Automated Generator Model Calibration with PredictiveGrid

Chen Wang, Kevin D. Jones, Chetan Mishra (Dominion Energy)
Luigi Vanfretti, Giuseppe Laera, Marcelo de Castro (RPI)
Project Overview


• Project aims at using streaming synchrophasor data on PredictiveGrid platform to automatically calibrate modularized generator models including controllers.

• Generator models are built using Modelica and exported using the FMI standard.

• Python and Jupyter Notebook to combine the data query and the optimized model parameters calibration process.
Dominion’s Needs for Model Calibration

- Dominion uses the same models used for planning and control design

- **Modeling challenges**
  - Conventional model validation require events happening but system mostly in ambient conditions.
  - Operation conditions change throughout the day due to changing nature of load, line switching, V setpoint change, etc.
  - Existing model needs to be updated due to unmodeled dynamics.
  - Difficult to do when models and data are segregated.

- **Vision: Data-driven modeling with PredictiveGrid and Modelica**
  - Quickly accessible synchrophasor data.
  - Portable model modules for various generator stations with enhanced functionalities to match to data (linearization).
  - Quickly do model validation and calibration “on-demand” to support planning and operation tasks.
PredictiveGrid
Envisioned Toolchain (Design)

Components Model Library (OpenIPSL + RPI development)
  Generator: GENROU, GENROE, ...
  AVR: AC7B, AC8B, ESST4B, ESST1A, ...
  PSS: PSS2A, PSS2B, ...
  Turbine Governor: IEEEGL, GAST, ...
  Compensator: IEEEVC, ...

Library of FMUs for Dominion generators
  FMU (Gen A at Dominion)
  FMU (Gen B at Dominion)
  FMU (Gen C at Dominion)

Python Library for Model Calibration
  PredictiveGrid Data Adaption
  Signal Pre-Processing
  FMU Simulation
  Simulation Results Comparison
  Parameter Calibration (optimization process)

Jupyter Notebooks (Model Calibration)
  PredictiveGrid: Synchrophasor Data Input (Ambient Data, Small Disturbances, Large Transients)
  Signal Processing
  FMU Parameters Calibration
  FMU Parameters Output

Jupyter Notebooks (Dominion Application)
  PredictiveGrid: Synchrophasor Data Query

Form Up
Call FMU
Executable python functions
Calibrated Generator Model
The Modelica Language and the OpenIPSL Library for Power System Modeling and Simulation

- Non-proprietary, object-oriented, equation-based modeling language for cyber physical systems.
- Open access (no paywall) & standardized language specification (link), maintained by the Modelica Association
- Open source Modelica Standard Library with more than 1,600 components models.
- Supported by 9 tools natively, both proprietary (Dymola, Modelon Impact, etc.) and Open Source (OpenModelica)
- A vast number of proprietary and open-source Modelica Libraries

OpenIPSL is an open-source Modelica library for power systems that:
- Contains a vast number of power system components for phasor time domain modeling and simulation of power systems (transmission and distribution)
- Several models have been verified against a number of reference tools (PSS/E, PSAT).

OpenIPSL enables:
- Unambiguous model exchange, use of model in Modelica-compliant tools.
- Formal mathematical description, no discretization w.r.t. specific integration method.
- Separation of models from tools and solvers.
- Using Dymola, as fast* as PSS/E (link).
OpenIPSL Library and Example

```
equation
// Interfacing outputs with the internal variables
XADIFD = Xadifd
ISORCE = Xadifd
EFD0 = efd0
FMECH0 = pm0

\[
\frac{dE_{pq}}{dt} = -\frac{1}{T_{pd0}} \left( EFD - XadIFD \right)
\]
\[
\frac{dE_{qd}}{dt} = -\frac{1}{T_{pq0}} \left( Xaqllq \right)
\]
\[
\frac{dPSIkd}{dt} = -\frac{1}{T_{pd0}} \left( E_{pq} - PSIkd - \left( Xpd - Xl \right) id \right)
\]
\[
\frac{dPSIqk}{dt} = -\frac{1}{T_{pq0}} \left( E_{pd} - PSIqk + \left( Xpq - Xl \right) id \right)
\]

\[
Te = PSIq - PSIq \cdot id
\]
\[
PSId = PSippd - Xppd \cdot id
\]
\[
PSIq = \left( -PSippq \right) - Xppq \cdot iq
\]
\[
PSippd = E_{pq} \cdot K3d + PSIId \cdot K4d
\]
\[
-PSippq = \left( -E_{pq} \cdot K3q \right) - PSIq \cdot K4q
\]
\[
PSipp = \sqrt{PSippd} \cdot PSippd - PSIppq \cdot PSIppq
\]
\[
Xadifd = K1d \cdot \left( E_{pq} - PSIkd - \left( Xpd - Xl \right) id \right) - E_{pq}
\]
\[
Xaqllq = K1q \cdot \left( E_{pd} - PSIqk + \left( Xpq - Xl \right) id \right) - E_{pd}
\]

// change sign for PSippq 3/3
ud = \left( -PSIq \right) - \left( Xl \right) id
uc = PSIId - \left( Xl \right) iq
```

// flow

```
+ id \cdot \left( Xd - Xpd \right) + SE_{exp} \left( PSIpp, S10, S12, 1, 1, 2 \right) PSIppd
-iq \cdot \left( Xq - Xpq \right) - SE_{exp} \left( PSIpp, S10, S12, 1, 1, 2 \right) \left( -1 \right) PSIippq \cdot \left( Xq - Xl \right)
.Xd - Xl
```

The Functional Mockup Interface Standard

- **FMI is an open access standard**, also from the Modelica Association.
- It defines a container and an interface to *exchange dynamic models* using a combination of XML files, binaries and C code zipped into a *single file, called a Functional Mock-up Unit (FMU) or .fmu*.

- **Supported by simulation 100+ tools!**

- FMI supports model export in two modes Co-Simulation (CS) and Model Exchange (ME)
  - With a Model Exchange FMU, the numerical solver is supplied by the importing tool. The solver in the importing tool will determine what time steps to use, and how to compute the states at the next time step.
  - With a Co-Simulation FMU, the numerical solver is embedded and supplied by the exporting tool. The importing tool sets the inputs, tells the FMU to step forward a given time, and then reads the outputs.
Integrating Models in PredictiveGrid

- **Challenge:** Typical generator plant models are isolated in simulation tool (PSS/E):
  - Limited to in-built capabilities of the tool
  - Not possible to deploy existing PSS/E model in PredictiveGrid platform.

- **Solution:** use Modelica and FMI to create a portable model! *However, the models needed were not available in OpenIPSL.*

- **Approach:**
  - Implement the model in Modelica and verify against PSS/E.
  - If results are the same, export Modelica model as an FMU
  - Deploy model in platform and build toolchain for model calibration:
    - Use Python functionalities to integrate the model.
    - Use Python and Jupyter notebooks to build calibration “notebook”

- SW-to-SW verification of the plant model (PSS@E vs. Modelica)

- Export Modelica model as FMU with source code

- Predictive Grid Integration:
  - Import measurements data
  - Implement signal processing of PMU data
  - Integrate the FMU by coupling model I/O data
  - Integrate tools for model calibration, i.e. optimization-based parameter estimation.

- Manually Update PSS/E Model Data (Could also be automated)
Models for Software-to-Software Verification

Plant configuration of the reference PSS@E model

<table>
<thead>
<tr>
<th>Plant Name</th>
<th>Generator</th>
<th>AVR</th>
<th>PSS</th>
<th>Governor</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC ST01</td>
<td>GENROE</td>
<td>ESST1A</td>
<td>PSS2A</td>
<td>IEEEG1</td>
</tr>
</tbody>
</table>

SMIB test system diagram in PSS@E (GEN01 = WC ST01)

Modelica Implementation using the OpenIPSL Library
Verification: Modelica (Dymola SW) vs PSS/E

Test: 3-phase fault to ground applied to bus FAULT of the test system at t=2sec for 0.15sec
Modelica Model for PMU-data Replay and FMI Export

- Model configuration of WC ST01 for FMU export:

**Legend**

1. Record with system data
2. Blocks with power flow data as a parameter.
3. Controlled voltage source
4. Generator model (GENROE)
5. Turbine Governor model (IEEEG1)
6. Power System Stabilizer model (PSS2A)
7. Automatic Voltage Regulator model (ESST1A)
8. Model interfaces giving the output active and reactive power of the generator (4)
9. Inputs for measurements
Modelica/FMI Model Calibration:

- **ModestPy** is an Open Source Python tool for parameter estimation.

- Developed by the University of Southern Denmark, compatible with Python 3 and possible to use in Linux (platform requirement).

- It facilitates parameter estimation in models compliant with Functional Mock-up Interface (FMI) standard. That means it works with both CS and ME FMUs!

- It uses a combination of global and local search methods (genetic algorithm, pattern search, truncated Newton method, L-BFGS-B, sequential least squares) that can be applied in a sequentially.

- For our proof-of-concept we have used a Co-Simulation FMU of the plant exported with source code to allow for its use on the platform.
  - The CS FMU showed a more stable behavior on the PingThings platform.
**Signal Processing**

Data is retrieved
- PMU stream is selected
- Time window is selected
- Sampling frequency is determined

Data is prepared
- Data passes a high pass filter (very low frequencies removed)
- Data passes a low pass filter (noise)
- Data is resampled (match time step of solver)

Final Signals for Model Coupling
- Current and voltage magnitudes and angles become phasors in per unit
- Calculated, positive sequence V, I, P and Q.
- Real and imag. parts of voltage are extracted
Model and Toolchain Integration

- Import a specific user defined library for connection to the platform and retrieve data
- Import standard Python modules for mathematical calculations, data processing and ModestPy tool after its installation
- Instantiation of the FMU
- Defining inputs/outputs after signal processing
- Defining parameters to be estimated
- Defining estimation algorithms and settings

```python
from Chetan_11h02 import *
conn = btrc6.connect("internal.api.dominion.predictivegrid")

# Instantiate FMU
fmu_file = 'NC_5011.fmu'
model = Model(fmu_file)

# Load definition of estimated parameters (name, default value, bounds)
est = {'t5SST1.K.A':(1.261,1.5),
'w5SST1.T.A':(1.44-0.06,0.0,0.0003),
'PK':((18300000.0,1830000.0,1830000.0),
'QK':((28000000.0,28000000.0,28000000.0))}
```
Testing: Parameter Estimation Under Ambient Conditions

- After a linear analysis of the plant, it has been noticed that the exciter could contribute to the anomalous behavior.
- Therefore, an estimation of the voltage regulator gain $K_a$ and time constant $T_a$ and the steady state active ($P_0$) and reactive power ($Q_0$), has been performed for ambient conditions.

**GA Algorithm**

**Nelder-Mead**

**Error**

estimation elapsed time $\approx 1431$s

Sequence of algorithms used for the estimation
Testing: Parameter Estimation Under a Transient

- The estimation of the voltage regulator gain $K_a$ and time constant $T_a$, active ($P_0$) and reactive power ($Q_0$), has been performed for transient conditions.

**GA algorithm**

**Nelder-Mead**

**Error**

estimation elapsed time $\approx 447s$

Sequence of algorithms used for the estimation
Proof-of-Concept: Parameter Estimation Results for 4 parameters

- From the results, the exciter gain $K_a$ (uncalibrated value 160) keeps a value of the same order of magnitude in both scenarios whereas the time constant $T_a$ (uncalibrated value 0.029s) has a difference of several orders of magnitude.
- More parameters for different parts of the model need to be included (e.g. turbine, PSS, etc).
- More scenarios and different combinations of parameters will be tested since the preliminary results could also be affected by correlation between parameters.

Calibration for Ambient

Calibration for Transient Event

estimates

\{'eSST1A1.K_A': 2.6426506177827225, 'eSST1A1.T_A': 0.24553453810083333, 'P0.K': 182750836.3976619, 'Q0.K': -34367843.8675238\}

estimates

\{'eSST1A1.K_A': 1.0379577408185557, 'eSST1A1.T_A': 2.189549442699525e-06, 'P0.K': 183852847.85619336, 'Q0.K': -26708844.37074689\}
Conclusions and Future Work

● **Open access, standards-based, portable and reusable modeling using Modelica and FMI:**
  ○ Open access, interoperable standards for modeling exchange provide model portability → new implemented models in OpenIPSL can now be used by Dominion (and others!) for multiple tasks.
  ○ Modelica and FMI standards provide great benefits for integration with modern platforms (e.g. cloud).
  ○ Model portability provides the flexibility to perform any type of simulation analysis without a specific tool dependency.

● **PredictiveGrid Platform:**
  ○ Availability of Python tools (i.e. ModestPy), allowed for quickly prototyping a new solution.
  ○ Custom Python routines for signal processing to couple models with data were also implemented.
  ○ This new prototype has helped identify feature enhancements and new functionalities needed in the platform to facilitate quicker development of new applications (e.g. AWS instance resources for optimization).

● **Proof of concept successfully implemented:**
  ○ Results show great promise for automation for model calibration within a synchrophasor utility platform.
  ○ Provides a framework that can be generalized for any other generator stations, FACTS devices, etc.
  ○ Open source tools (i.e. ModestPy) minimized development effort (no need to reinvent the wheel!)
  ○ Need to develop methods and tools for parameter selection and correlation analysis.

● **Future work:** enhance prototype and expand coverage for other stations in Dominion’s grid; implement new applications based on the developed models.
Thank you!