

Deploying and Calibrating Power Plant Models in a Cloud-Based Synchrophasor Platform

Luigi Vanfretti, PhD

Associate Professor

Rensselaer Polytechnic Institute

<http://ALSETLab.com>, luigi.vanfretti@gmail.com



ALSET *lab*

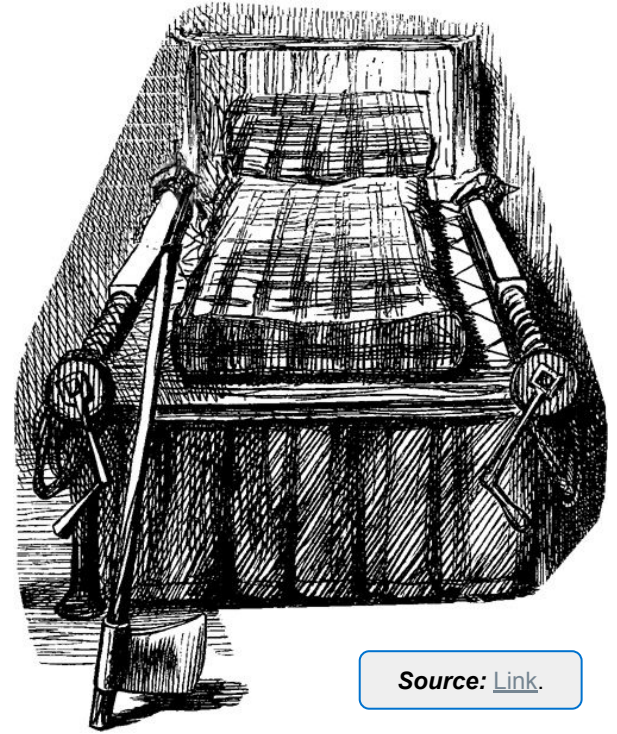
Contributors:

RPI, ALSETLab - Giuseppe Laera, Marcelo de Castro,
Dominion Energy - Chen Wang, Chetan Mishra, Kevin D. Jones



Overview

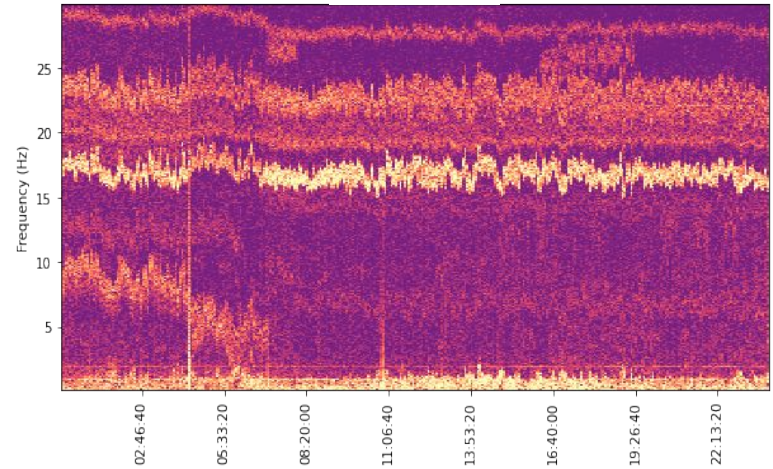
- Dominion Energy's needs for model calibration
- Background:
 - Modelica, OpenIPSL and the **FMI Standard**
- Developing a Cloud-Based Proof-of-Concept Prototype
 - Requirements
 - Model implementation in Modelica
 - Model verification vs. PSS/E
 - Model variant for calibration
 - Signal processing
 - Parameter estimation
 - Results
- Conclusions



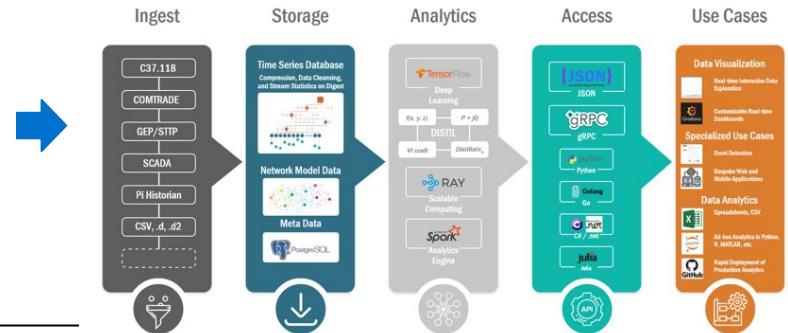
Source: [Link](#).

Dominion's Needs for Model Calibration

- Dominion's models for planning are used for operations analysis, forensics and control settings.
- 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:** Cloud-based Data-assisted modeling with Modelica-based technologies
 - Quickly accessible synchrophasor data using *PredictiveGrid™*
 - **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.



Voltage Magnitude Spectrogram at Unmodeled Generating Unit



Predictive Grid Synchrophasor Platform

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](#)

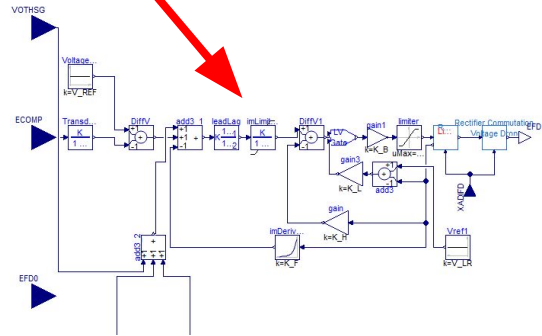
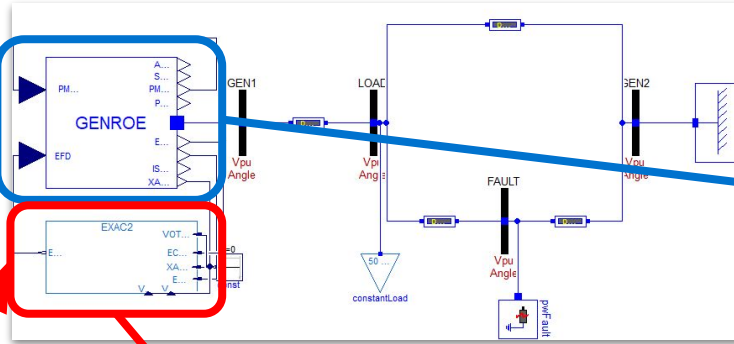


- [OpenIPSL](#) is an open-source Modelica library for power systems that:
 - Contains a vast number of power system components for phasor time domain modeling and simulation of power systems (transmission and distribution)
 - Several models have been verified against a number of reference tools ([PSS/E](#), PSAT).
- **OpenIPSL enables:**
 - **Unambiguous model exchange**, use of model in Modelica-compliant tools, and export with FMI.
 - **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



- ✓ OpenIPSL
 - Copyright
 - Examples
 - Electrical
 - Machines
 - PSAT
 - PSSE
 - GENSAL
 - GENROU
 - GENROE
 - Controls
 - PSAT
 - Simulink
 - PSSE
 - OEL
 - ES
 - ESST4B
 - EXAC2



```
//Interfacing outputs with the internal variables
XADIFD = Xadfd
ISORCE = Xadfd
EFD0 = efd0
PMECH0 = pm0
```

$$\frac{d Epq}{dt} = \frac{1}{Tp d0} (EFD - Xadfd)$$

$$\frac{d Epd}{dt} = \frac{1}{Tp q0} (-1) Xa q l l q$$

$$\frac{d PSIk d}{dt} = \frac{1}{Tpp d0} (Epq - PSIk d - (Xpd - Xl) id)$$

$$\frac{d PSIk q}{dt} = \frac{1}{Tpp q0} (Epd - PSIk q + (Xpq - Xl) iq)$$

$$Te = PSId \cdot iq - PSIk \cdot id$$

$$PSId = PSIp d - Xpp d \cdot id$$

$$PSIk = (-PSIp q) - Xpp q \cdot iq$$

$$PSIp d = Epq \cdot K3 d + PSIk d \cdot K4 q$$

$$-PSIp q = (-Epd \cdot K3 q) - PSIk q \cdot K4 q$$

$$PSIp q = \sqrt{PSIp d \cdot PSIp d + PSIp q \cdot PSIp q}$$

$$Xadfd = K1 d \cdot (Epq - PSIk d - (Xpd - Xl) id) + Epq$$

$$Xa q l l q = K1 q \cdot (Epd - PSIk q + (Xpq - Xl) iq) + Epd$$

```
//change sign for PSIp q 3/3
ud = (-PSIk) - Ra \cdot id
uq = PSId - Ra \cdot iq
//flow
```

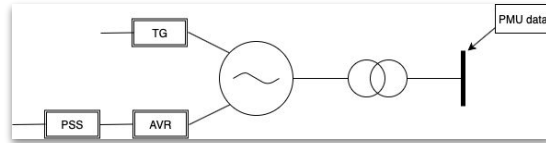
```
end GENROE
```

$$\begin{cases} + id \cdot (Xd - Xpd) + SE_exp(PSIp, S10, S12, 1, 1.2) PSIp d \\ - iq \cdot (Xq - Xpq) - \frac{SE_exp(PSIp, S10, S12, 1, 1.2) (-1) PSIp q \cdot (Xq - Xl)}{Xd - Xl} \end{cases}$$

Note: Modelica uses object-orientation, so the swing equations are not shown because they are inherited from a base class, as they are the same for all models.

Developing a Cloud-Based Proof-of-Concept Prototype

- **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 deploy the model in platform.
 - Use Python and Jupyter notebooks to build calibration “notebook” in cloud platform.

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

Export Modelica model as FMU *with source code*



Predictive Grid Integration:

- Query measurements data
- Implement signal processing of PMU data
- Couple the model (FMU) with PMU data
- Integrate tools for model calibration, i.e. optimization-based parameter estimation.

Manually Update PSS/E Model Data
(Could also be automated)

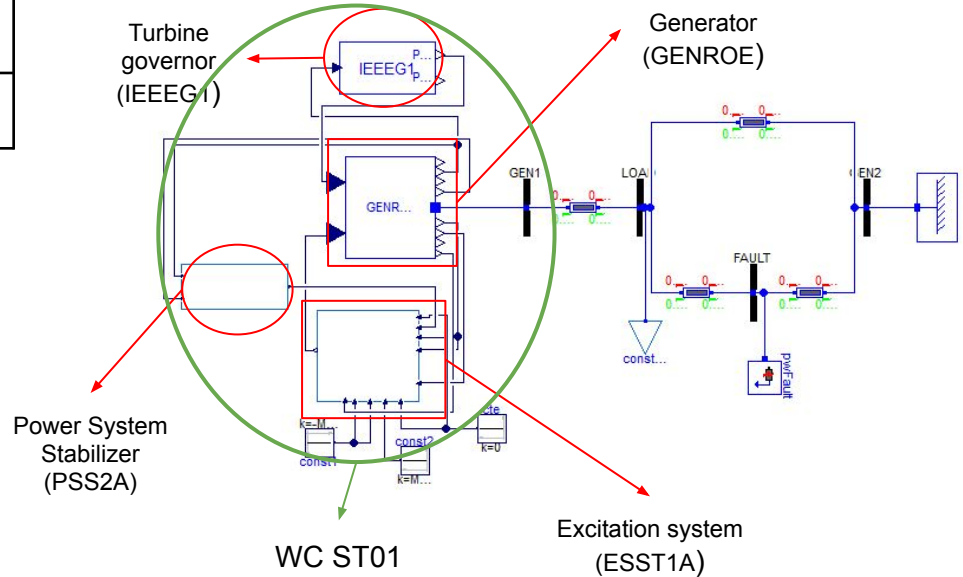
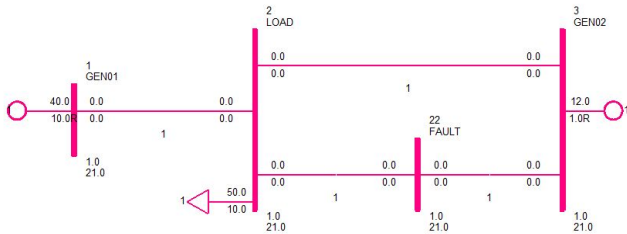
Models for Software-to-Software Verification

Plant configuration of the reference PSS@E model

Modelica Implementation using the OpenIPSL Library

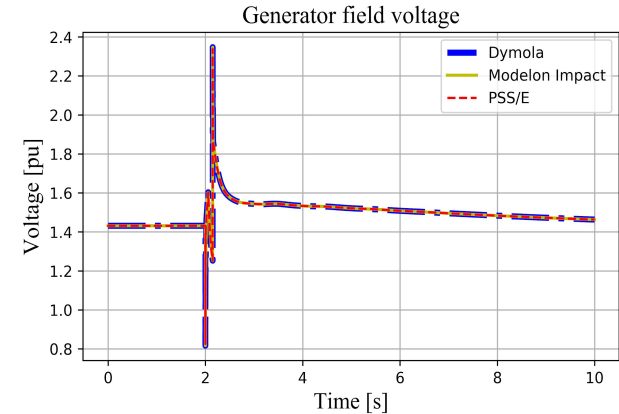
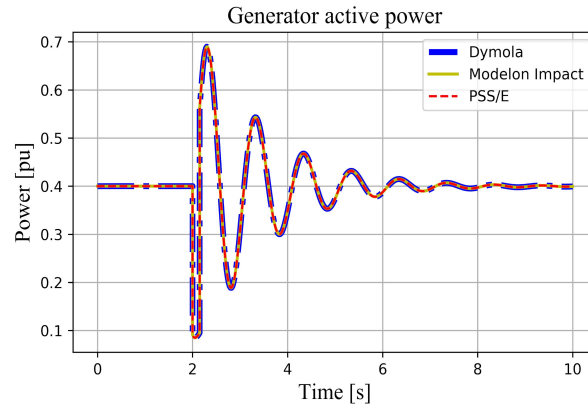
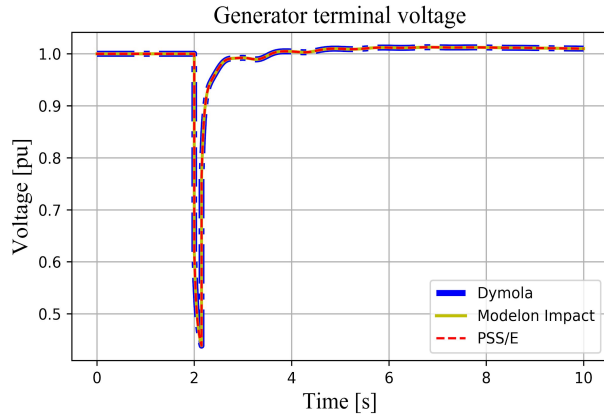
Plant Name	Generator	AVR	PSS	Turbine Governor
WC ST01	GENROE	ESST1A	PSS2A	IEEEG1

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



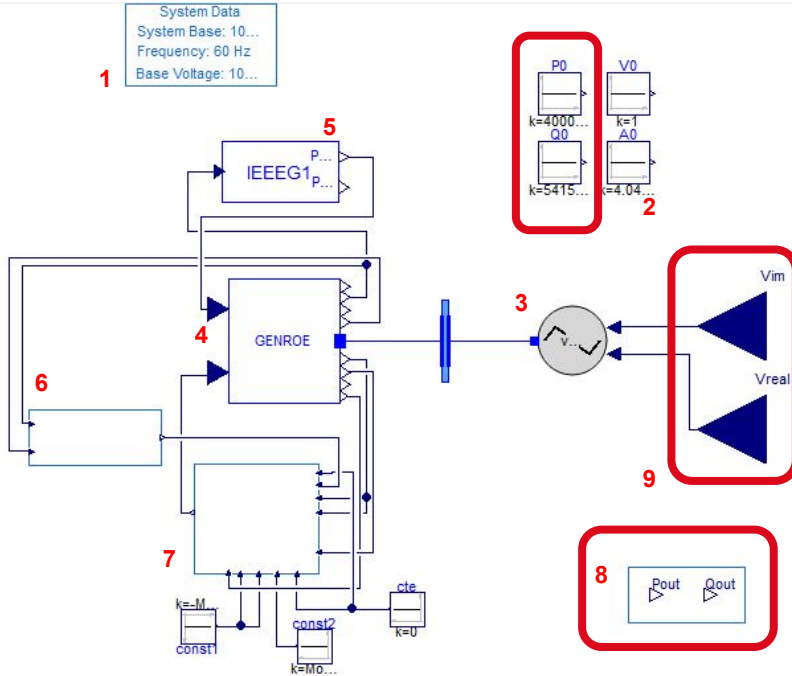
Verification: Modelica vs PSS/E

- Sample Test: 3-phase fault to ground applied to bus FAULT of the test system at t=2sec for 0.15sec
- Modelica model simulated in **2 different** Modelica-based **tools**:
 - Dymola: Modelica software tool from Dassault Systems, [link](#).
 - Modelon Impact: Modelica software tool from Modelon, [link](#).
- PSS/E: Siemens PTI, v33.
- Other verification examples online at: <https://alsetlab.github.io/NYPAModelTransformation/>



Model Variant: Coupling for PMU-data Replay and FMI Export

- Model configuration of WC ST01 for FMI export:



Legend

- Record with system data
- Blocks with power flow data as a parameter.
- Controlled voltage source
- Generator model (GENROE)
- Turbine Governor model (IEEEG1)
- Power System Stabilizer model (PSS2A)
- Automatic Voltage Regulator model (ESST1A)
- Model interfaces giving the output active and reactive power of the generator (4)
- Inputs for measurements

Coupling: Model, Measurements and Optimizer

Import a specific user defined library for connection to the platform and retrieve data

```
from Chetan_lib02 import *  
conn = btrdb.connect("internal.api.dominion.predictivegrid")
```

Import standard Python modules for mathematical calculations, data processing and ModestPy tool after its installation

```
import time  
import os  
import pandas as pd  
import numpy as np  
from modestpy import Estimation  
from modestpy.utilities.sysarch import get_sys_arch  
from modestpy.fmi.model import Model  
import matplotlib.pyplot as plt  
import seaborn as sns
```

Instantiation of the FMU

```
# Instantiate FMU  
fmu_file = 'WC_ST01.fmu'  
model = Model(fmu_file)
```

Defining inputs/outputs after signal processing

```
# Inputs  
inp = pd.DataFrame()  
t = tnew  
inp['time'] = t  
inp['Vreal'] = Vre  
inp['Vim'] = Vim  
inp = inp.set_index('time')  
  
# Load measurements (ideal results)  
ideal = pd.DataFrame()  
ideal['time'] = t  
ideal['Pout'] = P  
ideal['Qout'] = Q  
ideal = ideal.set_index('time')
```

Defining parameters to be estimated

```
# Load definition of estimated parameters (name, initial value, bounds)  
est = {'eSST1A1.K_A': (1.26, 1., 1.5),  
       'eSST1A1.T_A': (1.4e-06, 1.0e-6, 0.0001),  
       'P0.K': (183600000., 183000000., 184000000),  
       'Q0.K': (-28800000., -30000000., -28000000)}
```

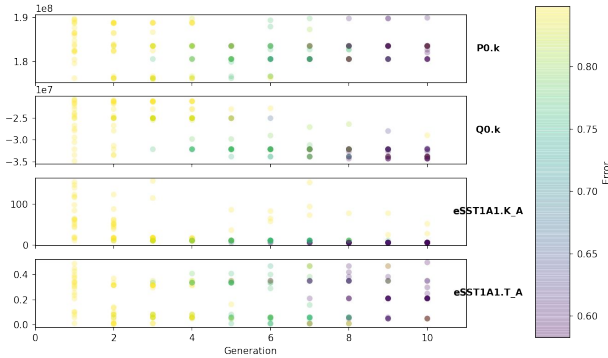
Defining estimation algorithms and settings

```
# Session  
session = Estimation(workdir, fmu_file, inp, known, est, ideal,  
                    lp_n=1, lp_len=None, lp_frame=None,  
                    vp=None,  
                    method='GA', 'SCIPY'),  
                    ga_opts={'maxiter': 10, 'tol': 1e-6, 'lhs': True},  
                    ps_opts={'maxiter': 100, 'tol': 1e-5},  
                    scipy_opts={'solver': 'Helder-Head',  
                                 'options': {'eps': 1e-6}},  
                    ftype='RMSE', seed=1,  
                    default_log=True, logfile='WC.log')
```

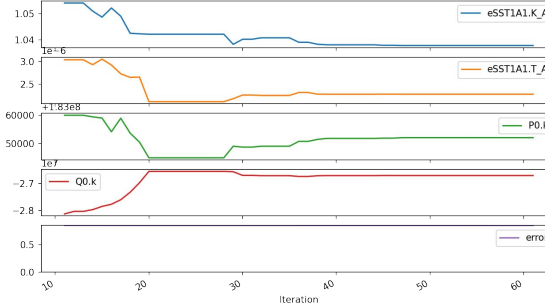
Parameter Estimation Under Ambient Conditions

- 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 **Ka** and time constant **Ta** and the steady state active (**P0**) and reactive power (**Q0**), has been performed for ambient conditions.

GA Algorithm

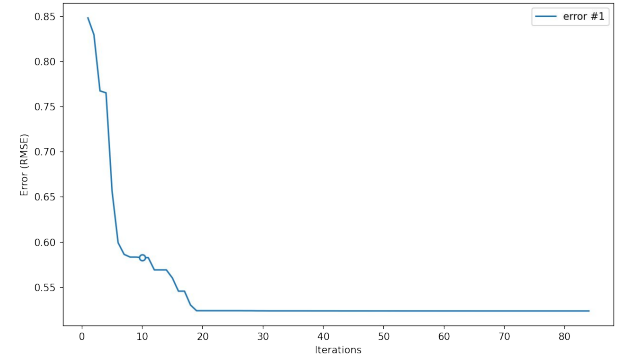


Nelder-Mead



estimation elapsed time $\approx 1431s$

Error



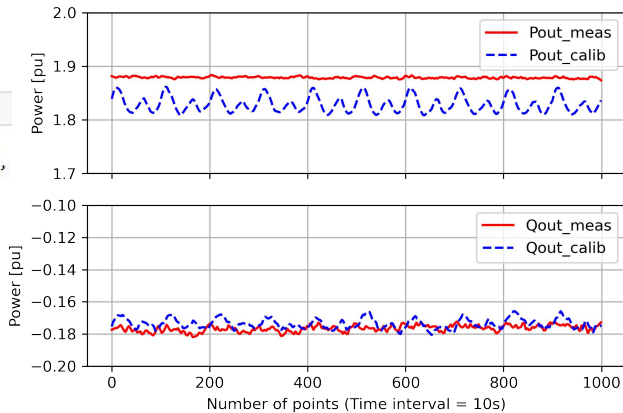
GA Nelder-Mead

Sequence of algorithms used for the estimation

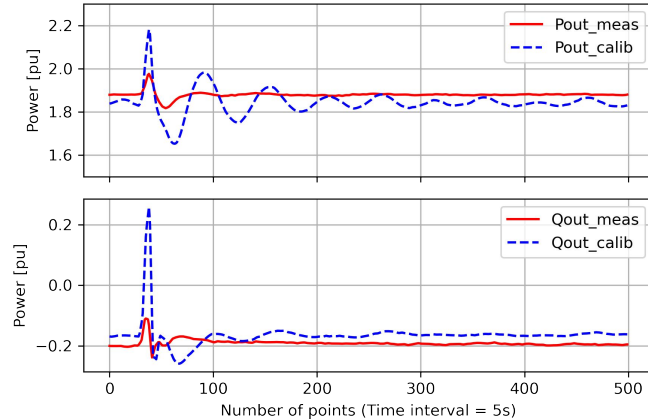
Results: Parameter Estimation Results for 4 parameters

- From the results, the exciter gain **Ka** (uncalibrated value 160) keeps a value of the same order of magnitude in both scenarios **whereas the time constant Ta (uncalibrated value 0.029s) has a difference of several orders of magnitude.**
- Current parameters being used do not represent dynamics accurately** (damped response (measurements) vs. undamped response of model):
 - More parameters for different parts of the model need to be included (e.g. turbine, PSS, etc).
 - Component models may need to be revisited (e.g. many parameters not used, modelers don't know why).
- More scenarios and different combinations of parameters will be tested since the preliminary results could also be affected by correlation between parameters:
 - Uncertainty quantification and sensitivity analysis methods need to be available in the platform.

Calibration for Ambient Conditions



Calibration for Transient Event



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.
- **Cloud-based PredictiveGrid Platform:**
 - Availability of Python tools, allowed for quickly prototyping a new solution in a cloud-based platform.
 - 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 used for optimization) reduced development effort.
 - 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.

14th *m*odelica Conference *FMI User Meeting*

Thank you!
Questions?

Luigi Vanfretti, PhD
Associate Professor
Rensselaer Polytechnic Institute
<http://ALSETLab.com>, luigi.vanfretti@gmail.com



ALSET *lab*



Extra Slides

Model Calibration: Parameter Estimation

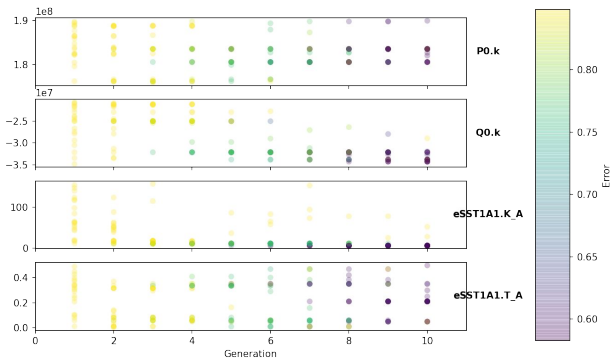


- ModestPy is an Open Source Python tool for parameter estimation.
- Developed by the University of Southern Denmark, compatible with Python 3 and possible to use in Linux (platform requirement).
- It facilitates parameter estimation in models compliant with Functional Mock-up Interface (FMI) standard. **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*

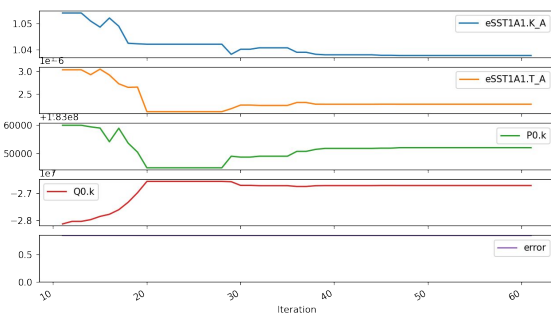
Testing: Parameter Estimation Under Ambient Conditions

- 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 **Ka** and time constant **Ta** and the steady state active (**P0**) and reactive power (**Q0**), has been performed for **ambient conditions**.

GA Algorithm

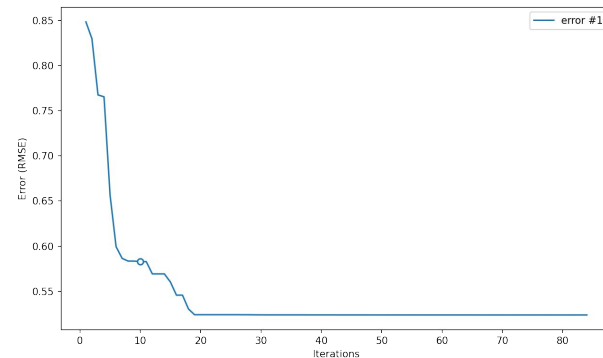


Nelder-Mead



estimation elapsed time $\approx 1431s$

Error

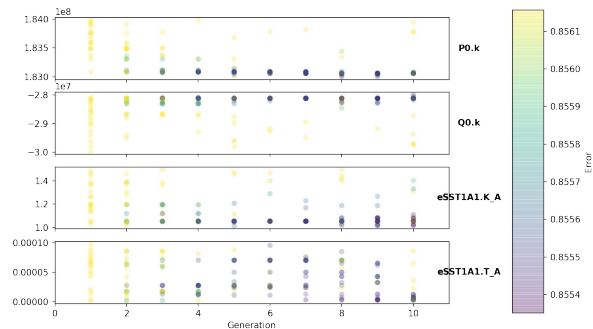


GA Nelder-Mead
Sequence of algorithms used for the estimation

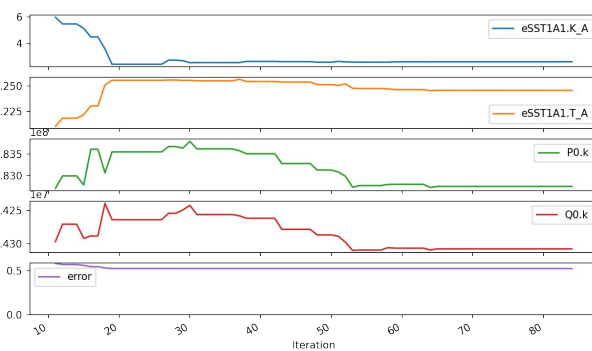
Testing: Parameter Estimation Under a Transient

- The estimation of the voltage regulator gain **Ka** and time constant **Ta**, active (**P0**) and reactive power (**Q0**), has been performed for **transient conditions**.

GA algorithm

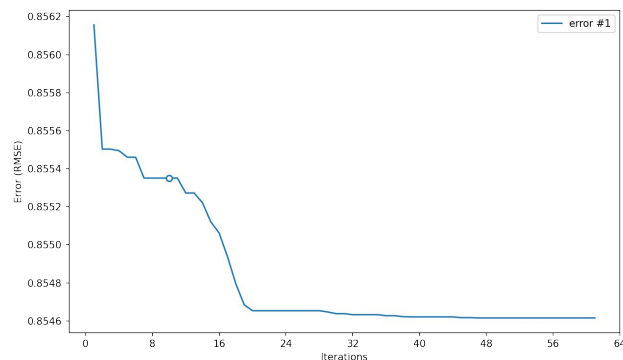


Nelder-Mead



estimation elapsed time $\approx 447s$

Error



Sequence of algorithms used for the estimation

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/or (source) 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

