Power Flow Record Structures to Initialize OpenIPSL Phasor Time-Domain Simulations with Python

Sergio A. Dorado-Rojas[1] Giuseppe Laera[1] Marcelo de Castro Fernandes[1] Tetiana Bogodorova[1] Luigi Vanfretti[1]

[1]Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, United States,
{dorads, laerag, decasm3, bogodt2, vanfrl}@rpi.edu

Abstract

This paper presents a tool to populate power flow results for phasor time-domain simulations with the Open Instance Power System Library (OpenIPSL). Our proposal takes advantage of the object-oriented philosophy of Modelica and introduces a data structure based on records to handle power flow data for a given network model. Such records constitute a user-friendly interface to change the guess values used to solve the initial condition of a dynamical simulation straightforwardly. Power flow calculations are carried out by the open-source Python library GridCal. We demonstrate the tool capabilities by generating power flow results for several grid models and comparing them with those obtained via proprietary tools such as PSS/E. Moreover, we provide tutorial materials to ease integrating the tool for a new/experienced OpenIPSL user.

Keywords: GridCal, OpenIPSL, Power Flow, Python, Records

1 Introduction

The Open Instance Power System Library (OpenIPSL) is an open-source library of power system component models written entirely in Modelica (Baudette et al. 2018). Beyond the inherent advantages of the Modelica language, OpenIPSL components are constantly cross-validated against commercial packages such as PSS/E, producing practically the same results (Laera 2016) and exhibiting the same or even better simulation performance (see Henningsson, Olsson, and Vanfretti (2019) and Dorado-Rojas, Navarro Catalán, et al. (2020)).

Successful use cases of the library come from a broad range of applications such as multi-domain simulation (Gomez et al. 2018), damping (Boersma et al. 2020) and parameter estimation (Podlaski et al. 2020) in power systems, dynamic stability assessment (Nohac et al. 2019), co-simulation for energy analysis (Gusain, Cvetkovic, and Palensky 2019), stability analysis of hydro-power grids (Winkler 2019), wind turbine control (Qin et al. 2019), cyber-attack evaluation (Pan, Gusain, and Palensky 2019), power system stability enhancement (Gonzalez-Torres et al. 2019), extremum seeking control (Müller et al. 2020), and data generation for machine learning applications (Dorado-Rojas, de Castro Fernandes, and Vanfretti 2020).

Motivation

Despite the library's usefulness, the main caveat is the absence of a systematic approach to link phasor time-domain simulations with static computations like power flows. Power flow computations are ubiquitous in any power system analysis. A power flow problem involves determining the system's voltage profiles and electrical power transfer across a network given the generator power injections and load consumption. Mathematically, it is a nonlinear vector algebraic equation commonly solved using an iterative method such as a Newton-Raphson algorithm. From the dynamical perspective, a power flow result represents an operating condition for which may contain a *potential* equilibrium for the underlying dynamical system. So, the power flow result represents the set of initial guesses to initialize a dynamic model and analyze an electrical grid's behavior subjected to a dynamical event. Observe that because the simplified algebraic representation of the power grid in the power flow problem, many of its solutions can result in operating conditions where an equilibrium may not exist when the system's dynamical model is considered.

OpenIPSL models contain a myriad of nonlinearities employed to represent dynamical behaviors more accurately. So, varying the initial condition of a dynamical simulation represents a critical step towards system assessment. So far, users have proposed ad-hoc solutions to generate power flow results (e.g., using Matpower[1] or PSS/E[2] as in Vanfretti et al. (2017)). However, despite valuable, these efforts do not completely fill the gap to easily provide power flow solutions to OpenIPSL models. The former approach replaces the power flow values in the `*.mo` file of the model directly, which is inconvenient from the user's point of view. The latter depends on proprietary software which might not be available to the base users of OpenIPSL. The OpenIPSL community, and users of other Modelica-based power system libraries (see Winkler (2017)), will more than welcome a systematic power flow approach based on open-source tools to integrate into their models quickly. Addressing this issue is the primary purpose of this paper.

[1]See https://github.com/dgusain1/InitialiseModelica
[2]See https://github.com/ALSETLab/Raw2Record

Contribution

This paper's main contribution is to bridge the gap between phasor time-domain simulation and static computations for OpenIPSL utilizing an open-source-based pipeline.

We propose a Modelica records structure to handle all power flow variables. Such nested records data structure enables a user to replace a power flow condition, typically composed of several algebraic variables, with a single click or one line of code. These records are created automatically from the model's `*.mo` file.

GridCal[3], and provides open-source Python library for power system computations, such as solving the power flow problem. A remarkable characteristic of GridCal is its built-in PSS/E parser. Consequently, users can parse `*.raw` files containing a grid static model's information to a GridCal internal grid representation. In our examples, we will use PSS/E files to construct GridCal models. This enables us to benchmark the GridCal power flow results against PSS/E outputs. By doing so, we bring confidence to Modelica tools in terms of the quality of results, showing that Modelica-based power system models can be initialized and result in the same initial condition and provide the same simulation results as proprietary tools.

So far, we summarize the main contributions of our work as follows:

- we bridge the gap between Modelica-based tools and conventional domain-specific power system tools;
- we automate the process of providing good initial guess values to solve the initialization process of Modelica-based power system models with the OpenIPSL library;
- we make Modelica-based tools more attractive for dynamic power system simulation, making it easier for users to use OpenIPSL models for analysis under multiple operating conditions by a power flow record structure;
- with the contributions above, we facilitate the potential adoption of and transition to Modelica-based tools by power system domain specialists.

Paper Structure

This paper is structured as follows: Section 2 gives a brief introduction to the power flow problem in electrical networks. In Section 3, we introduce the records structure proposed to handle power flow variables. We illustrate how this data container can be linked to OpenIPSL models in Section 4, where we also benchmark the power flow values against the results obtained with commercial tools. Finally, Section 5 concludes the work.

[3]GridCal is able to import and parse model description and parameter files from proprietary software such as PSS/E and DigSilent, and also widespread open-source libraries for power system analysis like Matpower. In contrast to many proprietary electrical grid software tools, GridCal runs on Windows, Linux, and macOS natively. It can be downloaded from https://github.com/SanPen/GridCal

2 The Power Flow Problem

The *power flow* (also incorrectly called *load flow*) problem is undoubtedly one of the most performed calculations in power system applications (Stott 1974). For instance, these calculations are carried out many times in Operation and Planning procedures for power grids. An application of a particular interest to this paper is that power flow solutions provide a *potential* starting guess to solve the initialization problem at which a dynamic simulation may start. Hence, a power flow can be considered one of the most critical problems to be solved when studying a power system (Milano 2009).

The problem, however, is not new, and nor are the techniques used to solve it. The first practical solutions began to appear in the mid-1950s with the aid of digital computers (Ward and Hale 1956) and a breakthrough came about a decade later. The development of incredibly efficient handling of sparse matrices (Tinney and Walker 1967) was paramount to the wide adoption of Newton-Raphson (NR) algorithm. As new issues to solve the power flow problem have arisen, a myriad of new techniques and methods have been proposed; however, NR-based techniques are still the most preeminent methods (Stott 1974; Milano 2009).

From the mathematical perspective, a power flow problem is posed as a set of nonlinear algebraic equations. Its solution will determine an operational point for a specified loading and generation condition in the power system. This operational point is defined by the voltage magnitude and angle in each bus of the system, together with the active and reactive powers generated and consumed in generation and load buses respectively. In the current paper, we will give a brief introduction to its formulation.

In most power flow formulations, the power system is assued to be perfectly balanced and operating at constant frequency (i.e. 50/60 Hz) which would allow it to be represented in its positive sequence equivalent circuit (Stott 1974). If a system can be represented in its positive sequence equivalent, it is then possible to assemble its nodal admittance matrix $\mathbf{Y}$ and to write the *nodal equation* as follows:

$$\bar{\mathbf{I}} = \mathbf{Y}\bar{\mathbf{V}}, \tag{1}$$

where $\bar{\mathbf{I}}$ is the nodal injection current phasor vector, $\bar{\mathbf{V}}$ is the nodal voltage phasor vector and $\mathbf{Y}$ is the admittance matrix, which is square and sparse.

We could use the nodal equation to compute the voltage at all nodes if all current injection measurements were available. Unfortunately, this is not the case in an electric grid where the known quantities differ from bus to bus. For instance, in a load node, active and reactive power consumption (P,Q) are assumed to be known. Likewise, in a generation bus, active power injection and voltage magnitude at the generator terminals are typically known (P,V). Then, the nodal equation has to be reformulated in terms of P,Q, and V. Because the steady-state relationship between power and voltage/current is nonlinear (and complex-valued), the linear nodal equation into a nonlinear set of complex-valued equations on P, Q, V, and the voltage phasor angle θ.

The exact formulation in the power system jargon is the following. For the mth bus, four variables are either specified or should be calculated in a power flow: active power injected in the bus in per unit P_m, reactive power injected in the bus in per unit Q_m, node voltage phasor magnitude in per unit V_m and node voltage phasor angle in radians θ_m. In load buses (identified as PQ buses) the variable that is known beforehand is the specified apparent power ($S_m^{\mathrm{sp}} = P_m^{\mathrm{sp}} + jQ_m^{\mathrm{sp}}$), while in generation buses (called PV buses) the specified variables are the voltage magnitude ($\bar{V}_m$) and active power (P_m^{sp}). In addition to that, there should be one *slack bus* which should have a specified voltage magnitude value and, most importantly, it should be responsible for providing the angle reference used for all other calculations (Stott 1974).

The power flow solution is achieved when the computation of active and reactive power via the nodal equation, using the solution values from the most recent iteration, matches the given data for active and reactive power. In other words, the solution occurs when the mismatch between specified and calculated power values is less than or equal to some given tolerance. We can write such a mismatch as

$$\Delta S_m = S_m^{\mathrm{sp}} - V_m I_m^* = P_m^{\mathrm{sp}} + jQ_m^{\mathrm{sp}} - \bar{V}_m \sum_{k \in \mathbf{K}_m} Y_{mk}^* \bar{V}_k^*, \quad (2)$$

where $\mathbf{K}_m$ is the set of buses k which are directly connected to bus m and the superscript $(^*)$ denotes the complex conjugate. By using the fact that $Y_{mk} = G_{mk} + jB_{mk}$ and expressing the phasor $\bar{V}_m$ as $\bar{V}_m = V_m(\cos\theta_m + j\sin\theta_m)$, we can split Equation (2) into its real and imaginary parts as:

$$\begin{cases} \Delta P_m = P_m^{\mathrm{sp}} - V_m \sum_{k \in \mathbf{K}_m} \left(G_{mk}\cos\theta_{mk} + B_{mk}\sin\theta_{mk}\right) V_k, \\ \Delta Q_m = Q_m^{\mathrm{sp}} - V_m \sum_{k \in \mathbf{K}_m} \left(G_{mk}\sin\theta_{mk} - B_{mk}\cos\theta_{mk}\right) V_k, \end{cases}$$

where $\theta_{mk} = \theta_m - \theta_k$. Note that, in this polar formulation, the unknown variables are the nodal voltage phasor magnitudes (V_m) and angles (θ_m).

As said previously, a power flow problem is typically solved using an NR algorithm. First, a nonlinear vector function $\mathbf{f} : \mathbb{R}^{2n} \mapsto \mathbb{R}^{2n}$ is defined, where n is the total number of nodes (buses). This function could be expressed as:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \Delta \mathbf{P} \\ \Delta \mathbf{Q} \end{bmatrix}, \quad (3)$$

where the $\Delta\mathbf{P}$ and $\Delta\mathbf{Q}$ are the n-row vectors $[\Delta P_m]$ and $[\Delta Q_m]$, respectively. In addition, $\mathbf{x} \in \mathbb{R}^{2n}$ is given by

$$\mathbf{x} = \begin{bmatrix} V_1 & \cdots & V_m & \cdots & V_n & \theta_1 & \cdots & \theta_m & \cdots & \theta_n \end{bmatrix}^T, \quad (4)$$

where the superscript ^{T} stands for transpose. To find the power flow solution, we state the following equation (Milano 2009).

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \quad (5)$$

Due to the nonlinear nature of $\mathbf{f}$, it is impossible to find a closed-form solution for such a problem. Therefore, it is necessary to use an iterative method such as a classical NR algorithm. The i-th iteration of the NR method is written as (Milano 2009)

$$\begin{cases} \Delta\mathbf{x}_i = [\mathbf{J}(\mathbf{x}_i)]^{-1}\mathbf{f}(\mathbf{x}_i), \\ \mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i, \end{cases} \quad (6)$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix of function $\mathbf{f}$. The iterative method will stop when $\mathbf{f}(\mathbf{x})$ is sufficiently close to $\mathbf{0}$ or, in other words, when its norm is less than a tolerance set by the user. Besides, there are many different ways to find $\mathbf{x}_0$, which is used to start the process described in (6). Generally, robust techniques usually allow to find a solution when using a flat start, i.e., when all voltage magnitudes are started as 1 per unit and all angles are started as 0 radians, or one could use a previous power flow solution can be used for the same system.

3 Power Flow Records Structure

One of the Modelica language's main advantages is the object-oriented paradigm that enables the user to create dynamic system models hierarchically. Such a structure allows the user to manage model parameters systematically.

A Modelica `record` is a data container used to store a wide range of information about a model, such as parameter values, simulation settings, or values of specific variables for several analysis conditions. Records permit changing a significant number of variables of a given model by modifying just one parameter that related hierarchically to many variables inside the data representation.

Records are a perfect structure for handling power flow values in a dynamic simulation. Suppose power flow results are handled using a record-based data structure. In that case, we could modify the power flow condition of an OpenIPSL model by varying only one attribute of the model, namely, the power flow record, rather than individually changing multiple variables.

The proposed power flow record structure is presented in Figure 1. A Python script called `create_pf_records` creates the Modelica files containing the `record` structure and places them inside the model's root folder in a directory called `PF_Data`. This function reads the `.mo` file of the model (`<model_name>.mo`) as a plain text file and uses several regular expressions to determine the number of buses, generators, loads, and transformers in the network. Such a script becomes handy when a user has an existing OpenIPSL model and would like to add a power flow records structure automatically.

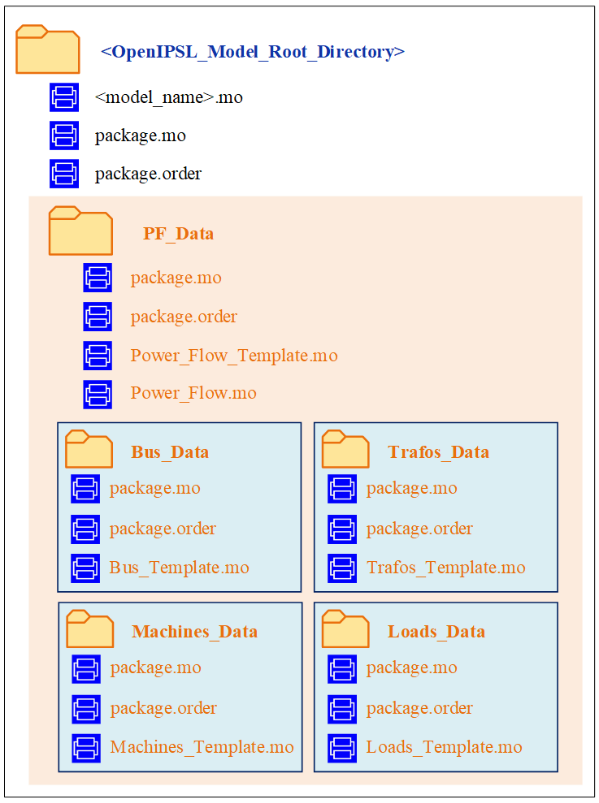


Figure 1. File structure of the power flow records inside the OpenIPSL model directory.

We define our power flow record structure in the class **Power_Flow** shown in Figure 2. **Power_Flow** is a record having a single attribute: a **replaceable record PowerFlow**. A replaceable condition allows the user having many power flow results for the same model (see Figure 4).

The main idea behind the proposed nested structure is that, by setting the value of **PowerFlow**, the user changes all the power flow variables at once. So, a model has a unique **Power_Flow** record whose power flow attribute is replaceable.

PowerFlow has four attributes, which are also records themselves: a record for bus voltages and angles (**Bus_Data**), another for transformer tap positions (**Trafos_Data**), a third one for active and reactive power consumption (**Load_Data**), and a fourth record for machine power dispatch (**Machines_Data**). Naturally, the number of variables inside each of these internal records depends on each particular power system model. For each record type, the variables are specified by the **partial record** templates called **Bus_Template**, **Trafos_Template**, **Machines_Template**, and **Loads_Template**, respectively.

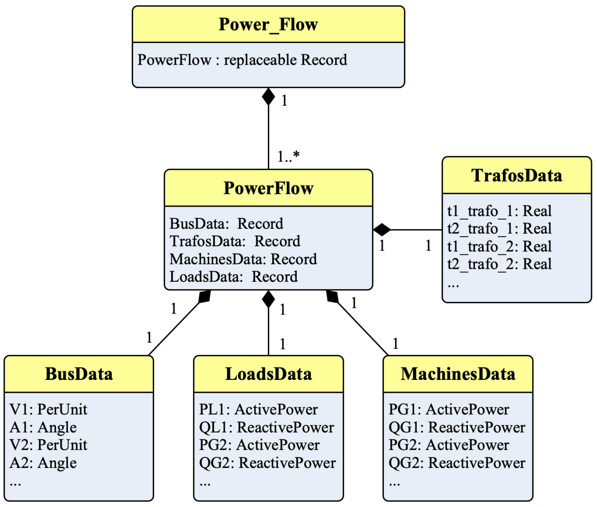


Figure 2. Class diagram for the proposed power flow record structure.

The numerical results are written by a parser function that translates the power flow result from a GridCal model computation into a format compatible with the Modelica record structure. This function is called **gridcal2rec**. **gridcal2rec** creates a **PowerFlow** instance placed inside `PF_Data`, whose attributes are four record instances: **Bus_Data**, **Trafos_Data**, **Machines_Data**, and **Loads_Data**.

4 Computing and Linking PF Records

This section describes how the records structure, illustrated previously, can be successfully applied to grid models of different sizes.

4.1 Creation of Records Structure

A user can integrate our proposed power flow structure into any existing OpenIPSL model using the code contained within the `pf2rec` library (available on GitHub). The records structure is instantiated by the `create_pf_records` function. Listing 1 presents a minimal example of creating a power flow record for the Single Machine Infinite Bus (SMIB) system.

Listing 1. Creation of the records structure

```
from pf2rec import *
import os

# Current working directory
_wd = os.getcwd()
_model_package = 'SMIB'

# Path to the model package directory
data_path = os.path.join(_wd, _model_package)
data_path = os.path.abspath(data_path)

path_mo = os.path.join(data_path,
    'SMIB_Base_case.mo')
path_mo = os.path.abspath(path_mo)

# Creating records structure
create_pf_records(_model_package, path_mo,
    data_path,
    openipsl_version = '2.0.0')
```

Note that the content of the Modelica model is saved within several `*.mo` files. This practice is encouraged since it allows to increase the complexity of the data layer of the model. As supplementary material to this paper, we provide a tutorial[4] showing the step-by-step construction of the SMIB system, the corresponding power flow records integration, and computation using GridCal.

The four arguments that we pass to **create_pf_records** are the name of the containing package (`_model_package`), the path to the `*.mo` file where the model is declared (`path_mo`), the path to the containing folder of the model package (`data_path`), and the OpenIPSL library version on which the model has been developed (`_version`). Here, the paths are constructed as absolute references thanks to the `os` library. Such a workaround is recommended to avoid any path problems since the records instantiation involves file/folder creation. The script places all the power flow record `*.mo` files inside a new directory called `PF_Data`. `PF_Data` is also added to the `package.order` file of the root package. In this way, the **record** structure is loaded with the model automatically. Once the power flow is created, the data structure in Figure 1 is shown as a nested subpackage, illustrated in Figure 3.

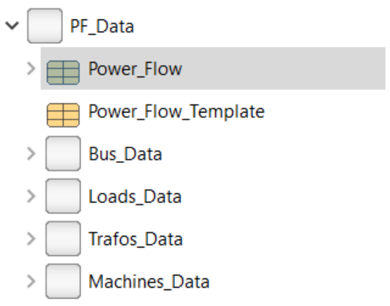


Figure 3. Power flow record structure as a nested subpackage in the model structure.

4.2 Power Flow Computation with GridCal

GridCal is a Python-based object-oriented software for the computation of power flow results. An example of using GridCal to compute power flow and Python to write the power flow solution into record is shown in Listing 2. In this case, a PSS/E `*.raw` file containing the static model information is translated into a GridCal object using the built-in parser class **FileOpen**. The `*.raw` contains the static model of the network, which is required for any power flow formulation. The `*.raw` parser allows us to benchmark the performance of GridCal against PSS/E in terms of power flow result accuracy. Furthermore, this feature reduces the cost of migrating a model from PSS/E to OpenIPSL since the user could initialize both models from the same `*.raw` file. However, the user can define their own grid models from scratch. The reader is referred to the GridCal documentation for network implementation examples.

After creating the grid object via the parser class, an instance of the **PowerFlowDriver** is declared: **pf**. **pf** is responsible for carrying out the power flow computation following user-specified settings (**options**). Recall from Equation (6) that the method for a power flow computation is constrained by the grid topology (i.e., the matrix $\mathbf{J}(\mathbf{x})$). Therefore, the **grid** object must be passed to the **PowerFlowDriver** constructor method for any power flow computation. The PSS/E `*.raw` file can store up to one power flow result. We take advantage of this fact and use that power flow as an initialization value for a base-case power flow computation in GridCal. The result of this base case should be the same power flow (within the solver's tolerance) as the one included in the PSS/E file. The power flow calculation is commanded by invoking the function **pf.run()**. The results are stored as an attribute of the **PowerFlowDriver** class.

Listing 2. Generation of power flow result using GridCal (PSS/E file input)

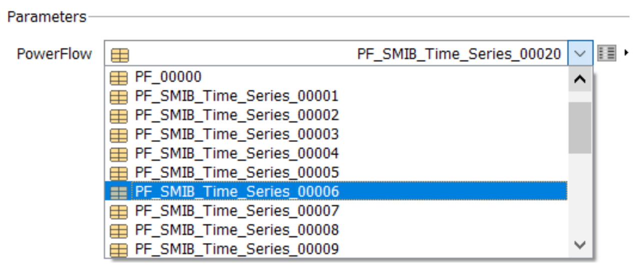
```
_wd = os.getcwd() # working directory
_model_package = 'SMIB'

# Path to the model package directory
data_path = os.path.join(_wd, _model_package)
data_path = os.path.abspath(data_path)

# Path to the PSSE `.raw` file
psse_raw_path = os.path.join(data_path, "
    PSSE_Files", "SMIB_Base_Case.raw")
psse_raw_path = os.path.abspath(psse_raw_path)

# Grid model in GridCal
file_handler = FileOpen(psse_raw_path)

# Creating grid object and setting options
grid = file_handler.open()
options = PowerFlowOptions(SolverType.NR,
    verbose = True,
    initialize_with_existing_solution = False,
    multi_core = False,
    tolerance = 1e-6,
    max_iter = 99,
    control_q = ReactivePowerControlMode.Direct)

pf = PowerFlowDriver(grid, options)
pf.run()

# Writing power flow results in records
gridcal2rec(grid = grid, pf = pf, model_name = '
    SMIB',
    data_path = data_path,
    pf_num = 0,
    export_pf_results = False)
```

Finally, the function **gridcal2rec** takes the grid information and the power flow driver information and writes the results as Modelica records, following the structure described in Section 3. The new files are placed within the `PF_Data` subfolder, housing the power flow record structure. They are also written automatically in-

[4]https://github.com/ALSETLab/SMIB_Tutorial/
and https://youtu.be/4qfKw9SAXFY

side the corresponding `package.order` file to become available to the user right after the computation is completed. The function **`gridcal2rec`** can be included in automation loops to perform a time series power flow. The resulting output is shown in Figure 4.

In Figure 5 one can notice how the power flow condition, defining several variables in a model, can be set either from the graphical interface or by redeclaring a single parameter in the text layer. To the authors' best knowledge, such a feature is not typically available in commercial power system software for dynamics. However, we can easily incorporate it into OpenIPSL models by exploiting the flexibility of object-oriented structure of the Modelica language.

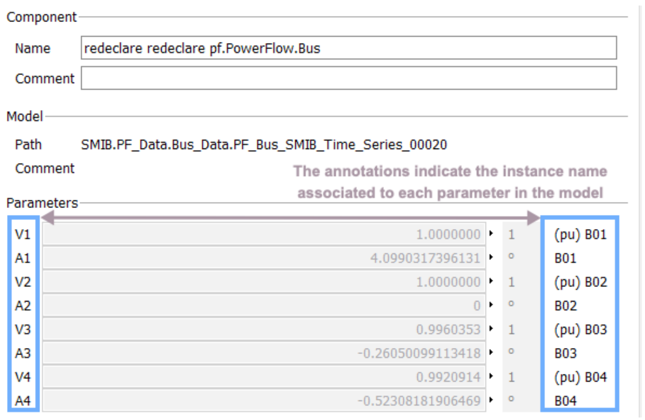

Parameters

PowerFlow: PF_SMIB_Time_Series_00020

- PF_00000
- PF_SMIB_Time_Series_00001
- PF_SMIB_Time_Series_00002
- PF_SMIB_Time_Series_00003
- PF_SMIB_Time_Series_00004
- PF_SMIB_Time_Series_00005
- PF_SMIB_Time_Series_00006
- PF_SMIB_Time_Series_00007
- PF_SMIB_Time_Series_00008
- PF_SMIB_Time_Series_00009

Figure 4. Multiple power flows within an OpenIPSL model.

A possible difficulty of our power flow generation tool is that the user must connect the power flow parameters in each device to the record manually. After several attempts, we noticed that it depended on how the user constructed a particular model, which is unpredictable. However, we included informative annotations in the record attributes to link the initialization values (see Figure 5) correctly.


Component

Name: redeclare redeclare pf.PowerFlow.Bus

Comment:

Model

Path: SMIB.PF_Data.Bus_Data.PF_Bus_SMIB_Time_Series_00020

Comment:

The annotations indicate the instance name associated to each parameter in the model

Parameters

V1	1.0000000	1	(pu) B01
A1	4.0990317396131	°	B01
V2	1.0000000	1	(pu) B02
A2	0	°	B02
V3	0.9960353	1	(pu) B03
A3	-0.26050099113418	°	B03
V4	0.9920914	1	(pu) B04
A4	-0.52308181906469	°	B04

Figure 5. Informative annotations to assist the user link the record attributes to the model correctly.

Despite this caveat, referencing of the power flow variables to the record must be done only *once*. Afterwards, the user must change the Powerflow attribute, not the record itself. Since the references point to the record on the top layer, they remain unchanged. A detailed example of this process in the tutorial[5] accompanying this paper.

[5]https://youtu.be/RMD8WEOi6r4

4.3 Scalability for Larger Models

We validated our approach in several systems of different number of buses (that defines the scale of the power flow problem) and of state variables (that defines the size of the complexity of dynamic simulation problem). Table 1 and Figure 6 summarize the characteristics of the benchmarked systems and illustrate the tool's performance in terms of execution time for record generation and power flow computation. The results correspond to the best scenario over 100 repetitions with 100 loops each. All models are available within the Application Examples of OpenIPSL.

Table 1. Scalability results on different systems

System (Buses)	*Number of Variables*		*Avg. Execution Time (over 100 loops)*	
	Algebraic	*State*	*Record Creation*	*Power Flow Computation*
SMIB (4)	99	9	4.08 ms ± 255 μs	31.6 ms ± 1 ms
IEEE 9 (9)	241	29	7.29 ms ± 287 μs	35.5 ms ± 1.55 ms
Kundur Two Areas (11)	244	20	5.07 ms ± 194 μs	37.4 ms ± 1.01 ms
AVRI (14)	16	233	5.77 ms ± 144 μs	35.9 ms ± 1.17 ms
Nordic 44 (44)	1294	6315	55.2ms ± 874 μs	349 ms ± 12.8 ms

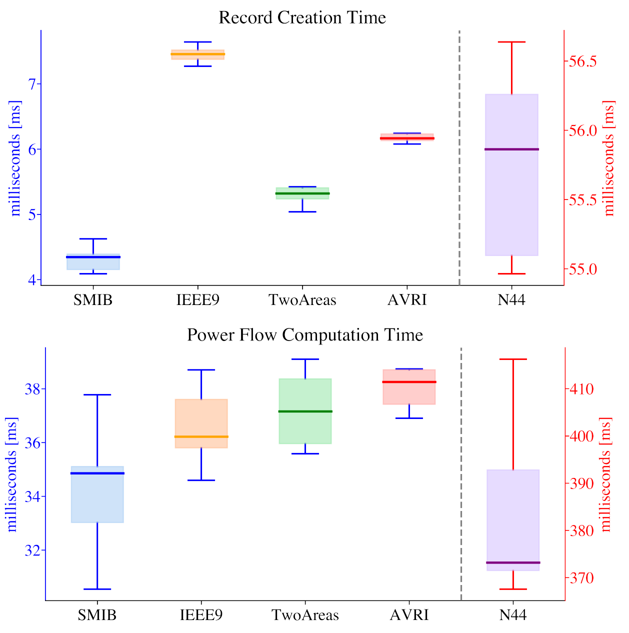


Figure 6. Execution time for record creation (top) and power flow computation (bottom). Observe that the results for the N44 are presented on a different scale.

Table 2. Power Flow Comparison between PSS/E and GridCal

System	*Bus*	*Voltage*						*Power*					
		Magnitude [pu]		*Absolute Error*	*Angle [deg]*		*Absolute Error*	*P [MW]*		*Absolute Error*	*Q [Mvar]*		*Absolute Error*
		PSS/E	*GridCal*		*PSS/E*	*GridCal*		*PSS/E*	*GridCal*		*PSS/E*	*GridCal*	
SMIB	1	1.0000	1.0000	-9.99×10^{-16}	4.04628	4.04627	-2.24×10^{-6}	40.000	40.000	0.00000	5.417	5.417	3.66×10^{-8}
	2	1.0000	1.0000	0.00000	0.00000	0.00000	0.00000	10.017	10.017	5.63×10^{-6}	8.007	8.007	3.83×10^{-8}
IEEE9	1	1.0400	1.0400	0.00000	0.00000	0.00000	0.00000	71.613	71.613	1.11×10^{-5}	25.592	25.592	4.12×10^{-6}
	2	1.0300	1.0300	0.00000	9.18220	9.18219	-4.36×10^{-6}	163.000	163.000	0.00000	8.925	8.925	-3.69×10^{-6}
	3	1.0250	1.0250	0.00000	4.64766	4.64766	-2.20×10^{-6}	85.000	85.000	0.00000	-12.503	-12.503	-1.23×10^{-5}
Two Areas	1	1.0300	1.0300	0.00000	27.07087	27.07086	-7.19×10^{-6}	700.000	700.000	0.00000	185.035	185.035	-2.56×10^{-5}
	2	1.0100	1.0100	0.00000	17.30648	17.30647	-7.33×10^{-6}	700.000	700.000	0.00000	234.624	234.624	-2.10×10^{-5}
	3	1.0300	1.0300	0.00000	0.00000	0.00000	0.00000	719.095	719.095	-2.58×10^{-5}	176.040	176.040	2.24×10^{-5}
	4	1.0100	1.0100	-9.99×10^{-15}	-10.19216	-10.19215	1.09×10^{-5}	700.000	700.000	0.00000	202.114	202.114	-4.49×10^{-5}
AVRI	1	1.0500	1.0500	0.00000	0.00000	0.00000	0.00000	-100.000	-100.000	0.00000	41.391	41.391	-4.25×10^{-6}
	8	1.0500	1.0500	0.00000	47.01978	47.01976	-1.89×10^{-5}	50.000	50.000	0.00000	19.795	19.795	-7.89×10^{-7}
	12	1.0500	1.0500	0.00000	43.26172	43.26170	-2.11×10^{-5}	50.000	50.000	0.00000	21.916	21.916	-4.04×10^{-6}
N44	3115	1.0000	1.0000	0.00000	-13.59220	-13.59220	1.12×10^{-6}	1114.875	1114.875	0.00000	-395.702	-395.702	1.37×10^{-5}
	6000	1.0050	1.0050	0.00000	-18.37864	-18.37864	-2.86×10^{-6}	1010.808	1010.808	0.00000	-400.800	-400.780	1.97×10^{-2}
	6500	1.0000	1.0000	0.00000	-25.88593	-25.88593	-3.62×10^{-6}	1093.284	1093.284	0.00000	882.375	882.375	-1.36×10^{-4}
	8500	1.0200	1.0200	-9.99×10^{-15}	-5.72443	-5.72443	5.95×10^{-7}	1952.664	1952.664	0.00000	596.683	596.683	2.58×10^{-4}

The Record Creation (RC) process is 5–7x faster than the power flow computation, as expected[6]. Both procedures scale up with the number of algebraic variables, directly related to the dimensionality of the power flow problem. Notice that increase in execution time to generate the records shows an exponential trend with respect to the size of the power flow problem (Figure 7), as expected.

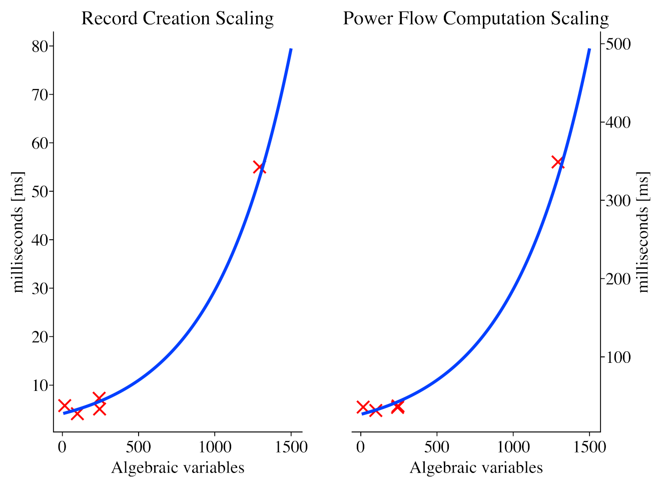


Figure 7. Exponential increase in execution time as a function of the number of algebraic variables in the model.

4.4 Result Validation with PSS/E

The validation against PSS/E of the power flow results obtained using GridCal has been performed on several test systems. In Table 2, a list of the tested networks is given. For each of the networks some buses have been selected indicating their voltage magnitude and angle, the injected/absorbed active and reactive powers of the generating units connected to the corresponding node. Those power flow results are compared with the corresponding calculations obtained from PSS/E including evaluation of an absolute error between the evaluated power flow and reference PSS/E power flow. The power flow values match with low tolerance errors that in some cases hit the machine precision. This shows the validity of the proposed approach of power flow calculation using the open-source Python library GridCal.

[6]The experiments were performed on an Intel Core i5 Quad-Core (2.0 GHz) processor, with 16 GB RAM DDR4 memory.

5 Conclusions

This article presents an approach to form a record-based data structure to handle power flow starting guesses for a dynamic simulation using the phasor-domain OpenIPSL library. A power flow computation, performed before running a phasor-domain simulation, specifies the starting equilibrium of the nonlinear system simulation. The record class architecture benefits directly from the object-oriented paradigm of the Modelica language, allowing management of *all* power flow variables from a *single* attribute in the model, a feature not common in specialized proprietary power system tools. Such structure can be extrapolated to other open-source Modelica-based power system libraries.

We provide a Python script to create the structure for any existing OpenIPSL model built on versions 1.5.0 or 2.0.0, in this way, naturally expanding capabilities of the library to perform dynamic simulations for different power flow initial conditions. The power flow record instances can be populated by an open-source Python library called GridCal, capable of producing numerically the same results as PSS/E for power flow computations. We also introduce a script to convert the GridCal power flow results to records directly.

From our perspective, the proposed methodology can be useful for users of existing OpenIPSL models, especially for those who study the behavior of the models under different power flow conditions. However, for large scale models the user would have to spend significant time linking the power flow variables to the record. To avoid the aforementioned issue, a model translation tool that translates the information from PSS/E `*.dyr` and

`*.raw` files into OpenIPSL `*.mo` models is currently under development. The tool will include the proposed record structure in this paper by default. In that case, the power flow variables will point to the record automatically. This will be a key advantage in helping power system analysts with the potential adoption and transition to Modelica-based tools.

Acknowledgements

The authors would like to thank Fernando Fachini, Ricardo Rincón Ballesteros, and Manuel Navarro Catalán for their support and feedback with the implementation of the software. We would also like to thank Santiago Peñate Vera for developing GridCal.

This research was funded in part by the New York State Energy Research and Development Authority (NYSERDA) through the Electric Power Transmission and Distribution (EPTD) High Performing Grid Program under agreement numbers 137940, 137951, in part by the Engineering Research Center Program of the National Science Foundation and the Department of Energy under Award EEC-1041877, in part by the CURENT Industry Partnership Program, in part by the Center of Excellence for NEOM Research at King Abdullah University of Science and Technology, and in part by Dominion Energy.

Parts of this material are based upon work supported by the U.S. Department of Energys Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

Legal Disclaimer

The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government, nor of any of the funding bodies listed in the acknowledgement.

References

Baudette, Maxime et al. (2018-01). “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. DOI: 10.1016/j.softx.2018.01.002.

Boersma, S. et al. (2020-09). “Enhanced Power System Damping Estimation via Optimal Probing Signal Design”. In: *2020 22nd European Conference on Power Electronics and Applications (EPE'20 ECCE Europe)*. IEEE, pp. 1–10. DOI: 10.23919/EPE20ECCEEurope43536.2020.9215892.

Dorado-Rojas, Sergio A., Marcelo de Castro Fernandes, and Luigi Vanfretti (2020). “Synthetic Training Data Generation for ML-based Small-Signal Stability Assessment”. In: *2020 IEEE SmartGridComm*.

Dorado-Rojas, Sergio A., Manuel Navarro Catalán, et al. (2020-11). “Performance Benchmark of Modelica Time-Domain Power System Automated Simulations using Python”. In: *Proceedings of the American Modelica Conference 2020*.

Gomez, Francisco J. et al. (2018). “Multi-Domain Semantic Information and Physical Behavior Modeling of Power Systems and Gas Turbines Expanding the Common Information Model”. In: *IEEE Access* 6, pp. 72663–72674. DOI: 10.1109/ACCESS.2018.2882311.

Gonzalez-Torres, J.C. et al. (2019). “Power system stability enhancement via VSC-HVDC control using remote signals: application on the Nordic 44-bus test system”. In: *15th IET International Conference on AC and DC Power Transmission (ACDC 2019)*. Institution of Engineering and Technology, 78 (6 pp.)–78 (6 pp.) ISBN: 978-1-83953-007-4.

Gusain, Digvijay, Milos Cvetkovic, and Peter Palensky (2019-06). “Energy Flexibility Analysis using FMUWorld”. In: *2019 IEEE Milan PowerTech*. IEEE, pp. 1–6. ISBN: 978-1-5386-4722-6. DOI: 10.1109/PTC.2019.8810433.

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019-02). “DAE Solvers for Large-Scale Hybrid Models”. In: pp. 491–502. DOI: 10.3384/ecp19157491.

Laera, Giuseppe (2016). *Modelica Norwegian Grid Models for iTesla and Model Validation Tasks*. Tech. rep.

Milano, Federico (2009). “Continuous Newton's method for power flow analysis”. In: *IEEE Transactions on Power Systems* 24.1, pp. 50–57.

Müller, Joscha et al. (2020-11). “A Modelica Library for Continuous and Discrete Extremum Seeking for Static and Dynamic Systems”. In: pp. 36–45.

Nohac, Karel et al. (2019-05). “Open Source Platforms for Dynamic Stability Assessment”. In: *2019 20th International Scientific Conference on Electric Power Engineering (EPE)*. IEEE, pp. 1–6. ISBN: 978-1-7281-1334-0.

Pan, Kaikai, Digvijay Gusain, and Peter Palensky (2019-01). “Modelica-Supported Attack Impact Evaluation in Cyber Physical Energy System”. In: *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, pp. 228–233. ISBN: 978-1-5386-8540-2.

Podlaski, Meaghan et al. (2020-11). “Parameter Estimation of User-Defined Control System Models for Itaipú Power Plant using Modelica and OpenIPSL”. In: pp. 139–148. DOI: 10.3384/ecp20169139.

Qin, Yining et al. (2019-08). “A JModelica.org Library for Power Grid Dynamic Simulation with Wind Turbine Control”. In: *IEEE Power and Energy Society General Meeting*. Vol. 2019-Augus. IEEE, pp. 1–5. ISBN: 9781728119816.

Stott, Brian (1974). “Review of load-flow calculation methods”. In: *Proceedings of the IEEE* 62.7, pp. 916–929.

Tinney, William F and John W Walker (1967). “Direct solutions of sparse network equations by optimally ordered triangular factorization”. In: *Proceedings of the IEEE* 55.11, pp. 1801–1809.

Vanfretti, Luigi et al. (2017-04). “An open data repository and a data processing software toolset of an equivalent Nordic grid model matched to historical electricity market data”. In: *Data in Brief* 11, pp. 349–357. ISSN: 23523409.

Ward, J B and H W Hale (1956). “Digital computer solution of power-flow problems [includes discussion]”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 75.3, pp. 398–404.

Winkler, Dietmar (2017-09). “Electrical Power System Modelling in Modelica - Comparing Open-source Library Options”. In: pp. 263–270. DOI: 10.3384/ecp17138263.

Winkler, Dietmar (2019-02). “Analysing the stability of an Islanded hydro-electric power system”. In: pp. 103–111. DOI: 10.3384/ecp18154103.