algebraic variables that are crucial for setting initial state values in the models it comprises. Central to object-oriented design, attributes like `final enablev_0=true` and `final enableangle_0=true` are strategically enabled for initializing values, while `final enableP_0=false` is disabled to comply with KCL, illustrating the model's ability to customize through selective inheritance. The `PwPin` instance, named `p`, exemplifies encapsulation, initializing its algebraic variables, `vr` and `vi`, from `v_0` and `angle_0`. Furthermore, the variables `v` and `angle`, in Lines 13 and 14, which represent the magnitude and angle of the voltage, are managed within the model to reflect its link with other components of the system. The calculations in for voltage (Lines 13-14) and zero current enforcement (Lines 15-16) through `p.ir` and `p.ii` not only confirm the model's functionality but also ensure its integration within the larger system, underscoring the efficacy and adaptability of object-oriented modeling for complex

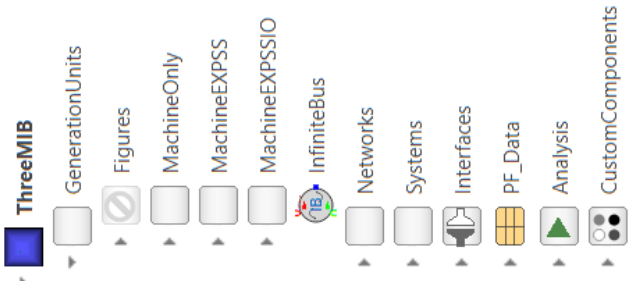


Figure 1. Package Structure

power systems.

Listing 1. Excerpt of the `OpenIPSL.Electrical.Buses.Bus` model

```
model Bus "Bus model"
  extends OpenIPSL.Electrical.Essentials.
      pfComponent(
    ...
    final enableP_0=false,
    ...
    final enablev_0=true,
    final enableangle_0=true);
  OpenIPSL.Interfaces.PwPin p(vr(start=v_0*
      cos(angle_0)), vi(start=v_0*sin(
      angle_0)));
  Types.PerUnit v(start=v_0) "Bus v.
      magnitude";
  Types.Angle angle(start=angle_0) "Bus v.
      angle";
  ...
equation
  v = sqrt(p.vr^2 + p.vi^2);
  angle = atan2(p.vi, p.vr);
  p.ir = 0;
  p.ii = 0;
  ...
end Bus;
```

Similarly to the bus component, other OpenIPSL component models are used to develop the system models described below. More information on the components available in OpenIPSL can be found in (L. Vanfretti et al. 2016; Baudette et al. 2018; De Castro et al. 2023).

2.2 System Modeling

This section explains how the object-oriented features of Modelica (Fritzson 2014) and OpenIPSL components are used to construct the multi-machine power system model from (F. De Marco, Rullo, and Martins 2021), as shown in Figure 4, allowing modular reusable components that simplify the design of the system model and enhance the simulation flexibility. The model consists of three generation units, six buses, two transmission lines, three transformers, three loads, and an infinite bus. Some of the sub-packages are explained below.

- *Generation Units*: The `GenerationUnits` subpackage is expanded in Figure 1 to show the internal structure. The sub-packages within offer various configurations of the three generation units named `G1`, `G2`, and `G3`:

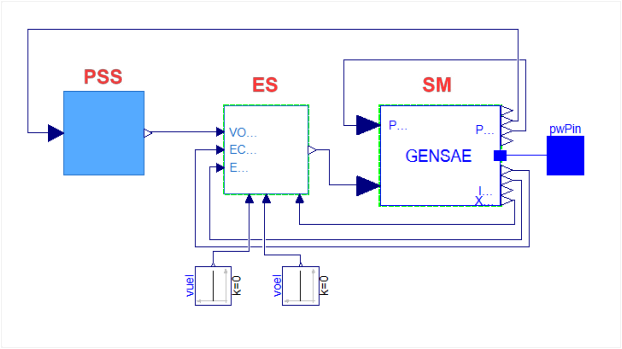


Figure 2. `GenerationUnits.MachineEXPSS.Generator1` Generator1 model diagram view

  - `MachineOnly`: Consists of only the synchronous machine (SM).
  - `MachineEXPSS`: Consists of a synchronous machine equipped with an excitation control system (ES) and power system stabilizer (PSS).
  - `MachineEXPSSIO`: Consists of a synchronous machine equipped with an excitation control system (ES) and power system stabilizer (PSS) along with an input and output (IO) interface.

The three generation units are chosen from `MachineEXPSS` to be used in the multi-machine model shown in Figure 4. Each unit is modeled as a separate component. The structure of each generation unit consists of a synchronous machine (SM), which is the primary component for generating electrical power, an excitation control system (ES) which regulates the field voltage, maintaining the terminal voltage stability, and a power system stabilizer (PSS) which provides damping of the power system oscillations by modulating the ES. The diagram view of one of the generation units `G1` is shown in Figure 2. The graphical placement and connections of the components ensure that the mathematical relationships are correctly established when `connect(...)` statements are generated. In Modelica, a `connect(...;...)` statement links the compatible ports of two components, enabling them to interact within the simulation environment as described in Chapter 9 (Modelica Association 2023).

The SM and ES are parameterized using the values of an implementation made in the Siemens PSS/E software. `*.raw` and `*.dyr` files from (Illinois Center for a Smarter Electric Grid (ICSEG) 2024) are used to set parameter values and to obtain a power flow solution that populates Modelica `records` within the `PF_Data` subpackage in Figure 1. These help provide an initial guess for the algebraic variables that are used to initialize the model (see more details in (Dorado-Rojas et al. 2021). PSS models are specifically parameterized according to optimized transfer functions from studies on PSS tuning for phase compensation(F. J. De Marco, Martins, and Ferraz 2012; F. De Marco, Rullo, and Martins 2021). Furthermore, the parameters of the individual components that are required

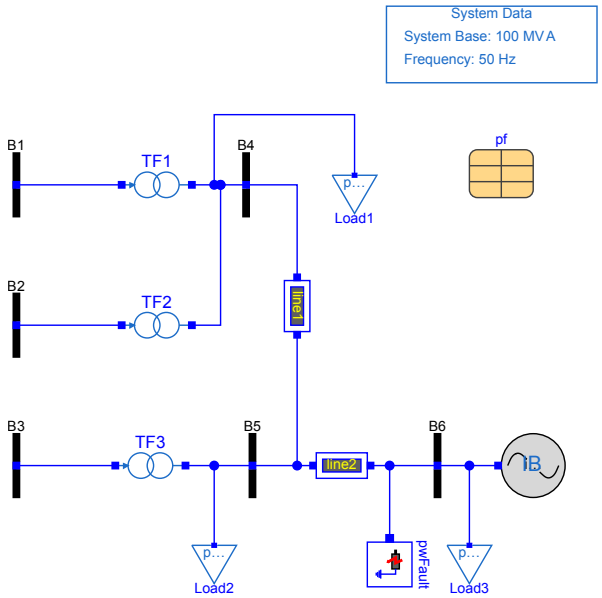


Figure 3. `ThreeMIB.Networks.BasePFnFault` multi-machine Power System Base Model

at higher levels, such as the model shown in Figure 4, are propagated to provide a user-friendly interaction.

- *Networks*: The sub-package named `Networks` is comprised of a `partial model` of the multi-machine system `ThreeMIB.Networks.Base` including the buses `B1`-`B6`, transmission lines `line1` and `line2`, the transformers `TF1`-`TF3`, and the `System Data` component. Instead of programmatically building it, this model is built graphically. The components are dragged and dropped, connected, and parameterized; the Modelica tool automatically generates the corresponding source code as shown in Listing 2.

Listing 2. Connect equations of the `partial model` called `ThreeMIB.Networks.Base`

```
1 partial model Base "Partial model
    containing network elements"
2   ...
3   OpenIPSL.Electrical.Branches.PwLine line1
      (R=0, X=0.036, G=0, B=0);
4   ...
5 equation
6   connect(B1.p, TF1.p);
7   connect(TF1.n, B4.p);
8   connect(B2.p, TF2.p);
9   connect(line1.n, line2.p);
10  ... (more connect equations follow)
11 end Base;
```

As observed in Listing 2, the instantiation of the `PwLine` component named as `line1` and the parameter `X=0.036` is set through a modifier, that is, it is changed from the default values. Similar code is generated for all other component instantiation and parameterized. Observe that there are fewer components in Figure 3 than those shown in Figure 4. This is because the model in 4 is built through inheritance, i.e., it inherits the components in Figure 3 and adds new ones. This method allows for the creation of varied system model variants from partial models, which are extended and customized through modifications. The automatically generated `connect` equations link the `PwPin` within each component instance. For example, as observed from Line 6 of Listing 2, `B1.p` is connected to `TF1.p`, thereby interfacing bus `B1` to transformer `TF1`. Similarly, line 9 shows how `line1` and `line 2` are interfaced through the `connect` equations. For illustration, this is labeled in red in Figure 4. Similar equations are automatically generated by the tool for other connections that were done graphically.

The partial model `ThreeMIB.Networks.Base` is extended by adding other components, namely the power flow component `pf`, the loads `Load1`-`Load3`, and the fault component `pwFault`. This does not include the generation units, which are discussed later. The resulting base model is called `ThreeMIB.Networks.BasePFnFault` shown in Figure 3.

- *Systems*: The sub-package `Systems` comprises the final model of the multi-machine power system, as shown in Figure 4. To create this, the `ThreeMIB.Networks.BasePFnFault` enclosed in the dotted blue box is extended and the three generation units `G1`, `G2`, and `G3` enclosed in the dotted green box are dragged and dropped from the `ThreeMIB.GenerationUnits.MachineEXPSS` package. Once connected to the corresponding buses, the generation units are parameterized with power flow data contained within the `pf` record component. The resulting model is the `ThreeMIB.Systems.Grid`, which can be readily used for typical power system time-domain simulations. This particular package also consists of the models built for linearization and nonlinear simulation which is discussed in detail in the following sections.

3 Deriving Linear Models

3.1 Refactoring Models for Linearization

This section discusses the creation of the linearized model, here referred to as "plant". To generate a model that can be linearized, the base model `ThreeMIB.Networks.Base` is extended and instantiated as `ThreeMIB.Systems.GridIO`, and the power flow component `pf` and fault component `pwFault` are added graphically. It is worth noting here that the load components added to this model `Load1`-`Load3` are chosen as those with an external input. Figure 5 shows the detailed extended model with the inputs in the green boxes and the outputs within the orange one. This is achieved by choosing the generation units from the package `ThreeMIB.GenerationUnits.MachineEXPSSIO` with an IO interface depicting a structure as shown in Figure 6. The inputs, enclosed within the dotted green boundary in Figure 6, are simply

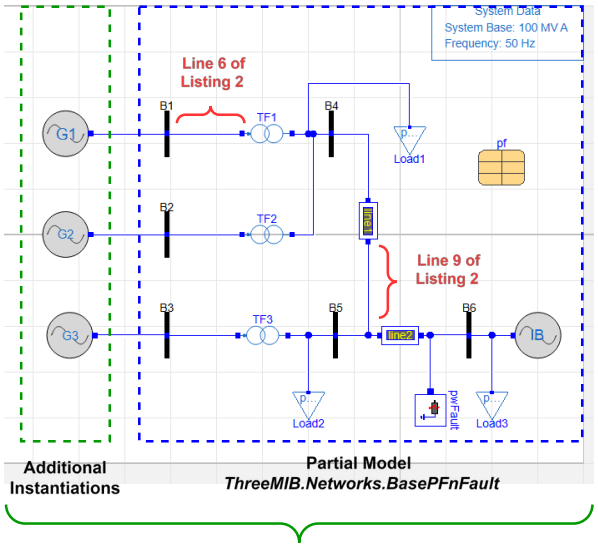


Figure 4. `ThreeMIB.Systems.Grid` multi-machine power system model

connected to a `RealInput` interface from the Modelica Standard Library (MSL). In the case of the outputs, `RealOutput` interfaces from the MSL need to be provided, which are shown enclosed by a dotted orange boundary in Figure 6. These interfaces must be linked to the desired output variables. This is carried out in the textual layer of the model, as shown in Listing 3.

Listing 3. Linking output variables to the `RealOutput` interfaces.

```
1  model GridIO
2    "Multimachine power grid model with input
         /output interfaces ..."
3    extends ThreeMIB.Interfaces.
         OutputsInterface;
4    extends ThreeMIB.Networks.Base(...
5    GenerationUnits.MachineEXPSSIO.
         Generator1EXPSSIO G1(...)
6    ...  // More instantiations follow
7  equation
8    SCRXin = G1.feedbackSCRX.y;
9    SCRXout = G1.sCRX.EFD;
10   Vt = G1.gENSAE.ETERM;
11   ANGLE = G1.gENSAE.ANGLE;
12   SPEED = G1.gENSAE.SPEED;
13   ... // More connect statements follow
14 end
```

Each of the `RealOutput` interfaces must be linked to the output of different components. For example, on Line 10 of Listing 3, the generator's terminal voltage `G1.gENSAE.ETERM` is linked to the interface `Vt`. This is done similarly for other machine variables. Meanwhile, to access the output of the PSS (which is the input of the ES), the `RealOutput` interface `SCRXin` is linked to `G1.feedbackSCRX.y` as seen in Line 8 and similarly the output of the ES, `SCRXout`, is linked to the field voltage `G1.sCRX.EFD` in Line 9. The plant model shown in Figure 5 can now be utilized as a block with the specified

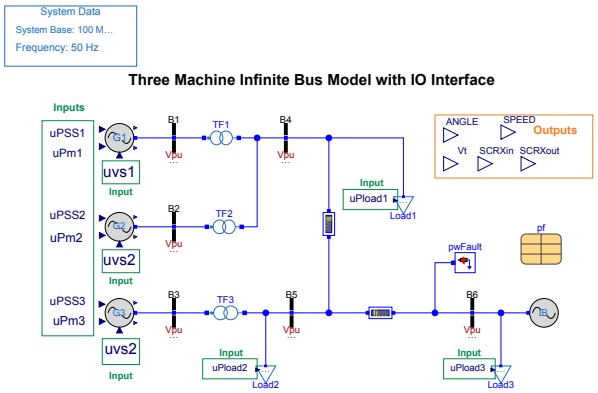


Figure 5. `ThreeMIB.Systems.GridIO` multi-machine power system model with IO interface

inputs and outputs for the analysis of the design of the control system. Figure 7 illustrates this concept where the inputs to the `ThreeMIB.Systems.GridIO` can be set to zero with only one desired functional input and output. The entire model enclosed in the red dotted lines is treated as a single-input-single-output (SISO) block. This modularity improves the adaptability and utility of the model in diverse linearization and simulation needs.

3.2 Linearization Process

Each Modelica-compliant tool, such as Dymola or OpenModelica, supports symbolic analysis to automatically generate a linear model from the same model used for non-linear time-domain simulation. Within Dymola, the `Modelica_LinearSystems2` (`MLin2`) library can be used to perform this task, which allows easy conversion of models to representations of linear time-invariant systems (Baur, Otter, and Thiele 2009). Listing 4 shows the command needed to linearize the model shown in Figure 5. A state space object and `*.mat` file are generated as the resulting output `ss` which is suitable for further analysis in Dymola or external tools, supporting tasks such as eigenvalue computation, frequency response analysis, and advanced control design such as pole placement and LQG controller design.

Listing 4. Linearization using Modelica_LinearSystems2

```
1 ss := Modelica_LinearSystems2.ModelAnalysis
      .Linearize("ThreeMIB.Systems.GridIO");
```

Once linearized, the system, input, and output matrices can be observed from Dymola's command window.

4 Nonlinear Simulation

This section explores the initialization process and the selection of solvers in time-domain simulations, demonstrating how these features can accommodate various use cases with models developed using OpenIPSL.

4.1 Initialization

Providing suitable initial guess values for large-system models under various operating conditions can be challenging. To address this, a Modelica `record` template within

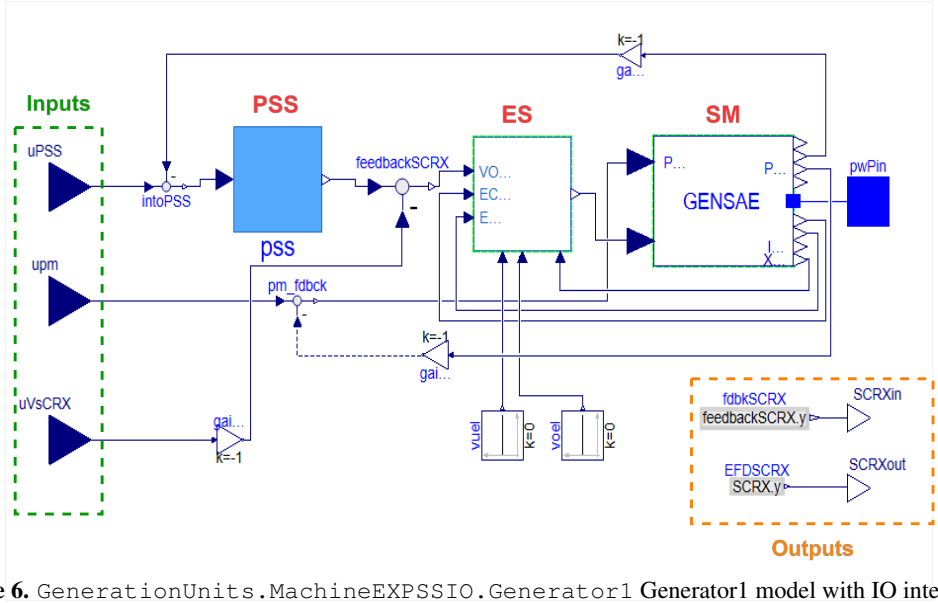


Figure 6. `GenerationUnits.MachineEXPSSIO.Generator1` Generator1 model with IO interfaces.

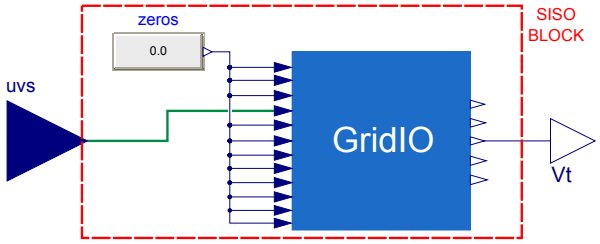


Figure 7. `ThreeMIB.Systems.GridIOsiso` multi-machine power system with IO interface used as a SISO Block

`ThreeMIB.PF_Data.powerflow` is associated with each component of the model to facilitate the entry of data from the power flow solution as starting values. This mapping is done once when creating the model. As shown in Figure 5, the component `pf` is added directly as the block of the yellow record template in the diagram. This allows for the selection of specific data values for buses, machines, loads, and transformers. This record structure can be automatically implemented using the `pf2rec` Python utility, which transforms the power flow simulation results into Modelica records (Dorado-Rojas et al. 2021).

Similarly, as mentioned in Section 2 the `OpenIPSL.Electrical.Essentials.pfComponent` can be provided with data that are used to calculate the starting values within each of the components that extend from the `pfComponent`. For example, it can be observed in Listing 1, how the start values for the real and imaginary parts of the voltage phasor, `vr` and `vi` are calculated from data of voltage magnitude and angle, `v_0` and `angle_0` (see Line 8).

4.2 Solvers

Domain-specific power system tools like Siemens PSS/E usually provide a single solver for which the models' equations have been discretized; a popular choice is to use the trapezoidal integration method combined with a Netwon-Raphson solver to solve the DAEs. This approach typically restricts simulations to a few seconds with a fixed time step. Modelica tools do not face this limitation when simulating the models from the OpenIPSL library. As noted in (Henningsson, Olsson, and Luigi Vanfretti 2019), Dymola has advanced solvers for sparse large-scale DAE models, enhancing the competitiveness of power system simulations with Modelica compared to Siemens PSS/E. To utilize these advanced features in Dymola, the utility `ThreeMIB.Utilities.SetupSolverSettings` offers a series of functions to enable or disable them. For example, it allows settings like `Advanced.Define.DAEsolver := true/false` and `Advanced.SparseActivate := true/false`, which activates the DAE solvers and optimizes for sparsity, respectively. Note that for linearization tasks, these advanced settings should be deactivated to ensure the generation of accurate state-space models.

5 Results

The power system models developed in this work are utilized to analyze the control system design developed in the study for PSS tuning using phase compensation (F. J. De Marco, Martins, and Ferraz 2012). Modelica tools provide means to visualize and analyze the results. Custom functions can be used with the necessary path to the models to perform the required analysis. Within Dymola, the `Modelica_LinearSystems2` (`MLin2`) library provides commands to directly linearize and plot the frequency response from the single-input-single-output version of the model in Figure 7 as observed in Listing 5.

Listing 5. Custom Function for Bode Plot using Modelica_LinearSystems2

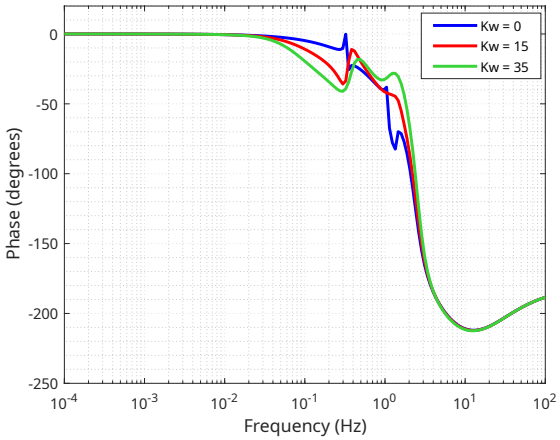
```
1 function bodeplot_GridIOsiso
2   extends Modelica.Icons.Function;
```

```
  input Modelica.Units.SI.Time tlin = 30;
algorithm
  ...
  // linearize and plot
  Modelica_LinearSystems2.ModelAnalysis.
      TransferFunctions(
  "OpenIPSL.ThreeMIB.Systems.GridIOsiso",
      simulationSetup=
  Modelica_LinearSystems2.Records.
      SimulationOptionsForLinearization(
  linearizeAtInitial=false,
  t_linearize=tlin));
end bodeplot_GridIOsiso;
```

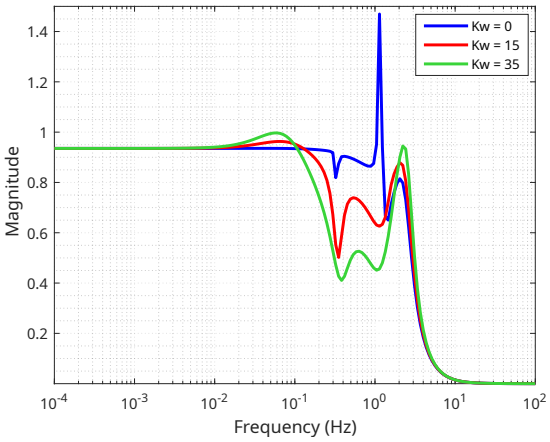
Figure 8 illustrates the frequency response of the system, showing both the magnitude and phase of the terminal voltage as functions of frequency for increasing values of the PSS gain (K_w). These values were obtained from the designs in (F. J. De Marco, Martins, and Ferraz 2012). The adjustment K_w adjusts the phase change introduced by the system as shown in Figure 8a. When the PSS is disabled by setting $K_w = 0$, the phase curve introduces a negative phase shift in the frequency response. Increasing K_w to 15 and 35 shows an improvement in phase around the resonant frequency, reducing the phase lag, which is crucial to effectively damp system oscillations. The magnitude plot as shown in Figure 8b reveals the system's sensitivity to frequency changes for different PSS gains. With increasing K_w, there are noticeable peaks in the magnitude response, particularly around the resonant frequencies, suggesting an enhanced ability of the PSS to counteract perturbations effectively. However, higher gains ($K_w = 35$) introduce sharp peaks that could lead to potential system instability under certain conditions.

The PSS is tuned by receiving a feedback signal from the rotor speed of the synchronous machine. Figure 9 demonstrates the time-domain simulation of the rotor speed of Generator 1 after a load disturbance at t = 30.5 seconds, clearly demonstrating the impact of varying PSS gain values on the stability of the system. With the PSS disabled ($K_w = 0$), the rotor speed experiences substantial oscillations, indicating poor damping characteristics. With an increase of K_w to 15 and 35 there is an improvement in damping performance, with the rotor speed quickly stabilizing and exhibiting minimal oscillatory behavior. This analysis underscores the effectiveness of PSS in enhancing the system's dynamic response to disturbances, highlighting the critical role of appropriate PSS tuning in maintaining system stability.

To further investigate this power system dynamics, Figure 10 provides further insight into the stability of the system by illustrating the pole positions of the `GridIOsiso` model under varying PSS gains. With the PSS disabled ($K_w = 0$), the poles marked with pink crosses highlight a critically damped system with potential for sustained oscillations. Increasing K_w to 15 (red) and 35 (dark red) shows a shift in the poles, which move toward the left in the complex plane. This indicates improved damping and stability, thus emphasizing the significant influence of PSS



(a) Phase for different values of K_w



(b) Magnitude for different values of K_w

Figure 8. Bode Plot of the GridIOsiso model for three values of the PSS gain (K_w)

tuning on the system dynamics. Moreover, this illustrates the value of complementing non-linear simulations with linear methods when assessing control system designs.

6 Conclusions

In this work, Modelica and the OpenIPSL library have been utilized to build a multi-machine power system model developed for the analysis of intra-plant and inter-area modes. The model is refactored and extended to implement a control system design and analyze its performance. This is done by exploiting the object-oriented modeling features of Modelica. Linearization capabilities provide an advantage over other domain-specific tools in implementing this design and performing an analysis of the model. Given the complexity of power systems and the critical role of stability and dynamic behavior as illustrated in Figure 10, careful control design analysis is essential to ensure the robustness of the system to dynamic conditions and disturbances.

This study demonstrates how Modelica and OpenIPSL

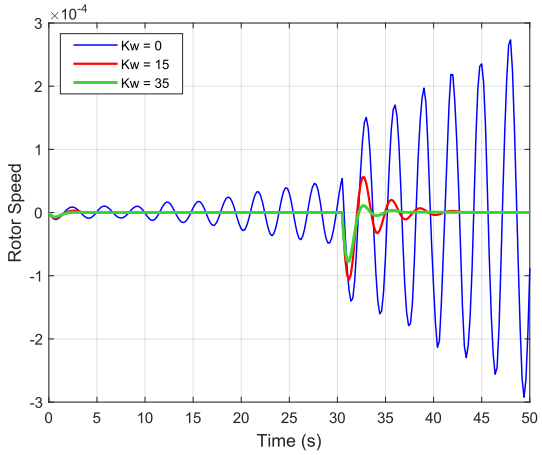


Figure 9. Time domain simulation of the rotor speed of G1 under a load disturbance at t= 30.5 sec. for different values of PSS gain (K_w)

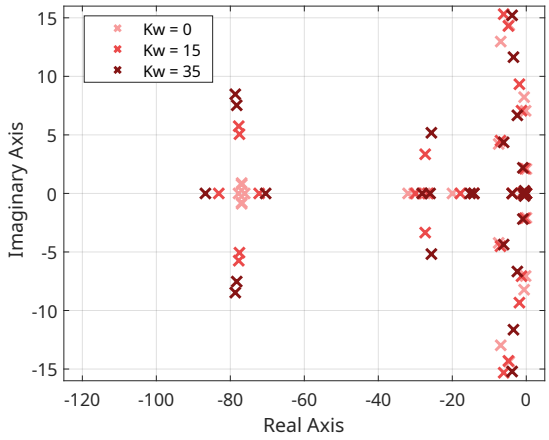


Figure 10. Poles of the GridIO model

can be used to assist in control design analysis and improve power system dynamic performance by testing control system designs. The modifications (model re-factoring and extension) aim to effectively address the complexities of power system stability studies effectively. Additional work includes the development of detailed examples of the actual design of the PSS using the unique features of Modelica and the integration of the models presented into the OpenIPSL library.

To access the models in this paper before they are integrated into OpenIPSL, the reader can find them in the following GitHub repository: `https://github.com/ALSETLab/AMCONF2024_ThreeMIB`

Acknowledgements

This paper is in part, based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

References

al., Peter Fritzson et (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Baudette, Maxime et al. (2018). "OpenIPSL: Open-instance power system library—update 1.5 to "iTesla power systems library (iPSL): A modelica library for phasor time-domain simulations"". In: *SoftwareX* 7, pp. 34–36.

Baur, Marcus, Martin Otter, and Bernhard Thiele (2009). "Modelica libraries for linear control systems". In: *Proceedings 7th Modelica Conference*. DOI: 1, pp. 593–602.

Brück, Dag et al. (2002). "Dymola for multi-engineering modeling and simulation". In: *Proceedings of modelica*. Vol. 2002. Citeseer.

Chow, J.H. and K.W. Cheung (1992). "A toolbox for power system dynamics and control engineering education and research". In: *IEEE Transactions on Power Systems* 7.4, pp. 1559–1564. DOI: 10.1109/59.207380.

De Castro, Marcelo et al. (2023). "Version [OpenIPSL 2.0.0]-[iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]". In: *SoftwareX* 21, p. 101277.

De Marco, Fernando, Pablo Rullo, and Nelson Martins (2021). "Synthetic power system models for PSS tuning and performance assessment". In: *2021 IEEE Electrical Power and Energy Conference (EPEC)*. IEEE, pp. 107–112.

De Marco, Fernando Javier, Nelson Martins, and Julio Cesar Rezende Ferraz (2012). "An automatic method for power system stabilizers phase compensation design". In: *IEEE Transactions on power systems* 28.2, pp. 997–1007.

Dorado-Rojas, Sergio A et al. (2021). "Power flow record structures to initialize openipsl phasor time-domain simulations with python". In: *Modelica Conferences*, pp. 147–154.

Fachini, Fernando, Srijita Bhattacharjee, et al. (2023). "Exploiting Modelica and the OpenIPSL for University Campus Microgrid Model Development". In: *Modelica Conferences*, pp. 285–292.

Fachini, Fernando, Luigi Vanfretti, et al. (2021). "Modeling and validation of renewable energy sources in the openipsl modelica library". In: *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 1–6.

Fritzson, Peter (2014). *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons.

Fritzson, Peter and Vadim Engelson (1998). "Modelica—A unified object-oriented language for system modeling and simulation". In: *ECOOP'98—Object-Oriented Programming: 12th European Conference Brussels, Belgium, July 20–24, 1998 Proceedings 12*. Springer, pp. 67–90.

Guironnet, Adrien et al. (2018). "Towards an Open-Source Solution using Modelica for Time-Domain Simulation of Power Systems". In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pp. 1–6. DOI: 10.1109/ISGTEurope.2018.8571872.

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019). "DAE Solvers for Large-Scale Hybrid Models." In: *Modelica*, pp. 157–050.

Illinois Center for a Smarter Electric Grid (ICSEG) (2024). *Three Machines Infinite Bus Benchmark System*. Available online: https://icseg.iti.illinois.edu/three-machines-infinite-bus-benchmark-system/.

Isaacs, Andrew (2017). "Simulation Technology: The Evolution of the Power System Network [History]". In: *IEEE Power and Energy Magazine* 15.4, pp. 88–102. DOI: 10.1109/MPE.2017.2690527.

Li, Wei, Luigi Vanfretti, and Yuwa Chompoobutrgool (2012). "Development and implementation of hydro turbine and governor models in a free and open source software package". In: *Simulation Modelling Practice and Theory* 24, pp. 84–102.

Martins, Nelson et al. (2000). "A small-signal stability program incorporating advanced graphical user interface". In: *Proceedings of the VII SEPOPE*.

Milano, Federico (2005). "An open source power system analysis toolbox". In: *IEEE Transactions on Power systems* 20.3, pp. 1199–1206.

Milano, Federico (2013). "A Python-based software tool for power system analysis". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Milano, Federico and Luigi Vanfretti (2009). "State of the art and future of OSS for power systems". In: *2009 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–7.

Modelica Association (2023). *Modelica Language Specification v3.6.0*. Available online: https://specification.modelica.org/maint/3.6/MLS.html. Accessed: 14 Aug 2024.

Nikolaev, Nikolay et al. (2020). "PSS/E Based Power System Stabilizer Tuning Tool". In: *2020 21st International Symposium on Electrical Apparatus & Technologies (SIELA)*. IEEE, pp. 1–6.

Plietzsch, Anton et al. (2022). "PowerDynamics.jl–An experimentally validated open-source package for the dynamical analysis of power grids". In: *SoftwareX* 17, p. 100861.

Vanfretti, L. et al. (2016). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2016.05.001. URL: https://www.sciencedirect.com/science/article/pii/S2352711016300097.

Vanfretti, Luigi et al. (2013). "Unambiguous power system dynamic modeling and simulation using modelica tools". In: *2013 IEEE Power & Energy Society General Meeting*. IEEE, pp. 1–5.

Winkler, Dietmar (2017). "Electrical power system modelling in modelica–comparing open-source library options". In.